Linux NET-HOWTO

Terry Dawson, terryd@extro.ucc.su.oz.au v2.0, 4 Jun 1994

This document describes how to obtain, install and configure the Linux NET-2 and NET-3 networking software.

Contents

1 Introduction. **1**

1.1 A brief development history. 1

2 Disclaimer. **2**

3 Questions already ? **2**

4 Related Documentation. **3**

4.1 New versions of this document. 4

4.2 Feedback. 4

5 NET-2/NET-3 Supported functionality. **4**

5.1 Supported Ethernet cards. 5

6 Getting the NET-2/NET-3 software. **6**

6.1 The kernel source. 6

6.2 The libraries. 6

6.3 The network configuration tool suite. 7

6.4 The network applications. 7

6.5 Additional drivers or packages. 8

7 Configuring the kernel. **8**

8 Configuring the Network Devices. **10**

8.1 What information do I need before I begin ? 11

8.1.1 IP Address. 11

8.1.2 Network Mask ('netmask'). 11

8.1.3 Network Address. 11

8.1.4 Broadcast Address. . . . 12
8.1.5 Router ('Gateway') Address. . . . 12
8.1.6 Nameserver Address. . . . 13
8.1.7 NOTE for SLIP/PLIP/PPP users. . . . 13
8.2 /etc/rc.d/rc.inet1,2 or /etc/rc.net . . . 13
8.2.1 rc.inet1 . . . 14
8.2.2 rc.inet2 . . . 14
8.3 Configuring the Loopback device (mandatory). . . . 14
8.4 Configuring an ethernet device. (optional) . . . 15
8.5 Configuring a SLIP device (optional) . . . 16
8.5.1 dip . . . 16
8.5.2 slattach . . . 17
8.5.3 When do I use which ? . . . 17
8.5.4 Static slip server with a dialup line and DIP. . . . 17
8.5.5 Dynamic slip server with a dialup line and DIP. . . . 18
8.5.6 Using DIP. . . . 18
8.5.7 Permament slip connection using a leased line and slattach. . . . 20
8.6 Configuring a PLIP device. (optional) . . . 21
8.6.1 PLIP cabling diagram. . . . 21

9 Routing. (mandatory) 22

9.1 Default Route. . . . 22
9.2 Proxy ARP. . . . 23
9.3 gated - the routing daemon. . . . 24

10 Configuring the network daemons. 24

10.1 /etc/rc.d/rc.inet2 (the second half of rc.net) . . . 24
10.1.1 inetd. . . . 25
10.1.2 syslogd. . . . 25
10.2 A sample rc.inet2 file. . . . 25
10.3 Name Resolution. . . . 28
10.3.1 /etc/hosts . . . 28
10.3.2 named - do I need thee ? . . . 29

10.3.3 /etc/networks 30
10.3.4 /etc/host.conf 30
10.3.5 /etc/resolv.conf 30
10.3.6 /etc/HOSTNAME 31
10.3.7 Setting your machine's name. 31
10.4 Other files. 32

11 Advanced Configurations. 32
11.1 Configuring Linux as a Slip Server. 32
11.1.1 /etc/net/diphosts 33
11.2 Using the Automounter Daemon - AMD. 34
11.2.1 What is an automounter, and why would I use one ? 34
11.2.2 Where to get AMD, the automounter daemon. 35
11.2.3 An example AMD configuration. 35

12 Experimental and Developmental modules. 37
12.1 PPP - Point to Point Protocol. 37
12.1.1 Where to obtain the PPP software. 37
12.1.2 Installing the PPP software. 37
12.1.3 Configuring and using the PPP software. 38
12.2 AX.25 - A protocol used by Amateur Radio Operators. 40
12.2.1 Where to obtain the AX.25 software. 40
12.2.2 Installing the AX.25 software. 41
12.2.3 Configuring and using the AX.25 software. 42
12.3 NIS - Sun Network Information System. 42
12.4 snmp agent. 43

13 Some Questions and Answers. 43

14 Known Bugs. 44

15 Copyright Message. 44

16 Miscellaneous, and Acknowledgements. 45

1 Introduction.

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

1.1 A brief development history.

Ross Biro <bir7@leland.stanford.edu> wrote the original kernel based networking code for Linux. He used ethernet drivers written by Donald Becker <becker@super.org>, 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 <iiitac@pyr.swan.ac.uk> took Fred's NET-2d code and set about debugging the code with the aim of producing a stable and working release of code for incorporation into the standard kernel releases. This code was called *NET-2D(ebugged)*, and has been incorporated into the standard kernel releases since some time before Linux vers 1.0 was released.

Fred continued developing his kernel network code, and produced NET-2E. A reference for it if you are interested in looking at Fred's new work is listed later on in this document.

With the release of Linux vers 1.0, Linus made a decision to continue supporting Alan's code as the *'standard'* network kernel code.

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

Unless otherwise stated, this document will refer to the network code included in the standard kernel releases. On the whole this document will server for Fred's code as well, but as the development paths are now seperate, it is possible that there will be differences between the two.

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.

Because it is a fresh implementation it still may 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. The networking code has a small team of dedicated people working on it, and a cast of thousands testing the code, and collecting and reporting bugs and problems. Any problem you experience if likely to have already been reported, and be being worked on, and will possible 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 may, and 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.

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.help*, 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 seaerching 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 prople 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 test or non-standard versions of code and utilities. The of the NET-2 code cannot be expected to offer support for the network code as distributed in any for, other than as described in this docuement, or as per distributed Alpha/Beta test instructions.

To join the Linux **NET** channel on the mail list server, send mail to:

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

with the line:

X-Mn-Admin: join NET

at the top of the message body (not the subject line).
```

Remember, keep in mind that the NET channel is for development discussions only.

Note also that a PPP list has been established. To join it, use the same procedure as for joining the NET channel, except specify PPP in place of NET in the `X-Mn-Admin:` field.

4 Related Documentation.

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.

This book supplements the NET-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 version is available in:

ftp.uni-erlangen.de

```
/pub/Linux/LOCAL/BETA/LDP/NAG
```

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.

You should also read the *Ethernet-HOWTO* if you intend using an Ethernet card with Linux. The *Ethernet-HOWTO* describes the ethernet drivers that are used by the Network code, their configuration, and other aspects of ethernet networking. If you are intending to use ethernet, then the *Ethernet-HOWTO* should be considered the definitive source of information relating to same. That is, if the *Ethernet-HOWTO* and this document disagree on any point, then believe the *Ethernet-HOWTO*.

For more general information on Unix network configuration another good place to look for help on setting up your network is the *O'Reilly and Associates* book *TCP/IP Network Administration*, (the one with the Crab on the cover). Keep in mind that the Linux Network code is now a fairly standard implementation of tcp/ip networking, this means that the commands to configure and use it will work in much the same way as for those for other unix operating systems. Keep in mind though that some of the arguments and options might differ slightly from those in the book.

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

athos.rutgers.edu

```
/runet/tcp-ip-admin.doc
/runet/tcp-ip-admin.ps
/runet/tcp-ip-intro.doc
/runet/tcp-ip-intro.ps
```

4.1 New versions of this document.

New version of this document can be retrieved by anonymous ftp from:

sunsite.unc.edu

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

or directly from me, <terryd@extro.ucc.su.oz.au>. It will also be posted to the newsgroups: comp.os.linux.announce, comp.os.linux.help, and news.answers periodically.

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, <terryd@extro.ucc.su.oz.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 instead of posting to one of the newsgroups, as I may miss it. Thanks.

5 NET-2/NET-3 Supported functionality.

The NET code is a complete kernel based implementation of tcp/ip for Linux. The NET-2 and NET-3 versions of code support:

Ethernet Cards

Most popular ethernet cards are supported.

SLIP (Serial Line IP) and PPP

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 performance over low speed lines.

PLIP (Parallel Lines IP)

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

NFS (Networked File System)

to allow you to remotely mount another machines filesystems.

AX.25 (A protocol used by Amateur Radio Operators)

Alan Cox has some experimental code available.

PI Card (An 8530 SCC based card used by Amateur Radio Operators)

An experimental PI Card driver is available.

The NET-2 and NET-3 network code does not yet currently support:

SPX/IPX/NCP (Novell) support

to allow Linux to serve and mount Novell network devices. This is being worked on.

Lan types other than ethernet

This means token ring, arcnet, FDDI, etc. An experimental Token Ring driver is being developed.

ISDN Support

this is being developed.

5.1 Supported Ethernet cards.

The standard linux kernel release supports the following type of Ethernet cards:

- NE2000/NE1000 and close compatibles.
- WD80*3 and close compatibles.
- SMC Ultra and close compatibles.
- 3c501 and close compatibles (not recommended).
- 3c503 and close compatibles.
- 3c509/3c579 and close compatibles.
- HP PCLAN and close compatibles.
- AT1500 and NE2100 (LANCE and PCnet-ISA) and close compatibles.
- AT1700 and close compatibles.
- DEPCA and close compatibles.
- D-Link DE600 pocket adaptor and close compatibles.
- AT-LAN-TEC/RealTek pocket adaptor and close compatibles.

As mentioned above, Linux supports otehr 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 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.

6.1 The kernel source.

The current kernel version is found in:

ftp.funet.fi

```
/pub/OS/Linux/PEOPLE/Linux/v1.1/v1.1.0.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 v.1.1.0.tar.gz | tar xvf -
```

6.2 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 libraries (libc-4.5.21) can be found in:

sunsite.unc.edu

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

You will need at least the following files:

- image-4.5.21.tar.gz
- inc-4.5.21.tar.gz
- extra-4.5.21.tar.gz
- release.libc-4.5.21

You **MUST** read **release.libc-4.5.21** before you install the libraries. Please note the single line in the release document regarding deleting the older version of `/usr/lib/libgcc.*` or else your compiles will not link properly. Please note that to use release 4.5.21 you will also need at least GCC version 2.5.7, and Linux kernel 1.0 or later.

6.3 The network configuration tool suite.

You will the utility suite that provides tools to configure your network support. The current NET-2 utility suite is available from:

sunacm.swan.ac.uk

```
/pub/misc/Linux/Networking/Programs/System/net032/net-0.32b.tar.gz
```

If you are using NET-3, you will need:

```
/pub/misc/Linux/Networking/Programs/System/net032/net-0.32d.tar.gz
```

instead. These packages include the network configuration programs such as *ifconfig*, *route*, *netstat* etc.

6.4 The network applications.

You will want a number of network application programs. These are programs like *telnet*, *ftp*, *finger* and their daemons at least. The tcp/ip application binaries and setup files are found in:

tsx-11.mit.edu

```
/pub/linux/packages/net/net-2/binaries/net-std.tar.z
/pub/linux/packages/net/net-2/binaries/net-ext.tar.z
```

NOTE: The `net-base.tar.z` package is now obselete and should not be used. It contains the original version of the network configuration utilities which will not work with the newer kernel releases. It does though contain a number of sample configuration files which will be useful for you to look at, so you might get it and untar it in a safe place (under `/tmp` perhaps) so that you can have a look at them.

You can unpack each of the packages above with the command:

```
# cd /
# gzip -dc filename.tar.z | tar xvvofp -
```

Please note that these are quite old versions of the network applications. These packages do though give you a good idea of where each of the programs go. In general you will find that the client programs will go in `/usr/bin` and the daemon programs (server) will go in `/usr/etc`. There are new version of the network applications available in both biary and source form, from:

sunacm.swan.ac.uk

```
/pub/misc/Linux/Networking/Programs/BSD/*
```

You will need to copy these into the same places, with the same permissions, as those you are replacing. A packaged release of the newer applications will be released soon.

6.5 Additional drivers or packages.

If you want o add some non-standard, or developmental code, such as AX.25 support, or PPP 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.

7 Configuring the kernel.

Before you can use any of the network tools, or configure any network devices, you must ensure that you 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 obtained and untarred the the kernel source already, and applied any patches that you might need to have applied to get any nonstandard or development 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 necesary 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 you ethernet hardware. For example, if you 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 *Ethernet-HOWTO* for more definitive information on this file and its effect on ethernet cards. This file also contains configurable parameters for PLIP, mostly timers, 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
*
 ...
 ...
TCP/IP networking (CONFIG_INET) [y] y
 ...
 ...
```

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

```
*
* Networking options
*
*
* (it is safe to leave these untouched)
*
*
* IP (required for now) y
*
Reverse ARP (CONFIG_INET_RARP) [y]
Assume subnets are local (CONFIG_INET_SNARL) [y]
Disable NAGLE algorithm (normally enabled) (CONFIG_TCP_NAGLE_OFF) [n]
*
```

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.

```
*
* Network device support
*
Network device support? (CONFIG_ETHERCARDS) [y] y
SLIP (serial line) support (CONFIG_SLIP) [y] y
 CSLIP compressed headers (SL_COMPRESSED) [n] n
PLIP (parallel port) support (CONFIG_PLIP) [n] n
NE2000/NE1000 support (CONFIG_NE2000) [n] n
WD80*3 support (CONFIG_WD80x3) [y] y
SMC Ultra support (CONFIG_ULTRA) [n] n
3c501 support (CONFIG_EL1) [n] n
3c503 support (CONFIG_EL2) [n] n
3c509/3c579 support (CONFIG_EL3) [n] n
HP PCLAN support (CONFIG_HPLAN) [n] n
AT1500 and NE2100 (LANCE and PCnet-ISA) support (CONFIG_LANCE) [n] n
AT1700 support (CONFIG_AT1700) [n] n
DEPCA support (CONFIG_DEPCA) [n] n
D-Link DE600 pocket adaptor support (CONFIG_DE600) [n] n
AT-LAN-TEC/RealTek pocket adaptor support (CONFIG_ATP) [n] n
*
```

This section if 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 without header compression, the WD80*3 driver, and nothing else. Simply answer '`y`' to whatever you want to play with, and '`n`' to that 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 Configuring the Network Devices.

'With some fencing wire and a pair of pliers I can build anything'

If everything has gone ok so far, then you will have a Linux kernel which support the network devices you intend to use, 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 devices 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 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.

8.1 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.

8.1.1 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 has the loopback port always uses the address `127.0.0.1`.

8.1.2 Network Mask ('netmask').

For performance reasons it is desirable to 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 tal to people outside your network, not to some people on your own network, then it is quite likely that you have specified an incorrect subnet mask.

Your network adminstrators 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.

8.1.3 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     =
```

8.1.4 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 soem 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 misconfigured.

8.1.5 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, the router is allocated an address when it needs one. 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 you address is `128.253.154.32`, then your router might be `128.253.154.1`. The router can 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 adminstrator to properly identify your router address.

If you're using only loopback then you don't need a router address. If you're using SLIP or PPP, then your router address will be your SLIP/PPP server address.

8.1.6 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.

8.1.7 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.

8.2 /etc/rc.d/rc.inet1,2 or /etc/rc.net

While the commands to configure your network devices can be typed manually each time, you will probably want to record them somewhere so that they 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 analagous to the MS-DOS *autoexec.bat* file, and *rc* might stand for *'runtime commands'*. By convention these files are kept under the */etc* directory. For NET-2/NET-3 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. **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 *rc.net*, 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.

8.2.1 rc.inet1

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

ifconfig */sbin/ifconfig* is used for configuring your interface 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.

8.2.2 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.

8.3 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 fucntion 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 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, eatablish 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 check 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
#
```

8.4 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'.

8.5 Configuring a SLIP device (optional)

SLIP (Serial Line Internet Protocol) allow syou to use tcp/up 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 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 used 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*

8.5.1 dip

dip (Dialup IP) is a smart program that is able to ser 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 performance 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.

8.5.2 slattach

slattach on the other is a very simple program that, that is very easy to use, but does not have the sophistication of *dip*. *slattach* is ideal to use where you have a permanent connection to your server, such as a physical cable, or a leased line.

8.5.3 When do I use which ?

When do you use which ? You would use dip when your link to the machine that is your slip server is a dialup modem, or some other termporary 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 unlink 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 initialise 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.

8.5.4 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 the 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 addresss (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 *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 ocnfigure the interface for you after commanding the modem to establish the call, and loggin 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.

8.5.5 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 netework 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, guidees 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 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.

8.5.6 Using DIP.

As explained earlier, *dip* is a powerful program that can simplify and automate the process of dialling 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 'chat 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` in the *net-32* package for an explanation. *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 from `tsx-11`, and the *net-032* distribution.

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 was supplied by `Paul Mossip`, and is probably a good starting point for you. You might like to save it as `/etc/dipscript`:

```
#
# Connection script for SLIP to knoware.nl.mugnet.org
#

  # Fetch the IP address of our target host.
main:

  # Set the desired serial port and speed.
  port /dev/cua0
  speed 38400

  # Reset the modem and terminal line.
  reset

  # Prepare for dialing.
  send ATZ1\r
```

```
  wait OK 4
  if $errlvl != 0 goto error
  dial 666-0999                    ## Change to your servers number!
  if $errlvl != 0 goto error
  wait CONNECT 60
  if $errlvl != 0 goto error

  # We are connected.  Login to the system.
login:
  sleep 3
  send \r\n\r\n
  wait gracelands> 20              ## Change to your servers prompt
  if $errlvl != 0 goto error
  send login\n
  wait name: 10                    ## Wait username: prompt
  if $errlvl != 0 goto erro
  send elvisp\n                    ## Change to your own!
  wait ord: 10                     ## Wait password prompt
  if $errlvl != 0 goto error
  send alive\n                     ## Change to your own!
  wait gracelands> 10
  if $errlvl != 0 goto error
  send slip\n                      ## Change to suit your server
  wait SLIP 30                     ## Wait for SLIP prompt
  if $errlvl != 0 goto error
  get $local remote 10             ## Assumes the server sends your IP..
  if $errlvl != 0 goto error       ## address as soon as you enter slip.
  get $remote gracelands           ## slip server address from /etc/hosts
done:
  print CONNECTED to $remote with address $rmtip we are $local
  default
  mode SLIP
  goto exit
error:
  print SLIP to $host failed.
exit:
#
# End dip script
```

The above example assumes you are calling a *dynamic* slip server, if you are calling a *static* slip server, then remove the lines:

```
get $local remote 10             ## Assumes the server sends your IP..
if $errlvl != 0 goto error       ## address as soon as you enter slip.
```

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 seperated 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 automaticaly 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, after the *dip* command has finished, manually 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.

8.5.7 Permament 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* tao set up your serial link. *slattach* is a very simple to use utility that will allow you to do just enough fucntionality 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 permament 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 deive 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 do want compression or not.

8.6 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 comparitvely expensive 16550AFN UART's for your serial ports.

The Linux *plip* interface is compatible with the *Crywyr 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*.

When compiling the kernel, there is only 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.

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

```
#
# Attach a PLIP interface
#
#  configure parallel port lp0 as a plip device
/sbin/ifconfig pl0 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* can be, or are, used.

8.6.1 PLIP cabling diagram.

plip has been designed to use cables with the same pinout sa 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
---------   -------------------------------
INIT        16 - 16
SLCTIN      17 - 17
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*
```

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**.

9 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 though you have more than one network interface then your choice is a little more complicated. You might for example 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 three commonly used routing mechanisms for unix network configurations. I'll briefly discuss each in turn.

9.1 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 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 vai your *slip server*.

To configure your default route, **add** thw 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 Route/Gateway Address.

9.2 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 or PPP dial-in server.

Normally when a 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.

If you want to use your machine as a server for other machines, you must get your machine to answer ARP requests for them 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/ppp 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 address you would need to **add** the following commands to 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 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.

9.3 gated - the routing daemon.

gated could be used in place of *proxy arp* and would certainly be much cleaner, but its primary use is if you want your linux machine to act as an *ip router*. It is an implentation of the *rip* protocol. *rip* stands for *'Routing Information Protocol'*. If you run *gated*, your linux machine will periodically broadcast a copy of it *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 accesible via your machine.

gated would normally be started from your `rc.inet2` which is covered in the next section.

There are a couple of *gated* distributions floating around, and I don't have details of any of them at the moment. I'll try and get some more information into the next version.

10 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 configruations 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.

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

You should by this stage have built an *rc* file to configure each of your network devices, with the correct addresses, and setup 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:

10.1.1 inetd.

inetd is a program that sites 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 incomine request for a particular service, eg *telnet*, or *ftp*, it will check the `/etc/services` file, and find what server program needs to be run to manage the request, will start it, and hand the connection over to it. Imagine it as a master server for your internet servers. It aslos have a few simple standard services inbuilt, these are *echo*, *discard* and *generate* services used for various types of network testing.

10.1.2 syslogd.

syslogd is a deemon 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.

10.2 A sample rc.inet2 file.

The following 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, and that it 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/etc"
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.
if [ ! -f ${NET}/named ]
then
        echo -n "named "
        ${NET}/named
fi

# Start the ROUTEd server.
if [ -f ${NET}/routed ]
then
        echo -n "routed "
        ${NET}/routed -q #-g -s
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!
```

10.3 Name Resolution.

Name Resolution is the process of converting a a hostname in the familiar dotted notatiion (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.

10.3.1 /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') also allows you to do the same name -¿ip address translation automatically. (Running *named* allows you to run your own nameserver on you linux machine). This file needs to contain at least an entry for `127.0.0.1` with the name `localhost`. If you're no 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 hstname 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.

10.3.2 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 comparitively difficult to setup 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 bottom line is this: **You do not need to run named** on your Linux system. The SLS instructions will probably tell you to run `hostcvt.build` to setup *named*. This is simply unnecessary unless you want to make your Linux system fucntion as a nameserver for other machines, in which case you probably should learn some more about *named* anyway. When looking up hostnames, your linux machine will first check the `/etc/hosts` file, and then ask the nameserver out on the net.

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 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 bizzarre, as you'd never need to call it.

10.3.3 /etc/networks

The `/etc/networks` file lists the names and address 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 simliar 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
```

10.3.4 /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.

10.3.5 /etc/resolv.conf

This file actually configures the system name resovler. This file contains two types of entries: The addresses of your nameservers (if any), and the name of your domain, if you ahve one. I fyou'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 127.253.154.5
```

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.

10.3.6 /etc/HOSTNAME

This is a new file, it contains the full hostname of your machine with the domain name included. It is used by the *hostname* command, to save you having to supply the hostname as an argument. For example, the machine above would have the file `/etc/HOSTNAME`:

```
goober.norelco.com
```

That's all.

10.3.7 Setting your machine's name.

After you have all of your network daemons, and all of the network code, up and running, there is one small task that remains to do, and that is to set the hostname for your machine. This is achieved by adding a command to your `/etc/rc.local`, or your `/etc/rc` file, **after** `rc.inet2`, or your `rc.net` has been run. You may already have this command in your file, so all you will have to do is modify it to your hostname.

```
# /etc/rc or /etc/rc.local
 ...
 ...
/bin/hostname -S
 ...
 ...
```

This will set your hostname to the name found in `/etc/HOSTNAME` if you have one, if you have decided not to, and you don't need to, then you can use this form:

```
# /etc/rc or /etc/rc.local
 ...
 ...
/bin/hostname -S loomer.vpizza.com
 ...
 ...
```

It is important that you give a full hostname, that is, including your domain name, as the *hostname* command attempts to set both the hostname and the domain name from the same command.

Important: The hostname in `/etc/HOSTNAME` **must** be a valid hostname. This means it must be found in `/etc/hosts` (or that you nameserver must be able to resolve it, but you should put it in your `/etc/hosts` in case your nameserver is down).

10.4 Other files.

There are of course many otehr files in the the `/etc` directory which you may need to dablle 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 avialable 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 everthing in the right place, you should be able to reboot you new kernel, and net away to your hearts content. However I strongly suggest that you keep a bootable copy of your old kernel and even possible a 'recovery disk', in case something goes wrong, so that you can get back in a fix it. You might try HJLu's 'single disk boot disk', or disk1 from an SLS distribution.

11 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.

11.1 Configuring Linux as a Slip Server.

Let me start by saying that some of the information below came from the *dip* man pages, where in fact how to run Linux as a slip server is briefly documented.

To configure Linux as a slip server, you need to create some special slip accounts for users, where *dip* (in slave mode) is used as the login shell. Fred suggests that he has a convention of having all of his slip accounts begin with a capital 'S', eg 'Sfredm'.

Because the login program won't accept arguments to the login shell, you will need to create a small program that looks like the following:

```
/* dip-i.c - from a mail message of Karl kkeyte@esoc.bitnet */
int main()
{
   execlp("dip", "dip", "-i", (char *) 0);
}
```

Compile it with: `gcc -O dip-i.c -o dip-i`

Give it permissions `555`. I recommend calling it `/usr/bin/dip-i` as shown below.

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

```
Sfredm:ij/SMxiTlGVCo:1004:10:UUNET:/tmp:/usr/bin/dip-i
|          |         |   |    |     |    \__ shell program running
|          |         |   |    |     |         dip -i 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 shell program *dip-i* which will execute the *dip* command in input mode (`-i`). *dip* now scans the `/etc/net/diphosts` file for an entry for the given user name. Therefore, each slip user must also have an entry in `/etc/net/diphosts`.

You will have to re-read section 'Proxy Arp' to arrange for your machine to *proxy arp* for the slip users who will be using your system if you want them to have access to any network that your server machine might be connected to.

11.1.1 /etc/net/diphosts

`/etc/net/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/net/diphosts` is as follows:

```
Suwalt::145.71.34.1:SLIP uwalt:CSLIP,1006
 ^    ^  ^            ^         ^     ^
 |    |  |            |         |     \___ MTU
 |    |  |            |         \_________ protocol (SLIP, CSLIP,
 |    |  |            |                   KISS, PPP)
 |    |  |            \___________________ comment field
 |    |  \_______________________________ IP address of the other
 |    |                                   side, or host.domain.name
 |    \___________________________________ unused (compat. with passwd)
 \________________________________________ login name (as returned by
                                           getpwuid(getuid()))
```

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

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

which specifies a slip link with MTU of 296, or

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

which specifies a cslip-capable link with MTU of 1006.

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.

11.2 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.

11.2.1 What is an automounter, and why would I use one ?

An *automounter* provides a convenient means of mounting filesystems on demand, i.e. when requried. 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 msut 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
  ```

11.2.2 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.

11.2.3 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 fileystems, instead 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 a unused tty console. This (*syslog*) can be changed to a 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 umounted when they log out.

12 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.

12.1 PPP - Point to Point Protocol.

The *Point to Point Protocol* is a modern and efficient protocol for conveying tcp/ip across serial links, that a lot of people use in place of slip. It offers enhanced functionality and security options.

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, in fact will tell you much more than I present here.

The code is still officially *ALPHA* which means you should exercise caution in using it. But there have been many reports of successful operation, and the authors feel confident that the new releases are quite stable.

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 and bringing the link into a functioning state and for authentication.

12.1.1 Where to obtain the PPP software.

The ppp software is available from:

ftp.gang.umass.edu

```
/usr/michael/linux-ppp-0.2.1.tgz
```

This file contains the kernel source, and the *pppd* source and binary. Version 0.2.1 is meant for use with kernels 1.0.x and 1.1.x.

12.1.2 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. Configuring the kernel is fairly easy, the following should work ok:

```
# cd /usr/src
# gzip -dc linux-ppp-0.2.1.tgz | tar xvf -
# cp /usr/src/linux-ppp-0.2.1/linux/ppp.c /usr/src/linux/drivers/net
# cp /usr/src/linux-ppp-0.2.1/pppd/ppp.h /usr/src/linux/include/linux
```

You will then need to uncomment the `CONFIG_PPP` line in `/usr/src/linux/config.in`. If you are running a version of the kernel that is 1.1.4 or higher, then you will also need to comment out or delete the macro definition of `NET02D` in the file `/usr/src/linux/drivers/net/ppp.c`

You can then do a *make config*, select PPP support, and follow with a *make dep;make*.

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

```
PPP: version 0.2.1 (4 channels)
TCP compression code copyright 1989 Regents of the University of California
PPP line discipline registered.
```

Now, try looking at the contents of `/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 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/etc` 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.

12.1.3 Configuring and using the PPP software.

Like slip, you can configure the PPP software as either a client or a server. The *chat* performs a similar function to the *dip* program in that it is used to automate the dialling 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* 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 have to invoke the *pppd* program with appropriate arguments.

The following example might looks a little confusing at first, but it is easier to understand if you can see that all it is doing it taking a command line for the *chat* program as its first argument and then others for itself later.

```
pppd connect 'chat -v "" ATDT5551212 CONNECT "" ogin: ppp word: password'\
      /dev/cua1 38400 debug crtscts modem defaultroute 192.1.1.17:
```

What this says is:

- Invoke the *chat* program with the command line:

  ```
  chat -v "" ATDT5551212 CONNECT "" ogin: ppp word: password
  ```

  Which says: Dial 5551212, wait for the 'CONNECT' string, transmit a carriage return, wait for the string 'ogin:', transmit the string 'ppp', wait for the string 'word:', transmit the string 'password', and quit.

- Use serial device `/dev/cua1`
- Set its speed to 38400 bps.
- *debug* means log status messages to *syslog*
- *crtscts* means use hardware handshaking to the modem, recommended.
- *modem* means that *pppd* will attempt to hang up the call before after making the call.
- *defaultroute* instructs *pppd* to add a routing entry that makes this the default route. In most cases this will be what you want.
- *192.1.1.17:* says to set the ppp interfaces address to `192.1.1.17`. This argument normally looks like x.x.x.x:y.y.y.y, where x.x.x.x is your ip address, and y.y.y.y is the ip address of the server. If you leave off the servers address, *pppd* will ask for it, and x.x.x.x will be set to your machines ip address.

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 client via a leased line. Configuring a PPP client via a leased line is almost as straightforward as for configuring slip with *slattach*. 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 connect 'echo connecting...' defaultroute noipdefault debug \
      kdebug 2 /dev/cua0 9600
```

This will echo a message to your screen, and 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/etc/pppd passive :192.1.2.23
```

The address that you provide will 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.

12.2 AX.25 - A protocol used by Amateur Radio Operators.

The *AX.25* protocol is used by Amateur Radio Operators worldwide. It offers both connected and connectionless modes of operation, and is used either by itself for point-point links, or to carry other protocols such as tcp/ip and netrom.

It is similar to X.25 level 2 in structure, with some extensions to make it more useful in the amateur radio environment.

Alan Cox has developed some kernel based AX.25 software support for Linux and these are available in ALPHA form for you to try.

12.2.1 Where to obtain the AX.25 software.

Teh AX.25 software is available from:

sunacm.swan.ac.uk

```
/pub/misc/Linux/Radio/*
```

You will find a number of directories, each containing different versions of the code. Since it is closely linked with the kernel code, you will need to ensure that you choose the version appropriate for the kernel version you are running. The following table shows the mapping between the two:

```
AX25007                 Prehistoric
AX25010                 Obsolete
AX25012                 Current AX.25 kernel release for 1.0.* kernels
AX25017                 Current AX.25 kernel release for 1.1.6+ kernels
patch10-11              Patch from AX.25 010 to 011
```

In each directory you will find at least two files, one called something like `krnl017.tgz`, and the other called something like `user017.tgz`. These are the kernel software, and the user programs respectively.

12.2.2 Installing the AX.25 software.

The software comes in two parts, the kernel drivers, and the user programs.

The kernel drivers. To install the kernel drivers, do the following:

```
# cd /usr/src
# gzip -dc krnl017.tgz | tar xvf -
```

you will need to uncomment the `CONFIG_AX25` define in the `/usr/src/linux/config.in` file.

You should then:

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

Be sure to answer 'yes' when you are asked if you should include the AX.25 support in the *make config* step.

The user programs. To install the user programs you should try:

```
# cd /
# gzip -dc user017.tgz | tar xvvof -
```

You should then:

```
# cd /usr/local/ax25/src
# make install
```

12.2.3 Configuring and using the AX.25 software.

Configuring an AX.25 port is very similar to configuring a slip device. The AX.25 software has been designed to work with a TNC in *kiss* mode. You will need to have the TNC preconfigured and connected.

You use the *axattach* program in much the same way as you would use the *slattach* program. For example:

```
# /usr/local/ax25/bin/axattach -s 4800 /dev/cua1 VK2KTJ &
```

would configure your `/dev/cua1` serial device to be a *kiss* interface at 4800 bps, with the hardware address `VK2KTJ`.

You would then use the *ifconfig* program to configure the ip address and netmask as for an ethernet device:

```
# /sbin/ifconfig sl0 44.136.8.5
# /sbin/ifconfig sl0 netmask 255.255.255.0
# /sbin/ifconfig sl0 broadcast 44.136.8.255
# /sbin/ifconfig sl0  arp mtu 257 up
```

To test it out, try the following:

```
/usr/local/ax25/bin/call VK2DAY via VK2RVT
```

The *call* program is a linemode terminal program for making ax.25 calls. It recognises lines that start with ' ' as command lines. The ' .' command will close the connection.

I haven't had a chance to try this code out yet, please refer to the man pages in `/usr/local/ax25/man` for more information.

12.3 NIS - Sun Network Information System.

There are in fact two NIS implementations being distributed. Firstly there is a rudimentary implementation int he standard *libc* ditribution which however requires binding to to servers via *ypbind* before use. A more clean implementation tending towwards the NIS+ implementation is called *NYS* written by `Peter Eriksson, <pen@lysator.liu.se>` and is available from:

ftp.funet.fi

```
/pub/OS/Linux/BETA/NYS/nys-0.26.tar.gz
```

An NIS style server can be retrieved from:

ftp.funet.fi

```
/pub/OS/Linux/BETA/NYS/ypserv-0.5.tar.gz
```

Check there are no newer versions, as this information might now be slightly dated.

Both of these are fully fucntional and they have been used extensively with no troubles to query Sun servers for NIS information like passwd/hosts/group etc. and don't require binding to arbitrary servers. In fact they allow you to specify servers for services and have the ability to select a yp/dns/file option for name/passwd/etc. resolution of sepcific services. They are extremely easy to set up, and recommended for client machines intgrating into larger network.

Clearly your network daemons and clients need to be recompiled to link with the shared library *libnsl.so* to make use of the YP facilities. This is fairly trivial and an NYS package of all network clients and daemons is currently being compiled.

If you have more detailed information on NIS, please email me.

12.4 snmp agent.

There is an experimental snmp agent for linux, that I lost the information for after an accidental deletion of 6 weeks worth of updating this document. I'll see if I can find it for the next release.

13 Some Questions and Answers.

Following are some questions and answers that are commonly asked.

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

Try: `cat /proc/version`

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.

dip doesn't work. How do I make it work ?

Check that the file permissions of dip are 6750, that is 'chmod 6750 dip'. Check also that dip is owned by root: 'chown root:dip dip'

With SLIP I can ping my server, but can't connect to anything.

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.

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

This is probably due to a routing problem. Reread the 'routing' section in this document, and ensure that you have a default route set that is pointing to you gateway, or server.

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 'kill -9' the dip process. When dip dies it should hang up the call.

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

If you have the Novell NFS Daemon code then it is easy, just NFS mount the Novell volume that you wish to use. If you don't, and you are really desperate to be able to do this, and you have a spare pc machine laying about, you are in luck. You can run a program called Stan's Own Server on the spare PC. First, configure the pc as a novell workstation with maps to the directories you want to nfs mount, then run SOS, and export those drive maps. SOS is available from spdcc.com:pub/sos/sossexe.zoo

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

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.

14 Known Bugs.

The Linux networking code is still an evolving thing. It still has bugs. For the most up to date information on what is known and what isn't, then join the NET channel.

15 Copyright Message.

'We're not ogres, nor are we silly'

The NET-2-HOWTO is copyright by Terry Dawson and Matt Welsh. A verbatim copy of this document may be reproduced and distributed in any medium, physical or electronic without permission of the authors. Translations are similarly permitted without express permission if such translations include a notice stating who performed the translation, and that it is a translation. Commercial redistribution is allowed and encouraged, however, the authors would like to be notified of any such distributions.

Short quotes may be used without prior consent by the authors. Derivative works and partial distributions of the NET-2-HOWTO must include either a verbatim copy of this file, or make a verbatim copy of this file available. If the latter is the case, a pointer to the verbatim copy must be stated at a clearly visible place.

In short, we wish to promote dissemination of this information through as many channels as possible. However, we wish to retain copyright on this HOWTO document, and would like to be notified of any plans to redistribute it. Further we desire that ALL information provided in this HOWTO be disseminated.

If you have any questions relating to the conditions of this copyright, please contact Matt Welsh, the Linux HOWTO coordinator, at: `mdw@sunsite.unc.edu`, or `+1 607 256 7372`.

16 Miscellaneous, and Acknowledgements.

This HOWTO has been completely rewritten using the new smgl tools that Matt Welsh put together. The tools seem to work just fine, and they are pretty simple to use. There are so many people who have contributed comments and suggsetions 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.

73 `Terry Dawson, <terryd@extro.ucc.su.oz.au>, or <terry@orac.dn.telecom.com.au>`