Linux NET-2/3-HOWTO

Terry Dawson, terry@perf.no.itg.telecom.com.au v3.5, 16 January 1996

This document aims to describe how to obtain, install and configure the Linux NET-2 and NET-3 networking software. Some answers to some of the more frequently asked questions are included in the appendix. This document is not designed to teach you about tcp/ip networking, though some information of this kind is included where possible. Pointers to other documentation which does teach tcp/ip networking principles are listed.

Contents

1 Introduction. **5**

1.1 Changes from the previous release. 6

1.2 A brief development history of Linux Networking. 6

2 Disclaimer. **6**

3 Questions already ? **7**

4 Related Documentation. (Where to learn about tcp/ip) **8**

4.1 New versions of this document. 10

4.2 Feedback. 10

5 Some of the terms used in this document. **10**

6 NET-3 Supported functionality. **11**

6.1 General Support 11

6.2 Supported Ethernet cards. 13

6.3 Support for Amateur Radio 15

7 Getting the NET-2/NET-3 software. **15**

7.1 The kernel source. 16

7.2 The libraries. 16

7.3 The network configuration tool suite. 17

7.4 The network applications. 18

7.5 Additional drivers or packages. 19

8 Configuring the kernel. 19

8.1 What do all those funky Networking options actually do? . . . 22

9 Configuring the Network Devices. 23

9.1 Configuring the special device files in /dev . . . 23

9.2 What information do I need before I begin ? . . . 24

9.2.1 IP Address. . . . 24

9.2.2 Network Mask ('netmask'). . . . 24

9.2.3 Network Address. . . . 24

9.2.4 Broadcast Address. . . . 25

9.2.5 Router ('Gateway') Address. . . . 25

9.2.6 Nameserver Address. . . . 26

9.2.7 NOTE for SLIP/PLIP/PPP users. . . . 26

9.3 /etc/rc.d/rc.inet1,2 or /etc/rc.net or /etc/init.d/network . . . 26

9.3.1 rc.inet1 . . . 27

9.3.2 rc.inet2 . . . 27

9.4 Configuring the Loopback device (mandatory). . . . 27

9.5 Configuring an ethernet device. (optional) . . . 29

9.6 Configuring a SLIP device (optional) . . . 30

9.6.1 dip . . . 30

9.6.2 slattach . . . 31

9.6.3 When do I use which ? . . . 31

9.6.4 Static SLIP server with a dialup line and DIP. . . . 31

9.6.5 Dynamic SLIP server with a dialup line and DIP. . . . 32

9.6.6 Using DIP. . . . 32

9.6.7 Permanent SLIP connection using a leased line and slattach. . . . 35

9.7 Configuring a PLIP device. (optional) . . . 36

9.7.1 PLIP cabling diagram. . . . 37

10 Routing. (mandatory) 38

10.1 Static/Manual Routes. . . . 38

10.2 Default Route. . . . 39

10.3 Proxy ARP. . . . 40

10.4 gated - the routing daemon. 41
10.4.1 Obtaining *gated* 41
10.4.2 Installing *gated* 41

11 Configuring the network daemons. 43
11.1 /etc/rc.d/rc.inet2 (the second half of rc.net) 43
11.1.1 inetd. 43
11.1.2 syslogd. 44
11.2 A sample rc.inet2 file. 44
11.3 Other necessary network configuration files. 47
11.3.1 A sample /etc/inetd.conf file. 47
11.3.2 A sample /etc/services file. 48
11.3.3 A sample /etc/protocols file. 50
11.4 Name Resolution. 50
11.4.1 /etc/hosts 50
11.4.2 named - do I need thee ? 51
11.4.3 /etc/networks 52
11.4.4 /etc/host.conf 52
11.4.5 /etc/resolv.conf 52
11.4.6 Configuring your Hostname - /etc/HOSTNAME 53
11.5 Other files. 54

12 Advanced Configurations. 54
12.1 PPP - Point to Point Protocol. 54
12.1.1 Why would I use PPP in place of SLIP ? 55
12.1.2 Where to obtain the PPP software. 55
12.1.3 Installing the PPP software. 55
12.1.4 Configuring and using the PPP software. 56
12.1.5 Where to obtain more information on PPP, or report bugs. 60
12.2 Configuring Linux as a Slip Server. 60
12.2.1 Slip Server using *sliplogin*. 60
12.2.2 Slip Server using *dip*. 64
12.2.3 SLIP server using the *dSLIP* package. 66

12.3 Using the Automounter Daemon - AMD. 67
12.3.1 What is an automounter and why would I use one ? 67
12.3.2 Where to get AMD, the automounter daemon. 68
12.3.3 An example AMD configuration. 68
12.4 Using Linux as a router 70
12.5 NIS - Sun Network Information System. 70

13 Experimental and Developmental modules. 70

13.1 snmp agent. 70
13.2 Experimental Token Ring driver 71
13.3 V.35 interface board 71
13.3.1 Internet Technologies International XNET-1 71
13.3.2 SDL Communications N2 card. 72
13.4 Demand Dial SLIP/PPP package 72
13.5 ISDN support 72
13.5.1 3Com-Sonix PC/Arpeggio driver 73
13.5.2 DIEHL SCOM card 73
13.5.3 ICN ISDN and/or Teles ISDN card and/or Creatix AVM ISDN board 73
13.5.4 Euro-ISDN card ? 73
13.5.5 Combinet's Everyware 1000 ISDN card 74
13.5.6 More information: 74
13.6 System-V Streams 74
13.7 ATM (Asynchronous Transfer Mode) support. 74
13.8 X.25 support. 74
13.9 Appletalk support. 75
13.9.1 Building the kernel with Appletalk DDP support 75
13.9.2 Building the Appletalk configuration tools and daemons. 75
13.9.3 Configuring the Appletalk software. 76
13.9.4 Exporting a Linux filesystems via Appletalk. 76
13.9.5 Sharing your Linux printer across Appletalk. 76
13.9.6 Starting the appletalk software. 77
13.9.7 Testing the appletalk software. 77
13.9.8 Caveats of the appletalk software. 77

13.10IP_MASQUERADE - allow a number of hosts to pretend to be one. 78
13.10.1Obtaining the 'masq' software. 78
13.10.2How do I install and compile the patch ? 78
13.10.3How do I configure the IP_MASQUERADE software ? 79
13.10.4Where is IP_MASQUERADE being discussed ? 81
13.11IPALIAS - multiple address per interface. 81
13.11.1Support for 1.2.* kernels 81
13.11.2Support for 1.3.50++ kernels 82

14 Diagnostic tools - How do I find out what is wrong? 82
14.1 ping - are you there? 83
14.2 traceroute - How do I get there? 83
14.3 tcpdump - capturing and displaying network activity. 84
14.4 icmpinfo - logs icmp messages received. 86

15 Some Frequently Asked Questions, with brief Answers. 86
15.1 General questions: 86
15.2 Error messages: 88
15.3 Routing questions: 88
15.4 Using Linux with Novell fileservers or NFS: 89
15.5 SLIP questions: 89
15.6 PPP questions. 91

16 Quick Guide - SLIP or PPP Server 91

17 Known Bugs. 92

18 Copyright Message. 92

19 Miscellaneous and Acknowledgements. 93

1 Introduction.

This is the Linux NET-2/3-HOWTO. This document is a complete rewrite of the earlier NET-FAQ and of the subsequent NET-2-HOWTO versions 1.0+. This document is for the new NET-2/NET-3 tcp/ip networking code for Linux kernels 1.0 and above.

1.1 Changes from the previous release.

```
Additions:

Corrections/Updates:
        updated ipalias to apply to new 1.3.* kernels
        pointed IPX questions to IPX-HOWTO
```

1.2 A brief development history of Linux Networking.

Ross Biro <biro@yggdrasil.com> wrote the original kernel based networking code for Linux. He used ethernet drivers written by Donald Becker <becker@cesdis1.gsfc.nasa.gov>, a *SLIP* driver written by Laurence Culhane <loz@holmes.demon.co.uk> and a D-Link driver by Bj0rn Ekwall <bj0rn@blox.se>.

The further development of the Linux networking code was later taken up by Fred van Kempen <waltje@hacktic.nl>, who took Ross's code and produced the *NET-2* release of network code. NET-2 went through a number of revisions until release NET-2d, when Alan Cox <iialan@iifeak.swan.ac.uk> set about debugging Fred's code with the aim of producing a stable and working release of code for incorporation into the standard kernel releases. This code was called originally called *NET-2D(ebugged)* and was incorporated into the standard kernel releases some time before Linux version 1.0 was released.

PPP support was added by Michael Callahan, <callahan@maths.ox.ac.uk> and Al Longyear, <longyear@netcom.com>, originally as patches to the kernel and in later releases as part of the standard kernel distribution.

The latest version of the code, NET-3, appears in kernel releases 1.1.5 and later and is essentially the same code, but with many fixes, corrections and enhancements.

Alan has added such features as IPX and AX.25 modules. Florian La Roche, <flla@stud.uni-sb.de> has produced an updated distribution of network applications.

NIIBE Yutaka has enhanced the PLIP driver.

Jonathan Naylor has taken up development work on the AX.25 code and has added many features including NetRom support.

Many other people have made contributions by way of bug fixes, ports of applications and by writing device drivers.

2 Disclaimer.

The Linux networking code is a brand new implementation of kernel based tcp/ip networking. It has been developed from scratch and is not a port of any existing kernel networking code.

The Linux networking is some of the newest and most innovative kernel based networking code around. There are many developers working on pressing it into service for many new applications and tasks and because of this it is growing rapidly. It is in a constant state of flux and it may still have a number of bugs

or problems with it and there may be a number of fixes and patches released. If you are worried about problems then just stick to the version of network code released with the standard kernel releases and utility sets. These standard kernel releases are denoted by an **even** number in the second digit, 1.2.7 for example is a production release. The kernel versions with an odd number as the second digit are **alpha** versions and you should expect to find problems or bugs with these version as they are *test* releases. The networking code has a small team of dedicated people working on it, with a cast of thousands testing the code, collecting and reporting bugs and problems, providing fixes for problems. Any problem you experience is likely to have already been reported and be being worked on and will possibly be corrected soon, so be patient, or if you can help, offer your assistance.

We do not and cannot know everything there is to know about the Linux network software. Please accept and be warned that this document probably does contain errors. Please read any README files that are included with any of the various pieces of software described in this document for more detailed and accurate information. We will attempt to keep this document as error-free and up-to-date as possible. Versions of software are current as at time of writing.

NOTE: While its name may appear similar to the *Berkeley Software Distribution NET-2 release*, the Linux network code actually has nothing at all to do with it. Please don't confuse them.

3 Questions already ?

'The only stupid question is the unasked one.'

If you have general configuration questions and you have been unable to find the answers after reading the other various HOWTO and FAQ files, then you would be best served to post them to *comp.os.linux.networking*, or, if you believe your question to be specifically related to the Linux Network code, then you could post it to the NET mailing list. **Please include as much relevant information as possible**, there is nothing more annoying than to have a bug or problem reported without sufficient information to even begin searching for it.

Version numbers and revisions of code, a detailed account of the problem, and the circumstances that caused it to occur, are essential. Trace and debug messages where available should also be considered mandatory.

If you have a question relating to the configuration of, or problems experienced with, **any** linux distribution, regardless of who has provided it, please contact the people who created the distribution first before attempting to report the problem to the network code developers. The reason for this is that some of the distributions use non-standard directory structures and supply test/non-standard versions of code and utilities. The developers of the NET-2 code cannot be expected to offer support for the network code as distributed in any form, other than as described in this document, or as per distributed Alpha test instructions.

To join the Linux **linux-net** channel on the mail list server, send mail to:

```
Majordomo@vger.rutgers.edu
```

with the line:

```
subscribe linux-net
```

as the message body and you will be subscribed. The subject line is ignored. Remember, keep in mind that the `linux-net` channel is for development discussions only.

A PPP list has been established. To join it, use the same procedure as for joining the `linux-net` channel, except specify `linux-ppp` in place of `linux-net` in the message body.

Note also that a `linux-hams` list has been established. This list has been established for the discussion of programs related to Amateur Radio. To join it, follow the same procedure as for joining the `linux-net` or `linux-ppp` channels, except specify `linux-hams` in place of `linux-net` in the message body.

4 Related Documentation. (Where to learn about tcp/ip)

If you are looking for information about tcp/ip networking that this HOWTO does not cover, then you might try the following sources, as they provide some very useful information.

`Olaf Kirch` has written a substantial document as part of the *Linux Documentation Project* entitled the *Linux Network Administration Guide*. This is an excellent document. It covers all aspects of setting up and using the tcp/ip networking under Linux, including NFS, UUCP, mail, News, nameserver etc.

Olaf's book supplements this HOWTO, taking up where this document leaves off. This document covers the installation and configuration of the NET code, i.e. 'How to put your machine on the net'. If you are new to unix networking, then I strongly urge you to obtain a copy and read it first. It will answer a lot of questions for you that are not within the scope of this document.

The current release version is available in:

sunsite.unc.edu

```
/pub/Linux/docs/linux-doc-project/network-guide/*
```

There are various versions of the document in this directory. The most common formats are supported, being plain ascii, Postscript, DVI, Latex and groff.

The *Linux Network Administrators Guide* is Copyright (c) by Olaf Kirch.

There are now quite a variety of companies publishing Linux documentation so if you want to avoid having to retrieve and print the document yourself you should have little trouble finding Olaf's book in a bound form over the counter at any good bookshop.

You should also read the other HOWTO documents relevant to networking with Linux.

They are:

The *Ethernet-HOWTO* `<http://sunsite.unc.edu/mdw/HOWTO/Ethernet-HOWTO.html>`, which you should read if you intend using an ethernet card with Linux. It includes a lot of detail on how to select, install and configure an ethernet card for Linux and on how to diagnose problems related to the ethernet driver.

The *PPP-HOWTO* `<http://sunsite.unc.edu/mdw/HOWTO/PPP-HOWTO.html>` if you intend using PPP.

The *IPX-HOWTO* `<http://sunsite.unc.edu/mdw/HOWTO/IPX-HOWTO.html>` if you would like information relating to IPX support for Linux.

The *Serial-HOWTO* <http://sunsite.unc.edu/mdw/HOWTO/Serial-HOWTO.html> if you intend using SLIP or PPP in server mode.

The *NIS-HOWTO* <http://sunsite.unc.edu/mdw/HOWTO/NIS-HOWTO.html> if you are interested in running a version of Sun's Network Information Service.

The *HAM-HOWTO* <http://sunsite.unc.edu/mdw/HOWTO/HAM-HOWTO.html> if you are interested in configuring and running amateur radio software.

The *Mail-HOWTO* <http://sunsite.unc.edu/mdw/HOWTO/Mail-HOWTO.html> and the *News-HOWTO* <http://sunsite.unc.edu/mdw/HOWTO/News-HOWTO.html> for some specific information on setting up Mail and News on your system.

The *UUCP-HOWTO* <http://sunsite.unc.edu/mdw/HOWTO/UUCP-HOWTO.html> if you will be connecting to the net via UUCP.

The *Firewall-HOWTO* <http://sunsite.unc.edu/mdw/HOWTO/Firewall-HOWTO.html> if you want to build a Linux based Firewall gateway for your network.

If you are after some basic tutorial information on tcp/ip networking generally, then I recommend you take a look at the following documents:

tcp/ip introduction

text version <ftp://athos.rutgers.edu/runet/tcp-ip-intro.doc>, *postscript version* <ftp://athos.rutgers.edu/runet/tcp-ip-intro.ps>.

tcp/ip administration

text version <ftp://athos.rutgers.edu/runet/tcp-ip-admin.doc>, *postscript version* <ftp://athos.rutgers.edu/runet/tcp-ip-admin.ps>.

If you are after some more detailed information on tcp/ip networking then I highly recommend:

```
"Internetworking with TCP/IP"
by Douglas E. Comer

ISBN 0-13-474321-0
Prentice Hall publications.
```

If you are wanting to learn about how to write network applications in a Unix compatible environment then I also highly recommend:

```
"Unix Network Programming"
by W. Richard Stevens

ISBN 0-13-949876-1
Prentice Hall publications.
```

4.1 New versions of this document.

If your copy of this document is more than two months old then I strongly recommend you obtain a newer version. The networking support for Linux is changing very rapidly with new enhancements and features, so this document also changes fairly frequently. The latest released version of this document can always be retrieved by anonymous ftp from:

sunsite.unc.edu

`/pub/Linux/docs/HOWTO/NET-2-HOWTO`

or:

`/pub/Linux/docs/HOWTO/other-formats/NET-2-HOWTO{-html.tar,ps,dvi}.gz`

or via the World Wide Web from the *Linux Documentation Project Web Server* `<http://sunsite.unc.edu/mdw/linux.html>`, at page: *NET-2-HOWTO* `<http://sunsite.unc.edu/mdw/HOWTO/NET-2-HOWTO.html>` or directly from me, `<94004531@postoffice.csu.edu.au>`. It will also be posted to the newsgroups: `comp.os.linux.networking`, `comp.os.linux.answers` and `news.answers` from time to time.

You can find `news.answers` FAQ postings, including this one, archived on `rtfm.mit.edu:/pub/usenet`.

4.2 Feedback.

Please send any comments, updates, or suggestions to me, `<94004531@postoffice.csu.edu.au>`. The sooner I get feedback, the sooner I can update and correct this document. If you find any problems with it, please mail me directly as I now very rarely read the newsgroup. You might also catch me as *terryd* on the `#linpeople` IRC channel on the `undernet` IRC network.

5 Some of the terms used in this document.

You will often see the terms `client` and `server` used in this document. They are normally fairly specific terms but in this document I have generalised their definitions a little so that they mean the following:

client

The machine or program that initiates an action or a connection for the purpose of gaining use of some service or data.

server

The machine or program that accepts incoming connections from multiple remote machines and provides a service or data to those.

These definitions are not very reliable either, but they provide a means of distinguishing the ends of peer to peer systems such as *SLIP* or *PPP* which truly do not actually have clients and servers.

Other terms you will see are:

IP address

This is a number that uniquely identifies a TCP/IP host on the network. The address is 4 bytes long and is usually represented in what is called the "dotted decimal notation", where each byte is represented in decimal from with dots '.' between them.

Hardware address

This is a number that uniquely identifies a host in a physical network at the media access layer. Examples of this are *Ethernet Addresses* and *AX.25 Addresses*.

datagram

A datagram is a discrete package of data and headers which contain addresses, which is the basic unit of transmission across an IP network. You might also hear this called a 'packet'.

MTU

The Maximum Transmission Unit (*MTU*) is a parameter that determines the largest datagram than can be transmitted by an IP interface without it needing to be broken down into smaller units. The MTU should be larger than the largest datagram you wish to transmit unfragmented. Note, this only prevents fragmentation locally, some other link in the path may have a smaller MTU and the datagram will be fragmented there. Typical values are 1500 bytes for an ethernet interface, or 576 bytes for a SLIP interface.

MSS

The Maximum Segment Size (*MSS*) is the largest quantity of data that can be transmitted at one time. If you want to prevent local fragmentation MSS would equal MTU-IP header.

window

The *window* is the largest amount of data that the receiving end can accept at a given point in time.

route

The *route* is the path that your datagrams take through the network to reach their destination.

ARP

This is an acronym for the *Address Resolution Protocol* and this is how a network machine associates an IP Address with a hardware address.

6 NET-3 Supported functionality.

The NET code is a complete kernel based implementation of tcp/ip for Linux.

6.1 General Support

The recent NET-3 versions of code support:

Ethernet Cards

most popular ethernet cards are supported. Including some portable, pocket adaptors and PCI.

SLIP (Serial Line IP) and PPP (Point to Point Protocol)

for tcp/ip networking over serial lines such as the telephone via modem, or a local cable between two machines.

Van Jacobsen Header Compression

for compressing the tcp/ip headers to improve SLIP/PPP performance over low speed lines.

PLIP (Parallel Lines IP)

to allow local connections between two machines using your printer ports.

EQL Load balancing

allows you to use two (or more) links to connect your machine to another machine or the Internet (provided your ISP supports it) to effectively double your bandwidth. New release kernels support this.

NFS (Networked File System)

to allow you to remotely mount another machines filesystems across a network connection.

IPX (Novell)

to allow you to write custom IPX applications, or to use Linux as an IPX router.

Sun's Network Information System - NIS

An NIS implementation has been ported to Linux should you wish to use it.

ARCNet

An ARCNet driver has been written and is included in recent kernels. It might not be as fast as ethernet but the cards are much cheaper.

IBM's Token Ring

to allow linux to be installed on a Token Ring lan. An experimental Token Ring driver has been written is included in recent kernels.

Appletalk

Or is this EtherTalk ? Either way, I think this will let you shares files and printers with your Macintosh. See 'Experimental and Developmental modules.' below.

WaveLan Wireless Lan Card support

to allow you to operate your linux machine in a mobile fashion or at some distance from your network. Support for the WaveLan Wireless lan card is now included in recent kernels.

ISDN

There is some experimental support for some proprietary ISDN cards available. See 'Experimental and Developmental modules.' below.

ATM

There is a team of programmers working to provide ATM support for Linux.

IP firewalling

to assist in configuring your Linux machine as a secure firewall gateway.

IP Accounting

to allow you to keep track of who is using how much of your network.

IP tunnelling

to allow mobile IP experimentation

The NET-3 network code does not yet currently support:

SPX/NCP (Novell Netware) support

to allow Linux to serve and mount Novell network filesystems or use Novell printers. This is being worked on but due to the proprietary nature of the product it may take some time to do.

FDDI

There is currently no support that I know of for FDDI cards for Linux.

System-V streams

there is a team of people working on System-V streams for Linux, details are presented later.

6.2 Supported Ethernet cards.

The 1.2.0 linux kernel release supports the following types of Ethernet cards:

- WD80*3 and close compatibles.
- SMC Ultra.
- AMD LANCE and PCnet (AT1500 and NE2100) and close compatibles.
- 3Com 3c501 (obsolete and very slow).
- 3Com 3c503.
- 3Com 3c505.
- 3Com 3c507.
- 3Com 3c509/3c579.
- Cabletron E21xx.
- DEPCA and close compatibles.

- EtherWorks 3.
- ARCNet.
- AT1700 (not clones).
- EtherExpress.
- NI5210 and close compatibles.
- NI6510.
- WaveLAN.
- HP PCLAN+ (27247B and 27252A).
- HP PCLAN (27245 and other 27xxx series).
- NE2000/NE1000 and close compatibles.
- SK_G16.
- Ansel Communications EISA 3200.
- Apricot Xen-II on board ethernet.
- DE425, DE434, DE435.
- Zenith Z-Note.
- AT-LAN-TEC/RealTek pocket adaptor.
- D-Link DE600 pocket adaptor and close compatibles.
- D-Link DE620 pocket adaptor and close compatibles.

Later versions of the Kernel software may support a wider variety of cards.

If you intend using an ethernet card with Linux you should read the *Ethernet-HOWTO* `<http://sunsite.unc.edu/mdw/HOWTO/Ethernet-HOWTO.html>` as it contains a lot of very useful information on the supported ethernet cards, including information on how to choose an ethernet card if you are intending to purchase some specifically for Linux.

As mentioned above, Linux supports other means of network connection if you don't have access to an ethernet card or connection. Many universities and businesses worldwide offer some form of dial-up network access. Generally these forms of access will offer an option of either SLIP or PPP access, so you will be well catered for. All you will need is a telephone modem, the one you already have may well be good enough and to configure your Linux system appropriately. There are sections below that describe exactly what you need.

6.3 Support for Amateur Radio

Linux now supports a number of features specifically for Amateur Radio. The latest alpha kernels are now distributed with standard support for:

AX.25 Support

Alan Cox and Jonathan Naylor have kernel based AX.25 socket support working.

NetRom Support

Jonathan Naylor has developed kernel based NetRom support. It is still experimental but is progressing well.

Ottawa PI Card

A mature driver for the Ottawa PI card has been developed by Dave Perry of the Ottawa Packet Radio Group.

Generic SCC card driver

A generic driver for SCC cards is now included in alpha kernels developed by Joerg Reuter.

A kernel user network link driver

for linking user programs to the kernel without messy pty's.

Further detail on the Amateur Radio support can be found in the *HAM-HOWTO* `<http://sunsite.unc.edu/mdw/HOWTO/HAM-HOWTO.html>`.

7 Getting the NET-2/NET-3 software.

Before you can configure the networking software you must obtain all of the bits and pieces that make it up. These include the current version of the kernel code (version 1.0 or later), the correct system libraries, the tcp/ip configuration programs and files (e.g. /sbin/ifconfig, /etc/hosts etc.) and finally a set of network application programs (such as telnet, ftp, rlogin etc.).

If you obtained Linux from a distribution you may already have all that you need. Check and make sure that you do. For example, some Linux distributions come with all of the network configuration files, binaries, libraries and kernel installed, so there's no reason to get the following files.

NOTE: they may be in directories and files different to those specified in this HOWTO document

If you **DO** have the network software, skip to the 'Configuring the kernel' section. If you **DO NOT** have the network software follow the following directions.

7.1 The kernel source.

Version 1.2.* of the Linux kernel is the *production* version. Any of the Linux kernels 1.3.* are development kernels. If you feel at all concerned about the possibility of having to patch and modify the kernel source, then you should stick to this release, as it will do most of what you want it to. In the case of the networking code though, I strongly suggest you just take a deep breath and follow the newer releases of code, as there have been many changes in the newer version kernels that affect networking. I know you hear it from everyone and everywhere, but when trying out any new version of kernel software you should always ensure that you have sufficient backups of your system just in case something goes seriously wrong while you are testing.

The current kernel version is found in:

ftp.funet.fi

```
/pub/OS/Linux/PEOPLE/Linus/v1.2/linux-1.2.13.tar.gz
```

This is a gzipped file, so you will need *gzip* to uncompress it.

To install it, try:

```
# cd /usr/src
# mv linux linux.old
# gzip -dc linux-1.2.13.tar.gz | tar xvf -
```

You may also find some files called `patch-1.2.1.gz ...` in the same directory. These are patch files. If you have a linux kernel that is version 1.2.1 then what this means is that you have linux kernel version 1.2.0 with patch 1 applied. So you don't need to patch 1. If there are any patch files that are greater than the version of kernel you have, you should obtain **all** of those above and apply them **in sequence** with something like the following commands:

```
# cd /usr/src
# for patchfile in .../patch*
> do
> gzip -dc $patchfile | patch -p0 2>>patch.errs
> done

 ...
```

Check the output file (patch.errs) and search for the string `fail`. If you can't find it then all of the patch files were applied ok. If it is there, then at least one of the patch files didn't apply correctly. If this happens what you should do is start again from a clean kernel archive and apply the patches one by one until you find the patch file that failed. If you can't work out why it didn't work then report it as a problem.

7.2 The libraries.

You'll want at **least** version 4.4.2 of *libc*, as there were problems with earlier version that affected subnet masks.

The current a.out libraries (libc-4.6.27) can be found in:

sunsite.unc.edu

```
/pub/Linux/GCC/
```

You will need at least the following files:

- image-4.6.27.tar.gz
- inc-4.6.27.tar.gz
- extra-4.6.27.tar.gz
- release.libc-4.6.27

When installing libc.4.6.27 you **MUST** read **release.libc-4.6.27** before you install the libraries. Please note that to use release 4.5.26 or later you will also need at least GCC version 2.6.2 and Linux kernel 1.1.52 or later.

7.3 The network configuration tool suite.

You will need the utility suite that provides tools to configure your network support.

The current NET-2 utility suite is available from:

ftp.linux.org.uk

```
/pub/linux/Networking/PROGRAMS/NetTools/net-tools-1.2.0.tar.gz
```

Because the kernel networking code is still changing some changes to the network tools have been necessary as new kernels are released, so you will need to choose the version that is appropriate for the kernel version you intend to use.

The filenames reflect the earliest version of kernel that the tools will work with. Please choose the filename whose version equals, or is less than the version of kernel source you intend to use.

To build and install the tools, you should try:

```
# cd /usr/src
# mkdir net-tools
# cd net-tools
# gzip -dc net-tools-1.2.0.tar.gz | tar xvf -
# make
```

This will automatically run the `Configure.sh` script. If everything makes ok, then:

```
# make install
```

If you use a kernel version 1.1.26 or earlier you should look in:

ftp.linux.org.uk

```
/pub/linux/Networking/PROGRAMS/Other/net032
```

In this directory you will find three versions of the network tools. The following table lists net-032 package name with the relevant kernel versions:

```
net-0.32d-net3.tar.gz     1.1.12+
net-0.32b.tar.gz          1.1.4+
net-0.32.old.tar.gz       pre 1.1.4 kernels
```

These packages include the essential network configuration programs such as *ifconfig*, *route*, *netstat* etc. These will be discussed later.

7.4 The network applications.

You will want a number of network application programs. These are programs like *telnet*, *ftp*, *finger* and their daemons at least. `Florian La Roche, <flla@stud.uni-sb.de>` has put together a fairly complete distribution of network applications in both binary and source form. The tcp/ip application binaries and some sample config files are found in:

ftp.funet.fi

```
/pub/OS/Linux/PEOPLE/Linus/net-source/base/NetKit-A-0.08.bin.tar.gz
/pub/OS/Linux/PEOPLE/Linus/net-source/base/NetKit-B-0.06.tar.gz
```

If there are newer versions then use the newer versions. Please read the `README` file **first** just to make sure that you have the necessary prerequisites.

Florian used to have a binary distribution of the networking applications (the B file) available but it is no longer there, so you will have to build the files yourself. You can use the following procedure:

```
# cd /usr/src
# gzip -dc NetKit-B-0.06.tar.gz | tar xpvlf -
# cd NetKit-B-0.06
```

Then, read the `README` file. You will need to edit the `Makefile` and set the `HAVE_SHADOW_PASSWORDS` define appropriately. I don't use shadow passwords, so I commented it out by placing a `#` at the start of the line. The rest should not need modifying, so then all you should have to do is:

```
# make
# make install
```

IMPORTANT NOTE: Florian has built and prepackaged these tar files for your convenience. Florian has attempted to make them as complete as possible and has included a distribution of the binaries found in

the net-tools-n.n.nn releases. Unfortunately Florian has chosen not to use the same directory structure as Alan did when he prepared the installation script for the net-tools. This will mean that you should be very careful when installing them. Florian will change this later so that this difference is not a problem, but until then, I suggest you do the following instead of the above:

```
-  Unpack the binaries somewhere safe:
# cd /usr/src
# mkdir NetKit
# cd NetKit
# gzip -dc NetKit-A-0.07.bin.tar.gz | tar xpvlf -
# gzip -dc NetKit-B-0.06.bin.tar.gz | tar xpvlf -

-  Remove Florians copies of the network tools previously described:
# rm ./bin/hostname ./sbin/route ./sbin/ifconfig ./sbin/netstat
# rm ./usr/sbin/arp ./usr/sbin/rarp ./usr/sbin/slattach

-  Copy Florian's files into their new home:
# cp -vrpd . /
```

7.5 Additional drivers or packages.

If you want to add some developmental, or Alpha/Beta test code, such as AX.25 support, you will need to obtain the appropriate support software for those packages. Please check the relevant sections for those packages in this document for more detail.

8 Configuring the kernel.

Before you can use any of the network tools, or configure any network devices, you must ensure that your kernel has the necessary network support built into it. The best way of doing this is to compile your own, selecting which options you want and which you don't.

Assuming you have obtained and untarred the kernel source already and applied any patches that you might need to have applied to get any nonstandard or developmental software installed, all you have to do is edit /usr/src/linux/drivers/net/CONFIG. This file has many comments to guide you in editing it and in general you will need to edit very little, as it has sensible defaults. In my case I don't need to edit it at all. This file is really necessary if your ethernet card is an unusual one, or is one that isn't automatically detected by the ethernet driver. It allows you to hard code some of the elements of your ethernet hardware. For example, if your ethernet card is a close, but not exact clone of a WD-8013, then you might have to configure the shared memory address to ensure the driver detects and drives the card properly. Please check the The *Ethernet-HOWTO* <http://sunsite.unc.edu/mdw/HOWTO/Ethernet-HOWTO> for more definitive information on this file and its effect on ethernet cards. This file also contains configurable parameters for PLIP, though the defaults should again be ok unless you have a particularly slow machine.

When you are happy that the CONFIG file is suitable for your purposes, then you can proceed to build the kernel. Your first step will be to edit the top level Makefile to ensure the kernel will be built with the

appropriate VGA settings and then you must run the kernel configuration program:

```
# cd /usr/src/linux
# make config
```

You will be asked a series of questions. There are four sections relevant to the networking code. They are the `General setup`, `Networking options`, `Network device support` and the `Filesystems` sections. The most difficult to configure is the `Network device support` section, as it is where you select what types of physical devices you want configured. On the whole you can just use the default values for the other sections fairly safely. The following will give you an idea of how to proceed:

```
*
* General setup
*
 ...
 ...
Networking support (CONFIG_NET) [y] y
 ...
 ...
```

In the `General setup` section you simply select whether you want network support or not. Naturally you must answer yes.

```
*
* Networking options
*
TCP/IP networking (CONFIG_INET) [y]
IP forwarding/gatewaying (CONFIG_IP_FORWARD) [n]
IP multicasting (CONFIG_IP_MULTICAST) [n]
IP firewalling (CONFIG_IP_FIREWALL) [n]
IP accounting (CONFIG_IP_ACCT) [n]
*
* (it is safe to leave these untouched)
*
PC/TCP compatibility mode (CONFIG_INET_PCTCP) [n]
Reverse ARP (CONFIG_INET_RARP) [n]
Assume subnets are local (CONFIG_INET_SNARL) [y]
Disable NAGLE algorithm (normally enabled) (CONFIG_TCP_NAGLE_OFF) [n]
The IPX protocol (CONFIG_IPX) [n]
*
```

The second half of the `Networking options` section allows you to enable or disable some funky features that you can safely accept the defaults on until you have some idea why you want to change them. They are described briefly later if you are interested.

```
*
```

```
*
* Network device support
*
Network device support? (CONFIG_NETDEVICES) [y]
Dummy net driver support (CONFIG_DUMMY) [n]
SLIP (serial line) support (CONFIG_SLIP) [y]
 CSLIP compressed headers (CONFIG_SLIP_COMPRESSED) [y]
 16 channels instead of 4 (SL_SLIP_LOTS) [n]
PPP (point-to-point) support (CONFIG_PPP) [y]
PLIP (parallel port) support (CONFIG_PLIP) [n]
Do you want to be offered ALPHA test drivers (CONFIG_NET_ALPHA) [n]
Western Digital/SMC cards (CONFIG_NET_VENDOR_SMC) [y]
WD80*3 support (CONFIG_WD80x3) [y]
SMC Ultra support (CONFIG_ULTRA) [n]
AMD LANCE and PCnet (AT1500 and NE2100) support (CONFIG_LANCE) [n]
3COM cards (CONFIG_NET_VENDOR_3COM) [n]
Other ISA cards (CONFIG_NET_ISA) [n]
EISA, VLB, PCI and on board controllers (CONFIG_NET_EISA) [n]
Pocket and portable adaptors (CONFIG_NET_POCKET) [n]
*
```

This section is the most important and the most involved. It is where you select what hardware devices you want to support. You can see that I have selected SLIP support with header compression, PPP, the WD80*3 driver and nothing else. Other options will appear depending on what you select. If you answered '`n`' to the '`SLIP..`' option you will not be presented with the `compressed SLIP` or `16 channel` options. Simply answer '`y`' to whatever you want to play with and '`n`' to those that you don't.

```
*
* Filesystems
*
 ...
 ...
/proc filesystem support (CONFIG_PROC_FS) [y]
NFS filesystem support (CONFIG_NFS_FS) [y]
 ...
 ...
```

If you wish to run an NFS client then you will want to include the NFS filesystem type. You will need to include the `/proc` filesystem because a number of the network utilities use it.

After you have completed the configuration, all that remains is to actually compile the kernel:

```
# make dep
# make
```

Don't forget to `make zlilo` if the new kernel compiles and tests ok.

8.1 What do all those funky Networking options actually do?

Newer kernels have a number of options that you are asked about when you do a `make config`. Generally you will not need to change these, but some of the options might be useful to you in certain circumstances.

TCP/IP networking

This one is obvious, it selects whether you configure the tcp/ip suite of protocols into your kernel. Chances are if you are reading this then you will want to answer 'y' to this one.

Dummy networking device

This was added to allow SLIP and PPP users to configure an address on their linux machine that would not be dependent on their serial link being established. It is an easy way to give your linux machine two addresses.

IP forwarding/gatewaying

This determines what your kernel will do when it receives a datagram that has a destination address that is not one of its own devices. You **must** have this option selected if you want your kernel to act as an IP router. Most SLIP and PPP servers will want this option selected.

IP multicasting

This is alpha test code support for IP multicasting, examples of which include services such as 'Internet Talk Radio' and live video. You will need additional programs to make use of this facility, this is just the kernel support.

IP firewalling

This option allows you to provide flexible security options for your linux machine. You can selectively enable/disable access to tcp/ip ports from any address ranges you choose. This also needs additional programs to support it.

IP accounting

This option is for those people that want to use their Linux machine to provide internet connectivity to others on a commercial basis. It allows you to count and record incoming and outgoing bytes on a per port and address basis. With the addition of suitable software this would allow you to produce seperate usage charges for each person using your systems networking capabilities.

PC/TCP compatibility mode

This option provides a work-around for a bug that causes problems when using the PC/TCP networking programs to talk to your linux machine. There is a PC/TCP bug which provokes a difficult to remedy Linux bug and this option prevents the two clashing. Normally you would leave this disabled, but if you have users on your network who use PC/TCP then you may have to enable this option to prevent problems.

Reverse ARP

This option allows you to configure the RARP protocol into your kernel. This option was added to allow the booting of Sun 3 systems. This is not generally very useful otherwise.

Assume subnets are local

This option selects whether you assume that your whole subnet is directly connected to your linux machine, or whether it might be bridged or otherwise subdivided at a lower layer. In practice it will make little difference if you leave it set at the default.

Disable NAGLE algorithm

This is a timing option that determines when a datagram should be transmitted. The default setting provides for the best throughput in most situations and you should leave this set as it is, as disabling it will degrade your throughput. This option can be selectively changed from within a program with a socket option and you would normally be much better off leaving it set at the default and specifically writing your programs to disable the NAGLE algorithm if they require extremely fast interactivity.

The IPX protocol

This option selects whether you compile the IPX protocol support into your kernel. The IPX protocol is an inter-networking protocol similar in function to the IP protocol. This protocol is one of those used by the Novell suite.

Amateur Radio AX.25 Level 2

This option selects whether you compile in the Amateur Radio AX.25 protocol suite. If you select this option then a new class of network sockets are available for programming. The AX.25 protocol is used primarily by Amateur Radio Operators for packet radio use.

9 Configuring the Network Devices.

If everything has gone ok so far, then you will have a Linux kernel which supports the network devices you intend to use and you also have the network tools with which to configure them. *Now comes the fun part!* You'll need to configure each of the devices you intend to use. This configuration generally amounts to telling each device things like what its IP address will be and what network it is connected to.

In past versions of this document I have presented near complete versions of the various configuration files and included comments to modify or delete lines from them as appropriate. From this version onwards I will take a slightly different approach which I hope will result in you having a complete set of uncluttered configuration files that you have built from scratch so you know exactly what is in them and why. I'll describe each of these files, and their function, as we come to them.

9.1 Configuring the special device files in /dev

You do not need to configure any special device files in the `/dev` directory for Linux Networking. Linux does not need or use them as other operating systems might. The devices are built dynamically in memory by the kernel and since they are only names there is no need for them to have an appearance directly to you. The kernel provides all of the programming hooks and interfaces that you need to utilize them effectively.

9.2 What information do I need before I begin ?

Before you can configure the networking software, you will need to know a number of pieces of information about your network connection. Your network provider or administrator will be able to provide you with most of them.

9.2.1 IP Address.

This is the unique machine address, in dotted decimal notation, that your machine will use. An example is `128.253.153.54`. Your network administrator will provide you with this information.

If you will be using a SLIP or plip connection you may not need this information, so skip it until we get to the SLIP device.

If you're using the loopback device only, ie no ethernet, SLIP or plip support, then you won't need an ip address as the loopback port always uses the address `127.0.0.1`.

9.2.2 Network Mask ('netmask').

For performance reasons it is desirable to limit the number of hosts on any particular segment of a network. For this reason it is common for network administrators to divide their network into a number of smaller networks, known as *subnets*, which each have a portion of the network addresses assigned to them. The *network mask* is a pattern of bits, which when overlayed onto an address on your network, will tell you which subnetwork it belongs to. This is very important for routing and if you find for example, that you can happily talk to people outside your network, but not to some people on your own network, then it is quite likely that you have specified an incorrect subnet mask.

Your network administrators will have chosen the netmask when the network was designed and therefore they should be able to supply you with the correct mask to use. Most networks are class-C subnetworks which use `255.255.255.0` as their netmask. Other larger networks use class-B netmasks (`255.255.0.0`). The NET-2/NET-3 code will automatically select a default mask when you assign an address to a device. The default assumes that your network has **not** been subnetted.

The NET-2/NET-3 code will choose the following masks by default:

```
For addresses with the first byte:
1-127         255.0.0.0         (Class A)
128-191       255.255.0.0       (Class B)
192+          255.255.255.0     (Class C)
```

if one of these doesn't work for you, try another. If this doesn't work ask your network administrator or local network guru (dime a dozen) for help.

You don't need to worry about a netmask for the loopback port, or if you are running SLIP/plip.

9.2.3 Network Address.

This is your IP address masked (bitwise AND) with your netmask. For example:

```
If your netmask is:           255.255.255.0
and your IP address is:       128.253.154.32    &&
                                               ---------------
your Network address is:      128.253.154.0     =
```

9.2.4 Broadcast Address.

'A shout is a whisper that everyone hears whether they need to or not'

This is normally your network address logically ORed with your netmask inverted. This is simpler than it sounds. For a Class-C network, with network mask `255.255.255.0`, your *Broadcast Address* will be your network address (calculated above), logically ORed with `0.0.0.255`, the network mask inverted.

A worked example might look like:

```
If your netmask is:          255.255.255.0      !
the netmask inverted is:       0.  0.  0.255    =
If your Network address is:  128.253.154.0      ||
                              ----------------
Your broadcast address is:   128.253.154.255    =
```

Note that for historical reasons some networks use the network address as the broadcast address. If you have any doubts contact your network administrator.

If you have access to a *sniffer*, or some other device capable of providing you with a trace of your network traffic, then you might be able to determine both the network and broadcast addresses by watching other traffic on the lan. Keep an eye open for, (or filter everything except), ethernet frames destined for the ethernet broadcast address: `ff:ff:ff:ff:ff:ff`. If any of them has an IP source address of your local router and the protocol ID is not ARP, then check the destination IP address, because this datagram may well be a RIP routing broadcast from your router, in which case the destination IP address will be your broadcast address.

Once again, if you're not sure, check with your network administrator, they'd rather help you, than have you connect your machine in a misconfigured way.

9.2.5 Router ('Gateway') Address.

'There must be some way out of here.'

This is the address of the machine that connects your network to the rest of the Internet. It is your 'gateway' to the outside world. A couple of conventions exist for allocating addresses to routers which your network might follow, they are: The router is the lowest numbered address on the network, the router is the highest numbered host on the network. Probably the most common is the first, where the router will have an address that is mostly the same as your own, except with a `.1` as the last byte. eg. if your address is `128.253.154.32`, then your router might be `128.253.154.1`. The router can in fact have any address valid on your network and function properly, the address doesn't matter at all. There may in fact even be more than one router on your network. You will probably need to talk to your network administrator to properly identify your router address.

If you're using only loopback then you don't need a router address. If you're using PPP then you also don't need your router address, because PPP will automatically determine the correct address for you. If you're using SLIP, then your router address will be your SLIP server address.

9.2.6 Nameserver Address.

Most machines on the net have access to a name server which translates human tolerable hostnames into machine tolerable addresses and *vice versa*. Your network administrators will again tell you the address of your nearest nameserver. You can in fact run a nameserver on your own machine by running *named*, in which case your nameserver address will be `127.0.0.1`, the *loopback* port address. However it is not required that you run *named* at all; see section 'named' for more information.

If you're only using loopback then you don't need to know the nameserver address since you're only going to be talking to your own machine.

9.2.7 NOTE for SLIP/PLIP/PPP users.

You may or may not in fact need to know any of the above information. Whether you do or not will depend on exactly how your network connection is achieved and the capabilities of the machine at the other end of the link. You'll find more detail in the section relevant to configuration of the SLIP/PLIP and PPP devices.

9.3 /etc/rc.d/rc.inet1,2 or /etc/rc.net or /etc/init.d/network

While the commands to configure your network devices can be typed manually each time, you will probably want to record them somewhere so that your network is configured automatically when you boot your machine.

The `'rc'` files are specifically designed for this purpose. For the non-unix-wizard: `'rc'` file are run at bootup time by the *init* program and start up all of the basic system programs such as *syslog*, *update* and *cron*. They are analogous to the MS-DOS *autoexec.bat* file and *rc* might stand for *'runtime commands'*. By convention these files are kept under the `/etc` directory. The Linux Filesystem Standard doesn't go so far as to describe exactly where your `rc` files should go, stating that it is ok for them to follow either the BSD (/etc/rc.*) or System-V (/etc/rc.d/rc*) conventions. Alan, Fred and I all use the System-V convention, so that is what you will see described here. This means that these files are found in `/etc/rc.d` and are called `rc.inet1` and `rc.inet2`. The first `rc` file that gets called at bootup time is `/etc/rc` and it in turn calls others, such as `rc.inet1`, which in turn might called `rc.inet2`. It doesn't really matter where they are kept, or what they are called, so long as *init* can find them.

In some distributions the `rc` file for the network is called `rc.net` and is in the `/etc` subdirectory. The `rc.net` file on these systems is simply the `rc.inet1` and the `rc.inet2` files combined into one file that gets executed.

Some equivalences:

Document Convention	Slackware	Debian and other newer.
`rc`	`rc.inet1`	`rc.inet2`

```
/etc/rc.d/rc.inet1    /etc/rc.net   /etc/init.d/network

/etc/rc.d/rc.inet2    /etc/rc.net   /etc/init.d/netbase
                                    /etc/init.d/netstd_init
                                    /etc/init.d/netstd_nfs
                                    /etc/init.d/netstd_misc
```

It doesn't matter where the commands appear, so long as you configure the interfaces before starting the network daemons and applications.

I will refer to these files as *rc.inet1* and *rc.inet2* and I keep them in the `/etc/rc.d`, so if you are using one of the distributions that uses some other convention, or you want to keep the files somewhere else, then you will have to make appropriate adjustments as you go.

We will be building these files from scratch as we go.

9.3.1 rc.inet1

The *rc.inet1* file configures the basic tcp/ip interfaces for your machine using two programs: */sbin/ifconfig* and */sbin/route*.

ifconfig

/sbin/ifconfig is used for configuring your interfaces with the parameters that they require to function, such as their IP address, network mask, broadcast addresses and similar. You can use the *ifconfig* command with no parameters to display the configuration of all network devices. Please check the *ifconfig* man page for more detail on its use.

route

/sbin/route is used to create, modify and delete entries in a table (the routing table) that the networking code will look at when it has a datagram that it needs to transmit. The routing table lists destination address and the interface that that address is reachable via. You can use the *route* command with no parameters to display the contents of the routing table. Please check the *route* man page for more detail on its use.

9.3.2 rc.inet2

The *rc.inet2* file starts any network daemons such as *inetd*, *portmapper* and so on. This will be covered in more detail in section 'rc.inet2', so for the moment we will concentrate on *rc.inet1*. I have mentioned this file here so that if you have some other configuration, such as a single *rc.net* file you will understand what the second half of it represents. it is important to remember that you must start your network applications and daemons **after** you have configured your network devices.

9.4 Configuring the Loopback device (mandatory).

The loopback device isn't really a hardware device. It is a software construct that looks like a physical interface. Its function is to happily allow you to connect to yourself and to test network software without

actually having to be connected to a network of any kind. This is great if you are developing network software and you have a SLIP connection. You can write and test the code locally and then when you are ready to test it on a live network, establish your SLIP connection and test it out. You won't hurt others users if your program misbehaves.

By convention, the loopback device always has an IP address of `127.0.0.1` and so you will use this address when configuring it.

The loopback device for Linux is called *'lo'*. You will now make the first entry into your *rc.inet1* file. The following code fragment will work for you:

```
#!/bin/sh
#
# rc.inet1   --  configures network devices.
#
# Attach the loopback device.
/sbin/ifconfig lo 127.0.0.1
#
# Add a route to point to the loopback device.
/sbin/route add 127.0.0.1
# End loopback
#
```

You have used the *ifconfig* program to give the loopback interface its IP address and *route* program to create an entry in the routing table that will ensure that all datagrams destined for `127.0.0.1` will be sent to the loopback port.

There are two important points to note here.

Firstly, the netmask and broadcast addresses have been allowed to take the default values for the loopback device described earlier in section 'Network Mask'. To see what they are, try the *ifconfig* program without any arguments.

```
# ifconfig
lo        Link encap Local Loopback
          inet addr 127.0.0.1  Bcast 127.255.255.255  Mask 255.0.0.0
          UP BROADCAST LOOPBACK RUNNING  MTU 2000  Metric 1
          RX packets 0 errors 0 dropped 0 overrun 0
          TX packets 30 errors 0 dropped 0 overrun 0
#
```

Secondly, its not obvious how the *route* command chose the loopback device as the device for the route to `127.0.0.1`. The *route* program is smart enough to know that `127.0.0.1` belongs to the network supported by the loopback device. It works this out by checking the IP address and the netmask. You can use the *route* command with no arguments to display the contents of the routing table:

```
# route
Kernel routing table
```

```
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
127.0.0.0       *               255.0.0.0       U     0      0       30 lo
#
```

Note: You might want to use the **-n** argument if your name resolver is not yet configured properly. The **-n** argument tells *route* to just display the **n**umeric addresses and to not bother looking up the name.

9.5 Configuring an ethernet device. (optional)

You'll only be interested in this section if you wish to configure an ethernet card, if not then skip on ahead to the next section.

To configure an ethernet card is only slightly more complicated than configuring the loopback device. This time you should probably specify explicitly the network mask and the broadcast address, unless you are sure that the defaults will work ok and they probably will.

For this you will need the IP address that you have been assigned, the network mask in use on your network and the broadcast address in use.

The first ethernet device for a *Linux* system is called *'eth0'*, the second *'eth1'* and so forth. You will now **add** a section to your *rc.inet1* file. The following code fragment will work for you if you change the addresses specified for real ones:

```
#
# Attach an ethernet device
#
#  configure the IP address, netmask and broadcast address.
/sbin/ifconfig eth0 IPA.IPA.IPA.IPA
/sbin/ifconfig eth0 netmask NMK.NMK.NMK.NMK
/sbin/ifconfig eth0 broadcast BCA.BCA.BCA.BCA
#
# add a network route to point to it:
/sbin/route add -net NWA.NWA.NWA.NWA device eth0
#
# End ethernet
#
```

Where:

IPA.IPA.IPA.IPA

represents your IP Address.

NMK.NMK.NMK.NMK

represents your netmask.

BCA.BCA.BCA.BCA

represents your Broadcast address.

NWA.NWA.NWA.NWA

represents your Network Address.

Note the use of the `-net` argument to the *route* command. This tells *route* that the route to be added is a route to a *network* and not to a *host*. There is an alternative method of achieving this, you can leave off the `-net` if you have the network address listed in the */etc/networks* file. This is covered later in section '/etc/networks'.

9.6 Configuring a SLIP device (optional)

SLIP (Serial Line Internet Protocol) allows you to use tcp/ip over a serial line, be that a phone line with a dialup modem, or a leased line of some sort. Of course to use SLIP you need access to a *SLIP-server* in your area. Many universities and businesses provide SLIP access all over the world.

Slip uses the serial ports on your machine to carry IP datagrams. To do this it must take control of the serial device. Slip device names are named *sl0*, *sl1* etc. How do these correspond to your serial devices ? The networking code uses what is called an *ioctl* (i/o control) call to change the serial devices into SLIP devices. There are two programs supplied that can do this, they are called *dip* and *slattach*

9.6.1 dip

dip (Dialup IP) is a smart program that is able to set the speed of the serial device, command your modem to dial the remote end of the link, automatically log you into the remote server, search for messages sent to you by the server and extract information for them such as your IP address and perform the *ioctl* necessary to switch your serial port into SLIP mode. *dip* has a powerful scripting ability and it is this that you can exploit to automate your logon procedure.

dip used to be supplied with the *net-tools*, but since development of *dip* is now seperate, you have to source it seperately. There have been a number of other versions of *dip* produced which offer a variety of new features. The `dip-uri` version seems to be the more popular, but I suggest you take a close look at each to determine which offers enhancements that you find useful. Since `dip-uri` is is so popular, the examples described in this document are based on current versions of it.

You can find it at:

sunsite.unc.edu

`/pub/Linux/system/Network/serial/dip337j-uri.tgz`

To install it, try the following:

```
#
# cd /usr/src
# gzip -dc dip337j-uri.tgz | tar xvf -
# cd dip.3.3.7j

<edit Makefile>
```

```
# make install
#
```

The `Makefile` assumes the existence of a group called *uucp*, but you might like to change this to either *dip* or *SLIP* depending on your configuration.

9.6.2 slattach

slattach as contrasted with *dip* is a very simple program, that is very easy to use, but does not have the sophistication of *dip*. It does not have the scripting ability, all it does is configure your serial device as a SLIP device. It assumes you have all the information you need and the serial line is established before you invoke it. *slattach* is ideal to use where you have a permanent connection to your server, such as a physical cable, or a leased line.

9.6.3 When do I use which ?

You would use *dip* when your link to the machine that is your SLIP server is a dialup modem, or some other temporary link. You would use *slattach* when you have a leased line, perhaps a cable, between your machine and the server and there is no special action needed to get the link working. See section 'Permanent Slip connection' for more information.

Configuring SLIP is much like configuring an Ethernet interface (read section 'Configuring an ethernet device' above). However there are a few key differences.

First of all, SLIP links are unlike ethernet networks in that there is only ever two hosts on the network, one at each end of the link. Unlike an ethernet that is available for use as soon are you are cabled, with SLIP, depending on the type of link you have, you may have to initialize your network connection in some special way.

If you are using *dip* then this would not normally be done at boot time, but at some time later, when you were ready to use the link. It is possible to automate this procedure. If you are using *slattach* then you will probably want to add a section to your *rc.inet1* file. This will be described soon.

There are two major types of SLIP servers: Dynamic IP address servers and static IP address servers. Almost every SLIP server will prompt you to login using a username and password when dialing in. *dip* can handle logging you in automatically.

9.6.4 Static SLIP server with a dialup line and DIP.

A static SLIP server in one in which you have been supplied an IP address that is exclusively yours. Each time you connect to the server, you will configure your SLIP port with that address. The static SLIP server will answer your modem call, possibly prompt you for a username and password, and then route any datagrams destined for your address to you via that connection. If you have a static server, then you may want to put entries for your hostname and IP address (since you know what it will be) into your `/etc/hosts`. You should also configure some other files such as: `rc.inet2`, `host.conf`, `resolv.conf`, `/etc/HOSTNAME` and

`rc.local`. Remember that when configuring `rc.inet1`, you don't need to add any special commands for your SLIP connection since it is *dip* that does all of the hard work for you in configuring your interface. You will need to give *dip* the appropriate information and it will configure the interface for you after commanding the modem to establish the call and logging you into your SLIP server.

If this is how your SLIP server works then you can move to section 'Using Dip' to learn how to configure *dip* appropriately.

9.6.5 Dynamic SLIP server with a dialup line and DIP.

A *dynamic* SLIP server is one which allocates you an IP address randomly, from a pool of addresses, each time you logon. This means that there is no guarantee that you will have any particular address each time, and that address may well be used by someone else after you have logged off. The network administrator who configured the SLIP server will have assigned a pool of address for the SLIP server to use, when the server receives a new incoming call, it finds the first unused address, guides the caller through the login process and then prints a welcome message that contains the IP address it has allocated and will proceed to use that IP address for the duration of that call.

Configuring for this type of server is similar to configuring for a static server, except that you must add a step where you obtain the IP address that the server has allocated for you and configure your SLIP device with that.

Again, *dip* does the hard work and new versions are smart enough to not only log you in, but to also be able to automatically read the IP address printed in the welcome message and store it so that you can have it configure your SLIP device with it.

If this is how your SLIP server works then you can move to section 'Using Dip' to learn how to configure *dip* appropriately.

9.6.6 Using DIP.

As explained earlier, *dip* is a powerful program that can simplify and automate the process of dialing into the SLIP server, logging you in, starting the connection and configuring your SLIP devices with the appropriate *ifconfig* and *route* commands.

Essentially to use *dip* you'll write a 'dip script', which is basically a list of commands that *dip* understands that tell *dip* how to perform each of the actions you want it to perform. See `sample.dip` that comes supplied with *dip* to get an idea of how it works. *dip* is quite a powerful program, with many options. Instead of going into all of them here you should looks at the *man* page, README and sample files that will have come with your version of *dip*.

You may notice that the `sample.dip` script assumes that you're using a static SLIP server, so you know what your IP address is beforehand. For dynamic SLIP servers, the newer versions of *dip* include a command you can use to automatically read and configure your SLIP device with the IP address that the dynamic server allocates for you. The following sample is a modified version of the `sample.dip` that came supplied with *dip337j-uri.tgz* and is probably a good starting point for you. You might like to save it as `/etc/dipscript` and edit it to suit your configuration:

```
#
# sample.dip    Dialup IP connection support program.
#
#               This file (should show) shows how to use the DIP
#       This file should work for Annex type dynamic servers, if you
#       use a static address server then use the sample.dip file that
#       comes as part of the dip337-uri.tgz package.
#
#
# Version:      @(#)sample.dip  1.40    07/20/93
#
# Author:       Fred N. van Kempen, <waltje@uWalt.NL.Mugnet.ORG>
#

main:
# Next, set up the other side's name and address.
# My dialin machine is called 'xs4all.hacktic.nl' (== 193.78.33.42)
get $remote xs4all.hacktic.nl
# Set netmask on sl0 to 255.255.255.0
netmask 255.255.255.0
# Set the desired serial port and speed.
port cua02
speed 38400

# Reset the modem and terminal line.
# This seems to cause trouble for some people!
reset

# Note! "Standard" pre-defined "errlevel" values:
#  0 - OK
#  1 - CONNECT
#  2 - ERROR
#
# You can change those grep'ping for "addchat()" in *.c...

# Prepare for dialing.
send ATQ0V1E1X4\r
wait OK 2
if $errlvl != 0 goto modem_trouble
dial 555-1234567
if $errlvl != 1 goto modem_trouble

# We are connected.  Login to the system.
login:
sleep 2
wait ogin: 20
if $errlvl != 0 goto login_trouble
send MYLOGIN\n
```

```
wait ord: 20
if $errlvl != 0 goto password_error
send MYPASSWD\n
loggedin:

# We are now logged in.
wait SOMEPROMPT 30
if $errlvl != 0 goto prompt_error

# Command the server into SLIP mode
send SLIP\n
wait SLIP 30
if $errlvl != 0 goto prompt_error

# Get and Set your IP address from the server.
#   Here we assume that after commanding the SLIP server into SLIP
#   mode that it prints your IP address
get $locip remote 30
if $errlvl != 0 goto prompt_error

# Set up the SLIP operating parameters.
get $mtu 296
# Ensure "route add -net default xs4all.hacktic.nl" will be done
default

# Say hello and fire up!
done:
print CONNECTED $locip ---> $rmtip
mode CSLIP
goto exit

prompt_error:
print TIME-OUT waiting for sliplogin to fire up...
goto error

login_trouble:
print Trouble waiting for the Login: prompt...
goto error

password:error:
print Trouble waiting for the Password: prompt...
goto error

modem_trouble:
print Trouble occurred with the modem...
error:
print CONNECT FAILED to $remote
quit
```

```
exit:
exit
```

The above example assumes you are calling a *dynamic* SLIP server, if you are calling a *static* SLIP server, then the `sample.dip` file that comes with *dip337j-uri.tgz* should work for you.

When *dip* is given the *get $local* command it searches the incoming text from the remote end for a string that looks like an IP address, ie strings numbers separated by '.' characters. This modification was put in place specifically for *dynamic* SLIP servers, so that the process of reading the IP address granted by the server could be automated.

The example above will automatically create a default route via your SLIP link, if this is not what you want, you might have an ethernet connection that should be your default route, then remove the *default* command from the script. After this script has finished running, if you do an *ifconfig* command, you will see that you have a device *sl0*. This is your SLIP device. Should you need to, you can modify its configuration manually, after the *dip* command has finished, using the *ifconfig* and *route* commands.

Please note that *dip* allows you to select a number of different protocols to use with the `mode` command, the most common example is *cSLIP* for SLIP with compression. Please note that both ends of the link must agree, so you should ensure that whatever you select agrees with what your server is set to.

The above example is fairly robust and should cope with most errors. Please refer to the *dip* man page for more information. Naturally you could, for example, code the script to do such things as redial the server if it doesn't get a connection within a prescribed period of time, or even try a series of servers if you have access to more than one.

9.6.7 Permanent SLIP connection using a leased line and slattach.

If you have a cable between two machines, or are fortunate enough to have a leased line, or some other permanent serial connection between your machine and another, then you don't need to go to all the trouble of using *dip* to set up your serial link. *slattach* is a very simple to use utility that will allow you just enough functionality to configure your connection.

Since your connection will be a permanent one, you will want to add some commands to your `rc.inet1` file. In essence all you need to do for a permanent connection is ensure that you configure the serial device to the correct speed and switch the serial device into SLIP mode. *slattach* allows you to do this with one command. **Add** the following to your `rc.inet1` file:

```
#
# Attach a leased line static SLIP connection
#
#  configure /dev/cua0 for 19.2kbps and cslip
/sbin/slattach -p cslip -s 19200 /dev/cua0 &
/sbin/ifconfig sl0 IPA.IPA.IPA.IPA pointopoint IPR.IPR.IPR.IPR up
#
# End static SLIP.
```

Where:

IPA.IPA.IPA.IPA

represents your IP address.

IPR.IPR.IPR.IPR

represents the IP address of the remote end.

slattach allocated the first unallocated SLIP device to the serial device specified. *slattach* starts with *sl0*. Therefore the first *slattach* command attaches SLIP device *sl0* to the serial device specified and *sl1* the next time, etc.

slattach allows you to configure a number of different protocols with the `-p` argument. In your case you will use either *SLIP* or *cSLIP* depending on whether you want to use compression or not. Note: both ends must agree on whether you want compression or not.

9.7 Configuring a PLIP device. (optional)

plip (Parallel Line IP), is like SLIP, in that it is used for providing a *point to point* network connection between two machines, except that it is designed to use the parallel printer ports on your machine instead of the serial ports. Because it is possible to transfer more than one bit at a time with a parallel port, it is possible to attain higher speeds with the *plip* interface than with a standard serial device. In addition, even the simplest of parallel ports, printer ports, can be used, in lieu of you having to purchase comparatively expensive 16550AFN UART's for your serial ports.

Please note that some laptops use chipsets that will not work with PLIP because they do not allow some combinations of signals that PLIP relies on, that printers don't use.

The Linux *plip* interface is compatible with the *Crynwyr Packet Driver PLIP* and this will mean that you can connect your Linux machine to a DOS machine running any other sort of tcp/ip software via *plip*.

You have two options in using the PLIP driver. You can either compile the driver into your kernel, or use the modules package to load the module dynamically. I recommend just compiling it into your kernel as it is probably the easiest and in most circumstances you will want the driver there all the time anyway.

When compiling the kernel, there is only one file that might need to be looked at to configure *plip*. That file is `/usr/src/linux/driver/net/CONFIG` and it contains *plip* timers in mS. The defaults are probably ok in most cases. You will probably need to increase them if you have an especially slow computer, in which case the timers to increase are actually on the **other** computer.

The driver assumes the following defaults:

```
device  i/o addr    IRQ
------  --------    -----
plip0   0x3BC           5
plip1   0x378           7
plip2   0x278           2 (9)
```

If your parallel ports don't match any of the above combinations then you can change the IRQ of a port using the *ifconfig* command using the 'irq' parameter. Be sure to enable IRQ's on your printer ports in your ROM BIOS if it supports this option.

To configure a *plip* interface, you will need to **add** the following lines to your `rc.inet1` file:

```
#
# Attach a PLIP interface
#
#  configure first parallel port as a plip device
/sbin/ifconfig plip0 IPA.IPA.IPA.IPA pointopoint IPR.IPR.IPR.IPR up
#
# End plip
```

Where:

IPA.IPA.IPA.IPA

represents your IP address.

IPR.IPR.IPR.IPR

represents the IP address of the remote machine.

The *pointopoint* parameter has the same meaning as for SLIP, in that it specifies the address of the machine at the other end of the link.

In almost all respects you can treat a *plip* interface as though it were a *SLIP* interface, except that neither *dip* nor *slattach* need be, nor can be, used.

9.7.1 PLIP cabling diagram.

plip has been designed to use cables with the same pinout as those commonly used by the better known of the MS-DOS based pc-pc file transfer programs.

The pinout diagram (taken from `/usr/src/linux/drivers/net/plip.c`) looks as follows:

```
Pin Name    Connect pin - pin
---------   -------------------------------
GROUND      25 - 25
D0->ERROR   2 - 15
ERROR->D0   15 - 2
D1->SLCT    3 - 13
SLCT->D1    13 - 3
D2->PAPOUT  4 - 12
PAPOUT->D2  12 - 4
D3->ACK     5 - 10
ACK->D3     10 - 5
D4->BUSY    6 - 11
```

```
BUSY->D4    11 - 6
D5          7*
D6          8*
D7          9*
STROBE      1*
FEED        14*
INIT        16*
SLCTIN      17*
```

Notes: Do not connect the pins marked with an asterisk '*'. Extra grounds are 18,19,20,21,22,23 and 24.

If the cable you are using has a metallic shield, it should be connected to the metallic DB-25 shell at **one end only**.

Warning: A miswired PLIP cable can destroy your controller card. Be very careful and double check every connection to ensure you don't cause yourself any unnecessary work or heartache.

While you may be able to run PLIP cables for long distances, you should avoid it if you can. The specifications for the cable allow for a cable length of about 1 metre or so. Please be very careful when running long plip cables as sources of strong electromagnetic fields such as lightning, power lines and radio transmitters can interfere with and sometimes even damage your controller. If you really want to connect two of your computers over a large distance you really should be looking at obtaining a pair of thin-net ethernet cards and running some coaxial cable.

10 Routing. (mandatory)

After you have configured all of your network devices you need to think about how your machine is going to route IP datagrams. If you have only one network device configured then your choice is easy, as all datagrams for any machine other than yours must go via that interface. If you have more than one network interface then your choice is a little more complicated. You might have both an ethernet device and SLIP connection to your machine at home. In this situation you must direct all datagrams for your machine at home via your SLIP interface and all else via the ethernet device. Routing is actually a very simple mechanism, but don't worry if you find it slightly difficult to understand at first; everybody does.

You can display the contents of your routing table by using the *route* command without any options.

There are four commonly used routing mechanisms for unix network configurations. I'll briefly discuss each in turn.

10.1 Static/Manual Routes.

Static routing, as its name implies, is 'hard coded' routing, that is, it will not change if your network suffers some failure, or if an alternate route becomes available. Static routes are often used in cases where you have a very simple network with no alternate routes available to a destination host, that is, there is only one possible network path to a destination host, or where you want to route a particular way to a host regardless of network changes.

In Linux there is a special use for manual routes and that is for adding a route to a SLIP or plip host where you have used the *ifconfig pointopoint* parameter. If you have a SLIP/plip link and have the *pointopoint* parameter specifying the address of the remote host, then you should add a static route to that address so that the ip routing software knows how to route datagrams to that address. The *route* command you would use for the SLIP/plip link via leased line example presented earlier would be:

```
#/sbin/route add IPR.IPR.IPR.IPR
```

Where:

IPR.IPR.IPR.IPR

represents the IP address of the remote end.

10.2 Default Route.

The *default route* mechanism is probably the most common and most useful to most end-user workstations and hosts on most networks. The *default* route is a special static route that matches every destination address, so that if there is no more specific route for a datagram to be sent to, then the *default* route will be used.

If you have a configuration where you have only a single ethernet interface, or a single SLIP interface device defined then you should point your default route via it. In the case of an ethernet interface, the Linux kernel knows where to send datagrams for any host on your network. It works this out using the network address and the network mask as discussed earlier. This means that the only datagrams the kernel won't know how to properly route will be those for people not on your network. To make this work you would normally have your default route point to your *router* address, as it is your means of getting outside of your local network. If you are using a SLIP connection, then your *SLIP server* will be acting as your *router*, so your default route will be via your *SLIP server*.

To configure your default route, **add** the following to your `rc.inet1` **after** all of your network device configurations:

```
#
# Add a default route.
#
/sbin/route add default gw RGA.RGA.RGA.RGA
#
```

Where:

RGA.RGA.RGA.RGA

represents your Router/Gateway Address.

10.3 Proxy ARP.

This method is ugly, hazard prone and should be used with extreme care, some of you will want to use it anyway.

Those with the greatest need for *proxy arp* will be those of you who are configuring your Linux machine as a SLIP dial-in server. For those of you who will be using PPP, the PPP daemon simplifies and automates this task, making it a lot safer to use.

Normally when a tcp/ip host on your ethernet network wants to talk to you, it knows your IP address, but doesn't know what hardware (ethernet) address to send datagrams to. The ARP mechanism is there specifically to provide that mapping function between network address and hardware address. The ethernet protocol provides a special address that is recognised by all ethernet cards, this is called the broadcast address. ARP works by sending a specially formatted datagram containing the IP address of the host it wishes to discover the hardware address of and transmits it to the ethernet broadcast address. Every host will receive this datagram and the host that is configured with the matching IP address will reply with its hardware address. The host that performed the arp will then know what hardware address to use for the desired IP address.

If you want to use your machine as a server for other machines, you must get your machine to answer ARP requests for their IP addresses on their behalf, as they will not be physically connected to the ethernet network. Lets say that you have been assigned a number of IP addresses on your local network that you will be offering to dial-in SLIP users. Lets say those addresses are: `128.253.154.120-124` and that you have an ethernet card with a hardware address of `00:00:C0:AD:37:1C`. (You can find the hardware address of your ethernet card by using the *ifconfig* command with no options). To instruct your Linux server to answer arp requests by proxy for these addresses you would need to **add** the following commands to the end of your `rc.inet1` file:

```
#
# Proxy ARP for those dialin users who will be using this
#           machine as a server:
#
/sbin/arp -s 128.263.154.120 00:00:C0:AD:37:1C pub
/sbin/arp -s 128.263.154.121 00:00:C0:AD:37:1C pub
/sbin/arp -s 128.263.154.122 00:00:C0:AD:37:1C pub
/sbin/arp -s 128.263.154.123 00:00:C0:AD:37:1C pub
/sbin/arp -s 128.263.154.124 00:00:C0:AD:37:1C pub
#
# End proxy arps.
```

The `pub` argument stands for *'publish'*. It is this argument that instructs your machine to answer requests for these addresses, even though they are not for your machine. When it answers it will supply the hardware address specified, which is of course its own hardware address.

Naturally you will need to ensure that you have routes configured in your linux server that point these addresses to the SLIP device on which they will be connecting.

If you are using PPP, you don't need to worry about manually messing with the arp table, as the *pppd* will manage those entries for you if you use the *proxyarp* parameter and as long as the IP addresses of the

remote machine and the server machine are in the same network. You will need to supply the netmask of the network on the server's *pppd* command line.

10.4 gated - the routing daemon.

gated could be used in place of *proxy arp* in some cases and would certainly be much cleaner, but its primary use is if you want your linux machine to act as an intelligent *ip router* for your network. *gated* provides support for a number of routing protocols. Among these are RIP, BGP, EGP, HELLO and OSPF. The most commonly used in small networks being *rip*. *rip* stands for *'Routing Information Protocol'*. If you run *gated*, configured for *rip*, your linux machine will periodically broadcast a copy of its *routing table* to your network in a special format. In this way, all of the other machines on your network will know what addresses are accessible via your machine.

gated can be used to replace *proxy arp* when all hosts on your network run either *gated* or *routed*. If you have a network where you use a mixture of manual and dynamic routes, you should mark any manual routes as `passive` to ensure that they aren't destroyed by *gated* because it hasn't received an update for them. The best way to add static routes if you are using *gated* is to add a `static` stanza to your `/etc/gated.conf` file. This is described below.

gated would normally be started from your `rc.inet2` which is covered in the next section. You might already see a daemon called *routed* running. *gated* is superior to *routed* in that it is more flexible and more functional. So you should use *gated* and not *routed*.

10.4.1 Obtaining *gated*

Gated is available from:

sunsite.unc.edu

`/pub/Linux/system/Network/daemons/gated11_bin.tgz`

10.4.2 Installing *gated*

The *gated* binary distribution comprises three programs and two sample configuration files.

The programs are:

gated

the actual *gated* daemon.

gdc

the operational user interface for *gated*. *gdc* is for controlling the *gated* daemon, stopping and starting it, obtaining its status and the like.

ripquery

a diagnostic tool to query the known routes of a gateway using either a 'rip query' or a 'rip poll'.

The configuration files are:

gated.conf

this is the actual configuration file for the *gated* daemon. It allows you to specify how *gated* will behave when it is running. You can enable and disable any of the routing protocols and control the behaviour of those routing protocols running.

gated.version

a text file that describes the version number of the *gated* daemon

The *gated* binary distribution will not install the *gated* files in the correct place for you. Fortunately there aren't very many, so its fairly simple to do.

To install the binaries try the following:

```
# cd /tmp
# gzip -dc .../gated.linux.bin.tgz | tar xvf -
# install -m 500 bin/gated /usr/sbin
# install -m 444 bin/gated.conf bin/gated.version /etc
# install -m 555 bin/ripquery bin/gdc /sbin
# rm -rf /tmp/bin
```

I keep the networking daemons in `/usr/sbin`, if yours are somewhere else then naturally you'll have to change the target directory. The sample *gated* configuration file included configures *gated* to emulate the old *routed* daemon. It will probably work for you in most circumstances and it looks like this:

```
#
#  This configuration emulates routed.  It runs RIP and only sends
#  updates if there are more than one interfaces up and IP forwarding is
#  enabled in the kernel.
#
#       NOTE that RIP *will not* run if UDP checksums are disabled in
#       the kernel.
#
rip yes ;
traceoptions all;
#
```

If you have any static routes you wish to add, you can add them in a `static` stanza appended to your `/etc/gated.conf` as follows:

```
#
static {
37.0.0.0 mask 255.0.0.0 gateway 44.136.8.97 ;
host 44.136.8.100 gateway 44.136.8.97 ;
} ;
#
```

The above example would create a static route to the Class A network `37.0.0.0` via gateway `44.136.8.97` and a static route to a host with address `44.136.8.100` via gateway `44.136.8.97`. If you do this you do not need to add the routes using the *route* command, *gated* will add and manage the routes for you.

To install the *man* files, try the following:

```
# cd /tmp
# gzip -dc .../gated.linux.man.tgz | tar xvf -
# install -m 444 man/*.8 /usr/man/man8
# install -m 444 man/*.5 /usr/man/man5
# rm -rf /tmp/man
```

The *man* files contain concise and detailed information on the configuration and use of *gated*. For information on configuring *gated*, refer to the `gated-config` *man* page.

11 Configuring the network daemons.

As mentioned earlier, there are other files that you will need to complete your network installation. These files concern higher level configurations of the network software. Each of the important ones are covered in the following sub-sections, but you will find there are others that you will have to configure as you become more familiar with the network suite.

11.1 /etc/rc.d/rc.inet2 (the second half of rc.net)

If you have been following this document you should at this stage have built an *rc* file to configure each of your network devices with the correct addresses and set up whatever routing you will need for your particular network configuration. You will now need to actually start some of the higher level network software.

Now would be a really good time to read Olaf's *Network Administrators Guide*, as it really should be considered the definitive document for this stage of the configuration process. It will help you decide what to include in this file and more importantly perhaps, what **not** to include in this file. For the security conscious it is a fair statement to say that the more network services you have running, the more likely the chance of your system having a security hole: Run only what you need.

There are some very important *daemons* (system processes that run in the background) that you will need to know a little about. The *man* pages will tell you more, but they are:

11.1.1 inetd.

inetd is a program that sits in the background and manages internet connection requests and the like. It is smart enough that you don't need to leave a whole bunch of servers running when there is nothing connected to them. When it sees an incoming request for a particular service, eg *telnet*, or *ftp*, it will check the `/etc/services` file, find what server program needs to be run to manage the request, start it and hand the connection over to it. Imagine it as a master server for your internet servers. It also has a few simple standard services inbuilt. These are *echo*, *discard* and *generate* services used for various types of network

testing. *inetd* doesn't manage **all** servers and services that you might run, but it manages most of the usual ones. Normally services such as `udp` based services, or services that manage their own connection multiplexing such as World Wide Web servers or muds would be run independently of *inetd*. Generally the documentation accompanying such servers will tell you whether to use *inetd* or not.

11.1.2 syslogd.

syslogd is a daemon that handles all system logging. It accepts messages generated for it and will distribute them according to a set of rules contained in `/etc/syslogd.conf`. For example, certain types of messages you will want to send to the console and also to a log file, where others you will want only to log to a file. *syslogd* allows you to specify what messages should go where.

11.2 A sample rc.inet2 file.

The following is a sample rc.inet2 file that Fred built. It starts a large number of servers, so you might want to trim it down to just those services that you actually want to run. To trim it down, simply delete or comment out the stanzas (*if* to *fi*) that you don't need. All each stanza does is test that the relevant module is a file, that it exists, echoes a comment that you can see when you boot your machine and then executes the commands with the arguments supplied to ensure that it runs happily in the background. For more detailed information on each of the deamons, check either the *Network Administrators Guide* or the relevant *man* pages.

```
#! /bin/sh
#
# rc.inet2      This shell script boots up the entire INET system.
#               Note, that when this script is used to also fire
#               up any important remote NFS disks (like the /usr
#               distribution), care must be taken to actually
#               have all the needed binaries online _now_ ...
#
# Version:      @(#)/etc/rc.d/rc.inet2  2.18    05/27/93
#
# Author:       Fred N. van Kempen, <waltje@uwalt.nl.mugnet.org>
#

# Constants.
NET="/usr/sbin"
IN_SERV="lpd"
LPSPOOL="/var/spool/lpd"

# At this point, we are ready to talk to The World...
echo -e "\nMounting remote file systems ..."
/bin/mount -t nfs -v            # This may be our /usr runtime!!!

echo -e "\nStarting Network daemons ..."
# Start the SYSLOG daemon.  This has to be the first server.
```

```
# This is a MUST HAVE, so leave it in.
echo -n "INET: "
if [ -f ${NET}/syslogd ]
then
echo -n "syslogd "
${NET}/syslogd
fi

# Start the SUN RPC Portmapper.
if [ -f ${NET}/rpc.portmap ]
then
echo -n "portmap "
${NET}/rpc.portmap
fi

# Start the INET SuperServer
# This is a MUST HAVE, so leave it in.
if [ -f ${NET}/inetd ]
then
echo -n "inetd "
${NET}/inetd
else
echo "no INETD found.  INET cancelled!"
exit 1
fi

# Start the NAMED/BIND name server.
# NOTE: you probably don't need to run named.
#if [ ! -f ${NET}/named ]
#then
#        echo -n "named "
#        ${NET}/named
#fi

# Start the ROUTEd server.
# NOTE: routed is now obsolete. You should now use gated.
#if [ -f ${NET}/routed ]
#then
#        echo -n "routed "
#        ${NET}/routed -q #-g -s
#fi

# Start the GATEd server.
if [ -f ${NET}/gated ]
then
echo -n "gated "
${NET}/gated
fi
```

```
# Start the RWHO server.
if [ -f ${NET}/rwhod ]
then
echo -n "rwhod "
${NET}/rwhod -t -s
fi

# Start the U-MAIL SMTP server.
if [ -f XXX/usr/lib/umail/umail ]
then
echo -n "umail "
/usr/lib/umail/umail -d7 -bd </dev/null >/dev/null 2>&1 &
fi

# Start the various INET servers.
for server in ${IN_SERV}
do
if [ -f ${NET}/${server} ]
then
                echo -n "${server} "
                ${NET}/${server}
fi
done

# Start the various SUN RPC servers.
if [ -f ${NET}/rpc.portmap ]
then
if [ -f ${NET}/rpc.ugidd ]
then
                echo -n "ugidd "
                ${NET}/rpc.ugidd -d
fi
if [ -f ${NET}/rpc.mountd ]
then
                echo -n "mountd "
                ${NET}/rpc.mountd
fi
if [ -f ${NET}/rpc.nfsd ]
then
                echo -n "nfsd "
                ${NET}/rpc.nfsd
fi

# Fire up the PC-NFS daemon(s).
if [ -f ${NET}/rpc.pcnfsd ]
then
                echo -n "pcnfsd "
```

```
                ${NET}/rpc.pcnfsd ${LPSPOOL}
fi
if [ -f ${NET}/rpc.bwnfsd ]
then
                echo -n "bwnfsd "
                ${NET}/rpc.bwnfsd ${LPSPOOL}
fi

fi
echo network daemons started.
# Done!
```

11.3 Other necessary network configuration files.

There are other network configuration files that you will need to configure if you want to have people connect to and use your machine as a host. If you have installed your linux from a distribution then you will probably already have copies of these files so just check them to make sure they look ok and if not you can use the following samples.

11.3.1 A sample /etc/inetd.conf file.

Your `/etc/rc.d/rc.inet2` file will have started *inetd*, *syslogd* and the various *rpc* servers for you. You will now need to configure the network daemons that will be managed by *inetd*. *inetd* uses a configuration file called `/etc/inetd.conf`.

The following is an example of how a simple configuration might look:

```
#
# The internal services.
#
# Authors:      Original taken from BSD UNIX 4.3/TAHOE.
#               Fred N. van Kempen, <waltje@uwalt.nl.mugnet.org>
#
echo    stream tcp nowait root  internal
echo    dgram  udp wait   root  internal
discard stream tcp nowait root  internal
discard dgram  udp wait   root  internal
daytime stream tcp nowait root  internal
daytime dgram  udp wait   root  internal
chargen stream tcp nowait root  internal
chargen dgram  udp wait   root  internal
#
# Standard services.
#
ftp     stream tcp nowait root  /usr/sbin/tcpd in.ftpd ftpd
telnet  stream tcp nowait root  /usr/sbin/tcpd in.telnetd
#
```

```
# Shell, login, exec and talk are BSD protocols.
#
shell   stream tcp nowait root  /usr/sbin/tcpd in.rshd
login   stream tcp nowait root  /usr/sbin/tcpd in.rlogind
exec    stream tcp nowait root  /usr/sbin/tcpd in.rexecd
talk    dgram  udp wait   root  /usr/sbin/tcpd in.talkd
ntalk   dgram  udp wait   root  /usr/sbin/tcpd in.talkd
#
# Status and Information services.
#
finger  stream tcp nowait root  /usr/sbin/tcpd in.fingerd
systat  stream tcp nowait guest /usr/sbin/tcpd /usr/bin/ps -auwwx
netstat stream tcp nowait guest /usr/sbin/tcpd /bin/netstat
#
# End of inetd.conf.
```

The *inetd* man page describes what each of the fields are, but put simply, each entry describes what program should be executed when an incoming connection is received on the socket listed as the first entry. Those entries which have `incoming` where the program name and arguments would be are those services that are provided internally by the *inetd* program.

The conversion between the service name in the first column and the actual socket number it refers to is performed by the `/etc/services` file.

11.3.2 A sample /etc/services file.

The `/etc/services` file is a simple table of Internet service names and the socket number and protocol is uses. This table is used by a number of programs including *inetd*, *telnet* and *tcpdump*. It makes life a little easier by allowing us to refer to services by name rather than by number.

The following is a sample of what a simple `/etc/services` file might look like:

```
#
# /etc/services - database of service name, socket number
#                 and protocol.
#
# Original Author:
#     Fred N. van Kempen, <waltje@uwalt.nl.mugnet.org>
#
tcpmux     1/tcp
echo       7/tcp
echo       7/udp
discard    9/tcp   sink null
discard    9/udp   sink null
systat     11/tcp  users
daytime    13/tcp
daytime    13/udp
netstat    15/tcp
```

```
chargen    19/tcp  ttytst source
chargen    19/udp  ttytst source
ftp-data   20/tcp
ftp        21/tcp
telnet     23/tcp
smtp       25/tcp  mail
time       37/tcp  timserver
time       37/udp  timserver
name       42/udp  nameserver
whois      43/tcp  nicname    # usually to sri-nic
domain     53/tcp
domain     53/udp
finger     79/tcp
link       87/tcp  ttylink
hostnames  101/tcp hostname   # usually to sri-nic
sunrpc     111/tcp
sunrpc     111/tcp portmapper # RPC 4.0 portmapper TCP
sunrpc     111/udp
sunrpc     111/udp portmapper # RPC 4.0 portmapper UDP
auth       113/tcp authentication
nntp       119/tcp usenet     # Network News Transfer
ntp        123/tcp            # Network Time Protocol
ntp        123/udp            # Network Time Protocol
snmp       161/udp
snmp-trap  162/udp
exec       512/tcp            # BSD rexecd(8)
biff       512/udp comsat
login      513/tcp            # BSD rlogind(8)
who        513/udp whod       # BSD rwhod(8)
shell      514/tcp cmd        # BSD rshd(8)
syslog     514/udp            # BSD syslogd(8)
printer    515/tcp spooler    # BSD lpd(8)
talk       517/udp            # BSD talkd(8)
ntalk      518/udp            # SunOS talkd(8)
route      520/udp routed     # 521/udp too
timed      525/udp timeserver
mount      635/udp            # NFS Mount Service
pcnfs      640/udp            # PC-NFS DOS Authentication
bwnfs      650/udp            # BW-NFS DOS Authentication
listen     1025/tcp listener  # RFS remote_file_sharing
ingreslock 1524/tcp           # ingres lock server
nfs        2049/udp           # NFS File Service
irc        6667/tcp           # Internet Relay Chat
# End of services.
```

The *telnet* entry tells us that the *telnet* service uses socket number 23 and the *tcp* protocol. The *domain* entry tells us that the Domain Name Service uses socket number 52 and both *tcp* and *udp* protocols. You should have an appropriate `/etc/services` entry for each `/etc/inetd.conf` entry.

11.3.3 A sample /etc/protocols file.

The `/etc/protocols` file is a table of protocol name with its corresponding protocol number. Since the number of protocols in use is small this file is quite trivial.

```
#
# /etc/protocols - database of protocols.
#
# Original Author:
#   Fred N. van Kempen, <waltje@uwalt.nl.mugnet.org>
#
ip   0   IP   # internet protocol
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
pup  12  PUP  # PARC universal packet protocol
udp  17  UDP  # user datagram protocol
idp  22  IDP
raw  255 RAW
#
# End of protocols.
```

11.4 Name Resolution.

Name Resolution is the process of converting a hostname in the familiar dotted notation (e.g. `tsx-11.mit.edu`) into an IP address which the network software understands. There are two principal means of achieving this in a typical installation, one simple and one more complex.

11.4.1 /etc/hosts

`/etc/hosts` contains a list of ip addresses and the hostnames they map to. In this way, you can refer to other machines on the network by name, as well as their ip address. Using a nameserver (see section 'named') allows you to do the same name-¿ip address translation automatically. (Running *named* allows you to run your own nameserver on your linux machine). This file needs to contain at least an entry for `127.0.0.1` with the name `localhost`. If you're not only using loopback, you need to add an entry for your ip address, with your full hostname (such as `loomer.vpizza.com`). You may also wish to include entries for your gateways and network addresses.

For example, if `loomer.vpizza.com` has the ip address `128.253.154.32`, the `/etc/hosts` file would contain:

```
# /etc/hosts
# List of hostnames and their ip addresses
127.0.0.1               localhost
128.253.154.32          loomer.vpizza.com loomer
# end of hosts
```

Once again you will need to edit this file to suit your own needs. If you're only using loopback, the only line in `/etc/hosts` should be for `127.0.0.1`, with both `localhost` and your hostname after it.

Note that in the second line, above, there are two names for `128.253.154.32`: `loomer.vpizza.com` and just `loomer`. The first name is the full hostname of the system, called the "Fully Qualified Domain Name" and the second is an alias for it. The second allows you to type only *rlogin loomer* instead of having to type the entire hostname. You should ensure that you put the Fully Qualified Domain Name in the line before the alias name.

11.4.2 named - do I need thee ?

'I dub thee ..'

named is the nameserver daemon for many unix-like operating systems. It allows your machine to serve the name lookup requests, not only for itself, but also for other machines on the network, that is, if another machine wants to find the address for `'goober.norelco.com'` and you have this machines address in your *named* database, then you can service the request and tell other machines what `'goobers'` address is.

Under older implementations of Linux tcp/ip, to create aliases for machine names, (even for your own machine), you had to run *named* on your Linux machine to do the hostname to IP address conversion. One problem with this is that *named* is comparatively difficult to set up properly and maintain. To solve this problem, a program called `hostcvt.build` was made available on Linux systems to translate your `/etc/hosts` file into the many files that make up *named* database files. However even with this problem overcome, *named* still uses CPU overhead and causes network traffic.

The only reason you may want to run *named* would be if:

- You're setting up a network of machines and need a nameserver for one of them and don't have a nameserver out on the net somewhere.
- Your network administrators want you to run your Linux system as a nameserver for some reason.
- You have a slow SLIP connection and want to run a small cache-only nameserver on your Linux machine so that you don't have to go out on the serial line for every name lookup that occurs. If you're only going to be connecting to a small number of hosts on the net and you know what their addresses are, then you can put them in your `hosts` file and not need to query a nameserver at all. Generally namelookup isn't that slow and should work fine over a SLIP link anyway.
- You want to run a nameserver for fun and excitement.

In general, **you do NOT need to run named**: this means that you can comment it out from your `rc.inet2` file and you don't have to run `hostcvt.build`. If you want to alias machine names, for example, if you want to refer to `loomer.vpizza.com` as just `loomer`, then you can add as alias in `/etc/hosts` instead. There is no reason to run *named* unless you have a specific requirement to do so. If you have access to a nameserver, (and your network administrators will tell you its address) and most networks do, then don't bother running *named*.

If you're only using loopback, you can run *named* and set your nameserver address to `127.0.0.1`, but since you are the only machine you can talk to, this would be quite bizarre, as you'd never need to call it.

11.4.3 /etc/networks

The `/etc/networks` file lists the names and addresses of your own, and other, networks. It is used by the *route* command and allows you to specify a network by name, should you so desire.

Every network you wish to add a route to using the *route* command should have an entry in the `/etc/networks` file, unless you also specify the `-net` argument in the *route* command line.

Its format is similar to that of `/etc/hosts` file above and an example file might look like:

```
#
# /etc/networks: list all networks that you wish to add route commands
#                for in here
#
default         0.0.0.0         # default route    - recommended
loopnet         127.0.0.0       # loopback network - recommended
mynet           128.253.154.0   # Example network CHANGE to YOURS
#
# end of networks
```

11.4.4 /etc/host.conf

The system has some library functions called the resolver library. This file specifies how your system will lookup host names. It should contain at least the following two lines:

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

These two lines tell the *resolve* libraries to first check the `/etc/hosts` file and then to ask the nameserver (if one is present). The *multi* entry allows you to have multiple IP addresses for a given machine name in `/etc/hosts`.

This file comes from the implementation of the *resolv+* bind library for Linux. You can find further documentation in the *resolv+(8)* man page if you have it. If you don't, it can be obtained from:

sunsite.doc.ic.ac.uk

```
/computing/comms/tcpip/nameserver/resolv+/resolv+2.1.1.tar.Z
```

This file contains the *resolv+.8* man page for the resolver library.

11.4.5 /etc/resolv.conf

This file actually configures the system name resolver and contains two types of entries: The addresses of your nameservers (if any) and the name of your domain, if you have one. If you're running your own nameserver (i.e running *named* on your Linux machine), then the address of your nameserver is `127.0.0.1`, the loopback address.

Your domain name is your fully qualified hostname (if you're a registered machine on the Internet, for example), with the hostname component removed. That is, if your full hostname is `loomer.vpizza.com`, then your domain name is `vpizza.com`, without the hostname `loomer`.

For example, if you machine is `goober.norelco.com` and has a nameserver at the address `128.253.154.5`, then your `/etc/resolv.conf` file would look like:

```
domain norelco.com
nameserver 128.253.154.5
```

You can specify more than one nameserver. Each one must have a *nameserver* entry in the `resolv.conf` file.

Remember, if you're running on loopback, you don't need a nameserver.

11.4.6 Configuring your Hostname - /etc/HOSTNAME

After you have configured everything else, there is one small task that remains, you need to configure your own machine with a name. This is so that application programs like *sendmail* can know who you are to accept mail and so that your machine can identify itself to other machines that it might be connected to.

There are two programs that are used to configure this sort of information, and they are commonly misused. They are *hostname* and *domainname*.

If you are using a release of `net-tools` earlier than `1.1.38` then you can include a command in your `/etc/rc` file that looks like this:

```
/bin/hostname -S
```

and this will cause the *hostname* command to read a file called `/etc/HOSTNAME` which it expects will contain a "Fully Qualified Domain Name", that is, your machines hostname **including** the domainname. It will split the F.Q.D.N. into its DNS hostname and domainname components and set them appropriately for you.

For example, the machine above would have the file `/etc/HOSTNAME`:

```
goober.norelco.com
```

If you are using the *hostname* that came with `net-tools-1.1.38` or later, then you would add a command at the end of your `/etc/rc.d/rc.inet1` file like:

```
/bin/hostname goober.norelco.com
```

or if you have upgraded from a previous release, you could add:

```
/bin/hostname -F /etc/HOSTNAME
```

and it would behave in the same way as for the earlier version.

The `/bin/domainname` command is for setting the **N.I.S.** domain name **NOT** the D.N.S. domain name. You do not need to set this unless you are running *NIS*, which is briefly described later.

11.5 Other files.

There are of course many other files in the `/etc` directory which you may need to dabble with later on. Instead of going into them here, I'm going to provide the bare minimum to get you on the net. More information is available in Olaf's *Network Administration Guide*. It picks up where this *HOWTO* ends and some more information will be provided in later versions of this document.

Once you have all of the files set up and everything in the right place, you should be able to reboot your new kernel and net away to your hearts content. However I strongly suggest that you keep a bootable copy of your old kernel and possibly even a 'recovery disk' in case something goes wrong so that you can get back in and fix it. You might try HJLu's 'single disk boot disk', or 'disk1' from a Linux distribution.

12 Advanced Configurations.

The configurations above have described how a typical Linux workstation might be configured for normal end-user operation. Some of you will have other requirements which will require slightly more advanced configurations. What follows are examples of some the more common of these.

The details of the AX.25, Ottawa PI and generic SCC drivers have been moved to the *HAM-HOWTO* `<http://sunsite.unc.edu/mdw/HOWTO/HAM-HOWTO.html>`.

12.1 PPP - Point to Point Protocol.

The *Point to Point Protocol* is a modern and efficient protocol for conveying multiple protocols, tcp/ip for one, across serial links, that a lot of people use in place of SLIP. It offers enhanced functionality, error detection and security options. It corrects a number of deficiencies that are found in SLIP and is suitable for both asynchronous links and synchronous links alike.

An important feature of PPP operation is the ability to negotiate such parameters as IP address allocation automatically and with ease and this feature will almost certainly be exploited by your PPP server. This feature allows a PPP client, with a specially formatted frame, to request its address from the server. In this way configuration is somewhat less messy than with SLIP, since this ability to retrieve your address must occur outside of the protocol.

The authors of the Linux port are `Michael Callahan, <callahan@maths.ox.ac.uk>` and `Al Longyear, <longyear@netcom.com>`. Most of this information has come from the documentation that accompanies the PPP software. The documentation is quite complete and will tell you much more than I present here.

The Linux PPP code is now well and truly a public release. The 1.0.0 Linux PPP code is based on Paul Mackerras's free PPP for BSD-derivative operating systems. The 1.0.0 release is based on version 2.1.1 of the free PPP code.

The PPP code comes in two parts. The first is a kernel module which handles the assembly and disassembly of the frames and the second is a set of protocols called LCP, IPCP, UPAP and CHAP, for negotiating link options, bringing the link into a functioning state and for authentication.

12.1.1 Why would I use PPP in place of SLIP ?

You would use PPP in place of SLIP for a few reasons. The most common are:

Your Internet Provider supports only PPP

The most obvious reason you would use PPP in favour of SLIP is when your Internet Provider supports PPP and not SLIP. Ok, I said it was obvious.

You have a normally noisy serial line

PPP provides a frame check sequence for each and every frame transmitted, SLIP does not. If you have a noisy serial line and you are using SLIP, your error correction will be performed end to end, that is between your machine and the destination machine, whereas with PPP the error detection occurs locally, between your machine and the PPP server. This makes for faster recovery from errors.

You need to make use of some other feature PPP offers.

PPP provides a number of features that SLIP does not. You might for example want to carry not only IP, but also DECNET, or AppleTalk frames over your serial link. PPP will allow you to do this.

12.1.2 Where to obtain the PPP software.

The PPP software is available from:

sunsite.unc.edu

`/pub/Linux/system/Networking/serial/ppp-2.1.2d.tar.gz`

This file contains the kernel source and the *pppd* source and binary. Version 1.0.0 is meant for use with kernels 1.0.x and 1.1.x. Version 2.1.2 is intended for kernel version 1.2.x, and the latest 2.2.0 pppd is intended for kernels 1.3.x.

12.1.3 Installing the PPP software.

Installation of the PPP software is fairly straightforward.

The kernel driver. Some support for *ppp* has been built into the kernel for some time. so you are advised to run a modern kernel. Configuring the kernel is fairly easy, the following should work ok:

```
# make config   (remembering to answer yes to PPP support)
# make dep
# make          (remember to install the new kernel after recompiling!)
```

When you reboot with the new kernel you should see messages at boot time that look something like these:

```
PPP: version 0.2.7 (4 channels) NEW_TTY_DRIVERS OPTIMIZE_FLAGS
TCP compression code copyright 1989 Regents of the University of California
PPP line discipline registered.
```

These indicate that the PPP support has in fact been compiled into your kernel.

Now, try looking at the contents of `/proc/net/dev`. You should be careful not to use *more* or *less* on the files in the `/proc` filesystem, as some of them check the filesize first and it is a feature of the `/proc` filesystem that the files are zero length, so use:

```
# cat /proc/net/dev
```

It should look something like this:

```
Inter-|   Receive                |  Transmit
 face |packets errs drop fifo frame|packets errs drop fifo colls carrier
lo:        0    0    0    0    0        0    0    0    0     0    0
ppp0:      0    0    0    0    0        0    0    0    0     0    0
ppp1:      0    0    0    0    0        0    0    0    0     0    0
ppp2:      0    0    0    0    0        0    0    0    0     0    0
ppp3:      0    0    0    0    0        0    0    0    0     0    0
```

This indicates that the kernel driver is installed correctly.

pppd To extract the ppp software and associated utilities the following should work ok:

```
# cd /usr/src
# gzip -dc ppp-2.1.2d.tar.gz | tar xvf -
```

If you want to recompile *pppd*, type *make* in the `pppd` subdirectory of the installation. There will be some warnings when compiling `lcp.c`, `upap.c` and `chap.c` but these are OK.

If you want to recompile *chat*, consult `README.linux` in the `chat` directory.

To install, type *make install* in the `chat` and `pppd` directories. This will put *chat* and *pppd* binaries in `/usr/sbin` and the `pppd.8` manual page in `/usr/man/man8`.

`pppd` needs to be run as `root`. You can either make it suid root or just use it when you are root. *make install* will try to install it suid root, so if you are root when you try to install it, it should work ok.

12.1.4 Configuring and using the PPP software.

Like SLIP, you can configure the PPP software as either a client or a server. The *chat* program performs a similar function to the *dip* program in that it is used to automate the dialing and login procedure to the remote machine, unlike *dip* though, it does not perform the *ioctl* to convert the serial line into a PPP line. This is performed by the *pppd* program. *pppd* can act as either the client or the server. When used as a client, it normally invokes the *chat* program to perform the connection and login and then it takes over by performing the *ioctl* to change the line discipline to *ppp*, performs a number of steps in configuring your machine to talk to the remote machine and then steps out of the way to let you operate.

Please refer to the *pppd* and *chat* man pages for more information. Please also refer to the README file that comes with the ppp software, as its description of the operation of these utilities is much more complete than I have described here.

Configuring a PPP client by dial-up modem. This is perhaps what most of you will want to do, so it appears first. You would use this configuration when you have a network provider who supports ppp by dialup modem. When you want to establish your connection you simply invoke the *pppd* program with no arguments. This configuration assumes that your PPP server will assign you an address dynamically. What to do if you have a static address is explained later.

The *pppd* program has a number of command line arguments that govern its behaviour. These arguments can all be placed in a configuration file so that the commands do not appear in the argument list when someone on the system uses the *ps* command. This is particularly important when *pppd* invokes the chat program as it needs the password to use to login to the server and you don't normally want this visible to users of the system.

The file that *pppd* uses is called the `/etc/ppp/options` file.

A good starting point for a 'typical' PPP installation might look like:

```
connect /etc/ppp/ppp-connect
/dev/ttyS1
19200
crtscts
modem
lock
asyncmap 0
defaultroute
```

This specifies that:

1. *pppd* should use the `/etc/ppp/ppp-connect` script to dial the modem and login.
2. *pppd* should use the `ttyS1` device at `19200` bits per second, with `crtscts` hardware handshaking and respect the modem signals (Carrier Detect particularly) to detect if the modem is still on-line or if the call has finished.
3. *pppd* should create a `lock` file for the `tty` device to ensure that other processes do not attempt to open it while *pppd* is using it.
4. That the line is '8 bit clean', that is, that all characters are able to pass freely across the line. The `asyncmap` option determines which characters the *pppd* should escape and by default this is set to all control characters. Most modern PPP servers are 8 bit clean and setting the `asyncmap` to `0` tells *pppd* not to escape any characters.
5. *pppd* should create a `defaultroute` via the ppp device when the connection is successfully established.

The next step is to configure the `/etc/ppp/ppp-connect` script. You would normally use the *chat* program that comes with *pppd* within this script as it is very simple to use and quite lightweight.

To start, you need to know what the login sequence for your PPP server looks like. The following is a fictional example that I'll base the sample script on:

```
CONNECT 14400

Welcome to XYZ PPP server!

login:                      <you enter your username>^M
password:                   <you enter your password>^M

Now entering PPP:
```

The *chat* program expects as arguments a send/expect sequence, meaning that it wants a series of '*what it should expect*, followed by *what it should send*' pairs. The `/etc/ppp/ppp-connect` should **not** not be world readable. A sample to deal with the hypothetical example above might look like:

```
#!/bin/sh
# A chat script to login to the XYZ PPP server.
#
NUM=5552857
UID=terryd
PASSWD=secret1
#
/usr/sbin/chat -v "" ATZ OK ATDT$NUM CONNECT ogin: $UID word: \\q$PASSWD PPP:
```

I've embellished the script with a sequence to initialise the modem before dialling (the `ATZ`. It is good practise to initialise your modem to something sensible before you start).

Note the `""` argument, this means '*wait for nothing*' and is used when you want to start the sequence by sending something.

The `-v` argument tells *chat* to be **verbose**. In this mode it will send a transcript of the logon to *syslog* so that you can see what is happening. Note the
`q` before the `$PASSWD` in the script. This tells *chat* not to echo the following text to *syslog*. In this way your password won't be logged in the system log.

You would want to use the `chmod 600 /etc/ppp/ppp-connect` to ensure that other users cannot read the script file and obtain your password.

To start your PPP session with the above configuration you would need only type: *pppd* and it all should happen automagically. You should observe the logon occurring in the system log and when it is finished you should see a `default` route in your routing table when you use the: `route -n` command pointed via your new `ppp0` device.

If you use a static IP address you can specify that by including a line similar to the following in your `/etc/ppp/options` file:

```
nnn.nnn.nnn.nnn:
```

Where: `nnn.nnn.nnn.nnn` is your IP addresss. Note this will only work if your Internet Service Providers is configured to allow you to do so. The colon ':' is important.

There are many other options you may want to read about that you can include in your `options` file. Please refer to the *pppd* and *chat* man pages for more information.

Configuring a PPP client via a leased line. Configuring a PPP client via a leased line is very simple. You will still use the *pppd* program, but since you won't need to establish the modem link the arguments to the chat program can be much simpler.

The example I'm presenting here assumes that the ppp server doesn't require any special login procedure. I do this because every login procedure will be different and if you are simply running a local connection then it is possible that you might have it set up this way.

```
pppd defaultroute noipdefault debug \
kdebug 2 /dev/cua0 9600
```

This will open the serial device, generate the *ioctl* to change it into a *ppp*device, set your default route via the *ppp* interface. The *noipdefault* argument instructs the *pppd* program to request the address to use for this device from the server. Debug messages will go to *syslog*. The *kdebug 2* argument causes the debug messages to be set to level 2, this will give you slightly more information on what is going on. It will use `/dev/cua0` at 9600 bps.

If your ppp server does require some sort of login procedure, you can easily use the *chat* program as in the example for the dialup server to perform that function for you.

Please refer to the *pppd* and *chat* man pages for more information. Please also refer to the README file that comes with the ppp software, as its description of the above is much more complete than I have described here.

Configuring a PPP server. Configuring a PPP server is similar to establishing a SLIP server. You can create a special 'ppp' account, which uses an executable script as its login shell. The `/etc/passwd` entry might look like:

```
ppp:EncPasswd:102:50:PPP client login:/tmp:/etc/ppp/ppplogin
```

and the `/etc/ppp/ppplogin` script might look like:

```
#!/bin/sh
exec /usr/sbin/pppd passive :192.1.2.23
```

The address that you provide will be the address that the **calling** machine will be assigned.

Naturally, if you want multiple users to have simultaneous access you would have to create a number of startup scripts and individual accounts for each to use, as you can only put one ip address in each script.

12.1.5 Where to obtain more information on PPP, or report bugs.

The *PPP-HOWTO* <http://sunsite.unc.edu/mdw/HOWTO/PPP-HOWTO.html> is an excellent reference to obtain more comprehensive information than I have provided here.

Most discussion on PPP for Linux takes place on the PPP mailing list.

To join the Linux **linux-ppp** channel on the mail list server, send mail to:

```
Majordomo@vger.rutgers.edu
```

with the line:

```
subscribe linux-ppp
```

in the message body. The subject line is ignored.

Please remember that when you are reporting bugs or problems you should include as much information relevant to the problem as you can to assist those that will help you understand your problem.

You might also like to check out:

RFCS 1548, 1331, 1332, 1333 and 1334. These are the definitive documents for PPP.

W. Richard Stevens also describes PPP in his book 'TCP/IP Illustrated Volume 1', (Addison-Wessley, 1994, ISBN 0-201-63346-9).

12.2 Configuring Linux as a Slip Server.

If you have a machine that is perhaps network connected, that you'd like other people be able to dial into and provide network services, then you will need to configure your machine as a server. If you want to use SLIP as the serial line protocol, then currently you have three options as to how to configure your Linux machine as a SLIP server. My preference would be to use the first presented, *sliplogin*, as it seems the easiest to configure and understand, but I will present a summary of each, so you make your mind.

12.2.1 Slip Server using *sliplogin*.

sliplogin is a program that you can use in place of the normal login shell for SLIP users that converts the terminal line into a SLIP line. It allows you to configure your Linux machine as either a *static address server*, users get the same address everytime they call in, or a *dynamic address server*, where users get an address allocated for them which will not necessarily be the same as the last time they called.

The caller will login as per the standard login process, entering their username and password, but instead of being presented with a shell after their login, *sliplogin* is executed which searches its configuration file (`/etc/slip.hosts`) for an entry with a login name that matches that of the caller. If it locates one, it configures the line as an 8bit clean line, and uses an *ioctl* call to convert the line discipline to SLIP. When this process is complete, the last stage of configuration takes place, where *sliplogin* invokes a shell script which configures the SLIP interface with the relevant ip address, netmask and sets appropriate routing in place.

This script is usually called `/etc/slip.login`, but in a similar manner to *getty*, if you have certain callers that require special initialisation, then you can create configuration scripts called `/etc/slip.login.loginname` that will be run instead of the default specifically for them.

There are either three or four files that you need to configure to get *sliplogin* working for you. I will detail how and where to get the software and how each is configured in detail. The files are:

- `/etc/passwd`, for the dialin user accounts.
- `/etc/slip.hosts`, to contain the information unique to each dial-in user.
- `/etc/slip.login`, which manages the configuration of the routing that needs to be performed for the user.
- `/etc/slip.tty`, which is required only if you are configuring your server for *dynamic address allocation* and contains a table of addresses to allocate
- `/etc/slip.logout`, which contains commands to clean up after the user has hung up or logged out.

Where to get *sliplogin* *sliplogin* can be obtained from:

sunsite.unc.edu

```
/pub/Linux/system/Network/serial/sliplogin-2.0.tar.gz
```

The tar file contains both source, precompiled binaries and a *man* page.

To ensure that only authorised users will be able to run *sliplogin* program, you should add an entry to your `/etc/group` file similar to the following:

```
 ..
SLIP::13:radio,fred
 ..
```

When you install the *sliplogin* package, the `Makefile` will change the group ownership of the *sliplogin* program to `SLIP`, and this will mean that only users who belong to that group will be able to execute it. The example above will allow only users radio and fred to execute *sliplogin*.

To install the binaries into your `/sbin` directory and the *man* page into section 8, do the following:

```
# cd /usr/src
# gzip -dc .../sliplogin-2.0.tar.gz | tar xvf -
# <..edit the Makefile if you don't use shadow passwords..>
# make install
```

If you want to recompile the binaries before installation, add a `make clean` before the `make install`. If you want to install the binaries somewhere else, you will need to edit the `Makefile` *install* rule.

Please read the `README` files that come with the package for more information.

Configuring `/etc/passwd` for Slip hosts. Normally you would create some special logins for Slip callers in your `/etc/passwd` file. A convention commonly followed is to use the *hostname* of the calling host with a capital 'S' prefixing it. So, for example, if the calling host is called `radio` then you could create a `/etc/passwd` entry that looked like:

```
Sradio:FvKurok73:1427:1:radio SLIP login:/tmp:/sbin/sliplogin
```

It doesn't really matter what the account is called, so long as it is meaningful to you.

Note: the caller doesn't need any special home directory, as they will not be presented with a shell from this machine, so `/tmp` is a good choice. Also note that *sliplogin* is used in place of the normal login shell.

Configuring `/etc/slip.hosts` The `/etc/slip.hosts` file is the file that *sliplogin* searches for entries matching the login name to obtain configuration details for this caller. It is this file where you specify the ip address and netmask that will be assigned to the caller and configured for their use. Sample entries for two hosts, one a static configuration for host `radio` and another, a dynamic configuration for user host `albert` might look like:

```
#
Sradio    44.136.8.99    44.136.8.100   0xffffff00   normal
Salbert   44.136.8.99    DYNAMIC        0xffffff00   compressed
#
```

The `/etc/slip.hosts` file entries are:

1. the login name of the caller.
2. ip address of the server machine, ie this machine.
3. ip address that the caller will be assigned. If this field is coded `DYNAMIC` then an ip address will be allocated based on the information contained in your `/etc/slip.tty` file discussed later. **Note:** you must be using at least version 1.3 of sliplogin for this to work.
4. the netmask assigned to the calling machine in hexadecimal notation eg 0xffffff00 for a Class C network mask.
5. optional parameters to enable/disable compression and other features.

Note: You can use either hostnames or IP addresses in dotted decimal notation for fields 2 and 3. If you use hostnames then those hosts must be resolvable, that is, your machine must be able to locate an ip address for those hostnames, otherwise the script will fail when it is called. You can test this by trying trying to telnet to the hostname, if you get the *Trying nnn.nnn.nnn...* message then your machine has been able to find an ip address for that name. If you get the message *Unknown host*, then it has not. If not, either use ip addresses in dotted decimal notation, or fix up your name resolver configuration (See section `Name Resolution`).

The most commonly used optional parameters for the `opt1` and `opt2` fields are:

normal

to enable normal uncompressed SLIP.

compressed

to enable van Jacobsen header compression (cSLIP)

Naturally these are mutually exclusive, you can use one or the other. For more information on the other options available, refer to the *man* pages.

Configuring the `/etc/slip.login` file. After *sliplogin* has searched the `/etc/slip.hosts` and found a matching entry, it will attempt to execute the `/etc/slip.login` file to actually configure the SLIP interface with its ip address and netmask.

The sample `/etc/slip.login` file supplied with the *sliplogin* package looks like this:

```
#!/bin/sh -
#
#       @(#)slip.login  5.1 (Berkeley) 7/1/90
#
# generic login file for a SLIP line.  sliplogin invokes this with
# the parameters:
#     $1       $2        $3       $4         $5        $6    $7-n
#   SLIPunit ttyspeed loginname local-addr remote-addr mask opt-args
#
/sbin/ifconfig $1 $4 pointopoint $5 mtu 1500 -trailers up
/sbin/route add $5
arp -s $5 <hw_addr> pub
exit 0
#
```

You will note that this script simply uses the *ifconfig* and *route* commands to configure the SLIP device with its ipaddress, remote ip address and netmask and creates a route for the remote address via the SLIP device. Just the same as you would if you were using the *slattach* command.

Note also the use of *Proxy ARP* to ensure that other hosts on the same ethernet as the server machine will know how to reach the dial-in host. The `<hw_addr>` field should be the hardware address of the ethernet card in the machine. If your server machine isn't on an ethernet network then you can leave this line out completely.

Configuring the `/etc/slip.logout` file. When the call drops out, you want to ensure that the serial device is restored to its normal state so that future callers will be able to login correctly. This is achieved with the use of the `/etc/slip.logout` file. It is quite simple in format.

```
#!/bin/sh -
#
#               slip.logout
```

```
#
/sbin/ifconfig $1 down
/sbin/route del $5
arp -d $5
exit 0
#
```

All it does is 'down' the interface and delete the manual route previously created. It also uses the *arp* command to delete any proxy arp put in place, again, you don't need the *arp* command in the script if your server machine does not have an ethernet port.

Configuring the `/etc/slip.tty` file. If you are using dynamic ip address allocation (have any hosts configured with the `DYNAMIC` keyword in the `/etc/slip.hosts` file, then you must configure the `/etc/slip.tty` file to list what addresses are assigned to what port. You only need this file if you wish your server to dynamically allocate addresses to users.

The file is a table that lists the *tty* devices that will support dial-in SLIP connections and the ip address that should be assigned to users who call in on that port.

Its format is as follows:

```
# slip.tty    tty -> IP address mappings for dynamic SLIP
# format: /dev/tty?? xxx.xxx.xxx.xxx
#
/dev/ttyS0      192.168.0.100
/dev/ttyS1      192.168.0.101
#
```

What this table says is that callers that dial in on port `/dev/ttyS0` who have their remote address field in the `/etc/slip.hosts` file set to `DYNAMIC` will be assigned an address of `192.168.0.100`.

In this way you need only allocate one address per port for all users who do not require an dedicated address for themselves. This helps you keep the number of addresses you need down to a minimum to avoid wastage.

12.2.2 Slip Server using *dip*.

Let me start by saying that some of the information below came from the *dip* man pages, where how to run Linux as a SLIP server is briefly documented. Please also beware that the following has been based on the *dip337j-uri.tgz* package and probably will not apply to other versions of *dip*.

dip has an input mode of operation, where it automatically locates an entry for the user who invoked it and configures the serial line as a SLIP link according to information it finds in the `/etc/diphosts` file. This input mode of operation is activated by invoking *dip* as *diplogin*. This therefore is how you use *dip* as a SLIP server, by creating special accounts where *diplogin* is used as the login shell.

The first thing you will need to do is to make a symbolic link as follows:

```
# ln -sf /usr/sbin/dip /usr/sbin/diplogin
```

You then need to add entries to both your `/etc/passwd` and your `/etc/diphosts` files. The entries you need to make are formatted as follows:

To configure Linux as a SLIP server with *dip*, you need to create some special SLIP accounts for users, where *dip* (in input mode) is used as the login shell. A suggested convention is that of having all SLIP accounts begin with a capital 'S', eg 'Sfredm'.

A sample `/etc/passwd` entry for a SLIP user looks like:

```
Sfredm:ij/SMxiTlGVCo:1004:10:Fred:/tmp:/usr/sbin/diplogin
^^          ^^         ^^  ^^    ^^    ^^    ^^
|           |          |   |     |     |     \__ diplogin as login shell
|           |          |   |     |     \_______ Home directory
|           |          |   |     \_____________ User Full Name
|           |          |   \___________________ User Group ID
|           |          \_______________________ User ID
|           \________________________________ Encrypted User Password
\____________________________________________ Slip User Login Name
```

After the user logs in, the *login(1)* program, if it finds and verifies the user ok, will execute the *diplogin* command. *dip*, when invoked as *diplogin* knows that it should automatically assume that it is being used a login shell. When it is started as *diplogin* the first thing it does is use the *getuid()* function call to get the userid of whoever has invoked it. It then searches the `/etc/diphosts` file for the first entry that matches either the userid or the name of the *tty* device that the call has come in on and configures itself appropriately. By judicious decision as to whether to give a user an entry in the `diphosts` file, or whether to let the user be given the default configuration you can build your server in such a way that you can have a mix of static and dynamically assigned address users.

dip will automatically add a 'Proxy-ARP' entry if invoked in input mode, so you do not need to worry about manually adding such entries.

Configuring `/etc/diphosts` `/etc/diphosts` is used by *dip* to lookup preset configurations for remote hosts. These remote hosts might be users dialing into your linux machine, or they might be for machines that you dial into with your linux machine.

The general format for `/etc/diphosts` is as follows:

```
 ..
Suwalt::145.71.34.1:145.71.34.2:255.255.255.0:SLIP uwalt:CSLIP,1006
ttyS1::145.71.34.3:145.71.34.2:255.255.255.0:Dynamic ttyS1:CSLIP,296
 ..
```

The fields are:

1. `login name`: as returned by getpwuid(getuid()) or tty name.
2. `unused`: compat. with passwd

3. `Remote Address`: IP address of the calling host, either numeric or by name
4. `Local Address`: IP address of this machine, again numeric or by name
5. `Netmask`: in dotted decimal notation
6. `Comment field`: put whatever you want here.
7. `protocol`: Slip, CSlip etc.
8. `MTU`: decimal number

An example `/etc/net/diphosts` entry for a remote SLIP user might be:

```
Sfredm::145.71.34.1:145.71.34.2:255.255.255.0:SLIP uwalt:SLIP,296
```

which specifies a SLIP link with remote address of 145.71.34.1 and MTU of 296, or:

```
Sfredm::145.71.34.1:145.71.34.2:255.255.255.0:SLIP uwalt:CSLIP,1006
```

which specifies a cSLIP-capable link with remote address 145.71.34.1 and MTU of 1006.

Therefore, all users who you wish to be allowed a statically allocated dial-up IP access should have an entry in the `/etc/diphosts` and if you want users who call a particular port to have their details dynamically allocated you must have an entry for the `tty` device and do not configure a user based entry. You should remember to configure at least one entry for each `tty` device that your dialup users use to ensure that a suitable configuration is available for them regardless of which modem they call in on.

When a user logs in, they will receive a normal login and password prompt, at which they should enter their SLIP-login userid and password. If they check out ok, then the user will see no special messages, they should just change into SLIP mode at their end and then they should be able to connect ok and be configured with the parameters from the `diphosts` file.

12.2.3 SLIP server using the *dSLIP* package.

Matt Dillon <`dillon@apollo.west.oic.com`> has written a package that does not only dial-in but also dial-out SLIP. Matt's package is a combination of small programs and scripts that manage your connections for you. You will need to have *tcsh* installed as at least one of the scripts requires it. Matt supplies a binary copy of the *expect* utility as it too is needed by one of the scripts. You will most likely need some experience with *expect* to get this package working to your liking, but don't let that put you off.

Matt has written a good set of installation instructions in the README file, so I won't bother repeating them.

You can get the *dSLIP* package from its home site at:

apollo.west.oic.com

```
/pub/linux/dillon_src/dSLIP203.tgz
```

or from:

sunsite.unc.edu

```
/pub/Linux/system/Network/serial/dSLIP203.tgz
```

Read the `README` file and create the `/etc/passwd` and `/etc/group` entries **before** doing a `make install`.

12.3 Using the Automounter Daemon - AMD.

This section has been supplied by `Mitch DSouza` and I've included it with minimal editing, as he supplied it. Thanks Mitch.

12.3.1 What is an automounter and why would I use one ?

An *automounter* provides a convenient means of mounting filesystems on demand, i.e. when required. This will reduce both the server and the client load and provides a great deal of flexibility even with non-NFS mounts. It also offers a redundancy mechanism whereby a mount point will automatically switch to a secondary server should a primary one be unavailable. A rather useful mount called the *union* mount gives the *automounter* the ability to *merge* the contents of multiple directories into a single directory. The documentation MST be read thoroughly to make full use of its extensive capabilities.

A few important points must be remembered - (in no particular order):

- *amd* maps are **not** compatible with Sun maps, which in turn are **not** compatible with HP maps *ad infinitum*. The point here however is that *amd* is freely available and compatible with all the systems mentioned above and more, thus giving you the ability to share maps if *amd* is installed throughout your network. Mitch uses it with a mixture of Linux/Dec/NeXt/Sun machines.
- *Sun* automount maps can be converted to *amd* style maps by using the *perl* script in the `contrib` directory - `automount2amd.pl`.
- You **must** have the *portmapper* running **before** starting *amd*.
- UFS mounts **do not** timeout.
- UFS mounts, in the case of Linux **only**, have been extended to deal with **all** varieties of native filesystems (i.e. minix, ext, ext2, xiafs ...) with the default being minix. This undocumented feature is accessed in the *opts* option like:

  ```
  ..., opts:=type=msdos,conv=auto
  ```

- Do not mount over existing directories unless you use a *direct* automount option, otherwise it is like mounting your disk on `/home` when some user directory is `/home/fred`.
- **Always** turn on full logging with the '-x all' option to *amd* if you have any troubles. Check also what the command:

  ```
  % amq -ms
  ```

reports, as it will indicate problems as they occur.

- GNU *getopt()* is too clever for its own good sometimes. You should always use '–' before the non-options e.g.

```
# /etc/amd -x all -l syslog -a /amd -- /net /etc/amd.net
```

12.3.2 Where to get AMD, the automounter daemon.

amd can be obtained from:

sunsite.unc.edu

```
/pub/Linux/system/Misc/mount/amd920824upl67.tar.gz
```

This contains ready-to-run binaries, full sources and documentation in texinfo format.

12.3.3 An example AMD configuration.

You do not configure the *automounter* from the `/etc/fstab` file, which you will already be using to contain information about your filesystems, instead it is command line driven.

To mount two *nfs* filesystems using your `/etc/fstab` file you would use two entries that looked like:

```
server-1:/export/disk  /nfs/server-1  nfs  defaults
server-2:/export/disk  /nfs/server-2  nfs  defaults
```

i.e. you were *nfs* mounting `server-1` and `server-2` on your linux disk on the `/nfs/server-1` and `/nfs/server-2` directories.

After commenting out, or deleting the above lines from your `/etc/fstab` file, you could *amd* to perform the same task with the following syntax:

```
/etc/amd -x all -l syslog -a /amd -- /nfs /etc/amd.server
|      | |    | |       | |     |  | |  | |             |
|      | |    | |       | |     |  | |  | |             |
`------' `----' `-------' `-----' _' `--' `-------------'
|        |      |         |      |   |    |
(1)      (2)    (3)       (4)    (5) (6)  (7)
```

Where:

1. The full *amd* binary path (obviously optional) depending on your $PATH setting, so just *'amd'* may be specified here.

2. '-x all' means turn full logging on. Read the documentation for the other logging levels

3. '-l syslog' means log the message via the *syslog* daemon. This could mean put it to a file, dump it, or pass it, to an unused tty console. This (*syslog*) can be changed to the name of a file, i.e. '-l foo' will record to a file called *foo*.

4. '-a /amd' means use the `/amd` directory as a temporary place for automount points. This directory is created automatically by *amd* and should be removed before starting *amd* in your `/etc/rc` scripts.

5. '–' means tell *getopt()* to stop attempting to parse the rest of the command line for options. This is especially useful when specifying the 'type:=' options on the command line, otherwise *getopt()* tries to decode it incorrectly.

6. '/nfs' is the **real** nfs mount point. Again this is automatically created and should **not** generally contain subdirectories unless the 'type:=direct' option is used.

7. The *amd* map (i.e. a file) named 'amd.server' contains the lines:

```
# /etc/amd.server
/defaults    opts:=rw;type:=nfs
server-1     rhost:=server-1;rfs:=/export/disk
server-2     rhost:=server-2;rfs:=/export/disk
```

Once started and successfully running, you can query the status of the mounts with the command:

```
% amq -ms
```

Now if you say:

```
% ls /nfs
```

you should see no files. However the command:

```
% ls /nfs/server-1
```

will mount the host 'server-1' automatically. *voila*! *amd* is running. After the default timeout has expired, this will automatically be unmounted. Your `/etc/password` file could contain entries like:

```
 ...
linus:EncPass:10:0:God:/nfs/server-1/home/linus:/bin/sh
mitch:EncPass:20:10:Mitch DSouza:/nfs/server-1/home/mitch:/bin/tcsh
matt:EncPass:20:10:Matt Welsh:/nfs/server-1/home/matt:/bin/csh
```

which would mean that when Linus, Matt, or Mitch are logged in, their home directory will be remotely mounted from the appropriate server and unmounted when they log out.

12.4 Using Linux as a router

Linux will function just fine as a router. You should run a routing daemon such as *gated*, or if you have simple routing requirements use hard coded routes. If you are using a late version kernel (1.1.*) then you should ensure that you have answered 'y' to:

```
IP forwarding/gatewaying (CONFIG_IP_FORWARD) [y] y
```

when building your kernel.

Olaf Kirch's Network Administrators Guide discusses network design and routing issues and you should read it for more information. A reference to it is in the "Related Documentation" section of this document.

12.5 NIS - Sun Network Information System.

There is now an *NIS-HOWTO* <http://sunsite.unc.edu/mdw/HOWTO/NIS-HOWTO.html> which you should read if you are interested in using NIS. It details how to obtain, install and configure the NIS system for Linux.

13 Experimental and Developmental modules.

There are a number of people developing new features and modules for the Linux networking code. Some of these are in quite an advanced state (read *working*) and it is these that I intend to include in this section until they are standard release code, when they will be moved forward.

The AX25 section has been relocated into the *HAM-HOWTO* <http://sunsite.unc.edu/mdw/HOWTO/HAM-HOWTO.html> because it was fairly specific to Amateur Radio.

13.1 snmp agent.

There is an experimental snmp agent for linux, ported from the cmu-snmp source by Erik Schoenfelder, <schoenfr@ibr.cs.tu-bs.de>.

It is available from:

ftp.ibr.cs.tu-bs.de

```
/pub/local/cmu-snmp2.1.2l3-src.tar.gz
```

Please **read** the file called cmu-snmp2.1.2l3.README, as it contains information that you will need to know about the package.

This package provides a nearly complete MIB-II variable set. and parts of the host MIB. Setting of system group variables is provided. The private community string is settable in the config file.

nstat.tar.gz contains a formatter of the output from /proc/net/snmp called *nstat*.

You will need Linux v1.1.60 and libc v4.6.27 or higher to compile and run the agent.

13.2 Experimental Token Ring driver

An experimental Token Ring driver is being developed by Peter De Schrijver <stud11@cc4.kuleuven.ac.be>. His latest version, at the time of writing is available in the Linux 1.3.* kernel release kernels. For older kernel versions the driver patches are located at:

linux3.cc.kuleuven.ac.be

/pub/Linux/TokenRing/TokenRing.patch-1.2.0.gz

ftp.cs.kuleuven.ac.be

/pub/unix/linux/TokenRing.patch-1.1.64.gz

There are a number of patch files against various kernel versions. Just pick the one that suits your kernel.

Note also that there are versions of the network tools to suit Token Ring in the same directory at linux3.cc.kuleven.ac.be.

Most boards based on IBM's TROPIC chipset should work now. The following boards are known to be working with the driver :

- IBM Token Ring Adapter II
- IBM Token Ring 16/4 Adapter
- IBM Token Ring Adapter/A
- IBM Token Ring 16/4 Adapter/A
- HyperRing Classic 16/4

Boards which use the TI chipset or busmastering DMA won't work with the current driver. However someone is working on a driver for the IBM busmaster adapters.

13.3 V.35 interface board

V.35 is a C.C.I.T.T. standard interface that provides a high speed balanced serial interface suitable for speeds up to about 2 Mbps. The use of differential pair balanced transmission allows the V.35 interface to support longer cables than can the more familiar V.24/RS232C type interface and higher data rates.

13.3.1 Internet Technologies International XNET-1

Matti Aarnio <mea@mea.utu.fi> represents *Internet Technologies International* <mailto:intec@digiw.fi>. They have an 8 bit X.21 card available called the XNET-1 which will work for V.35 with an appropriate cable. You can obtain information on the card itself from:

```
Internet Technologies International OY,
Helsinki, Finland  <intec@digiw.fi>
Internet Technologies International OY (Ltd)
PO BOX 51
00641 Helsinki
FINLAND
Fax: +356-0-7523939
```

The driver for the card is available from *ftp.funet.fi* <ftp://ftp.funet.fi/pub/Linux/kernel/xnet-sync-driver.tar.gz>.

13.3.2 SDL Communications N2 card.

SDL Communications <http://www.ici.net/cust_pages/sdl/sdl.html> produce a variety of controller cards supporting V.35 for ISA based machines.

The beta release driver can be found at:

ftp.sdlcomm.com

```
/business/sdlcomm/n2/linux
```

For further information you can contact *Rubin Dhillon at SDL Sales* <mailto:sales@sdlcomm.com>

13.4 Demand Dial SLIP/PPP package

Eric Schenk <schenk@cs.toronto.edu> has written a demand dial daemon that will work with either SLIP or PPP. It relies on you having a SLIP device configured which the daemon connects to via a pty. When your SLIP connection is not active all datagrams for non local hosts will be routed to this device and the daemon will detect them, when it receives a datagram it executes a script to activate your network link and then reroutes datagrams to that link.

The software is available at:

sunsite.unc.edu

```
/pub/Linux/system/Network/serial/diald-0.7.tar.gz
```

Note: You must configure your kernel so that it includes the SLIP driver, even if you only want to run PPP.

The included documentation describes how to install and configure the software.

13.5 ISDN support

There are various ISDN support efforts underway. The ones that I know of are:

13.5.1 3Com-Sonix PC/Arpeggio driver

Alan Cox has started a driver for the 3Com arpeggio ISDN card. It is currently capable of supporting Async PPP only.

sunsite.unc.edu

```
/pub/Linux/kernel/patches/network/sonix.tgz
```

13.5.2 DIEHL SCOM card

sunsite.unc.edu

```
/pub/Linux/kernel/patches/network/isdndrv-0.1.1.tar.gz
```

13.5.3 ICN ISDN and/or Teles ISDN card and/or Creatix AVM ISDN board

Fritz Elfert, Jan den Ouden and Thinking Objects have written a linux driver for the ICN ISDN card. Some of the documentation for this driver is in German. There seem to be various versions of this driver around that apparently support other ISDN cards, if you obtain one and it doesn't support your card take a look at another of the sites. You can obtain them from:

ftp.franken.de

```
/pub/isdn4linux/
```

or: **ftp.gwdg.de**

```
/pub/linux/misc/isdn4linux/
```

13.5.4 Euro-ISDN card ?

Matthias Urlichs <urlichs@smurf.noris.de> has developed some experimental Streams based ISDN support for Linux. The most recent version was for kernel version 1.3.* and is reported to work.

It is available at:

ftp.uni-stuttgart.de

```
/pub/systems/linux/isdn/ftp.noris.de/
```

Be warned, the documentation is in German and the code is designed to support the european ISDN network. Matthias is working on a english version of the documentation.

13.5.5 Combinet's Everyware 1000 ISDN card

Joel Katz <stimpson@panix.com> has developed a driver for Combinet's `Everyware 1000` series of plug-in ISDN cards.

It is available from: **sunsite.unc.edu**

```
/pub/Linux/kernel/patches/network/combinet1000isdn-1.02.tar.gz
```

13.5.6 More information:

Harald Milz has put together a World Wide Web page for Linux ISDN support and it is available at *Linux ISDN Web Page* <http://www.ix.de/ix/linux/linux-isdn.html>. A more generic site containing information about ISDN is available at *Dan Kegel's ISDN home page* <http://alumni.caltech.edu/ dank/isdn/>.

13.6 System-V Streams

The LiS (Linux Streams) project has formed to develop and implement System-V Streams for Linux. They have a World Wide Web page at *Linux Streams Web Page* <http://www.uc3m.es/ LiS/> where the status of the project can be obtained and where you can locate contact information and test code.

13.7 ATM (Asynchronous Transfer Mode) support.

Werner Almesberger has put together a World Wide site on the topic of Linux support for ATM. It is available at *lrcwww.epfl.ch* <http://lrcwww.epfl.ch/linux-atm/>. There is experimental software which supports raw ATM connections and some basic IP over ATM available at the pre-alpha stage and this can be obtained from the Web Site. The site contains a good deal of interesting information on ATM.

A mail list has been established to discuss ATM for Linux and you can subscribe by sending a mail message with the body `subscribe linux-atm` to: `majordomo@vger.rutgers.edu`

13.8 X.25 support.

Many people have enquired about X.25 support for Linux. There is some development work taking place but the apparently small community of people that are interested in it don't seem to have enough momentum to produce very quickly. If you are interested in X.25 support then you might volunteer some assistance to help accelerate the existing development effort. The only information that I have available on X.25 development efforts are a note on Alan Cox's webserver to say that someone has volunteered to do the job and the following World Wide Web site which provides information about two development efforts that seem to be making some progress:

X.25 for Linux is at: *www.ping.be* <http://www.ping.be/ ping0778/linux-x.25/>

13.9 Appletalk support.

If you have some Apple machines on your network that you'd like your linux machine to be able to share filesystems and printers with then the linux appletalk support is what you are after.

The following information was supplied mostly by Cees de Groot <cg@tricbbs.fn.sub.org who compiled it from his own experience with assistance from Dana Basken <dana@millenium.tiac.net> and others on the net.

13.9.1 Building the kernel with Appletalk DDP support

Alan Cox has developed some experimental and working support for the Apple Ethertalk specification. It is now a part of the later 1.2.* revision kernels and the new linux 1.3.* revision kernels so to build the kernel support for it you need only compile up a recent kernel, make sure you select the following configuration option:

```
#
# cd /usr/src/linux
# make config
 ...
 ...
    Appletalk DDP ? y
 ...
 ...
#
```

13.9.2 Building the Appletalk configuration tools and daemons.

You will need some additional software as the kernel support only provides enough to do basic networking, none of the higher layer protocols are implemented in the kernel.

Wesley Craig <netatalk@umich.edu represents a team called the Research Systems Unix Group at the University of Michigan and they have produced a package called *netatalk* which provides software that implements the Appletalk protocol stack and some useful utilities.

You can obtain it from:

University of Michigan <ftp://terminator.rs.itd.umich.edu/unix/netatalk/>

you will need at least version 1.3.3b2 as earlier versions did not support Linux.

To build and install that package something like:

```
# cd /usr/src
# tar xvfz .../netatalk
- You may want to edit the 'Makefile' at this point, specifically to change
  the DESTDIR variable which defines where the files will be installed later.
  The default of /usr/local/atalk is fairly safe.
- You should then just have to make the software. My 'uname' program wasn't
```

```
  in the /bin directory so I edited the Makefile to reflect this change too.
# make
- as root:
# make install
```

You should then end up with a whole range of new commands to play with.

13.9.3 Configuring the Appletalk software.

The first thing you need to do to make it all work is add the contents of the services.atalk file to your /etc/services file, just append it to the end of the file.

The next step is to create the appletalk configuration files in the `/usr/local/atalk/etc` directory (or wherever you installed the package).

The first file to create is the `/usr/local/atalk/etc/Atalkd.conf` file. To start this file needs only one line that gives the name of the network device that support the network that your Apple machines are on:

```
eth0
```

The Appletalk daemon program will add extra details after it is run.

13.9.4 Exporting a Linux filesystems via Appletalk.

You can export filesystems from your linux machine to the network so that Apple machine on the network can share them.

To do this you must configure the `/usr/local/atalk/etc/AppleVolumes.system` file.

Full details on how to do this and what the various options are can be found in the *afpd* man page.

A sample would look like:

```
/tmp Scratch
/home/ftp/pub Public
```

Which would export your `/tmp` filesystem as AppleShare Volume 'Scratch' and your ftp public directory as AppleShare Volume 'Public'. The volume names are not mandatory, the daemon will choose some for you, but it won't hurt to specify them anyway.

13.9.5 Sharing your Linux printer across Appletalk.

You can share your linux printer with your Apple machines quite simply. You need to run the *papd* program which is the Appletalk Printer Access Protocol Daemon. When you run this program it will accept requests from your Apple machines and spool the print job to your local line printer daemon for printing.

You need to edit the `/usr/local/atalk/etc/papd.conf` file to configure the daemon. The syntax of this file is the same as that of your usual `/etc/printcap` file. The name you give to the definition is registered with the Appletalk naming protocol, NBP.

A sample configuration might look like:

```
TricWriter:\
   :pr=lp:op=cg:
```

Which would make a printer named 'TricWriter' available to your Appletalk network and all accepted jobs would be printed to the linux 'lp' printer using *lpd*.

13.9.6 Starting the appletalk software.

Ok, you should now be ready to test this basic configuration. There is an *rc* file supplied with the *netatalk* package that should work ok for you, so all you should have to do is:

```
# /usr/local/atalk/etc/rc.atalk
```

and all should startup and run ok. You should see no error messages and the software will send messages to the console indicating each stage as it starts.

13.9.7 Testing the appletalk software.

To test that the software is functioning properly, go to one of your Apple machines, pull down the Apple menu, select the Chooser, click on AppleShare, and your Linux box should appear.

13.9.8 Caveats of the appletalk software.

- You may need to start the Appletalk support before you configure your IP network. If you have problems starting the Appletalk programs, or if after you start them you have trouble with your IP network, then try starting the Appletalk software before you run your `/etc/rc.d/rc.inet1` file.
- The *afpd* (Apple Filing Protocol Daemon) severely messes up your hard disk. Below the mount points it creates a couple of directories: `.AppleDesktop` and `Network Trash Folder`. Then, for each directory you access it will create a `.AppleDouble` below it so it can store resource forks, etc. So think twice before exporting `/`, you will have a great time cleaning up afterwards.
- The *afpd* program expects clear text passwords from the Macs. Security could be a problem, so be very careful when you run this daemon on a machine connected to the Internet, you have yourself to blame if somebody nasty does something bad.
- The existing diagnostic tools such as *netstat* and *ifconfig* don't support Appletalk. The raw information is available in the `/proc/net/` directory if you need it.

13.10 IP_MASQUERADE - allow a number of hosts to pretend to be one.

If you are one of the many people who have some sort of local network at home, whether it be a couple of machine connected via a SLIP/plip/ppp link to each other, or a number of machines connected to a private ethernet network who also has a dial-up Internet connection then you have probably at one time or another wanted the machines not directly connected to be able to communicate with the Internet. This is normally achieved by getting a valid register IP network address for your local network and asking your Internet Service Provider to kindly provide a network route to your network connection instead of the single host route they were previously providing. Unfortunately many Internet Service Providers will ask you to provide a good deal more dollars for the privilege of better utilising the bandwidth of your connection and you probably don't much like that idea.

IP_MASQUERADING provides a clever solution to this problem by making all of the machines on your network look like one very active networked machine. It does this by performing real-time, on the fly address translations. Most of the code was authored by Pauline Middelink <middelin@polyware.iaf.nl>.

Ken Eaves <keves@eves.com provided most the information contained below so if you find the following helpful, thank Ken.

13.10.1 Obtaining the 'masq' software.

The IP_Masquerade software currently comes as a patch to the later linux 1.2.* series kernels and a new version of the *ipfw* command. The *ipfw* program that comes with the `net-tools` software releases to configure it does not yet have the 'masquerade' option built into it. The kernel you are using must support the `ip_firewalling` option for ip_masquerade to work.

The patch file is available from:

ftp.eves.com

```
/pub/masq
```

In this directory you will find patch files, new versions of the *ipfw* command and Ken's original Masquerade FAQ on which this text is based.

The patch file is called `masq-patch.1.2.n`, so you must retrieve it. The files `ipfw` and `ipfw-for-1.3.10` are for kernel version 1.2.* and 1.3.10 respectively.

13.10.2 How do I install and compile the patch ?

Assuming you have the source to a recent 1.2.* version kernel installed in the `/usr/src/linux` directory you would do the following:

```
#
# cd /usr/src
# patch -p0 <.../masq-patch.1.2.n 2>patch.errs
- check the patch.errs file for the string 'fail'. If you can't find
  it then the patch applied successfully. If you can find it then something
```

```
  may have gone wrong double check that you have the correct version of
  kernel source.
# cd /usr/src/linux
# make config
- You must answer 'y' to:
  ...
  IP: forwarding/gatewaying (CONFIG_IP_FORWARD) [y]
  IP: firewalling (CONFIG_IP_FIREWALL) [y]
  IP: masquerading (ALPHA) [y]
  ...
- The rest of the options you should answer as you normally would.
# make dep
# make clean
- Build as you normally would, normally this would be:
# make zlilo
#
```

After making sure that you are actually running your newly built kernel you need only ensure that you have a working copy of the *ipfw* program. *ipfw* is normally installed in the `/sbin` so after testing the masquerade version you can simply copy it to `/sbin/ipfw`. This is all that is necessary to get the kernel support installed.

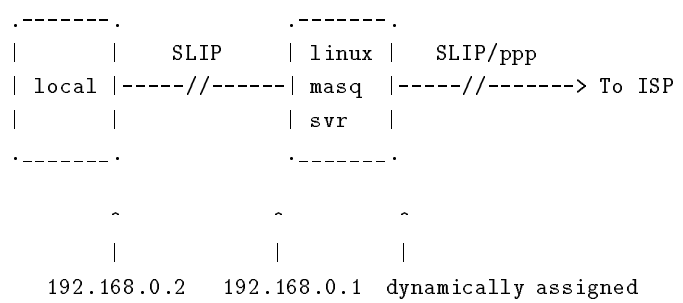
13.10.3 How do I configure the IP_MASQUERADE software ?

How you configure the IP_MASQUERADE function depends completely on how your network is built. The general idea is as follows: all of your local machine that you want to have access to the Internet must have their network default route pointing to your linux machine supporting IP_MASQUERADE. Secondly, you must have an entry in the linux firewall table that says that your local machines should be masqueraded.

To configure the IP-MASQUERADE function you use the *ipfw* command. The *ipfw* command is normally used to configure the Linux firewall facility and because your masquerading host is acting as a firewall of sorts *ipfw* has been modified to take on this additional function.

I'll present two example configurations that will probably cover most situations, if you have anything more complicated than these designs then you probably have the experience to work it out for yourself.

Two machines connected to each other by SLIP or ppp:

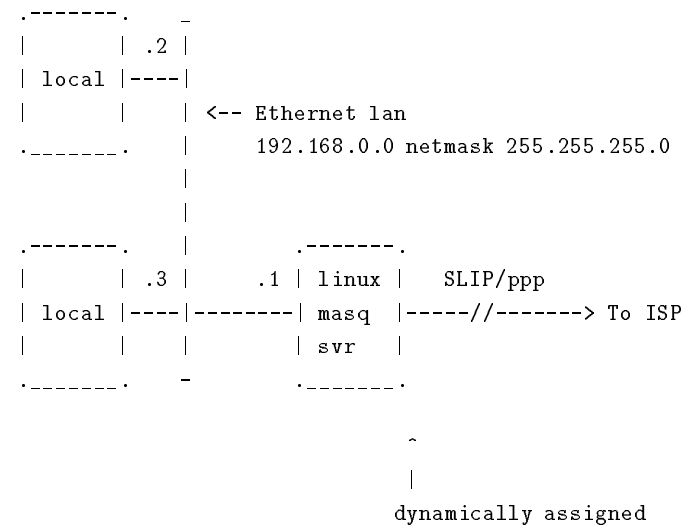


Your /etc/rc.d/rc.inet1 file would have something like the following appended:

```
 ...
 ...
# Allow machine 192.168.0.2 to be masqueraded to everybody
ipfw a m all from 192.168.0.2/32 to 0.0.0.0/0
 ...
 ...
```

Note the `iii.iii.iii.iii/nn` syntax. You might wonder what the `/nn` means and how you calculate it. This syntax is a means of encoding both an ip address and a netmask. What this syntax says is all addresses in which the first `nn` bits match those of `iii.iii.iii.iii`. The above example is for a single host, so all bits must match, since there are 32 bits in an IP address the syntax is `/32`.

A private ethernet network of machines sharing a single internet link:



For this configuration you have a single entry that describes the whole network, so you would use something like:

```
 ...
 ...
# Allow all machines on network 192.168.0.0 to be masqueraded to everybody
ipfw a m all from 192.168.0.2/24 to 0.0.0.0/0
 ...
 ...
```

Again note the relationship between the netmask (255.255.255.0) and the *ipfw* syntax. A class C network (255.255.255.0) has 24 bits in its netmask.

To make the job a little easier for you, here is a table of common network masks against matching bits:

```
netmask             ipfw
---------------     ----
```

```
255.0.0.0           /8
255.255.0.0.0       /16
255.255.255.0       /24
255.255.255.128     /25
255.255.255.192     /26
255.255.255.224     /27
255.255.255.240     /28
255.255.255.248     /29
255.255.255.255     /32
```

13.10.4 Where is IP_MASQUERADE being discussed ?

There was a support mailing list at masq@eves.com. Unfortunately it has now closed down.

13.11 IPALIAS - multiple address per interface.

If you are running a world wide web server and want to supports multiple virtual servers, or your network support multiple subnets on the same piece of wire then you'll need to be able to have your machine respond to more than one address per interface.

A facility called *ipalias* allows you to configure more than one address per interface.

Robert Sanders <rsanders@mindspring.com> has written linux support for this feature as patches against version 1.2.* kernels and Juan Jose Ciarlante <jjciarla@raiz.uncu.edu.ar> has developed a brand new and very sophisticated implementation for version 1.3.50++ kernels.

13.11.1 Support for 1.2.* kernels

You can obtain the patch from:

ftp.procyon.com

```
/pub/linux/ipalias
```

To install it you need the 1.2.1 kernel source installed and then:

```
# cd /usr/src/linux
# patch -p1 .../alias-patch-1.2.1-v1
# make config
# make dep; make clean; make zlilo
```

You will also need to get the `alias-net-tools.tar.gz` file as it contains new versions of the *ifconfig* and *netstat* commands.

Configuring ip aliases for 1.2.*. To configure an ip alias use the new *ifconfig* command as follows:

```
# ifconfig eth0 alias 198.71.200.5
```

This presumes that you already have the device configured to operate on your network and that you want `198.71.200.5` configured as a second address.

13.11.2 Support for 1.3.50++ kernels

You do not need any patches, simply answer yes to the question: `Network aliasing (CONFIG_NET_ALIAS) [N/y/?]` when prompted during `make config`.

Configuring IP_ALIAS for 1.3.50++ The new ip_alias facilities actually create new network interfaces to more properly. To check if your kernel actually has proper support for the new facility you should look for `/proc/net/alias*`. If your kernel has been properly built then two file will exist, `aliases` and `alias_types` in the `/proc/net` directory.

To configure aliases you use a special trick using a normal unmodified *ifconfig* program. The new alias ports are specified by including a suffix for the physical port name as follows:

```
# ifconfig eth0:0 192.168.1.1
# route add -host 192.168.1.1 dev eth0:0
```

Note that aliases start at '0'. So `eth0:1` would be the second alias for the physical `eth0` device.

To delete an alias you again use the *ifconfig* command as follows:

```
# ifconfig eth0:0- 0
```

The '-' character signifies to the kernel that the alias should be deleted and all routes and arp entries that point to it should be deleted.

14 Diagnostic tools - How do I find out what is wrong?

In this section I'll briefly describe some of the commonly used diagnostic tools that are available for your Linux network and how you might use them to identify the cause of your network problems, or to teach yourself a bit more about how tcp/ip networking works. I'll gloss over some of the detail of how the tools work because this document is not an appropriate forum for describing that sort of detail, but I hope I'll have presented enough information that you'll have an understanding of how to use the tool and to better understand the relevant *man* page or other documentation.

14.1 ping - are you there?

The *ping* tool is located in the NetKit-B distribution as detailed above in the 'Network Applications' section. *ping*, as the name implies, allows you to transmit a datagram at another host that it will reflect back at you if it is alive and working ok and the network in between is also ok. In its simplest form you would simply say:

```
# ping gw
PING gw.vk2ktj.ampr.org (44.136.8.97): 56 data bytes
64 bytes from 44.136.8.97: icmp_seq=0 ttl=254 time=35.9 ms
64 bytes from 44.136.8.97: icmp_seq=1 ttl=254 time=22.1 ms
64 bytes from 44.136.8.97: icmp_seq=2 ttl=254 time=26.0 ms
^C

--- gw.vk2ktj.ampr.org ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 22.1/28.0/35.9 ms
#
```

What *ping* has done is resolved the hostname to an address and using the *icmp* protocol has transmitted an *icmp echo request* datagram to the remote host periodically. For each *echo request* that the remote host receives it will formulate an *icmp echo reply* datagram which it will transmit back to you. Each line beginning with '64 bytes from ...' represents an *echo reply* received in response to an *echo request*. Each line tells you the address of the host that sent you the reply, the sequence number to which the reply was for, the *time to live* field and the total *round trip time* that was taken. The *round trip time* is the time between when the *echo request* datagram is transmitted and the corresponding *echo reply* is received. This can be used as a measure of how fast or slow the network connection between the two machines is.

The last two lines tell you how many datagrams were transmitted, how many valid responses were received and what percentage of the datagrams were lost. The percentage lost figure is a measure of how good or error free the network connection is. High percentage lost figures indicate such problems as a high error rate on a link somewhere between the hosts, exhausted capacity on a router or link somewhere, or high collision rate on an ethernet lan. You can use *ping* to identify where this problem might be by running *ping* sessions to each of the routed points that make up the network path. When you find that you can ping somewhere without any datagram loss, but pinging anywhere past there causes you packet loss, you can deduce that the problem lies somewhere between those two points.

14.2 traceroute - How do I get there?

The *traceroute* tool is found in the NetKit-A distribution detailed earlier. *traceroute* is primarily used for testing and displaying the path that your network connection would take to a destination host. *traceroute* also uses the *icmp* protocol, but it uses a clever trick to get each point along the path to send it back a reply as it creeps its way along. Its trick is to manually manipulate the *time to live* field of the datagrams it transmits. The *time to live* field is a mechanism that ensures that rogue datagrams do not get caught in a routing loop. Each time a datagram passes through a router it decrements the *time to live* field by one. If

the *time to live* reaches zero then that router or host sends an *icmp time to live expired* message back to the host who transmitted the datagram to let it know the datagram has expired. *traceroute* uses this mechanism by sending a series of *udp* datagrams with the *time to live* beginning set at one and incrementing each step it takes. By recording the addresses from the *icmp time to live expired* replies it receives in response to the datagrams dying it can determine the path taken to get to the destination. An example of its use would look something like:

```
# traceroute minnie.vk1xwt.ampr.org
traceroute to minnie.vk1xwt (44.136.7.129), 30 hops max, 40 byte packets
 1  gw (44.136.8.97)  51.618 ms  30.431 ms  34.396 ms
 2  gw.uts (44.136.8.68) 2017.322 ms  2060.121 ms 1997.793 ms
 3  minnie.vk1xwt (44.136.7.129) 2205.335 ms  2319.728 ms  2279.643 ms
#
```

The first column tells us how many hops away (what the *ttl* value was), the second column is the hostname and address that responded if it could be resolved or just its address if it could not. The third, fourth and fifth columns are the round trip time for three consecutive datagrams to that point. This tells us that the first hop in the network route is via gw.vk2ktj and the three figures following are the round trip times to that router. The next hop was via gw.uts.ampr.org and minnie.vk1xwt.ampr.org is one hop further away. You can deduce information about the network route by looking at the difference in times between each step in the route. You can see that the round trip times to gw are fairly fast, it is an ethernet connected host. gw.uts is substantially slower to get to than gw, it is across a low speed radio link, so you have the ethernet time plus the radio link time added together. minnie.vk1xwt is only slightly slower than gw.uts, they are connected via a high speed network.

If you perform a traceroute and you see the string `!N` appear after the time figure, this indicates that your traceroute program received a *network unreachable* response. This message tells you that the host or router who sent you the message did not know how to route to the destination address. This normally indicates that there is a network link down somewhere. The last address listed is as far as you get before you find the faulty link.

Similarly if you see the string `!H` this indicates that a *host unreachable* message has been received. This might suggest that you got as far as the ethernet that the remote host is connected to, but the host itself is not responding or is faulty.

14.3 tcpdump - capturing and displaying network activity.

Adam Caldwell <acaldwel@103mort2.cs.ohiou.edu¿ has ported the *tcpdump* utility to linux. *tcpdump* allows you to take traces of network activity by intercepting the datagrams on their way in and out of your machine. This is useful for diagnosing difficult to identify network problems.

You can find the source and binaries at: *103mor2.cs.ohiou.edu* `<ftp://ftp.funet.fi/pub/OS/Linux/PEOPLE/Linus/net-source/tools/tcpdump-3.0.3.tar.gz>`

tcpdump decodes each of the datagrams that it intercepts and displays them in a slightly cryptic looking format in text. You would use *tcpdump* if you were trying to diagnose a problem like protocol errors, or strange disconnections, as it allows you to actually see what has happened on the network. To properly use

tcpdump you would need some understanding of the protocols and how they work, but it is useful for simpler duties such as ensuring that datagrams are actually leaving your machine on the correct port if you are trying to diagnose routing problems and for seeing if you are receiving datagrams from remote destinations.

A sample of tcpdump output looks like this:

```
# tcpdump -i eth0
tcpdump: listening on eth0
13:51:36.168219 arp who-has gw.vk2ktj.ampr.org tell albert.vk2ktj.ampr.org
13:51:36.193830 arp reply gw.vk2ktj.ampr.org is-at 2:60:8c:9c:ec:d4
13:51:37.373561 albert.vk2ktj.ampr.org > gw.vk2ktj.ampr.org: icmp: echo request
13:51:37.388036 gw.vk2ktj.ampr.org > albert.vk2ktj.ampr.org: icmp: echo reply
13:51:38.383578 albert.vk2ktj.ampr.org > gw.vk2ktj.ampr.org: icmp: echo request
13:51:38.400592 gw.vk2ktj.ampr.org > albert.vk2ktj.ampr.org: icmp: echo reply
13:51:49.303196 albert.vk2ktj.ampr.org.1104 > gw.vk2ktj.ampr.org.telnet: S 700506986:700506986(0) win 512 <mss
13:51:49.363933 albert.vk2ktj.ampr.org.1104 > gw.vk2ktj.ampr.org.telnet: . ack 1103372289 win 14261
13:51:49.367328 gw.vk2ktj.ampr.org.telnet > albert.vk2ktj.ampr.org.1104: S 1103372288:1103372288(0) ack 700506
13:51:49.391800 albert.vk2ktj.ampr.org.1104 > gw.vk2ktj.ampr.org.telnet: . ack 134 win 14198
13:51:49.394524 gw.vk2ktj.ampr.org.telnet > albert.vk2ktj.ampr.org.1104: P 1:134(133) ack 1 win 2048
13:51:49.524930 albert.vk2ktj.ampr.org.1104 > gw.vk2ktj.ampr.org.telnet: P 1:28(27) ack 134 win 14335

 ..
#
```

When you start *tcpdump* without arguments it grabs the first (lowest numbered) network device that is not the loopback device. You can specify which device to monitor with a command line argument as shown above. *tcpdump* then decodes each datagram transmitted or received and displays them, one line each, in a textual form. The first column is obviously the time the datagram was transmitted or received. The remainder of the line is then dependent on the type of datagram. The first two lines in the sample are what an arp request from albert.vk2ktj for gw.vk2ktj look like. The next four lines are two pings from albert.vk2ktj to gw.vk2ktj, note that *tcpdump* actually tells you the name of the *icmp* datagram transmitted or received. The greater-than (>) symbol tells you which way the datagram was transmitted, that is, from who, to who. It points from the sender, to the receiver. The remainder of the sample trace are the establishment of a telnet connection from albert.vk2ktj to gw.vk2ktj.

The number or name at the end of each hostname tells you what socket number is being used. *tcpdump* looks in your /etc/services file to do this translation.

tcpdump explodes each of the fields and so you can see the values of the window and mss parameters in some of the datagrams.

The *man* page documents all of the options available to you.

Note for PPP users: The version of *tcpdump* that is currently available does not support the PPP suite of protocols. Al Longyear has produced a pair of patches to correct this, but these have not been built into a *tcpdump* distribution yet. The patch files are located in the same directory on `sunsite.unc.edu` as the *tcpdump* package.

14.4 icmpinfo - logs icmp messages received.

ICMP then Internet Control Message Protocol conveys useful information about the health of your IP network. Often ICMP messages are received and acted on silently with you never knowing of their presence. *icmpinfo* is a tool that will allow you to view ICMP messages much like tcpdump does. Laurent Demailly <dl@hplyot.obspm.fr> took the bsd ping source and modified it heavily.

Version 1.10 is available from:

hplyot.obspm.fr

```
/net/icmpinfo-1.10.tar.gz
```

Compilation is as simple as:

```
# cd /usr/src
# cd icmpinfo-1.10
# gzip -dc icmpinfo-1.10.tar.gz | tar xvf -
# make
```

You must be root to run *icmpinfo*. *icmpinfo* can either decode to the tty it was called from or send its output to the *syslog* utility.

To test out how it works, try running *icmpinfo* and starting a *traceroute* to a remote host. You will see the icmp messages that *traceroute* uses listed on the output.

15 Some Frequently Asked Questions, with brief Answers.

Following are some questions and answers that are commonly asked.

15.1 General questions:

I have only a dialin terminal access to a machine on the net, can I use this as a network connection ?

Yes you can, take a look at TERM. TERM allows you you to run network connections over a normal terminal session. It requires some modifications to the network applications to work with it, but binaries and sources are available for the most common ones already. take a look at the *TERM-HOWTO* <http://sunsite.unc.edu/mdw/HOWTO/Term-HOWTO.html> for lots more information.

When I telnet/ftp/rlogin to my machine it takes forever, why?

You do not have your name resolver configured properly. Reread the section on /etc/resolv.conf. You must have at least a nameserver entry configured.

I want to build my own standalone network, what addresses do I use ?

RFC1597 has specifically reserved some IP addresses for private networks. You should use these as they prevent anything nasty happening if you accidentally get connected to the Internet. The addresses reserved are:

```
10.0.0.0        -   10.255.255.255
172.16.0.0      -   172.31.255.255
192.168.0.0     -   192.168.255.255
```

Note, reserved network addresses are of classes A, B and C, so you are not restricted in your network design or size. Since you won't be connecting to the Internet it doesn't matter if you use the same address as some other group or network, just so long as the addresses you use are unique within your network.

I want the machines on my local network to work via my dial-ip Internet connection, how do I do it?

You have three options. They are as follows:

- Ask your Internet Service Provider to provide a network route to you instead of a host route. This is likely to be expensive but is the best performing and technical option.
- Set up your machine that supports your dial-up link as a 'SOCKS' based firewall. This will inherently provide a *proxy* function where the outside world thinks it is your router machine doing the talking where in fact it is doing it on behalf of the machines on your network. Check the *Firewall-HOWTO* `<http://sunsite.unc.edu/mdw/HOWTO/Firewall-HOWTO.html>` for detailed information on how to do this. This will require the machines on your network to have client programs that support the SOCKS capability.
- Configure and run the `IP-Masquerade` software. This is probably the simplest way, as it is fairly straightforward to do and doesn't require special clients on your machines. This software very cleverly performs address translations that make the machines on your network look like they are the machine supporting your dial-up connection. There are some limitations to how well this will work, but it is much lighter-weight than running a full firewall gateway.

If ftp.linux.org.uk is down, how do I get the files specified ?

Alan's archive is mirrored on:

ftp.Uni-Mainz.DE

`/pub/Linux/packages/Net2Debugged`

and/or:

ftp.infomagic.com

`/pub/mirrors/linux/sunacm`

How do I know what version of kernel/net code I am running ?

The network code and kernel now have synchronised version numbers, so try:

```
uname -a
```

or:

```
cat /proc/version
```

How do I change the message that telnet users are given at connect?

The `/etc/issue` is the message that is given to normal getty users when they login. Some *telnetd* programs use a different file `/etc/issue.net` instead. So if you find that changing your `issue` file doesn't work, try changing the other.

15.2 Error messages:

I keep getting the error 'eth0: transmit timed out'. What does this mean?

This usually means that your Ethernet cable is unplugged, or that the setup parameters for your card (I/O address, IRQ, etc.) are not set correctly. Check the messages at boot time and make sure that your card is recognized with the correct Ethernet address. If it is, check that there is no conflict with any other hardware in your machine, eg you might have a soundblaster sharing the same IRQ or i/o control port.

I get errors 'check Ethernet cable' when using the network.

You probably have your Ethernet card configured incorrectly. Double check the settings in /usr/src/linux/drivers/net/CONFIG. If this checks out ok, you may in fact have a cabling problem, check the cables are plugged in securely.

15.3 Routing questions:

Why do I get the message 'obsolete route request' when I use the route command ?

You are using a version of route that is older than your kernel. You should upgrade to a newer version of route. Refer to the "The network configuration tool suite" section of this document for information on where to obtain the tool set.

Why do I get a 'network unreachable' message when I try and network?

This message means that yours, or some other, machine doesn't know how to route to the host that you are attempting to ping or connect to. If it occurs for all hosts that you try, then it is probable that you don't have your default route set up properly, reread the 'routing' section.

I can ping my server/gateway, but can't ping or connect to anyone remote.

This is probably due to a routing problem. Reread the 'routing' section in this document. If this looks ok, then make sure that the host you are attempting to connect to has a route to you. If you are a dialin user then this is a common cause of problems, ensure that your server is either running a routing program like *gated* or *routed*, or that it is *'proxy arping'* for you, otherwise you will be able to get datagrams to the remote host, but it won't know how to return datagrams to you.

15.4 Using Linux with Novell fileservers or NFS:

How do I use my existing Novell fileserver with my Linux machine ?

You should read the *IPX-HOWTO* <http://sunsite.unc.edu/mdw/HOWTO/IPX-HOWTO.html> as it has information on Linux software that will allow you to do this without having to worry about using NFS.

Files get corrupted when running NFS over a network.

Certain vendors (Sun primarily) shipped many machines running NFS without UDP checksums. Great on ethernet, suicide otherwise. UDP checksums can be enabled on most file servers. Linux has it enabled by default from pl13 onwards - but both ends need to have it enabled...

Why are my NFS files all read only ?

The Linux NFS server defaults to read only. RTFM the 'exports' and nfsd manual pages. With non Linux servers you may also need to alter /etc/exports

15.5 SLIP questions:

What do I do if I don't know my SLIP servers address ?

dip doesn't really need to know the address of your SLIP server for SLIP to function. The *remote* option was added as a convenience so that *dip* could automate the *ifconfig* and *route* commands for you. If you don't know and cannot find out the address of your SLIP server then Peter D. Junger `Junger@samsara.law.cwru.edu` has suggested that he simply used his own address wherever a dip script called for a remote address. This is a small kludge but it works ok, as the server's address never actually appears in the SLIP headers anyway.

'dip' only works for root. How do I make it work for others?

dip needs to be setuid root to do some of the things it needs to do, such as modifying the routing table. Uri Blumenthal recommends the following:

- Create a new group called *dip* in your `/etc/group` file and place each person who you want to allow dial out operation in it.
- Then when logged in as root, do the following:

```
# chown root.dip /usr/bin/dip
# chmod u=rx,g=x,o= /usr/bin/dip
# chmod u+s /usr/bin/dip
```

Dial-In users will be restricted in what they can do by what is contained in the `/etc/diphosts` file.

I get `DIP: tty: set_disc(1): Invalid argument`, why?

This usually suggests that your kernel has not been compiled with *SLIP* support in it. Check that `/proc/net/dev` contains devices called `sl0`, `sl1` etc. It could also mean that your version of dip is very old. You should upgrade to a newer version.

When I ping a host I get 'wrong data byte #17...', why?

This generally means that you have your modem configured for XON/XOFF flow control. SLIP **must** have an eight bit clean line, so you cannot use XON/XOFF flow control. Hardware handshaking works better anyway, use it.

With SLIP I can ping my server and other hosts, but telnet or ftp don't work.

This is most likely caused by a disagreement on the use of header compression between your server and your machine. Double check that both ends either are, or are not, using compression. They must match.

How can I hang up the phone line when I'm done using SLIP?

If you use dip to dial out on the SLIP line, just 'dip -k' should do the trick. If not, try to kill the dip process that is running. When dip dies it should hang up the call. To give it the best chance to clean up after itself, try killing the process in the following sequence: 'kill <pid¿', 'kill -hup <pid¿' and finally, if the dip process still refuses to die, try 'kill -9 <pid¿'. The same philosophy should be applied to all unix processes that you are attempting to kill.

I see a lot of overrun errors on my SLIP port, why ?

The older network tools incorrectly report number of packets compressed as the number of packets overrun. This has been corrected and shouldn't occur of you are running the new version kernel and tools. If it still is it probably indicates that your machine isn't keeping up with the rate of data incoming. If you are not using 16550AFN UARTs then you should upgrade to them. 16450, or 8250 generate an interrupt for every character they receive and are therefore very reliant on the processor to be able to find time to stop what it is doing an collect the character from them to ensure none get lost. The 16550AFN has a 16 character FIFO and they only generate interrupts when the FIFO is nearly full, or when they have had character waiting, this means that less interrupts get generated for the same amount of data and that less time is spent servicing your serial port. If you want to use multiple serial ports you should mandatorily upgrade to 16550AFN UARTs anyway.

Can I use two SLIP interfaces ?

Yes. If you have, for example, three machines which you would like to interconnect, then you most certainly could use two SLIP interfaces on one machine and connect each of the other machines to it. Simply configure the second interface as you did the first. NOTE that the second interface will require a different IP address to the first. You may need to play with the routing a bit to get it to do what you want, but it should work.

I have a multiport i/o card, how do I use more than 4 SLIP ports ?

The kernel SLIP comes with a default of a maximum of 4 SLIP devices configured, this is set in the `/usr/src/linux/drivers/net/slip.h` file. To increase it, say to `16`, change the `#define SL_NRUNIT` to `16`, in place of the `4` that will be there. You also need to edit `/usr/src/linux/drivers/net/Space.c` and add sections for `sl4`, `sl5` etc. You can copy the existing driver definition as a template to make it easier. You will need to recompile the kernel for the change to take effect.

15.6 PPP questions.

You should refer to the *PPP-HOWTO* <http://sunsite.unc.edu/mdw/HOWTO/PPP-HOWTO.html> for a list of PPP questions and answers compiled by Al Longyear.

16 Quick Guide - SLIP or PPP Server

Configuring your linux machine as a SLIP or PPP server is a deceptively simple thing to do. The actual process is simple, but there are a number of different aspects to the configuration and understanding how each of the stages interact with each other is what will help you diagnose any problems you experience. Here are the steps that you must follow to configure your linux machine as a SLIP or PPP server:

1. Assemble your hardware. Avoid IRQ and shared memory conflicts. Test each of the serial ports by connecting a dumb terminal to each of the ports and use a null modem cable and a comms program like *minicom* or *seyon* to talk to each. Make sure you can send and receive characters. If you intend running a number of serial ports then try to use a smart serial board and use `16550AFN` UART's. This will help ease some of the work of handling interrupts generated by the serial ports.

2. Build your kernel, make sure it has networking configured, `IP Forwarding` enabled if you want users to route via your server and `SLIP` or `PPP` configured as is appropriate for your configuration. Make sure you configure `CSLIP` if you wish to use it. Double check you have `IP Forwarding` enabled if you don't then dialin users will only be able to connect directly to your machine.

3. Install your kernel. Test the kernel. Check the `/proc/net/dev` file and make sure that you have `sl*` `ppp*` devices listed. If not then you have probably made some error in configuring your kernel, or you are not actually running your new kernel. Double check that you have actually run lilo to install the new kernel.

4. Configure a *getty* on the serial port(s) that you wish to use for your incoming calls. You should refer to the *Serial-HOWTO* <http://sunsite.unc.edu/mdw/HOWTO/Serial-HOWTO.html> for a description of how to do this. Remember to configure your modem so that the `DCD` pin tracks received carrier, this is how most *getty* programs detect an incoming call (`mgetty` is an exception to this and uses the `RING` message so you must be sure you have your modem configured to generate it: `ATQ0V1` is usually enough).

5. Test the *getty* to make sure it works. It is important that you do this before you start worrying about the actual SLIP or PPP configuration. Try dialing into your system, you should get a `login:` prompt and be able to login to a shell account normally. Check for lost or garbled characters that might indicate a flow control problem.

6. Decide how you want your server to allocate addresses. If you want your users to get the same address each time they call then you want a `static` server, if you want to minimise the number of addresses you use and don't care what address your users are allocated then you want a `dynamic` server.

7. Decide how you are going to build the SLIP or PPP server, whether you are going to use *sliplogin*, *dip* or *dSLIP* for SLIP for example. If you wish to use the *sliplogin* package then refer to the "Slip Server

using Sliplogin" section. If you are going to use *dip* then refer to section "Slip Server using DIP". If you want to build your SLIP server using the *dSLIP* package then refer to section "Slip Server using dSLIP". For PPP you will need to use the *pppd* program, so make sure you read the *PPP-HOWTO* <http://sunsite.unc.edu/mdw/HOWTO/PPP-HOWTO.html> on how to configure this. Ensure you have the appropriate software, that it is a recent version and that it is compiled and installed if necessary.

8. If the addresses you are allocating to you SLIP users are part of your ethernet network, then make sure you read the Proxy ARP section and configure a proxy arp for each address. You may do this in the `/etc/sliploginand /etc/SLIPlogout` files if you are using the *sliplogin* package. If you are configuring a PPP server then make sure you use the *pppd* `proxyarp` option. If the addresses you are allocating are from a network seperate to your ethernet network then you will need to use a routing daemon program like *gated*. If you use *gated* then refer to the "gated" section.

9. Test your SLIP/PPP server.

10. Get someone else to test your SLIP/PPP server.

17 Known Bugs.

The Linux networking code is still an evolving thing. It still has bugs though they are becoming less frequently reported now. The *Linux Networking News* <http://iifeak.swan.ac.uk/NetNews.html> is a World Wide Web page maintained by Alan Cox which contains information on the status of the NET-3 networking code. You can obtain information on what is known and what isn't, by reading the `/usr/src/linux/net/inet/README` file that accompanies the kernel source, or by joining the `linux-net` channel.

18 Copyright Message.

The NET-2-HOWTO is Copyright (c) 1995 Terry Dawson and Matt Welsh.

Permission is granted to make and distribute verbatim copies of this document provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this document under the conditions for verbatim copies above, provided a notice clearly stating that the document is a modified version is also included in the modified document.

Permission is granted to copy and distribute translations of this document into another language, under the conditions specified above for modified versions.

Permission is granted to convert this document into another media under the conditions specified above for modified versions provided the requirement to acknowledge the source document is fulfilled by inclusion of an obvious reference to the source document in the new media. Where there is any doubt as to what defines 'obvious' the copyright owner reserves the right to decide.

19 Miscellaneous and Acknowledgements.

There are so many people who have contributed comments and suggestions for this update that I have forgotten who you are. **Thanks**.

Please, if you have any comments or suggestions then mail them to me. I'm fairly busy these days, so I might not get back to you straight away, but I will certainly consider any suggestion you have.

The Linux networking code has come a long way and it hasn't been an easy trip, but the developers, all of them, have done an excellent job in getting together something that is functional, versatile, flexible and free for us to use. We all owe them a great debt of thanks. Linus, Ross, Fred, Alan, the Alpha/Beta testers, the tools developers and those offering moral support have all contributed to the code as it is today.

For those that have an itch they want to scratch, happy hacking, here it is.

regards `Terry Dawson, vk2ktj.`

`<94004531@postoffice.csu.edu.au>, or <terry@perf.no.telecom.com.au>`