Linux Networking HOWTO – DocBook Rev .02

Joshua Drake

This is a LinuxPorts.Com Document for the Linux Documentation Project and has been sponsored in part by the Open Source Documentation Fund.

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Chapter 1. Introduction.

New versions of this document can always be found first at http://www.linuxports.com/.

Previously the document was called the Net3/4 and Net−3 Howto's. I believe that may not have been obvious enough for certain readers. So, I have renamed it the Networking−HOWTO.

This document has an auto−contribute function located at LinuxPorts.Com. (www.linuxports.com/howto/networking/updates.phtml) If you have something that you like like to add to this document you may do so at LinuxPorts.Com page. (www.linuxports.com/howto/networking/updates.phtml)

1.1. Why Don't you cover "X" Topic?

We have received many email requests for certain topics to be covered with in this document and although we have tried to make this document the best it can be we are missing some information. If there is something you would like to see us cover, please contact us. We will do the best we can to include it.

Our company CommandPrompt, Inc. will also document in an expedited fashion for a fee. Yes, this is free documentation but we are consultants and expedition costs money. We will always do our best to insure that the documentation herein is accurate, timely and up to date but we must feed out families :). If you are interested in having a TOPIC documented please contact us at info@commandprompt.com.

Chapter 2. Document History

The original NET−FAQ was written by Matt Welsh and Terry Dawson to answer frequently asked questions about networking for Linux at a time before the Linux Documentation Project had formally started. It covered the very early development versions of the Linux Networking Kernel. The NET−2−HOWTO superceded the NET−FAQ and was one of the original LDP HOWTO documents, it covered what was called version 2 and later version 3 of the Linux kernel Networking software. This document in turn supercedes it and relates only to version 4 of the Linux Networking Kernel or more specifically kernel releases 2.x and 2.2.x.

Previous versions of this document became quite large because of the enormous amount of material that fell within its scope. To help reduce this problem a number of HOWTO's dealing with specific networking topics have been produced. This document will provide pointers to them where relevant and cover those areas not yet covered by other documents.

2.1. Feedback

We are always interested in feedback. Please contact us at: poet@linuxports.com.

Again, if you find anything erroneous or anything you would like to see added, please contact us.

Chapter 3. How to use this HOWTO.

This document is organized top−down. The first sections include informative material and can be skipped if you are not interested; what follows is a generic discussion of networking issues, and you must ensure you understand this before proceeding to more specific parts. The rest, ``technology specific'' information is grouped in three main sections: Ethernet and IP−related information, technologies pertaining to widespread PC hardware and seldom−used technologies.

The suggested path through the document is thus the following:

Read the generic sections:

> These sections apply to every, or nearly every, technology described later and so are very important for you to understand. On the other hand, I expect many of the readers to be already confident with this material.

Consider your network:

> You should know how your network is, or will be, designed and exactly what hardware and technology types you will be implementing.

Read the ``Ethernet and IP'' section if you are directly connected a LAN or the Internet:

> This section describes basic Ethernet configuration and the various features that Linux offers for IP networks, like firewalling, advanced routing and so on.

Read the next section if you are interested in low−cost local networks or dial−up connections:

> The section describes PLIP, PPP, SLIP and ISDN, the widespread technologies used on personal workstations.

Read the technology specific sections related to your requirements:

> If your needs differ from IP and/or common hardware, the final section covers details specific to non−IP protocols and peculiar communication hardware.

Do the configuration work:

> You should actually try to configure your network and take careful note of any problems you have.

Look for further help if needed:

> If you experience problems that this document does not help you to resolve then read the section related to where to get help or where to report bugs.

Have fun!

> Networking is fun, enjoy it.

3.1. Conventions used in this document

No special convention is used here, but you must be warned about the way commands are shown. Following the classic Unix documentation, any command you should type to your shell is prefixed by a prompt. This howto shows "`user%`" as the prompt for commands that do not require superuser privileges, and "`root#`" as the prompt for commands that need to run as root. I chose to use "`root#`" instead of a plain "`#`" to prevent confusion with snapshots from shell scripts, where the hash mark is used to define comment lines.

When ``Kernel Compile Options'' are shown, they are represented in the format used by *menuconfig*. They should be understandable even if you, (like me), are not used to *menuconfig*. If you are in doubt about the options' nesting, running the program once can't do anything, but help.

Chapter 4. General Information about Linux Networking.

4.1. Linux Networking Resources.

There are a number of places where you can find good information about Linux networking.

There are a wealth of Consultants available. A searchable listing can be found at http://www.linuxports.com/.

Alan Cox, the current maintainer of the Linux kernel networking code maintains a world wide web page that contains highlights of current and new developments in linux Networking at: www.uk.linux.org.

There is a newsgroup in the Linux news hierarchy dedicated to networking and related matters, it is: comp.os.linux.networking

There is a mailing list to which you can subscribe where you may ask questions relating to Linux networking. To subscribe you should send a mail message:

```
        To: majordomo@vger.rutgers.edu
        Subject: anything at all
        Message:
        subscribe linux-net
```

Please remember when reporting any problem to include as much relevant detail about the problem as you can. Specifically you should specify the versions of software that you are using, especially the kernel version, the version of tools such as *pppd/* or *dip* and the exact nature of the problem you are experiencing. This means taking note of the exact syntax of any error messages you receive and of any commands that you are issuing.

4.2. Where to get some non−linux−specific network information.

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

tcp/ip introduction:

> This document comes as both a text version and a postscript version.

tcp/ip administration:

> This document comes as both a text version and a postscript version.

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

"*Internetworking with TCP/IP, Volume 1: principles, protocols and architecture*, by Douglas E. Comer, ISBN 0−13−227836−7, Prentice Hall publications, Third Edition, 1995."

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

"*Unix Network Programming*, by W. Richard Stevens, ISBN 0−13−949876−1, Prentice Hall publications, 1990."

A second edition of this book is appearing on the bookshelves; the new book is made up of three volumes: check Prenice−Hall's web site to probe further.

You might also try the comp.protocols.tcp−ip newsgroup.

An important source of specific technical information relating to the Internet and the tcp/ip suite of protocols are RFC's. RFC is an acronym for `Request For Comment' and is the standard means of submitting and documenting Internet protocol standards. There are many RFC repositories. Many of these sites are ftp sites and other provide World Wide Web access with an associated search engine that allows you to search the RFC database for particular keywords.

One possible source for RFC's is at Nexor RFC database.

Chapter 5. Generic Network Configuration Information.

The following subsections you will pretty much need to know and understand before you actually try to configure your network. They are fundamental principles that apply regardless of the exact nature of the network you wish to deploy.

5.1. What do I need to start ?

Before you start building or configuring your network you will need some things. The most important of these are:

5.1.1. Current Kernel source(Optional).

Please note:

The majority of current distributions come with networking enabled, therefore it may not be required to recompile the kernel. If you are running well known hardware you should be just fine. For example: 3COM NIC, NE2000 NIC, or an Intel NIC. However if you find yourself in the position that you do need to update the kernel, the following information is provided.

Because the kernel you are running now might not yet have support for the network types or cards that you wish to use you will probably need the kernel source so that you can recompile the kernel with the appropriate options.

For users of the major distributions such as Redhat, Caldera, Debian, or Suse this no longer holds true. As long as you stay within the mainstream of hardware there should be no need to recompile your kernel unless there is a very specific feature that you need.

You can always obtain the latest kernel source from ftp.cdrom.com. This is not the official site but they have LOTS of bandwidth and capacity. The official site is kernel.org but please use the above if you can. Please remember that ftp.kernel.org is seriously overloaded. Use a mirror.

Normally the kernel source will be untarred into the `/usr/src/linux` directory. For information on how to apply patches and build the kernel you should read the Kernel−HOWTO. For information on how to configure kernel modules you should read the ``Modules mini−HOWTO''. Also, the `README` file found in the kernel sources and the `Documentation` directory are very informative for the brave reader.

Unless specifically stated otherwise, I recommend you stick with the standard kernel release (the one with the even number as the second digit in the version number). Development release kernels (the ones with the odd second digit) may have structural or other changes that may cause problems working with the other software on your system. If you are uncertain that you could resolve those sorts of problems in addition to the potential for there being other software errors, then don't use them.

5.1.2. IP Addresses, an Explanation.

Internet Protocol Addresses are composed of four bytes. The convention is to write addresses in what is called `dotted decimal notation'. In this form each byte is converted to a decimal number, (0−255), dropping any leading zero's unless the number is zero and written with each byte separated by a `.' character. By convention each interface of a host or router has an IP address. It is legal for the same IP address to be used on each interface of a single machine in some circumstances but usually each interface will have its own address.

Internet Protocol Networks are contiguous sequences of IP addresses. All addresses within a network have a number of digits within the address in common. The portion of the address that is common amongst all addresses within the network is called the `network portion' of the address. The remaining digits are called the `host portion'. The number of bits that are shared by all addresses within a network is called the netmask and it is role of the netmask to determine which addresses belong to the network it is applied to and which don't. For example, consider the following:

```
        -----------------  ---------------
        Host Address       192.168.110.23
        Network Mask       255.255.255.0
        Network Portion    192.168.110.
        Host portion                  .23
        -----------------  ---------------
        Network Address    192.168.110.0
        Broadcast Address  192.168.110.255
        -----------------  ---------------
```

Any address that is 'bitwise anded' with its netmask will reveal the address of the network it belongs to. The network address is therefore always the lowest numbered address within the range of addresses on the network and always has the host portion of the address coded all zeroes.

The broadcast address is a special address that every host on the network listens to in addition to its own unique address. This address is the one that datagrams are sent to if every host on the network is meant to receive it. Certain types of data like routing information and warning messages are transmitted to the broadcast address so that every host on the network can receive it simultaneously. There are two commonly used standards for what the broadcast address should be. The most widely accepted one is to use the highest possible address on the network as the broadcast address. In the example above this would be `192.168.110.255`. For some reason other sites have adopted the convention of using the network address as the broadcast address. In practice it doesn't matter very much which you use but you must make sure that every host on the network is configured with the same broadcast address.

For administrative reasons some time early in the development of the IP protocol some arbitrary groups of addresses were formed into networks and these networks were grouped into what are called classes. These classes provide a number of standard size networks that could be allocated. The ranges allocated are:

```
        ----------------------------------------------------------------------------
```

```
| Network             | Netmask        | Network Addresses                  |
| Class                        |                                 |
-----------------------------------------------------------------------------
|    A    | 255.0.0.0                                    | 0.0.0.0     - 127.255.255.255    |
|    B    | 255.255.0.0                          | 128.0.0.0  - 191.255.255.255 |
|    C    | 255.255.255.0                        | 192.0.0.0  - 223.255.255.255 |
|Multicast| 240.0.0.0                            | 224.0.0.0  - 239.255.255.255 |
-----------------------------------------------------------------------------
```

What addresses you should use depends on exactly what it is that you are doing. You may have to use a combination of the following activities to get all the addresses you need:

Installing a linux machine on an existing IP network

If you wish to install a linux machine onto an existing IP network then you should contact whoever administers the network and ask them for the following information:

- Host IP Address
- IP network address
- IP broadcast address
- IP netmask
- Router address
- Domain Name Server Address

You should then configure your linux network device with those details. You can not make them up and expect your configuration to work.

Building a brand new network that will never connect to the Internet

If you are building a private network and you never intend that network to be connected to the Internet then you can choose whatever addresses you like. However, for safety and consistency reasons there have been some IP network addresses that have been reserved specifically for this purpose. These are specified in RFC1597 and are as follows:

```
-----------------------------------------------------------
|         RESERVED PRIVATE NETWORK ALLOCATIONS            |
-----------------------------------------------------------
| Network | Netmask        | Network Addresses            |
| Class   |                |                              |
-----------------------------------------------------------
|    A    | 255.0.0.0      | 10.0.0.0    - 10.255.255.255  |
|    B    | 255.255.0.0    | 172.16.0.0  - 172.31.255.255  |
|    C    | 255.255.255.0  | 192.168.0.0 - 192.168.255.255 |
-----------------------------------------------------------
```

You should first decide how large you want your network to be and then choose as many of the addresses as you require.

5.2. Where should I put the configuration commands ?

There are a few different approaches to Linux system boot procedures. After the kernel boots, it always executes a program called `*init*'. The *init* program then reads its configuration file called `/etc/inittab` and commences the boot process. There are a few different flavours of *init* around, although everyone is now converging to the System V (Five) flavor, developed by Miguel van Smoorenburg.

Despite the fact that the *init* program is always the same, the setup of system boot is organized in a different way by each distribution.

Usually the `/etc/inittab` file contains an entry looking something like:

```
        si::sysinit:/etc/init.d/boot
```

This line specifies the name of the shell script file that actually manages the boot sequence. This file is somewhat equivalent to the `AUTOEXEC.BAT` file in MS−DOS.

There are usually other scripts that are called by the boot script and often the network is configured within one of many of these.

The following table may be used as a guide for your system:

```
---------------------------------------------------------------------------
Distrib. | Interface Config/Routing           | Server Initialization
---------------------------------------------------------------------------
Debian   | /etc/init.d/network                | /etc/rc2.d/*
---------------------------------------------------------------------------
Slackware| /etc/rc.d/rc.inet1                 | /etc/rc.d/rc.inet2
---------------------------------------------------------------------------
RedHat   | /etc/rc.d/init.d/network           | /etc/rc.d/rc3.d/*
---------------------------------------------------------------------------
```

Note that Debian and Red Hat use a whole directory to host scripts that fire up system services (and usually information does not lie within these files, for example Red Hat systems store all of system configuration in files under `/etc/sysconfig`, whence it is retrieved by boot scripts). If you want to grasp the details of the boot process, my suggestion is to check */etc/inittab* and the documentation that accompanies *init*. Linux Journal is also going to publish an article about system initialization, and this document will point to it as soon as it is available on the web.

Most modern distributions include a program that will allow you to configure many of the common sorts of network interfaces. If you have one of these then you should see if it will do what you want before attempting a manual configuration.

```
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
        Distrib   | Network configuration program
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
        RedHat    | /usr/bin/netcfg
        Slackware | /sbin/netconfig
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
```

5.3. Creating your network interfaces.

In many Unix operating systems the network devices have appearances in the */dev* directory. This is not so in Linux. In Linux the network devices are created dynamically in software and do not require device files to be present.

In the majority of cases the network device is automatically created by the device driver while it is initializing and has located your hardware. For example, the ethernet device driver creates `eth[0..n]` interfaces sequentially as it locates your ethernet hardware. The first ethernet card found becomes `eth0`, the second `eth1` etc.

In some cases though, notably *slip* and *ppp*, the network devices are created through the action of some user program. The same sequential device numbering applies, but the devices are not created automatically at boot time. The reason for this is that unlike ethernet devices, the number of active *slip* or *ppp* devices may vary during the uptime of the machine. These cases will be covered in more detail in later sections.

5.4. Configuring a network interface. Kernels 2.0 and 2.2

When you have all of the programs you need and your address and network information you can configure your network interfaces. When we talk about configuring a network interface we are talking about the process of assigning appropriate addresses to a network device and to setting appropriate values for other configurable parameters of a network device. The program most commonly used to do this is the *ifconfig* (interface configure) command.

Typically you would use a command similar to the following:

```
        root# ifconfig eth0 192.168.0.1 netmask 255.255.255.0 up
```

In this case I'm configuring an ethernet interface `` `eth0' `` with the IP address `` `192.168.0.1' `` and a network mask of `` `255.255.255.0' ``. The `*up*' that trails the command tells the interface that it should become active, but can usually be omitted, as it is the default. To shutdown an interface, you can just call ``` ``ifconfig eth0 down'' ```.

The kernel assumes certain defaults when configuring interfaces. For example, you may specify the network address and broadcast address for an interface, but if you don't, as in my example above, then the kernel will make reasonable guesses as to what they should be based on the netmask you supply and if you don't supply a netmask then on the network class of the IP address configured. In my example the kernel would assume that it is a class–C network being configured on the interface and configure a network address of `` `192.168.0.0' `` and a broadcast address of `` `192.168.0.255' `` for the interface.

There are many other options to the *ifconfig* command. The most important of these are:

up

> This option activates an interface (and is the default).

down

> This option deactivates an interface.

[−]arp

> This option enables or disables use of the address resolution protocol on this interface

[−]allmulti

> This option enables or disables the reception of all hardware multicast packets. Hardware multicast enables groups of hosts to receive packets addressed to special destinations. This may be of importance if you are using applications like desktop videoconferencing but is normally not used.

mtu N

> This parameter allows you to set the *MTU* of this device.

netmask <addr>

> This parameter allows you to set the network mask of the network this device belongs to.

irq <addr>

> This parameter only works on certain types of hardware and allows you to set the IRQ of the hardware of this device.

[−]broadcast [addr]

> This parameter allows you to enable and set the accepting of datagrams destined to the broadcast address, or to disable reception of these datagrams.

[−]pointopoint [addr]

This parameter allows you to set the address of the machine at the remote end of a point to point link such as for *slip* or *ppp*.

hw <type <addr>

This parameter allows you to set the hardware address of certain types of network devices. This is not often useful for ethernet, but is useful for other network types such as AX.25.

With the release of Kernel 2.2 there are a number of options available that are not listed above. Some of the most interesting are tunneling and IPV6 options. The ifconfig paramaters for kernel 2.2 are listed below.

interface

The name of the interface. This is usually a driver name followed by a unit number, for example eth0 for the first Ethernet interface.

up

This flag causes the interface to be activated. It is implicitly specified if an address is assigned to the interface.

down

This flag causes the driver for this interface to be shut down.

[−]arp

Enables or disables the use of the ARP protocol on this interface.

[−]promisc

Enables or disables the promiscuous mode of the interface. If selected, all packets on the network will be received by the interface.

[−]allmulti

Enables or disables all−multicast mode. If selected, all multicast packets on the network will be received by the interface.

metric N

This parameter sets the interface metric.

mtu N

This parameter sets the Maximum Transfer Unit (MTU) of an interface.

dstaddr addr

Sets the remote IP address for a point−to−point link (such as PPP). This keyword is now obsolete; use the pointopoint keyword instead.

netmask addr

Sets the IP network mask for this interface. This value defaults to the usual class A, B or C network mask (as derived from the interface IP address), but it can be set to any value.

add addr prefixlen

Adds an IPv6 address to an interface.

del addr prefixlen

Removes an IPv6 address from an interface.

tunnel aa.bb.cc.dd

Creates a new SIT (IPv6−in−IPv4) device, tunnelling to the given destination.

irq addr

Sets the interrupt line used by this device. Not all devices can dynamically change their IRQ set− ting.

io_addr addr

Sets the start address in I/O space for this device.

mem_start addr

Set the start address for shared memory used by this device. Only a few devices need this.

media type

Sets the physical port or medium type to be used by the device. Not all devices can change this set− ting, and those that can vary in what values they support. Typical values for type are 10base2 (thin Ethernet), 10baseT (twisted−pair 10Mbps Ethernet), AUI (external transceiver) and so on. The special medium type of auto can be used to tell the driver to auto−sense the media. Again, not all drivers can do this.

[−]broadcast [addr]

If the address argument is given, set the protocol broadcast address for this interface. Otherwise, set (or clear) the IFF_BROADCAST flag for the interface.

[−]pointopoint [addr]

This keyword enables the point−to−point mode of an interface, meaning that it is a direct link between two machines with nobody else listening on it. If the address argument is also given, set the pro− tocol address of the other side of the link, just like the obsolete dstaddr keyword does. Otherwise, set or clear the IFF_POINTOPOINT flag for the interface.

hw class address

Set the hardware address of this interface, if the device driver supports this operation. The keyword must be followed by the name of the hardware class and the printable ASCII equivalent of the hardware address. Hardware classes currently supported include ether (Ethernet), ax25 (AMPR AX.25), ARCnet and netrom (AMPR NET/ROM).

multicast

Set the multicast flag on the interface. This should not normally be needed as the drivers set the flag correctly themselves.

address

The IP address to be assigned to this interface.

txqueuelen length

Set the length of the transmit queue of the device. It is useful to set this to small values for slower devices with a high latency (modem links, ISDN) to prevent fast bulk transfers from disturbing inter− active traffic like telnet too much.

You may use the *ifconfig* command on any network interface. Some user programs such as *pppd* and *dip* automatically configure the network devices as they create them, so manual use of *ifconfig* is unnecessary.

5.5. Configuring your Name Resolver.

The `*Name Resolver*' is a part of the linux standard library. Its prime function is to provide a service to convert human−friendly hostnames like ``ftp.funet.fi`' into machine friendly IP addresses such as `128.214.248.6`.

5.5.1. What's in a name ?

You will probably be familiar with the appearance of Internet host names, but may not understand how they are constructed, or deconstructed. Internet domain names are hierarchical in nature, that is, they have a tree−like structure. A `*domain*' is a family, or group of names. A `*domain*' may be broken down into `*subdomain*'. A `*toplevel domain*' is a domain that is not a subdomain. The Top Level Domains are specified in RFC−920. Some examples of the most common top level domains are:

COM

Commercial Organizations

EDU

Educational Organizations

GOV

Government Organizations

MIL

Military Organizations

ORG

Other organizations

NET

Internet–Related Organizations

Country Designator

these are two letters codes that represent a particular country.

For historical reasons most domains belonging to one of the non–country based top level domains were used by organizations within the United States, although the United States also has its own country code `` `.us' ``. This is not true any more for `.com` and `.org` domains, which are commonly used by non–us companies.

Each of these top level domains has subdomains. The top level domains based on country name are often next broken down into subdomains based on the `com`, `edu`, `gov`, `mil` and `org` domains. So for example you end up with: `com.au` and `gov.au` for commercial and government organizations in Australia; note that this is not a general rule, as actual policies depend on the naming authority for each domain.

The next level of division usually represents the name of the organization. Further subdomains vary in nature, often the next level of subdomain is based on the departmental structure of the organization but it may be based on any criterion considered reasonable and meaningful by the network administrators for the organization.

The very left–most portion of the name is always the unique name assigned to the host machine and is called the `*hostname*', the portion of the name to the right of the hostname is called the `*domainname*' and the complete name is called the `*Fully Qualified Domain Name*'.

To use Terry's host as an example, the fully qualified domain name is `` `perf.no.itg.telstra.com.au' ``. This means that the host name is `` `perf' `` and the domain name is `` `no.itg.telstra.com.au' ``. The domain name is based on a top level domain based on his country, Australia and as his email address belongs to a commercial organization, `` `.com' `` is there as the next level domain. The name of the company is (was) `` `telstra' `` and their internal naming structure is based on organizational structure, in this case the machine belongs to the Information Technology Group, Network Operations section.

Usually, the names are fairly shorter; for example, my ISP is called ``` ``systemy.it'' ``` and my non–profit organization is called ``` ``linux.it'' ```, without any `com` and `org` subdomain, so that my own host is just called ``` ``morgana.systemy.it'' ``` and `rubini@linux.it` is a valid email address. Note that the owner of a domain has the rights to register hostnames as well as subdomains; for example, the LUG I belong to uses the domain `pluto.linux.it`, because the owners of `linux.it` agreed to open a subdomain for the

LUG.

5.5.2. What information you will need.

You will need to know what domain your hosts name will belong to. The name resolver software provides this name translation service by making requests to a `*Domain Name Server*', so you will need to know the IP address of a local nameserver that you can use.

There are three files you need to edit, I'll cover each of these in turn.

5.5.3. /etc/resolv.conf

The `/etc/resolv.conf` is the main configuration file for the name resolver code. Its format is quite simple. It is a text file with one keyword per line. There are three keywords typically used, they are:

domain

This keyword specifies the local domain name.

search

This keyword specifies a list of alternate domain names to search for a hostname

nameserver

This keyword, which may be used many times, specifies an IP address of a domain name server to query when resolving names

An example `/etc/resolv.conf` might look something like:

```
        domain maths.wu.edu.au
        search maths.wu.edu.au wu.edu.au
        nameserver 192.168.10.1
        nameserver 192.168.12.1
```

This example specifies that the default domain name to append to unqualified names (ie hostnames supplied without a domain) is `maths.wu.edu.au` and that if the host is not found in that domain to also try the `wu.edu.au` domain directly. Two nameservers entry are supplied, each of which may be called upon by the name resolver code to resolve the name.

5.5.4. /etc/host.conf

The `/etc/host.conf` file is where you configure some items that govern the behaviour of the name resolver code. The format of this file is described in detail in the `` `resolv+ ``' man page. In nearly all circumstances the following example will work for you:

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

This configuration tells the name resolver to check the `/etc/hosts` file before attempting to query a nameserver and to return all valid addresses for a host found in the `/etc/hosts` file instead of just the first.

5.5.5. /etc/hosts

The `/etc/hosts` file is where you put the name and IP address of local hosts. If you place a host in this file then you do not need to query the domain name server to get its IP Address. The disadvantage of doing this is that you must keep this file up to date yourself if the IP address for that host changes. In a well managed system the only hostnames that usually appear in this file are an entry for the loopback interface and the local hosts name.

```
        # /etc/hosts
        127.0.0.1      localhost loopback
        192.168.0.1    this.host.name
```

You may specify more than one host name per line as demonstrated by the first entry, which is a standard entry for the loopback interface.

5.5.6. Running a name server

If you want to run a local nameserver, you can do it easily. Please refer to the DNS−HOWTO and to any documents included in your version of *BIND* (Berkeley Internet Name Domain).

5.6. Configuring your loopback interface.

The `` `loopback ``' interface is a special type of interface that allows you to make connections to yourself. There are various reasons why you might want to do this, for example, you may wish to test some network software without interfering with anybody else on your network. By convention the IP address `127.0.0.1' has been assigned specifically for loopback. So no matter what machine you go to, if you open a telnet connection to 127.0.0.1 you will always reach the local host.

Configuring the loopback interface is simple and you should ensure you do (but note that this task is usually performed by the standard initialization scripts).

```
        root# ifconfig lo 127.0.0.1
        root# route add −host 127.0.0.1 lo
```

We'll talk more about the *route* command in the next section.

5.7. Routing.

Routing is a big topic. It is easily possible to write large volumes of text about it. Most of you will have fairly simple routing requirements, some of you will not. I will cover some basic fundamentals of routing only. If you are interested in more detailed information then I suggest you refer to the references provided at the start of the document.

Let's start with a definition. What is IP routing ? Here is one that I'm using:

"IP Routing is the process by which a host with multiple network connections decides where to deliver IP datagrams it has received."

It might be useful to illustrate this with an example. Imagine a typical office router, it might have a PPP link off the Internet, a number of ethernet segments feeding the workstations and another PPP link off to another office. When the router receives a datagram on any of its network connections, routing is the mechanism that it uses to determine which interface it should send the datagram to next. Simple hosts also need to route, all Internet hosts have two network devices, one is the loopback interface described above and the other is the one it uses to talk to the rest of the network, perhaps an ethernet, perhaps a PPP or SLIP serial interface.

Ok, so how does routing work ? Each host keeps a special list of routing rules, called a routing table. This table contains rows which typically contain at least three fields, the first is a destination address, the second is the name of the interface to which the datagram is to be routed and the third is optionally the IP address of another machine which will carry the datagram on its next step through the network. In linux you can see this table by using the following command:

```
        user% cat /proc/net/route
```

or by using either of the following commands:

```
user% /sbin/route −n
user% netstat −r
```

The routing process is fairly simple: an incoming datagram is received, the destination address (who it is for) is examined and compared with each entry in the table. The entry that best matches that address is selected and the datagram is forwarded to the specified interface. If the gateway field is filled then the datagram is forwarded to that host via the specified interface, otherwise the destination address is assumed to be on the network supported by the interface.

To manipulate this table a special command is used. This command takes command line arguments and converts them into kernel system calls that request the kernel to add, delete or modify entries in the routing table. The command is called `*route*'.

A simple example. Imagine you have an ethernet network. You've been told it is a class−C network with an address of `192.168.1.0`. You've been supplied with an IP address of `192.168.1.10` for your use and have been told that `192.168.1.1` is a router connected to the Internet.

The first step is to configure the interface as described earlier. You would use a command like:

```
root# ifconfig eth0 192.168.1.10 netmask 255.255.255.0 up
```

You now need to add an entry into the routing table to tell the kernel that datagrams for all hosts with addresses that match `192.168.1.*` should be sent to the ethernet device. You would use a command similar to:

```
root# route add −net 192.168.1.0 netmask 255.255.255.0 eth0
```

Note the use of the ``−net`' argument to tell the route program that this entry is a network route. Your other choice here is a ``−host`' route which is a route that is specific to one IP address.

This route will enable you to establish IP connections with all of the hosts on your ethernet segment. But what about all of the IP hosts that aren't on your ethernet segment ?

It would be a very difficult job to have to add routes to every possible destination network, so there is a special trick that is used to simplify this task. The trick is called the ``default`' route. The `default` route matches every possible destination, but poorly, so that if any other entry exists that matches the required address it will be used instead of the `default` route. The idea of the `default` route is simply to enable you to say "and everything else should go here". In the example I've contrived you would use an entry like:

```
root# route add default gw 192.168.1.1 eth0
```

The `gw' argument tells the route command that the next argument is the IP address, or name, of a gateway or router machine which all datagrams matching this entry should be directed to for further routing.

So, your complete configuration would look like:

```
root# ifconfig eth0 192.168.1.10 netmask 255.255.255.0 up
root# route add −net 192.168.1.0 netmask 255.255.255.0 eth0
root# route add default gw 192.168.1.1 eth0
```

If you take a close look at your network `rc' files you will find that at least one of them looks very similar to this. This is a very common configuration.

Let's now look at a slightly more complicated routing configuration. Let's imagine we are configuring the router we looked at earlier, the one supporting the PPP link to the Internet and the lan segments feeding the workstations in the office. Lets imagine the router has three ethernet segments and one PPP link. Our routing configuration would look something like:

```
root# route add −net 192.168.1.0 netmask 255.255.255.0 eth0
root# route add −net 192.168.2.0 netmask 255.255.255.0 eth1
root# route add −net 192.168.3.0 netmask 255.255.255.0 eth2
root# route add default ppp0
```

Each of the workstations would use the simpler form presented above, only the router needs to specify each of the network routes separately because for the workstations the `default` route mechanism will capture all of them letting the router worry about splitting them up appropriately. You may be wondering why the default route presented doesn't specify a `gw'. The reason for this is simple, serial link protocols such as PPP and SLIP only have two hosts on their network; (one at each end). To specify the host at the other end of the link as the gateway is pointless and redundant, as there is no other choice, so you do not need to specify a gateway for these types of network connections. Other network types such as ethernet, arcnet or token ring do require the gateway to be specified as these networks support large numbers of hosts on them.

5.7.1. So what does the *routed* program do ?

The routing configuration described above is best suited to simple network arrangements where there are only ever single possible paths to destinations. When you have a more complex network arrangement things get a little more complicated. Fortunately for most of you this won't be an issue.

The big problem with `manual routing' or `static routing' as described, is that if a machine or link fails in your network then the only way you can direct your datagrams another way, if another way exists, is by manually intervening and executing the appropriate commands. Naturally this is clumsy, slow, impractical and hazard prone. Various techniques have been developed to automatically adjust routing tables in the event of network failures where there are alternate routes, all of these techniques are loosely grouped by the term `dynamic routing protocols'.

You may have heard of some of the more common dynamic routing protocols. The most common are probably RIP (Routing Information Protocol) and OSPF (Open Shortest Path First Protocol). The Routing Information Protocol is very common on small networks such as small−medium sized corporate networks or building networks. OSPF is more modern and more capable at handling large network configurations and better suited to environments where there is a large number of possible paths through the network. Common implementations of these protocols are: `*routed*' − RIP and `*gated*' − RIP, OSPF and others. The `*routed*' program is normally supplied with your Linux distribution or is included in the `NetKit' package detailed above.

An example of where and how you might use a dynamic routing protocol might look something like the following:

```
13;    192.168.1.0 /                        192.168.2.0 /
       255.255.255.0                        255.255.255.0
     -                                    -
     |                                    |
     |   /-----\                /-----\   |
     |   |     |ppp0   //    ppp0|     |   |
eth0 |---|  A  |------//---------|  B  |---| eth0
     |   |     |     //         |     |   |
     |   \-----/                \-----/   |
     |      \ ppp1            ppp1 /      |
     -       \                    /       -
              \                  /
               \                /
                \              /
                 \            /
                  \          /
                   \        /
                    \      /
                     \    /
                 ppp0\   /ppp1
                    /-----\
                    |     |
                    |  C  |
                    |     |
                    \-----/
                       |eth0
                       |
                  |---------|
```

```
                  192.168.3.0 /
                     255.255.255.013;
```

We have three routers A, B and C. Each supports one ethernet segment with a Class C IP network (netmask 255.255.255.0). Each router also has a PPP link to each of the other routers. The network forms a triangle.

It should be clear that the routing table at router A could look like:

```
        root# route add −net 192.168.1.0 netmask 255.255.255.0 eth0
        root# route add −net 192.168.2.0 netmask 255.255.255.0 ppp0
        root# route add −net 192.168.3.0 netmask 255.255.255.0 ppp1
```

This would work just fine until the link between router A and B should fail. If that link failed then with the routing entry shown above hosts on the ethernet segment of A could not reach hosts on the ethernet segment on B because their datagram would be directed to router A's ppp0 link which is broken. They could still continue to talk to hosts on the ethernet segment of C and hosts on the C's ethernet segment could still talk to hosts on B's ethernet segment because the link between B and C is still intact.

But wait, if A can talk to C and C can still talk to B, why shouldn't A route its datagrams for B via C and let C send them to B ? This is exactly the sort of problem that dynamic routing protocols like RIP were designed to solve. If each of the routers A, B and C were running a routing daemon then their routing tables would be automatically adjusted to reflect the new state of the network should any one of the links in the network fail. To configure such a network is simple, at each router you need only do two things. In this case for Router A:

```
        root# route add −net 192.168.1.0 netmask 255.255.255.0 eth0
        root# /usr/sbin/routed
```

The `*routed*' routing daemon automatically finds all active network ports when it starts and sends and listens for messages on each of the network devices to allow it to determine and update the routing table on the host.

This has been a very brief explanation of dynamic routing and where you would use it. If you want more information then you should refer to the suggested references listed at the top of the document.

The important points relating to dynamic routing are:

1. You only need to run a dynamic routing protocol daemon when your Linux machine has the possibility of selecting multiple possible routes to a destination. An example of this would be if you plan to use IP Masquerading.

2. The dynamic routing daemon will automatically modify your routing table to adjust to changes in your network.
3. RIP is suited to small to medium sized networks.

5.8. Configuring your network servers and services.

Network servers and services are those programs that allow a remote user to make user of your Linux machine. Server programs listen on network ports. Network ports are a means of addressing a particular service on any particular host and are how a server knows the difference between an incoming telnet connection and an incoming ftp connection. The remote user establishes a network connection to your machine and the server program, the network daemon program, listening on that port accepts the connection and executes. There are two ways that network daemons may operate. Both are commonly employed in practice. The two ways are:

standalone

The network daemon program listens on the designated network port and when an incoming connection is made it manages the network connection itself to provide the service.

slave to the inetd server

The *inetd* server is a special network daemon program that specializes in managing incoming network connections. It has a configuration file which tells it what program needs to be run when an incoming connection is received. Any service port may be configured for either of the tcp or udp protcols. The ports are described in another file that we will talk about soon.

There are two important files that we need to configure. They are the `/etc/services` file which assigns names to port numbers and the `/etc/inetd.conf` file which is the configuration file for the *inetd* network daemon.

5.8.1. `/etc/services`

The `/etc/services` file is a simple database that associates a human friendly name to a machine friendly service port. Its format is quite simple. The file is a text file with each line representing and entry in the database. Each entry is comprised of three fields separated by any number of whitespace (tab or space) characters. The fields are:

name port/protocol aliases # comment

name

A single word name that represents the service being described.

port/protocol

This field is split into two subfields.

port

A number that specifies the port number the named service will be available on. Most of the common services have assigned service numbers. These are described in `RFC−1340`.

protocol

This subfield may be set to either `tcp` or `udp`.

It is important to note that an entry of `18/tcp` is very different from an entry of `18/udp` and that there is no technical reason why the same service needs to exist on both. Normally common sense prevails and it is only if a particular service is available via both `tcp` and `udp` that you will see an entry for both.

aliases

Other names that may be used to refer to this service entry.

Any text appearing in a line after a `#' character is ignored and treated as a comment.

5.8.1.1. An example `/etc/services` file.

All modern linux distributions provide a good `/etc/services` file. Just in case you happen to be building a machine from the ground up, here is a copy of the `/etc/services` file supplied with an old Debian distribution:

```
# /etc/services:
# $Id: Net-HOWTO.sgml,v 1.1 2000/07/19 16:04:26 gferg Exp $
#
# Network services, Internet style
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, most entries here have two entries
# even if the protocol doesn't support UDP operations.
# Updated from RFC 1340, ``Assigned Numbers'' (July 1992).  Not all ports
# are included, only the more common ones.

tcpmux          1/tcp                           # TCP port service multiplexer
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
qotd            17/tcp          quote
msp             18/tcp                          # message send protocol
msp             18/udp                          # message send protocol
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp-data        20/tcp
```

```
ftp             21/tcp
ssh             22/tcp                            # SSH Remote Login Protocol
ssh             22/udp                            # SSH Remote Login Protocol
telnet          23/tcp
# 24 - private
smtp            25/tcp          mail
# 26 - unassigned
time            37/tcp          timserver
time            37/udp          timserver
rlp             39/udp          resource          # resource location
nameserver      42/tcp          name              # IEN 116
whois           43/tcp          nicname
re-mail-ck      50/tcp                            # Remote Mail Checking Protocol
re-mail-ck      50/udp                            # Remote Mail Checking Protocol
domain          53/tcp          nameserver        # name-domain server
domain          53/udp          nameserver
mtp             57/tcp                            # deprecated
bootps          67/tcp                            # BOOTP server
bootps          67/udp
bootpc          68/tcp                            # BOOTP client
bootpc          68/udp
tftp            69/udp
gopher          70/tcp                            # Internet Gopher
gopher          70/udp
rje             77/tcp          netrjs
finger          79/tcp
www             80/tcp          http              # WorldWideWeb HTTP
www             80/udp                            # HyperText Transfer Protocol
link            87/tcp          ttylink
kerberos        88/tcp          kerberos5 krb5    # Kerberos v5
kerberos        88/udp          kerberos5 krb5    # Kerberos v5
supdup          95/tcp
# 100 - reserved
hostnames       101/tcp         hostname          # usually from sri-nic
iso-tsap        102/tcp         tsap              # part of ISODE.
csnet-ns        105/tcp         cso-ns            # also used by CSO name server
csnet-ns        105/udp         cso-ns
rtelnet         107/tcp                           # Remote Telnet
rtelnet         107/udp
pop-2           109/tcp         postoffice        # POP version 2
pop-2           109/udp
pop-3           110/tcp                           # POP version 3
pop-3           110/udp
sunrpc          111/tcp         portmapper        # RPC 4.0 portmapper TCP
sunrpc          111/udp         portmapper        # RPC 4.0 portmapper UDP
auth            113/tcp         authentication tap ident
sftp            115/tcp
uucp-path       117/tcp
nntp            119/tcp         readnews untp     # USENET News Transfer Protocol
ntp             123/tcp
ntp             123/udp                           # Network Time Protocol
netbios-ns      137/tcp                           # NETBIOS Name Service
netbios-ns      137/udp
netbios-dgm     138/tcp                           # NETBIOS Datagram Service
netbios-dgm     138/udp
netbios-ssn     139/tcp                           # NETBIOS session service
netbios-ssn     139/udp
imap2           143/tcp                           # Interim Mail Access Proto v2
imap2           143/udp
snmp            161/udp                           # Simple Net Mgmt Proto
snmp-trap       162/udp         snmptrap          # Traps for SNMP
cmip-man        163/tcp                           # ISO mgmt over IP (CMOT)
```

```
cmip-man        163/udp
cmip-agent      164/tcp
cmip-agent      164/udp
xdmcp           177/tcp                         # X Display Mgr. Control Proto
xdmcp           177/udp
nextstep        178/tcp         NeXTStep NextStep       # NeXTStep window
nextstep        178/udp         NeXTStep NextStep       # server
bgp             179/tcp                         # Border Gateway Proto.
bgp             179/udp
prospero        191/tcp                         # Cliff Neuman's Prospero
prospero        191/udp
irc             194/tcp                         # Internet Relay Chat
irc             194/udp
smux            199/tcp                         # SNMP Unix Multiplexer
smux            199/udp
at-rtmp         201/tcp                         # AppleTalk routing
at-rtmp         201/udp
at-nbp          202/tcp                         # AppleTalk name binding
at-nbp          202/udp
at-echo         204/tcp                         # AppleTalk echo
at-echo         204/udp
at-zis          206/tcp                         # AppleTalk zone information
at-zis          206/udp
z3950           210/tcp         wais            # NISO Z39.50 database
z3950           210/udp         wais
ipx             213/tcp                         # IPX
ipx             213/udp
imap3           220/tcp                         # Interactive Mail Access
imap3           220/udp                         # Protocol v3
ulistserv       372/tcp                         # UNIX Listserv
ulistserv       372/udp
#
# UNIX specific services
#
exec            512/tcp
biff            512/udp         comsat
login           513/tcp
who             513/udp         whod
shell           514/tcp         cmd             # no passwords used
syslog          514/udp
printer         515/tcp         spooler         # line printer spooler
talk            517/udp
ntalk           518/udp
route           520/udp         router routed   # RIP
timed           525/udp         timeserver
tempo           526/tcp         newdate
courier         530/tcp         rpc
conference      531/tcp         chat
netnews         532/tcp         readnews
netwall         533/udp                         # -for emergency broadcasts
uucp            540/tcp         uucpd           # uucp daemon
remotefs        556/tcp         rfs_server rfs  # Brunhoff remote filesystem
klogin          543/tcp                         # Kerberized `rlogin' (v5)
kshell          544/tcp         krcmd           # Kerberized `rsh' (v5)
kerberos-adm    749/tcp                         # Kerberos `kadmin' (v5)
#
webster         765/tcp                         # Network dictionary
webster         765/udp
#
# From ``Assigned Numbers'':
#
#> The Registered Ports are not controlled by the IANA and on most systems
```

```
#> can be used by ordinary user processes or programs executed by ordinary
#> users.
#
#> Ports are used in the TCP [45,106] to name the ends of logical
#> connections which carry long term conversations.  For the purpose of
#> providing services to unknown callers, a service contact port is
#> defined.  This list specifies the port used by the server process as its
#> contact port.  While the IANA can not control uses of these ports it
#> does register or list uses of these ports as a convenience to the
#> community.
#
ingreslock      1524/tcp
ingreslock      1524/udp
prospero-np     1525/tcp                # Prospero non-privileged
prospero-np     1525/udp
rfe             5002/tcp                # Radio Free Ethernet
rfe             5002/udp                # Actually uses UDP only
bbs             7000/tcp                # BBS service
#
#
# Kerberos (Project Athena/MIT) services
# Note that these are for Kerberos v4 and are unofficial.  Sites running
# v4 should uncomment these and comment out the v5 entries above.
#
kerberos4       750/udp         kdc     # Kerberos (server) udp
kerberos4       750/tcp         kdc     # Kerberos (server) tcp
kerberos_master 751/udp                 # Kerberos authentication
kerberos_master 751/tcp                 # Kerberos authentication
passwd_server   752/udp                 # Kerberos passwd server
krb_prop        754/tcp                 # Kerberos slave propagation
krbupdate       760/tcp         kreg    # Kerberos registration
kpasswd         761/tcp         kpwd    # Kerberos "passwd"
kpop            1109/tcp                # Pop with Kerberos
knetd           2053/tcp                # Kerberos de-multiplexor
zephyr-srv      2102/udp                # Zephyr server
zephyr-clt      2103/udp                # Zephyr serv-hm connection
zephyr-hm       2104/udp                # Zephyr hostmanager
eklogin         2105/tcp                # Kerberos encrypted rlogin
#
# Unofficial but necessary (for NetBSD) services
#
supfilesrv      871/tcp                 # SUP server
supfiledbg      1127/tcp                # SUP debugging
#
# Datagram Delivery Protocol services
#
rtmp            1/ddp                   # Routing Table Maintenance Protocol
nbp             2/ddp                   # Name Binding Protocol
echo            4/ddp                   # AppleTalk Echo Protocol
zip             6/ddp                   # Zone Information Protocol
#
# Debian GNU/Linux services
rmtcfg          1236/tcp                # Gracilis Packeten remote config server
xtel            1313/tcp                # french minitel
cfinger         2003/tcp                # GNU Finger
postgres        4321/tcp                # POSTGRES
mandelspawn     9359/udp        mandelbrot      # network mandelbrot

# Local services
```

In the real world, the actual file is always growing as new services are being created. If you fear your own copy is incomplete, I'd suggest to copy a new `/etc/services` from a recent distribution.

5.8.2. /etc/inetd.conf

The `/etc/inetd.conf` file is the configuration file for the *inetd* server daemon. Its function is to tell *inetd* what to do when it receives a connection request for a particular service. For each service that you wish to accept connections for you must tell *inetd* what network server daemon to run and how to run it.

Its format is also fairly simple. It is a text file with each line describing a service that you wish to provide. Any text in a line following a `#' is ignored and considered a comment. Each line contains seven fields separated by any number of whitespace (tab or space) characters. The general format is as follows:

```
service  socket_type  proto  flags  user  server_path  server_args
```

service

Is the service relevant to this configuration as taken from the `/etc/services` file.

socket_type

This field describes the type of socket that this entry will consider relevant, allowable values are: `stream`, `dgram`, `raw`, `rdm`, or `seqpacket`. This is a little technical in nature, but as a rule of thumb nearly all `tcp` based services use `stream` and nearly all `udp` based services use `dgram`. It is only very special types of server daemons that would use any of the other values.

proto

The protocol to be considered valid for this entry. This should match the appropriate entry in the `/etc/services` file and will typically be either `tcp` or `udp`. Sun RPC (Remote Procedure Call) based servers will use `rpc/tcp` or `rpc/udp`.

flags

There are really only two possible settings for this field. This field setting tells *inetd* whether the network server program frees the socket after it has been started and therefore whether *inetd* can start another one on the next connection request, or whether *inetd* should wait and assume that any server daemon already running will handle the new connection request. Again this is a little tricky to work out, but as a rule of thumb all `tcp` servers should have this entry set to `nowait` and most `udp` servers should have this entry set to `wait`. Be warned there are some notable exceptions to this, so let the example guide you if you are not sure.

user

This field describes which user account from `/etc/passwd` will be set as the owner of the network daemon when it is started. This is often useful if you want to safeguard against security risks. You can set the user of an entry to the `nobody` user so that if the network server security is breached the possible damage is minimized. Typically this field is set to `root` though, because many servers require root privileges in order to function correctly.

server_path

This field is pathname to the actual server program to execute for this entry.

server_args

This field comprises the rest of the line and is optional. This field is where you place any command line arguments that you wish to pass to the server daemon program when it is launched.

5.8.2.1. An example `/etc/inetd.conf`

As for the `/etc/services` file all modern distributions will include a good `/etc/inetd.conf` file for you to work with. Here, for completeness is the `/etc/inetd.conf` file from the Debian distribution.

```
# /etc/inetd.conf:  see inetd(8) for further informations.
#
# Internet server configuration database
#
#
# Modified for Debian by Peter Tobias <tobias@et-inf.fho-emden.de>
#
# <service_name> <sock_type> <proto> <flags> <user> <server_path> <args>
#
# Internal services
#
#echo           stream  tcp     nowait  root    internal
#echo           dgram   udp     wait    root    internal
discard         stream  tcp     nowait  root    internal
discard         dgram   udp     wait    root    internal
daytime         stream  tcp     nowait  root    internal
daytime         dgram   udp     wait    root    internal
#chargen        stream  tcp     nowait  root    internal
#chargen        dgram   udp     wait    root    internal
time            stream  tcp     nowait  root    internal
time            dgram   udp     wait    root    internal
#
# These are standard services.
#
telnet  stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.telnetd
ftp     stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.ftpd
#fsp    dgram   udp     wait    root    /usr/sbin/tcpd  /usr/sbin/in.fspd
#
# Shell, login, exec and talk are BSD protocols.
#
shell   stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.rshd
login   stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.rlogind
#exec   stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.rexecd
```

```
talk    dgram   udp     wait    root    /usr/sbin/tcpd  /usr/sbin/in.talkd
ntalk   dgram   udp     wait    root    /usr/sbin/tcpd  /usr/sbin/in.ntalkd
#
# Mail, news and uucp services.
#
smtp    stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.smtpd
#nntp   stream  tcp     nowait  news    /usr/sbin/tcpd  /usr/sbin/in.nntpd
#uucp   stream  tcp     nowait  uucp    /usr/sbin/tcpd  /usr/lib/uucp/uucico
#comsat dgram   udp     wait    root    /usr/sbin/tcpd  /usr/sbin/in.comsat
#
# Pop et al
#
#pop−2  stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.pop2d
#pop−3  stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.pop3d
#
# `cfinger' is for the GNU finger server available for Debian.  (NOTE: The
# current implementation of the `finger' daemon allows it to be run as `root'.)
#
#cfinger stream tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.cfingerd
#finger stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.fingerd
#netstat        stream  tcp     nowait  nobody  /usr/sbin/tcpd  /bin/netstat
#systat stream  tcp     nowait  nobody  /usr/sbin/tcpd  /bin/ps −auwwx
#
# Tftp service is provided primarily for booting.  Most sites
# run this only on machines acting as "boot servers."
#
#tftp   dgram   udp     wait    nobody  /usr/sbin/tcpd  /usr/sbin/in.tftpd
#tftp   dgram   udp     wait    nobody  /usr/sbin/tcpd  /usr/sbin/in.tftpd /boot
#bootps dgram   udp     wait    root    /usr/sbin/bootpd        bootpd −i −t 120
#
# Kerberos authenticated services (these probably need to be corrected)
#
#klogin         stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.rlogind −k
#eklogin        stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.rlogind −k −x
#kshell         stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.rshd −k
#
# Services run ONLY on the Kerberos server (these probably need to be corrected)
#
#krbupdate      stream tcp      nowait  root    /usr/sbin/tcpd  /usr/sbin/registerd
#kpasswd        stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/kpasswdd
#
# RPC based services
#
#mountd/1       dgram   rpc/udp wait    root    /usr/sbin/tcpd  /usr/sbin/rpc.mountd
#rstatd/1−3     dgram   rpc/udp wait    root    /usr/sbin/tcpd  /usr/sbin/rpc.rstatd
#rusersd/2−3    dgram   rpc/udp wait    root    /usr/sbin/tcpd  /usr/sbin/rpc.rusersd
#walld/1        dgram   rpc/udp wait    root    /usr/sbin/tcpd  /usr/sbin/rpc.rwalld
#
# End of inetd.conf.
ident           stream  tcp     nowait  nobody  /usr/sbin/identd        identd −i
```

5.9. Other miscellaneous network related configuration files.

There are a number of miscellaneous files relating to network configuration under linux that you might be interested in. You may never have to modify these files, but it is worth describing them so you know what they contain and what they are for.

5.9.1. /etc/protocols

The `/etc/protocols` file is a database that maps protocol id numbers against protocol names. This is used by programmers to allow them to specify protocols by name in their programs and also by some programs such as *tcpdump* to allow them to display names instead of numbers in their output. The general syntax of the file is:

```
 protocolname  number  aliases
```

The `/etc/protocols` file supplied with the Debian distribution is as follows:

```
# /etc/protocols:
# $Id: Net-HOWTO.sgml,v 1.1 2000/07/19 16:04:26 gferg Exp $
#
# Internet (IP) protocols
#
#       from: @(#)protocols     5.1 (Berkeley) 4/17/89
#
# Updated for NetBSD based on RFC 1340, Assigned Numbers (July 1992).

ip      0       IP              # internet protocol, pseudo protocol number
icmp    1       ICMP            # internet control message protocol
igmp    2       IGMP            # Internet Group Management
ggp     3       GGP             # gateway-gateway protocol
ipencap 4       IP-ENCAP        # IP encapsulated in IP (officially ``IP'')
st      5       ST              # ST datagram mode
tcp     6       TCP             # transmission control protocol
egp     8       EGP             # exterior gateway protocol
pup     12      PUP             # PARC universal packet protocol
udp     17      UDP             # user datagram protocol
hmp     20      HMP             # host monitoring protocol
xns-idp 22      XNS-IDP         # Xerox NS IDP
rdp     27      RDP             # "reliable datagram" protocol
iso-tp4 29      ISO-TP4         # ISO Transport Protocol class 4
xtp     36      XTP             # Xpress Tranfer Protocol
ddp     37      DDP             # Datagram Delivery Protocol
idpr-cmtp       39      IDPR-CMTP       # IDPR Control Message Transport
rspf    73      RSPF            # Radio Shortest Path First.
vmtp    81      VMTP            # Versatile Message Transport
ospf    89      OSPFIGP         # Open Shortest Path First IGP
ipip    94      IPIP            # Yet Another IP encapsulation
encap   98      ENCAP           # Yet Another IP encapsulation
```

5.9.2. `/etc/networks`

The `/etc/networks` file has a similar function to that of the `/etc/hosts` file. It provides a simple database of network names against network addresses. Its format differs in that there may be only two fields per line and that the fields are coded as:

```
networkname networkaddress
```

An example might look like:

```
        loopnet    127.0.0.0
        localnet   192.168.0.0
        amprnet    44.0.0.0
```

When you use commands like the *route* command, if a destination is a network and that network has an entry in the `/etc/networks` file then the route command will display that network name instead of its address.

5.10. Network Security and access control.

Let me start this section by warning you that securing your machine and network against malicious attack is a complex art. I do not consider myself an expert in this field at all and while the following mechanisms I describe will help, if you are serious about security then I recommend you do some research of your own into the subject. There are many good references on the Internet relating to the subject, including the Security−HOWTO

An important rule of thumb is: `*Don't run servers you don't intend to use*'. Many distributions come configured with all sorts of services configured and automatically started. To ensure even a minimum level of safety you should go through your `/etc/inetd.conf` file and comment out (*place a `#' at the start of the line*) any entries for services you don't intend to use. Good candidates are services such as: `shell`, `login`, `exec`, `uucp`, `ftp` and informational services such as `finger`, `netstat` and `systat`.

There are all sorts of security and access control mechanisms, I'll describe the most elementary of them.

5.10.1. /etc/ftpusers

The `/etc/ftpusers` file is a simple mechanism that allows you to deny certain users from logging into your machine via ftp. The `/etc/ftpusers` file is read by the ftp daemon program (*ftpd*) when an incoming ftp connection is received. The file is a simple list of those users who are disallowed from logging in. It might looks something like:

```
# /etc/ftpusers - users not allowed to login via ftp
root
uucp
bin
mail
```

5.10.2. /etc/securetty

The `/etc/securetty` file allows you to specify which `tty` devices `root` is allowed to login on. The `/etc/securetty` file is read by the login program (usually */bin/login*). Its format is a list of the tty devices names allowed, on all others `root` login is disallowed:

```
# /etc/securetty - tty's on which root is allowed to login
tty1
tty2
tty3
tty4
```

5.10.3. The *tcpd* hosts access control mechanism.

The *tcpd* program you will have seen listed in the same `/etc/inetd.conf` provides logging and access control mechanisms to services it is configured to protect.

When it is invoked by the *inetd* program it reads two files containing access rules and either allows or denies access to the server it is protecting accordingly.

It will search the rules files until the first match is found. If no match is found then it assumes that access should be allowed to anyone. The files it searches in sequence are: `/etc/hosts.allow`, `/etc/hosts.deny`. I'll describe each of these in turn. For a complete description of this facility you should refer to the appropriate *man* pages (`hosts_access(5)` is a good starting point).

5.10.3.1. /etc/hosts.allow

The `/etc/hosts.allow` file is a configuration file of the */usr/sbin/tcpd* program. The `hosts.allow` file contains rules describing which hosts are *allowed* access to a service on your machine.

The file format is quite simple:

```
        # /etc/hosts.allow
        #
        # <service list>: <host list> [: command]
```

service list

This is a comma delimited list of server names that this rule applies to. Example server names are: `ftpd`, `telnetd` and `fingerd`.

host list

This is a comma delimited list of host names. You may also use IP addresses here. You may additionally specify hostnames or addresses using wildcard characters to match groups of hosts. Examples include: `gw.vk2ktj.ampr.org` to match a specific host, `.uts.edu.au` to match any hostname ending in that string, `44.` to match any IP address commencing with those digits. There are some special tokens to simplify configuration, some of these are: `ALL` matches every host, `LOCAL` matches any host whose name does not contain a `.' ie is in the same domain as your machine and `PARANOID` matches any host whose name does not match its address (name spoofing). There is one last token that is also useful. The `EXCEPT` token allows you to provide a list with exceptions. This will be covered in an example later.

command

This is an optional parameter. This parameter is the full pathname of a command that would be executed everytime this rule is matched. It could for example run a command that would attempt to identify who is logged onto the connecting host, or to generate a mail message or some other warning to a system administrator that someone is attempting to connect. There are a number of expansions that may be included, some common examples are: `%h` expands to the name of the connecting host or address if it doesn't have a name, `%d` the daemon name being called.

An example:

```
# /etc/hosts.allow
#
# Allow mail to anyone
in.smtpd: ALL
# All telnet and ftp to only hosts within my domain and my host at home.
telnetd, ftpd: LOCAL, myhost.athome.org.au
# Allow finger to anyone but keep a record of who they are.
fingerd: ALL: (finger @%h | mail -s "finger from %h" root)
```

5.10.3.2. /etc/hosts.deny

The `/etc/hosts.deny` file is a configuration file of the */usr/sbin/tcpd* program. The `hosts.deny` file contains rules describing which hosts are *disallowed* access to a service on your machine.

A simple sample would look something like this:

```
# /etc/hosts.deny
#
# Disallow all hosts with suspect hostnames
ALL: PARANOID
#
# Disallow all hosts.
ALL: ALL
```

The `PARANOID` entry is really redundant because the other entry traps everything in any case. Either of these entry would make a reasonable default depending on your particular requirement.

Having an `ALL: ALL` default in the `/etc/hosts.deny` and then specifically enabling on those services and hosts that you want in the `/etc/hosts.allow` file is the safest configuration.

5.10.4. /etc/hosts.equiv

The `hosts.equiv` file is used to grant certain hosts and users access rights to accounts on your machine without having to supply a password. This is useful in a secure environment where you control all machines, but is a security hazard otherwise. Your machine is only as secure as the least secure of the trusted hosts. To maximize security, don't use this mechanism and encourage your users not to use the `.rhosts` file as well.

5.10.5. Configure your *ftp* daemon properly.

Many sites will be interested in running an anonymous *ftp* server to allow other people to upload and download files without requiring a specific userid. If you decide to offer this facility make sure you configure the *ftp* daemon properly for anonymous access. Most *man* pages for `ftpd(8)` describe in some length how to go about this. You should always ensure that you follow these instructions. An important tip is to not use a copy of your `/etc/passwd` file in the anonymous account `/etc` directory, make sure you strip out all account details except those that you must have, otherwise you will be vulnerable to brute force password cracking techniques.

5.10.6. Network Firewalling.

Not allowing datagrams to even reach your machine or servers is an excellent means of security. This is covered in depth in the Firewall−HOWTO, and (more concisely) in a later section of this document.

5.10.7. Other suggestions.

Here are some other, potentially religious suggestions for you to consider.

sendmail

Despite its popularity the *sendmail* daemon appears with frightening regularity on security warning announcements. Its up to you, but I choose not to run it.

NFS and other Sun RPC services

Be wary of these. There are all sorts of possible exploits for these services. It is difficult finding an option to services like NFS, but if you configure them, make sure you are careful with who you allow mount rights to.

Chapter 6. Ethernet Information

This section covers information specific to Ethernet and the configuring of Ethernet Cards.

6.1. Supported Ethernet Cards

6.1.1. 3Com

- 3Com 3c501 − `avoid like the plague'' (3c501 driver)
- 3Com 3c503 (3c503 driver), 3c505 (3c505 driver), 3c507 (3c507 driver), 3c509/3c509B (ISA) / 3c579 (EISA)
- 3Com Etherlink III Vortex Ethercards (3c590, 3c592, 3c595, 3c597) (PCI), 3Com Etherlink XL Boomerang (3c900, 3c905) (PCI) and Cyclone (3c905B, 3c980) Ethercards (3c59x driver) and 3Com Fast EtherLink Ethercard (3c515) (ISA) (3c515 driver)
- 3Com 3ccfe575 Cyclone Cardbus (3c59x driver)
- 3Com 3c575 series Cardbus (3c59x driver) (ALL PCMCIA ??)

6.1.2. AMD, ATT, Allied Telesis, Ansel, Apricot

- AMD LANCE (79C960) / PCnet−ISA/PCI (AT1500, HP J2405A, NE1500/NE2100)
- ATT GIS WaveLAN
- Allied Telesis AT1700
- Allied Telesis LA100PCI−T
- Allied Telesyn AT2400T/BT ("ne" module)
- Ansel Communications AC3200 (EISA)
- Apricot Xen−II / 82596

6.1.3. Cabletron, Cogent, Crystal Lan

- Cabletron E21xx
- Cogent EM110
- Crystal Lan CS8920, Cs8900

6.1.4. Danpex, DEC, Digi, DLink

- Danpex EN−9400
- DEC DE425 (EISA) / DE434/DE435 (PCI) / DE450/DE500 (DE4x5 driver)
- DEC DE450/DE500−XA (dc21x4x) (Tulip driver)
- DEC DEPCA and EtherWORKS
- DEC EtherWORKS 3 (DE203, DE204, DE205)
- DECchip DC21x4x "Tulip"
- DEC QSilver's (Tulip driver)
- Digi International RightSwitch
- DLink DE−220P, DE−528CT, DE−530+, DFE−500TX, DFE−530TX

6.1.5. Fujitsu, HP, ICL, Intel

- Fujitsu FMV−181/182/183/184
- HP PCLAN (27245 and 27xxx series)
- HP PCLAN PLUS (27247B and 27252A)
- HP 10/100VG PCLAN (J2577, J2573, 27248B, J2585) (ISA/EISA/PCI)
- ICL EtherTeam 16i / 32 (EISA)
- Intel EtherExpress
- Intel EtherExpress Pro

6.1.6. KTI, Macromate, NCR NE2000/1000, Netgear, New Media

- KTI ET16/P−D2, ET16/P−DC ISA (work jumperless and with hardware−configuration options)
- Macromate MN−220P (PnP or NE2000 mode)
- NCR WaveLAN
- NE2000/NE1000 (be careful with clones)
- Netgear FA−310TX (Tulip chip)
- New Media Ethernet

6.1.7. PureData, SEEQ, SMC

* PureData PDUC8028, PDI8023
* SEEQ 8005
* SMC Ultra / EtherEZ (ISA)
* SMC 9000 series
* SMC PCI EtherPower 10/100 (DEC Tulip driver)
* SMC EtherPower II (epic100.c driver)

6.1.8. Sun Lance, Sun Intel, Schneider, WD, Zenith, IBM, Enyx

* Sun LANCE adapters (kernel 2.2 and newer)
* Sun Intel adapters (kernel 2.2 and newer)
* Schneider and Koch G16
* Western Digital WD80x3
* Zenith Z−Note / IBM ThinkPad 300 built−in adapter
* Znyx 312 etherarray (Tulip driver)

6.2. General Ethernet Information

Ethernet devices names are `eth0', `eth1', `eth2' etc. The first card detected by the kernel is assigned `eth0' and the rest are assigned sequentially in the order they are detected.

Once you have your kernel properly built to support your ethernet card then configuration of the card is easy.

Typically you would use something like (which most distributions already do for you, if you configured them to support your ethernet):

```
root# ifconfig eth0 192.168.0.1 netmask 255.255.255.0 up
root# route add −net 192.168.0.0 netmask 255.255.255.0 eth0
```

Most of the ethernet drivers were developed by Donald Becker

6.3. Using 2 or more Ethernet Cards in the same machine

6.3.1. If your driver is a module (Normal with newer distros)

The module will typically can detect all of the installed cards.

Information fromt he detection is stored in the file:

/etc/conf.modules.

Consider that a user has 3 NE2000 cards, one at 0x300 one at 0x240, and one at 0x220. You would add the following lines to the /etc/conf.modules file:

```
        alias eth0 ne
        alias eth1 ne
        alias eth2 ne
        options ne io=0x220,0x240,0x300
```

What this does is tell the program *modprobe* to look for 3 NE based cards at the following addresses. It also states in which order they should be found and the device they should be assigned.

Most ISA modules can take multiple comma separated I/O values. For example:

```
        alias eth0 3c501
        alias eth1 3c501
        options eth0 −o 3c501−0 io=0x280 irq=5
        options eth1 −o 3c501−1 io=0x300 irq=7
```

The −o option allows for a unique name to be assigned to each module. The reason for this is that you can not have two copies of the same module loaded.

The irq= option is used to specify the hardware IRQ and the io= to specify the different io ports.

By default, the Linux kernel only probes for one Ethernet device, you need to pass command line arguments to the kernel in order to force detection of furter boards.

To learn how to make your ethernet card(s) working under Linux you should refer to the Ethernet−HOWTO.

Chapter 7. IP Related Information

This section covers information specific to IP.

7.1. DHCP and DHCPD

DHCP is an acronym for Dynamic Host Configuration Protocol. The creation of DHCP has made configuring the network on multiple hosts extremely simple. Instead of having to configure each host separately you can assign all of the commonly used parameters by the hosts using a DHCP server.

Each time the host boots up it will broadcast a packet to the network. This packet is a call to any DHCP servers that are located on the same segment to configure the host.

DHCP is extermely useful in assigning items such as the IP address, Netmask, and gateway of each host.

7.2. DHCP Client Setup for users of LinuxConf

Under linux as the user root start the program linuxconf. This program comes with all versions of redhat and works with X as well as the console. It also works for SuSe, and Caldera.

```
Select Networking
----------------->Basic Host Information
----------------->Select Enable
----------------->Set Config Mode DHCP
```

7.3. DHCP Server Setup for Linux

Retrieve DHCPD if your machine does not already have it installed. Get DHCPD

Quick Note: MAKE SURE YOU HAVE MULTICAST ENABLED IN THE KERNEL.

If there is not a binary distribution for your version of linux you will have to compile DHCPD.

Edit your /etc/rc.d/rc.local to reflect an addition of a route for 255.255.255.255.

Quoted from DHCPd README:

In order for dhcpd to work correctly with picky DHCP clients (e.g., Windows 95), it must be able to send packets with an IP destination address of 255.255.255.255. Unfortunately, Linux insists on changing 255.255.255.255 into the local subnet broadcast address (here, that's 192.5.5.223). This results in a DHCP protocol violation, and while many DHCP clients don't notice the problem, some (e.g., all Microsoft DHCP clients) do. Clients that have this problem will appear not to see DHCPOFFER messages from the server.

Type the following as root:

route add −host 255.255.255.255 dev eth0

If the message appears:

255.255.255.255: Unknown host

Try adding the following entry to your /etc/hosts file:

255.255.255.255 dhcp

Then, try:

route add −host dhcp dev eth0

7.3.1. Options for DHCPD

Now you need to configure DHCPd. In order to do this you will have to create or edit /etc/dhcpd.conf. There is a graphical interface for dhcpd configuration under linuxconf. This makes configuring and managing DHCPD extremely simple.

If you want to configure it by hand follow instructions below. I suggest configuring it by hand at least once. It will help in the diagnostics that a GUI can't give you. Unfortunately Micrsoft doesn't believe this.

The easiest thing to do is assign IP addresses randomly. Below is a sample configuration file that shows this type of setup.

```
# Sample /etc/dhcpd.conf
# (add your comments here)
default−lease−time 1200;
max−lease−time 9200;
option subnet−mask 255.255.255.0;
option broadcast−address 192.168.1.255;
option routers 192.168.1.254;
option domain−name−servers 192.168.1.1, 192.168.1.2;
option domain−name "mydomain.org";

subnet 192.168.1.0 netmask 255.255.255.0 {
range 192.168.1.10 192.168.1.100;
range 192.168.1.150 192.168.1.200;
}
```

This will allow the DHCP server to assign the client an IP address from the range 192.168.1.10−192.168.1.100 or 192.168.1.150−192.168.1.200.

It will lease an IP address for 1200 seconds if the client doesn't request a longer time frame. Otherwise the maximum (allowed) lease the server will allow is 9200 seconds. The server send the following paramaters to the client:

Use 255.255.255.0 as your subnet mask Use 192.168.1.255 as your broadcast address Use 192.168.1.254 as your default gateway USE 192.168.1.1 and 192.168.1.2 as your DNS servers.

If you specify a WINS server for your Windows clients you need to include the following option in the dhcpd.conf file.

option netbios−name−servers 192.168.1.1;

You can also assign specific IP addresses based on clients ethernet *MAC* address e.g.

```
  host haagen {
     hardware ethernet 08:00:2b:4c:59:23;
     fixed−address 192.168.1.222;
}
```

This will assign IP address 192.168.1.222 to a client with ethernet *MAC* address of 08:00:2b:4c:59:23.

7.3.2. Starting the server

In most cases the DHCP installation doesn't create a dhcpd.leases file. Therefore before you start the server you must type to create an empty file:

touch /var/state/dhcp/dhcpd.leases

To start the DHCP server, simply type (or include in the bootup scripts)

/usr/sbin/dhcpd

This will start dhcpd on eth0 device. If you need to start it on another device simply supply it on the command line e.g.

/usr/sbin/dhcpd eth1

If you wish to test the configuration for any oddities you can start dhcpd with the debugging mode. Typing the command below will allow you to see exactly what is going on with the server.

/usr/sbin/dhcpd −d −f

Boot up a client take a look at the console of the server. You will see a number of debugging messages come up.

Your done

7.4. EQL − multiple line traffic equaliser

The EQL device name is `eql'. With the standard kernel source you may have only one EQL device per machine. EQL provides a means of utilizing multiple point to point lines such as PPP, slip or plip as a single logical link to carry tcp/ip. Often it is cheaper to use multiple lower speed lines than to have one high speed line installed.

Kernel Compile Options:

```
        Network device support  --->
            [*] Network device support
            <*> EQL (serial line load balancing) support
```

To support this mechanism the machine at the other end of the lines must also support EQL. Linux, Livingstone Portmasters and newer dial−in servers support compatible facilities.

To configure EQL you will need the eql tools which are available from: metalab.unc.edu.

Configuration is fairly straightforward. You start by configuring the eql interface. The eql interface is just like any other network device. You configure the IP address and mtu using the *ifconfig* utility, so something like:

```
        root# ifconfig eql 192.168.10.1 mtu 1006
```

Next you need to manually initiate each of the lines you will use. These may be any combination of point to point network devices. How you initiate the connections will depend on what sort of link they are, refer to the appropriate sections for further information.

Lastly you need to associate the serial link with the EQL device, this is called `enslaving' and is done with the *eql_enslave* command as shown:

```
        root# eql_enslave eql sl0 28800
        root# eql_enslave eql ppp0 14400
```

The `*estimated speed*' parameter you supply *eql_enslave* doesn't do anything directly. It is used by the EQL driver to determine what share of the datagrams that device should receive, so you can fine tune the balancing of the lines by playing with this value.

To disassociate a line from an EQL device you use the *eql_emancipate* command as shown:

```
        root# eql_emancipate eql sl0
```

You add routing as you would for any other point to point link, except your routes should refer to the `eql` device rather than the actual serial devices themselves, typically you would use:

```
        root# route add default eql
```

The EQL driver was developed by Simon Janes, `simon@ncm.com`.

7.5. IP Accounting (for Linux−2.0)

The IP accounting features of the Linux kernel allow you to collect and analyze some network usage data. The data collected comprises the number of packets and the number of bytes accumulated since the figures were last reset. You may specify a variety of rules to categorize the figures to suit whatever purpose you may have. This option has been removed in kernel 2.1.102, because the old ipfwadm−based firewalling was replaced by ``ipfwchains''.

Kernel Compile Options:

```
        Networking options  --->
            [*] IP: accounting
```

After you have compiled and installed the kernel you need to use the *ipfwadm* command to configure IP accounting. There are many different ways of breaking down the accounting information that you might choose. I've picked a simple example of what might be useful to use, you should read the *ipfwadm* man page for more information.

Scenario: You have a ethernet network that is linked to the internet via a PPP link. On the ethernet you have a machine that offers a number of services and that you are interested in knowing how much traffic is generated by each of ftp and world wide web traffic, as well as total tcp and udp traffic.

You might use a command set that looks like the following, which is shown as a shell script:

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

```
        # Flush the accounting rules
        ipfwadm −A −f
        #
        # Set shortcuts
        localnet=44.136.8.96/29
        any=0/0
        # Add rules for local ethernet segment
        ipfwadm −A in  −a −P tcp −D $localnet ftp−data
        ipfwadm −A out −a −P tcp −S $localnet ftp−data
        ipfwadm −A in  −a −P tcp −D $localnet www
        ipfwadm −A out −a −P tcp −S $localnet www
        ipfwadm −A in  −a −P tcp −D $localnet
        ipfwadm −A out −a −P tcp −S $localnet
        ipfwadm −A in  −a −P udp −D $localnet
        ipfwadm −A out −a −P udp −S $localnet
        #
        # Rules for default
        ipfwadm −A in  −a −P tcp −D $any ftp−data
        ipfwadm −A out −a −P tcp −S $any ftp−data
        ipfwadm −A in  −a −P tcp −D $any www
        ipfwadm −A out −a −P tcp −S $any www
        ipfwadm −A in  −a −P tcp −D $any
        ipfwadm −A out −a −P tcp −S $any
        ipfwadm −A in  −a −P udp −D $any
        ipfwadm −A out −a −P udp −S $any
        #
        # List the rules
        ipfwadm −A −l −n
        #
```

The names ``ftp−data'' and ``www'' refer to lines in `/etc/services`. The last command lists each of the Accounting rules and displays the collected totals.

An important point to note when analyzing IP accounting is that *totals for all rules that match will be incremented* so that to obtain differential figures you need to perform appropriate maths. For example if I wanted to know how much data was not ftp nor www I would substract the individual totals from the rule that matches all ports.

```
root# ipfwadm −A −l −n
IP accounting rules
 pkts bytes dir prot source               destination          ports
    0     0 in  tcp  0.0.0.0/0            44.136.8.96/29       * −> 20
    0     0 out tcp  44.136.8.96/29       0.0.0.0/0            20 −> *
   10  1166 in  tcp  0.0.0.0/0            44.136.8.96/29       * −> 80
   10   572 out tcp  44.136.8.96/29       0.0.0.0/0            80 −> *
  252 10943 in  tcp  0.0.0.0/0            44.136.8.96/29       * −> *
  231 18831 out tcp  44.136.8.96/29       0.0.0.0/0             * −> *
    0     0 in  udp  0.0.0.0/0            44.136.8.96/29       * −> *
    0     0 out udp  44.136.8.96/29       0.0.0.0/0            * −> *
    0     0 in  tcp  0.0.0.0/0            0.0.0.0/0            * −> 20
    0     0 out tcp  0.0.0.0/0            0.0.0.0/0            20 −> *
   10  1166 in  tcp  0.0.0.0/0            0.0.0.0/0            * −> 80
   10   572 out tcp  0.0.0.0/0            0.0.0.0/0            80 −> *
  253 10983 in  tcp  0.0.0.0/0            0.0.0.0/0            * −> *
```

```
 231 18831 out tcp  0.0.0.0/0            0.0.0.0/0            * -> *
   0     0 in  udp  0.0.0.0/0            0.0.0.0/0            * -> *
   0     0 out udp  0.0.0.0/0            0.0.0.0/0            * -> *
```

7.5.1. IP Accounting (for Linux−2.2)

The new accounting code is accessed via ``IP Firewall Chains''. See the IP chains home page for more information. Among other things, you'll now need to use *ipchains* instead of `ipfwadm` to configure your filters. (From `Documentation/Changes` in the latest kernel sources).

7.6. IP Aliasing

There are some applications where being able to configure multiple IP addresses to a single network device is useful. Internet Service Providers often use this facility to provide a `customized' to their World Wide Web and ftp offerings for their customers. You can refer to the ``IP−Alias mini−HOWTO'' for more information than you find here.

Kernel Compile Options:

```
        Networking options  --->
            ....
            [*] Network aliasing
            ....
            <*> IP: aliasing support
```

After compiling and installing your kernel with IP_Alias support; configuration is very simple. The aliases are added to virtual network devices associated with the actual network device. A simple naming convention applies to these devices being `<devname>:<virtual dev num>`, e.g. `eth0:0`, `ppp0:10` etc. Note that the the ifname:number device can only be configured *after* the main interface has been set up.

For example, assume you have an ethernet network that supports two different IP subnetworks simultaneously and you wish your machine to have direct access to both. You could use something like:

```
        root# ifconfig eth0 192.168.1.1 netmask 255.255.255.0 up
        root# route add -net 192.168.1.0 netmask 255.255.255.0 eth0

        root# ifconfig eth0:0 192.168.10.1 netmask 255.255.255.0 up
        root# route add -net 192.168.10.0 netmask 255.255.255.0 eth0:0
```

To delete an alias you simply add a `−' to the end of its name and refer to it and is as simple as:

```
        root# ifconfig eth0:0− 0
```

All routes associated with that alias will also be deleted automatically.

7.7. IP Firewall (for Linux−2.0)

IP Firewall and Firewalling issues are covered in more depth in the Firewall−HOWTO. IP Firewalling allows you to secure your machine against unauthorized network access by filtering or allowing datagrams from or to IP addresses that you nominate. There are three different classes of rules; incoming filtering, outgoing filtering and forwarding filtering. Incoming rules are applied to datagrams that are received by a network device. Outgoing rules are applied to datagrams that are to be transmitted by a network device. Forwarding rules are applied to datagrams that are received and are not for this machine, ie. datagrams that would be routed.

Kernel Compile Options:

```
        Networking options  --->
            [*] Network firewalls
            ....
            [*] IP: forwarding/gatewaying
            ....
            [*] IP: firewalling
            [ ] IP: firewall packet logging
```

Configuration of the IP firewall rules is performed using the *ipfwadm* command. As I mentioned earlier, security is not something I am expert at, so while I will present an example you can use, you should do your own research and develop your own rules if security is important to you.

Probably the most common use of IP firewall is when you are using your linux machine as a router and firewall gateway to protect your local network from unauthorized access from outside your network.

The following configuration is based on a contribution from Arnt Gulbrandsen, `<agulbra@troll.no>`.

The example describes the configuration of the firewall rules on the Linux firewall/router machine illustrated in this diagram:

```
-                                   -
 \                                  | 172.16.37.0
  \                                 |   /255.255.255.0
   \                 ---------      |
    |  172.16.174.30 | Linux |      |
NET =================|  f/w  |------|    ..37.19
```

```
   |    PPP         | router|      |  --------
  /                ---------       |--| Mail |
 /                                 |  | /DNS |
/                                  |  --------
-                                  -
```

The following commands would normally be placed in an `rc` file so that they were automatically started each time the system boots. For maximum security they would be performed after the network interfaces are configured, but before the interfaces are actually brought up to prevent anyone gaining access while the firewall machine is rebooting.

```
        #!/bin/sh

        # Flush the 'Forwarding' rules table
        # Change the default policy to 'accept'
        #
        /sbin/ipfwadm -F -f
        /sbin/ipfwadm -F -p accept
        #
        # .. and for 'Incoming'
        #
        /sbin/ipfwadm -I -f
        /sbin/ipfwadm -I -p accept

        # First off, seal off the PPP interface
        # I'd love to use '-a deny' instead of '-a reject -y' but then it
        # would be impossible to originate connections on that interface too.
        # The -o causes all rejected datagrams to be logged. This trades
        # disk space against knowledge of an attack of configuration error.
        #
        /sbin/ipfwadm -I -a reject -y -o -P tcp -S 0/0 -D 172.16.174.30

        # Throw away certain kinds of obviously forged packets right away:
        # Nothing should come from multicast/anycast/broadcast addresses
        #
        /sbin/ipfwadm -F -a deny -o -S 224.0/3 -D 172.16.37.0/24
        #
        # and nothing coming from the loopback network should ever be
        # seen on a wire
        #
        /sbin/ipfwadm -F -a deny -o -S 127.0/8 -D 172.16.37.0/24

        # accept incoming SMTP and DNS connections, but only
        # to the Mail/Name Server
        #
        /sbin/ipfwadm -F -a accept -P tcp -S 0/0 -D 172.16.37.19 25 53
        #
        # DNS uses UDP as well as TCP, so allow that too
        # for questions to our name server
        #
        /sbin/ipfwadm -F -a accept -P udp -S 0/0 -D 172.16.37.19 53
        #
        # but not "answers" coming to dangerous ports like NFS and
        # Larry McVoy's NFS extension.  If you run squid, add its port here.
        #
        /sbin/ipfwadm -F -a deny -o -P udp -S 0/0 53 \
```

```
                -D 172.16.37.0/24 2049 2050

        # answers to other user ports are okay
        #
        /sbin/ipfwadm -F -a accept -P udp -S 0/0 53 \
                -D 172.16.37.0/24 53 1024:65535

        # Reject incoming connections to identd
        # We use 'reject' here so that the connecting host is told
        # straight away not to bother continuing, otherwise we'd experience
        # delays while ident timed out.
        #
        /sbin/ipfwadm -F -a reject -o -P tcp -S 0/0 -D 172.16.37.0/24 113

        # Accept some common service connections from the 192.168.64 and
        # 192.168.65 networks, they are friends that we trust.
        #
        /sbin/ipfwadm -F -a accept -P tcp -S 192.168.64.0/23 \
                -D 172.16.37.0/24 20:23

        # accept and pass through anything originating inside
        #
        /sbin/ipfwadm -F -a accept -P tcp -S 172.16.37.0/24 -D 0/0

        # deny most other incoming TCP connections and log them
        # (append 1:1023 if you have problems with ftp not working)
        #
        /sbin/ipfwadm -F -a deny -o -y -P tcp -S 0/0 -D 172.16.37.0/24

        # ... for UDP too
        #
        /sbin/ipfwadm -F -a deny -o -P udp -S 0/0 -D 172.16.37.0/24
```

Good firewall configurations are a little tricky. This example should be a reasonable starting point for you. The *ipfwadm* manual page offers some assistance in how to use the tool. If you intend to configure a firewall, be sure to ask around and get as much advice from sources you consider reliable and get someone to test/sanity check your configuration from the outside.

7.7.1. IP Firewall (for Linux−2.2)

The new firewalling code is accessed via ``IP Firewall Chains''. See the IP chanins home page for more information. Among other things, you'll now need to use *ipchains* instead of `ipfwadm` to configure your filters. (From `Documentation/Changes` in the latest kernel sources).

We are aware that this is a sorely out of date statement and we are currently working on getting this section more current. You can expect a newer version sometime in 2000.

7.8. IPIP Encapsulation

Why would you want to encapsulate IP datagrams within IP datagrams? It must seem an odd thing to do if you've never seen an application of it before. Ok, here are a couple of common places where it is used: Mobile−IP and IP−Multicast. What is perhaps the most widely spread use of it though is also the least well known, Amateur Radio.

Kernel Compile Options:

```
        Networking options  --->
            [*] TCP/IP networking
            [*] IP: forwarding/gatewaying
            ....
            <*> IP: tunneling
```

IP tunnel devices are called `` `tunl0' ``, `` `tunl1' `` etc.

"But why ?". Ok, ok. Conventional IP routing rules mandate that an IP network comprises a network address and a network mask. This produces a series of contiguous addresses that may all be routed via a single routing entry. This is very convenient, but it means that you may use any particular IP address while you are connected to the particular piece of network to which it belongs. In most instances this is ok, but if you are a mobile netizen then you may not be able to stay connected to the one place all the time. IP/IP encapsulation (IP tunneling) allows you to overcome this restriction by allowing datagrams destined for your IP address to be wrapped up and redirected to another IP address. If you know that you're going to be operating from some other IP network for some time you can set up a machine on your home network to accept datagrams to your IP address and redirect them to the address that you will actually be using temporarily.

7.8.1. A tunneled network configuration.

```
 192.168.1/24                          192.168.2/24

     -                                     -
     |      ppp0 =            ppp0 =       |
     |  aaa.bbb.ccc.ddd  fff.ggg.hhh.iii   |
     |                                     |
     |   /-----\                /-----\    |
     |   |     |       //       |     |    |
     |---|  A  |------//--------|  B  |---|
     |   |     |     //         |     |    |
     |   \-----/                \-----/    |
     |                                     |
     -                                     -
```

The diagram illustrates another possible reason to use IPIP encapsulation, virtual private networking. This example presupposes that you have two machines each with a simple dial up internet connection. Each host is allocated just a single IP address. Behind each of these machines are some private local area networks configured with reserved IP network addresses. Suppose that you want to allow any host on network A to connect to any host on network B, just as if they were properly connected to the Internet with a network route. IPIP encapsulation will allow you to do this. Note, encapsulation does not solve the problem of how you get the hosts on networks A and B to talk to any other on the Internet, you still need tricks like IP Masquerade for that. Encapsulation is normally performed by machine functioning as routers.

Linux router `A' would be configured with a script like the following:

```
        #!/bin/sh
        PATH=/sbin:/usr/sbin
        mask=255.255.255.0
        remotegw=fff.ggg.hhh.iii
        #
        # Ethernet configuration
        ifconfig eth0 192.168.1.1 netmask $mask up
        route add −net 192.168.1.0 netmask $mask eth0
        #
        # ppp0 configuration (start ppp link, set default route)
        pppd
        route add default ppp0
        #
        # Tunnel device configuration
        ifconfig tunl0 192.168.1.1 up
        route add −net 192.168.2.0 netmask $mask gw $remotegw tunl0
```

Linux router `B' would be configured with a similar script:

```
        #!/bin/sh
        PATH=/sbin:/usr/sbin
        mask=255.255.255.0
        remotegw=aaa.bbb.ccc.ddd
        #
        # Ethernet configuration
        ifconfig eth0 192.168.2.1 netmask $mask up
        route add −net 192.168.2.0 netmask $mask eth0
        #
        # ppp0 configuration (start ppp link, set default route)
        pppd
        route add default ppp0
        #
        # Tunnel device configuration
        ifconfig tunl0 192.168.2.1 up
        route add −net 192.168.1.0 netmask $mask gw $remotegw tunl0
```

The command:

```
        route add −net 192.168.1.0 netmask $mask gw $remotegw tunl0
```

reads: `Send any datagrams destined for `192.168.1.0/24` inside an IPIP encap datagram with a destination address of `aaa.bbb.ccc.ddd`'.

Note that the configurations are reciprocated at either end. The tunnel device uses the `` `gw `` ' in the route as the *destination* of the IP datagram in which it will place the datagram it has received to route. That machine must know how to decapsulate IPIP datagrams, that is, it must also be configured with a tunnel device.

7.8.2. A tunneled host configuration.

It doesn't have to be a whole network you route. You could for example route just a single IP address. In that instance you might configure the `tunl` device on the `remote' machine with its home IP address and at the A end just use a host route (and Proxy Arp) rather than a network route via the tunnel device. Let's redraw and modify our configuration appropriately. Now we have just host `` `B `` ' which to want to act and behave as if it is both fully connected to the Internet and also part of the remote network supported by host `` `A `` ':

```
192.168.1/24

    -
    |      ppp0 =                ppp0 =
    |  aaa.bbb.ccc.ddd      fff.ggg.hhh.iii
    |
    |   /-----\                 /-----\
    |   |     |       //        |     |
    |---|  A  |------//---------|  B  |
    |   |     |     //          |     |
    |   \-----/                 \-----/
    |                      also: 192.168.1.12
    -
```

Linux router `` `A `` ' would be configured with:

```
        #!/bin/sh
        PATH=/sbin:/usr/sbin
        mask=255.255.255.0
        remotegw=fff.ggg.hhh.iii
        #
        # Ethernet configuration
        ifconfig eth0 192.168.1.1 netmask $mask up
        route add -net 192.168.1.0 netmask $mask eth0
        #
        # ppp0 configuration (start ppp link, set default route)
        pppd
        route add default ppp0
        #
```

```
        # Tunnel device configuration
        ifconfig tunl0 192.168.1.1 up
        route add −host 192.168.1.12 gw $remotegw tunl0
        #
        # Proxy ARP for the remote host
        arp −s 192.168.1.12 xx:xx:xx:xx:xx:xx pub
```

Linux host `B' would be configured with:

```
        #!/bin/sh
        PATH=/sbin:/usr/sbin
        mask=255.255.255.0
        remotegw=aaa.bbb.ccc.ddd
        #
        # ppp0 configuration (start ppp link, set default route)
        pppd
        route add default ppp0
        #
        # Tunnel device configuration
        ifconfig tunl0 192.168.1.12 up
        route add −net 192.168.1.0 netmask $mask gw $remotegwtunl0
```

This sort of configuration is more typical of a Mobile−IP application. Where a single host wants to roam around the Internet and maintain a single usable IP address the whole time. You should refer to the Mobile−IP section for more information on how that is handled in practice.

7.9. IP Masquerade

Many people have a simple dialup account to connect to the Internet. Nearly everybody using this sort of configuration is allocated a single IP address by the Internet Service Provider. This is normally enough to allow only one host full access to the network. IP Masquerade is a clever trick that enables you to have many machines make use of that one IP address, by causing the other hosts to look like the machine supporting the dial−up connection; hence the term masquerade. There is a small caveat; the masquerade function nearly always works only in one direction. That is, the masqueraded hosts can make calls out, but they cannot accept or receive network connections from remote hosts. This means that some network services do not work such as *talk*, and others such as *ftp* must be configured to operate in passive (PASV) mode to operate. Fortunately the most common network services such as *telnet*, World Wide Web and *irc* do work just fine.

Kernel Compile Options:

```
        Code maturity level options  --->
            [*] Prompt for development and/or incomplete code/drivers
        Networking options  --->
            [*] Network firewalls
            ....
```

```
            [*] TCP/IP networking
            [*] IP: forwarding/gatewaying
            ....
            [*] IP: masquerading (EXPERIMENTAL)
```

Normally you have your linux machine supporting a SLIP or PPP dial−up line just as it would if it were a standalone machine. Additionally it would have another network device configured, perhaps an ethernet, configured with one of the reserved network addresses. The hosts to be masqueraded would be on this second network. Each of these hosts would have the IP address of the ethernet port of the linux machine set as their default gateway or router.

A typical configuration might look something like this:

```
-                                   -
 \                                  | 192.168.1.0
  \                                 |   /255.255.255.0
   \                 ---------      |
    |                | Linux | .1.1 |
NET =================| masq  |------|
    |    PPP/slip    | router|      |  --------
   /                 ---------      |--| host |
  /                                 |  |      |
 /                                  |  --------
-                                   -
```

7.9.1. Masquerading with IPFWADM (Kernels 2.0.x)

The most relevant commands for this configuration are:

```
        # Network route for ethernet
        route add −net 192.168.1.0 netmask 255.255.255.0 eth0
        #
        # Default route to the rest of the internet.
        route add default ppp0
        #
        # Cause all hosts on the 192.168.1/24 network to be masqueraded.
        ipfwadm −F −a m −S 192.168.1.0/24 −D 0.0.0.0/0
```

7.9.2. Masquerading with IPCHAINS

This is similar to using IPFWADM, but the command structure has changed:

```
        # Network route for ethernet
        route add −net 192.168.1.0 netmask 255.255.255.0 eth0
        #
        # Default route to the rest of the internet.
        route add default ppp0
        #
        # Cause all hosts on the 192.168.1/24 network to be masqueraded.
        ipchains −A forward −s 192.168.1.0/24 −j MASQ
```

You can get more information on the Linux IP Masquerade feature from the IP Masquerade Resource Page. Also, a *very* detailed document about masquesrading is the ``IP−Masquerade mini−HOWTO'' (which also intructs to configure other OS's to run with a Linux masquerade server).

7.10. IP Transparent Proxy

IP transparent proxy is a feature that enables you to redirect servers or services destined for another machine to those services on this machine. Typically this would be useful where you have a linux machine as a router and also provides a proxy server. You would redirect all connections destined for that service remotely to the local proxy server.

Kernel Compile Options:

```
        Code maturity level options  --->
                [*] Prompt for development and/or incomplete code/drivers
        Networking options  --->
                [*] Network firewalls
                ....
                [*] TCP/IP networking
                ....
                [*] IP: firewalling
                ....
                [*] IP: transparent proxy support (EXPERIMENTAL)
```

Configuration of the transparent proxy feature is performed using the *ipfwadm* command

An example that might be useful is as follows:

```
        root# ipfwadm −I −a accept −D 0/0 telnet −r 2323
```

This example will cause any connection attempts to port `telnet` (23) on any host to be redirected to port 2323 on this host. If you run a service on that port, you could forward telnet connections, log them or do whatever fits your need.

A more interesting example is redirecting all `http` traffic through a local cache. However, the protocol used by proxy servers is different from native http: where a client connects to `www.server.com:80` and asks for `/path/page`, when it connects to the local cache it contacts `proxy.local.domain:8080` and asks for `www.server.com/path/page`.

To filter an `http` request through the local proxy, you need to adapt the protocol by inserting a small server, called `transproxy` (you can find it on the world wide web). You can choose to run `transproxy` on port 8081, and issue this command:

```
root# ipfwadm −I −a accept −D 0/0 80 −r 8081
```

The `transproxy` program, then, will receive all connections meant to reach external servers and will pass them to the local proxy after fixing protocol differences.

7.11. IPv6

Just when you thought you were beginning to understand IP networking the rules get changed! IPv6 is the shorthand notation for version 6 of the Internet Protocol. IPv6 was developed primarily to overcome the concerns in the Internet community that there would soon be a shortage of IP addresses to allocate. IPv6 addresses are 16 bytes long (128 bits). IPv6 incorporates a number of other changes, mostly simplifications, that will make IPv6 networks more managable than IPv4 networks.

Linux already has a working, but not complete, IPv6 implementation in the `2.2.*` series kernels.

7.12. IPv6 Linux resources

IPv6−HOWTO

IPv6 for Linux

Linux IPv6 RPM Project

IPv6 FAQ/HOWTO

7.13. Mobile IP

The term "IP mobility" describes the ability of a host that is able to move its network connection from one point on the Internet to another without changing its IP address or losing connectivity. Usually when an IP host changes its point of connectivity it must also change its IP address. IP Mobility overcomes this problem by allocating a fixed IP address to the mobile host and using IP encapsulation (tunneling) with automatic routing to ensure that datagrams destined for it are routed to the actual IP address it is currently using.

A project is underway to provide a complete set of IP mobility tools for Linux. The Status of the project and tools may be obtained from the: Linux Mobile IP Home Page.

7.14. Multicast

IP Multicast allows an arbitrary number of IP hosts on disparate IP networks to have IP datagrams simultaneously routed to them. This mechanism is exploited to provide Internet wide "broadcast" material such as audio and video transmissions and other novel applications.

Kernel Compile Options:

```
Networking options  --->
        [*] TCP/IP networking
        ....
        [*] IP: multicasting
```

A suite of tools and some minor network configuration is required. Please check the Multicast−HOWTO for more information on Multicast support in Linux.

7.15. Traffic Shaper − Changing allowed bandwidth

The traffic shaper is a driver that creates new interface devices, those devices are traffic−limited in a user−defined way, they rely on physical network devices for actual transmission and can be used as outgoing routed for network traffic.

The shaper was introduced in Linux−2.1.15 and was backported to Linux−2.0.36 (it appeared in `2.0.36−pre−patch−2` distributed by Alan Cox, the author of the shaper device and maintainer of Linux−2.0).

The traffic shaper can only be compiled as a module and is configured by the *shapecfg* program with commands like the following:

```
        shapecfg attach shaper0 eth1
        shapecfg speed shaper0 64000
```

The shaper device can only control the bandwidth of outgoing traffic, as packets are transmitted via the shaper only according to the routing tables; therefore, a ``route by source address'' functionality could help in limiting the overall bandwidth of specific hosts using a Linux router.

Linux−2.2 already has support for such routing, if you need it for Linux−2.0 please check the patch by Mike McLagan, at `ftp.invlogic.com`. Refer to `Documentation`networking/shaper.txt for further information about the shaper.

If you want to try out a (tentative) shaping for incoming packets, try out `rshaper−1.01` (or newer), from ftp.systemy.it.

Chapter 8. Advanced Networking with Kernel 2.2

Kernel 2.2 has advanced the routing capabilities of Linux quite a bit. Unfortunately the documentation for using these new capabilities is almost impossible to find, even if it does exist.

I have put some time into it and have been able to do a little with it. I will add more as I have time and help to figure out what it all means.

In kernel 2.0 and below Linux used the standard *route* command to place routes in a single routing table. If you were to type *netstat −rn* at the Linux prompt you could see and example.

In the newer kernels (2.1 and above) you have another option. This option is rule based and allows you to have multiple routing tables. The new rules allow a great deal of flexibility in deciding how a packet is handled. You can choose between routes based not only on the destination address, but the source address, TOS, or incoming device.

8.1. The Basics

Listing the Routing Table:

ip route

Now on my machine this equates to the following output:

```
207.149.43.62 dev eth0  scope link
207.149.43.0/24 dev eth0  proto kernel  scope link  src 207.149.43.62
default via 207.149.43.1 dev eth0
```

The first line:

207.149.43.62 dev eth0 scope link is the route for the interface

The second line:

207.149.43.0/24 dev eth0 proto kernel scope link src 207.149.43.62 Is the route that says *everything that goes to 207.149.43.0 needs to go out 207.149.43.62*.

The third line:

default via 207.149.43.1 dev eth0 is the default route.

8.1.1. Using the information

Now that we have walked through a basic routing table. Lets see how we use it. First read the Policy routing text. If you get confused, don't worry −− it is a confusing text. It will give you the run down on everything that the new routing code can do.

8.2. Adding a route with the new ip tools

In the previous section we spoke about listing the routing table and what the basics of that listing meant. Well, luckily the output is very similar to the syntax that you would use to implement that exact routing table on your own.

```
ip route add 207.149.43.62 dev eth0  scope link
ip route add 207.149.43.0/24 dev eth0 proto kernel scope link src 207.149.43.62
ip route add 127.0.0.0/8 dev lo  scope link
ip route add default via 207.149.43.1 dev eth0
```

As you can see the output and input are almost exact except for the adding of the *ip route add* in front of the line.

Note: We are aware that the documentation on Routing with 2.2 is sorely lacking. In fact, I think EVERYONE is aware of it. If you have any experience in this please contact us at poet@linuxports.com we would like to get information you have to help further document this.

8.3. Using NAT with Kernel 2.2

The IP Network Address Translation facility is pretty much the standardized big brother of the Linux IP Masquerade facility. It is specified in some detail in RFC−1631 at your nearest RFC archive. NAT provides features that IP−Masquerade does not that make it eminently more suitable for use in corporate firewall router designs and larger scale installations.

An alpha implementation of NAT for Linux 2.0.29 kernel has been developed by Michael.Hasenstein, `Michael.Hasenstein@informatik.tu-chemnitz.de`. Michaels documentation and implementation are available from:

Linux IP Network Address Web Page

The much improved TCP/IP stack of Linux 2.2 kernel has NAT functionality built−in. This facility seems to obsolete the work by Michael Hasenstein (Michael.Hasenstein@informatik.tu−chemnitz.de).

To get it work you need kernel with enabled CONFIG_IP_ADVANCED_ROUTER, CONFIG_IP_MULTIPLE_TABLES (aka policy routing) and CONFIG_IP_ROUTE_NAT (aka fast NAT). Also, if you want to use finer grained NAT rules, you may also want to turn on firewalling (CONFIG_IP_FIREWALL) and CONFIG_IP_ROUTE_FWMARK. To actually operate these kernel features you will need the "ip" program by Alexey Kuznyetsov from ftp://ftp.inr.ac.ru/ip−routing/.

Incoming datagrams NAT

Now, to translate addresses of incoming datagrams, following command is used:

```
          ip route add nat <ext-addr>[/<masklen>] via <int-addr>
```

This will make incoming packet destined to "ext−addr" (the address visible from outside internet) to have its destination address field rewritten to "int−addr" (the address in your internal network, behind your gateway/firewall). The packet is then routed further according local routing table. You can translate either single host addresses, or complete blocks. *Examples:*

```
        ip route add nat 195.113.148.34 via 192.168.0.2
        ip route add nat 195.113.148.32/27 via 192.168.0.0
```

First command will make internal address 192.168.0.2 accessible as 195.113.148.34. The second example shows remapping block 192.168.0.0−31 to 195.113.148.32−63.

Chapter 9. Kernel 2.2 IP Command Reference (Work In Progress)

9.1. ip

If you have the iproute2 tools installed, executing the ip command will allow the basic syntax to be displayed.

```
13;[root@jd Net4]# ip
Usage: ip [ OPTIONS ] OBJECT { COMMAND | help }
where  OBJECT := { link | addr | route | rule | neigh | tunnel |
                   maddr | mroute | monitor }
       OPTIONS := { −V[ersion] | −s[tatistics] | −r[esolve] |
                    −f[amily] { inet | inet6 | dnet | link } | −o[neline] }13;
```

There are also several options available:

−V, −Version −− print the version of the ip utility you are using and exit.

−s, −stats, −statistics −− obtain more output on the speficied device. You can issue this option more than once to increase the amount of information being displayed.

−f, −family followed by a protocol family identifier such as: inet, inet6 or link. −− Specify the exact protocol family to use, inet uses the standard IPv4 (e.g.; current internet standard), inet6 uses IPv6 (ground breadking, never to be implemented internet standard), and link (a physical link). If you do not present the option the protocol family is guessed and if not enough information is present it will fallback to the default setting.

−o, −oneline Show the output each device record in a single line.

−r, −resolve Use the system resolver (e.g.; DNS) to print actual names versus IP numbers.

OBJECT Is the object/device to manage or get information about. The current device types understood by the current implementation are:

- link −− The network device e.g.; eth0 or ppp0 .
- address −− The IP (IP or IPv6) address on the specified device.
- neigh −− The ARP or NDISC cache entry.
- route −− The routing table entry.
- rule −− The rule in routing policy database.
- maddress −− The multicast address.
- mroute −− The multicast route cache entry.
- tunnel −− Whether or not to tunnel over IP.

The amount of possible options allowed on each object type depend on the type of action being taken. As a basic rule it is possible to *add*, *delete*, or to show the object(s), but not all object will allow additional commands to be used. Of course, command help is available for all objects and when used will print out a list of available sytanx conventions for the given object.

If you do not give a command, the default command will be assumed. Typically the default command is to list the objects or if the the objects can not be listed to provide the standard help output.

ARGUMENTS is the list of arguments that can be passed to the command. The number of arguments depend on the command and object being used. There is two types of arguments:

Flags consist of a keyword followed by a value. For convenience each command has some default parameters, which can be left out for easier use. For example, the paramater *dev>* defaults to an *ip link.*

Mistakes... thank god for smart coders. All the operations within the ip commands are dynamic. If the sytanx of the ip utility fails, it will not change the configuration of the system. As always, there is an exception to this rule, the *ip link* command which is used to change part of a device parameters.

It is difficult to list all the error messages (especially, about syntax errors), however as rule their meaning is clear in the context of the command. The most common mistakes are: 1. Netlink is not configured in the kernel. The message is: Cannot open netlink socket: Invalid value

2. RTNETLINK is not configured in the kernel. In this case one of the following messages may be printed, depending on the command: Cannot talk to rtnetlink: Connection refused Cannot send dump request: Connection refused

3. Option CONFIG_IP_MULTIPLE_TABLES was not selected when configuring kernel. In this case any attempt to use commandip rule will fail, e.g.;

jd@home $ ip rule list RTNETLINK error: Invalid argument dump terminated

Chapter 10. Using common PC hardware

10.1. ISDN

The Integrated Services Digital Network (ISDN) is a series of standards that specify a general purpose switched digital data network. An ISDN `call' creates a synchronous point to point data service to the destination. ISDN is generally delivered on a high speed link that is broken down into a number of discrete channels. There are two different types of channels, the `B Channels' which will actually carry the user data and a single channel called the `D channel' which is used to send control information to the ISDN exchange to establish calls and other functions. In Australia for example, ISDN may be delivered on a 2Mbps link that is broken into 30 discrete 64kbps B channels with one 64kbps D channel. Any number of channels may be used at a time and in any combination. You could for example establish 30 separate calls to 30 different destinations at 64kbps each, or you could establish 15 calls to 15 different destinations at 128kbps each (two channels used per call), or just a small number of calls and leave the rest idle. A channel may be used for either incoming or outgoing calls. The original intention of ISDN was to allow Telecommunications companies to provide a single data service which could deliver either telephone (via digitised voice) or data services to your home or business without requiring you to make any special configuration changes.

There are a few different ways to connect your computer to an ISDN service. One way is to use a device called a `Terminal Adaptor' which plugs into the Network Terminating Unit that you telecommunications carrier will have installed when you got your ISDN service and presents a number of serial interfaces. One of those interfaces is used to enter commands to establish calls and configuration and the others are actually connected to the network devices that will use the data circuits when they are established. Linux will work in this sort of configuration without modification, you just treat the port on the Terminal Adaptor like you would treat any other serial device. Another way, which is the way the kernel ISDN support is designed for allows you to install an ISDN card into your Linux machine and then has your Linux software handle the protocols and make the calls itself.

Kernel Compile Options:

```
        ISDN subsystem  --->
                <*> ISDN support
                [ ] Support synchronous PPP
                [ ] Support audio via ISDN
                < > ICN 2B and 4B support
                < > PCBIT-D support
                < > Teles/NICCY1016PC/Creatix support
```

The Linux implementation of ISDN supports a number of different types of internal ISDN cards. These are those listed in the kernel configuration options:

- ICN 2B and 4B
- Octal PCBIT−D
- Teles ISDN−cards and compatibles

Some of these cards require software to be downloaded to them to make them operational. There is a separate utility to do this with.

Full details on how to configure the Linux ISDN support is available from the */usr/src/linux/Documentation/isdn/* directory and an FAQ dedicated to *isdn4linux* is available at www.lrz−muenchen.de. (You can click on the english flag to get an english version).

A note about PPP. The PPP suite of protocols will operate over either asynchronous or synchronous serial lines. The commonly distributed PPP daemon for Linux `*pppd*' supports only asynchronous mode. If you wish to run the PPP protocols over your ISDN service you need a specially modified version. Details of where to find it are available in the documentation referred to above.

10.2. PLIP for Linux−2.0

PLIP device names are `plip0', `plip1 and plip2.

Kernel Compile Options:

```
        Network device support  --->
            <*> PLIP (parallel port) support
```

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 (a cabling diagram in included in the cabling diagram section later in this document). Because it is possible to transfer more than one bit at a time with a parallel port, it is possible to attain higher speeds with the *PLIP* interface than with a standard serial device. In addition, even the simplest of parallel ports, printer ports, can be used in lieu of you having to purchase comparatively expensive 16550AFN UART's for your serial ports. PLIP uses a lot of CPU compared to a serial link and is most certainly not a good option if you can obtain some cheap ethernet cards, but it will work when nothing else is available and will work quite well. You should expect a data transfer rate of about 20 kilobytes per second when a link is running well.

The PLIP device drivers competes with the parallel device driver for the parallel port hardware. If you wish to use both drivers then you should compile them both as modules to ensure that you are able to select which port you want to use for PLIP and which ports you want for the printer driver. Refer to the ``Modules mini−HOWTO'' for more information on kernel module configuration.

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

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

In the 2.0.* series kernel the PLIP devices are mapped to i/o port and IRQ as follows:

```
        device  i/o     IRQ
        ------  -----   ---
        plip0   0x3bc   5
```

```
plip1   0x378   7
plip2   0x278   2
```

If your parallel ports don't match any of the above combinations then you can change the IRQ of a port using the *ifconfig* command using the `irq' parameter (be sure to enable IRQ's on your printer ports in your ROM BIOS if it supports this option). As an alternative, you can specify ``io='' annd ``irq='' options on the *insmod* command line, if you use modules. For example:

```
root# insmod plip.o io=0x288 irq=5
```

PLIP operation is controlled by two timeouts, whose default values 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. A program called *plipconfig* exists that allows you to change these timer settings without recompiling your kernel. It is supplied with many Linux distributions.

To configure a *PLIP* interface, you will need to invoke the following commands (or *add* them to your initialization scripts):

```
root# /sbin/ifconfig plip1 localplip pointopoint remoteplip
root# /sbin/route add remoteplip plip1
```

Here, the port being used is the one at I/O address 0x378; *localplip* amd *remoteplip* are the names or IP addresses used over the PLIP cable. I personally keep them in my `/etc/hosts` database:

```
# plip entries
192.168.3.1   localplip
192.168.3.2   remoteplip
```

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

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

Further information on PLIP may be obtained from the ``PLIP mini−HOWTO''.

10.2.1. PLIP for Linux−2.2

During development of the 2.1 kernel versions, support for the parallel port was changed to a better setup.

Kernel Compile Options:

```
        General setup  --->
            [*] Parallel port support
        Network device support  --->
            <*> PLIP (parallel port) support
```

The new code for PLIP behaves like the old one (use the same *ifconfig* and *route* commands as in the previous section, but initialization of the device is different due to the advanced parallel port support.

The ``first'' PLIP device is always called ``plip0'', where first is the first device detected by the system, similarly to what happens for Ethernet devices. The actual parallel port being used is one of the available ports, as shown in `/proc/parport`. For example, if you have only one parallel port, you'll only have a directory called `/proc/parport/0`.

If your kernel didn't detect the IRQ number used by your port, ``insmod plip'' will fail; in this case just write the right number to `/proc/parport/0/irq` and reinvoke *insmod*.

Complete information about parallel port management is available in the file `Documentation/parport.txt`, part of your kernel sources.

10.3. PPP

Do to the nature of PPP, its size, complexity, and flexbility it has been moved to its own HOWTO. The PPP−HOWTO is still a Linux Documentation Project document but its official home is at the LinuxPorts.Com website PPP section.

10.4. SLIP client − (Antiquated)

SLIP devices are named `` `sl0' ``, `` `sl1' `` etc. with the first device configured being assigned `` `0' `` and the rest incrementing sequentially as they are configured.

Kernel Compile Options:

```
        Network device support  --->
            [*] Network device support
            <*> SLIP (serial line) support
            [ ]  CSLIP compressed headers
            [ ]  Keepalive and linefill
            [ ]  Six bit SLIP encapsulation
```

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

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

10.4.1. dip

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

You can find it at: metalab.unc.edu.

To install it, try the following:

```
        user% tar xvzf dip337o−uri.tgz
        user% cd dip−3.3.7o
        user% vi Makefile
        root# make install
```

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

10.4.2. slattach

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

10.4.3. When do I use which ?

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

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

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

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

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

10.4.4. Static SLIP server with a dialup line and DIP.

A static SLIP server is one in which you have been supplied an IP address that is exclusively yours. Each time you connect to the server, you will configure your SLIP port with that address. The static SLIP server will answer your modem call, possibly prompt you for a username and password, and then route any datagrams destined for your address to you via that connection. If you have a static server, then you may want to put entries for your hostname and IP address (since you know what it will be) into your `/etc/hosts`. You should also configure some other files such as: `rc.inet2`, `host.conf`, `resolv.conf`, `/etc/HOSTNAME` and `rc.local`. Remember that when configuring `rc.inet1`, you don't need to add any special commands for your SLIP connection since it is *dip* that does all of the hard work for you in configuring your interface. You will need to give *dip* the appropriate information and it will configure the interface for you after commanding the modem to establish the call and logging you into your SLIP server.

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

10.4.5. Dynamic SLIP server with a dialup line and DIP.

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

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

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

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

10.4.6. Using DIP.

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

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

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

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

main:
# Next, set up the other side's name and address.
# My dialin machine is called 'xs4all.hacktic.nl' (== 193.78.33.42)
get $remote xs4all.hacktic.nl
```

```
# Set netmask on sl0 to 255.255.255.0
netmask 255.255.255.0
# Set the desired serial port and speed.
port cua02
speed 38400

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

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

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

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

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

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

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

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

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

prompt_error:
```

```
print TIME−OUT waiting for sliplogin to fire up...
goto error

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

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

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

exit:
exit
```

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

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

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

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

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

10.4.7. Permanent SLIP connection using a leased line and slattach.

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

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

```
        #
        # Attach a leased line static SLIP connection
        #
        #  configure /dev/cua0 for 19.2kbps and cslip
        /sbin/slattach -p cslip -s 19200 /dev/cua0 38;
        /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 allocates the first unallocated SLIP device to the serial device specified. *slattach* starts with *sl0*. Therefore the first *slattach* command attaches SLIP device *sl0* to the serial device specified and *sl1* the next time, etc.

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

10.4.8. SLIP server.

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

10.4.9. Slip Server using *sliplogin*.

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

The caller will login as per the standard login process, entering their username and password, but instead of being presented with a shell after their login, *sliplogin* is executed which searches its configuration file (`/etc/slip.hosts`) for an entry with a login name that matches that of the caller. If it locates one, it configures the line as an 8bit clean line, and uses an *ioctl* call to convert the line discipline to SLIP. When this process is complete, the last stage of configuration takes place, where *sliplogin* invokes a shell script which configures the SLIP interface with the relevant ip address, netmask and sets appropriate routing in place. This script is usually called `/etc/slip.login`, but in a similar manner to *getty*, if you have certain callers that require special initialization, then you can create configuration scripts called `/etc/slip.login.loginname` that will be run instead of the default specifically for them.

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

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

10.4.10. Where to get *sliplogin*

You may already have the *sliplogin* package installed as part of your distribution, if not then *sliplogin* can be obtained from: metalab.unc.edu. The tar file contains both source, precompiled binaries and a *man* page.

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

```
 ..
slip::13:radio,fred
 ..
```

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

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

```
# cd /usr/src
# gzip −dc .../sliplogin−2.1.1.tar.gz | tar xvf −
# cd sliplogin−2.1.1
# <..edit the Makefile if you don't use shadow passwords..>
# make install
```

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

10.4.11. Configuring `/etc/passwd` for Slip hosts.

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

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

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

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

10.4.12. Configuring `/etc/slip.hosts`

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

```
#
Sradio   44.136.8.99   44.136.8.100  255.255.255.0  normal      −1
Salbert  44.136.8.99   DYNAMIC       255.255.255.0  compressed  60
#
```

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

1. The login name of the caller.
2. The ip address of the server machine, ie this machine.
3. The ip address that the caller will be assigned. If this field is coded `DYNAMIC` then an ip address will be allocated based on the information contained in your `/etc/slip.tty` file discussed later. *Note:* you must be using at least version 1.3 of sliplogin for this to work.
4. The netmask assigned to the calling machine in dotted decimal notation eg 255.255.255.0 for a Class C network mask.
5. The slip mode setting which allows you to enable/disable compression and slip other features. Allowable values here are "`normal`" or "`compressed`".
6. A timeout parameter which specifies how long the line can remain idle (no datagrams received) before the line is automatically disconnected. A negative value disables this feature.
7. Optional arguments.

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

The most common slip modes are:

normal

to enable normal uncompressed SLIP.

compressed

to enable van Jacobsen header compression (cSLIP)

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

10.4.13. Configuring the `/etc/slip.login` file.

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

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

```
#!/bin/sh –
#
#       @(#)slip.login  5.1 (Berkeley) 7/1/90
#
# generic login file for a SLIP line.  sliplogin invokes this with
# the parameters:
#     $1       $2       $3    $4, $5, $6 ...
```

```
#   SLIPunit ttyspeed   pid   the arguments from the slip.host entry
#
/sbin/ifconfig $1 $5 pointopoint $6 mtu 1500 -trailers up
/sbin/route add $6
arp -s $6 <hw_addr> pub
exit 0
#
```

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

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

10.4.14. Configuring the `/etc/slip.logout` file.

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

```
        #!/bin/sh -
        #
        #               slip.logout
        #
        /sbin/ifconfig $1 down
        arp -d $6
        exit 0
        #
```

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

10.4.15. Configuring the `/etc/slip.tty` file.

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

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

Its format is as follows:

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

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

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

10.4.16. Slip Server using *dip*.

Let me start by saying that some of the information below came from the *dip* man pages, where how to run Linux as a SLIP server is briefly documented. Please also beware that the following has been based on the *dip337o−uri.tgz* package and probably will not apply to other versions of *dip*.

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

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

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

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

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

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

```
Sfredm:ij/SMxiTlGVCo:1004:10:Fred:/tmp:/usr/sbin/diplogin
^^         ^^        ^^  ^^   ^^   ^^   ^^
|          |         |   |    |    |    \__ diplogin as login shell
|          |         |   |    |    \_______ Home directory
|          |         |   |    \____________ User Full Name
|          |         |   \_________________ User Group ID
|          |         \_____________________ User ID
|          \_______________________________ Encrypted User Password
\__________________________________________ Slip User Login Name
```

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

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

10.4.17. Configuring `/etc/diphosts`

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

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

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

The fields are:

1. `Login name`: as returned by getpwuid(getuid()) or tty name.
2. `Unused`: compat. with passwd
3. `Remote Address`: IP address of the calling host, either numeric or by name
4. `Local Address`: IP address of this machine, again numeric or by name
5. `Netmask`: in dotted decimal notation
6. `Comment field`: put whatever you want here.
7. `Protocol`: Slip, CSlip etc.
8. `MTU`: decimal number

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

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

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

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

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

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

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

10.4.18. SLIP server using the *dSLIP* package.

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

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

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

apollo.west.oic.com

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

or from:

metalab.unc.edu

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

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

Chapter 11. Other Network Technologies

The following subsections are specific to particular network technologies. The information contained in these sections does not necessarily apply to any other type of network technology. The topics are sorted alphabetically.

11.1. ARCNet

ARCNet device names are `arc0e', `arc1e', `arc2e' etc. or `arc0s', `arc1s', `arc2s' etc. The first card detected by the kernel is assigned `arc0e' or `arc0s' and the rest are assigned sequentially in the order they are detected. The letter at the end signifies whether you've selected ethernet encapsulation packet format or RFC1051 packet format.

Kernel Compile Options:

```
        Network device support  --->
            [*] Network device support
            <*> ARCnet support
            [ ]   Enable arc0e (ARCnet "Ether-Encap" packet format)
            [ ]   Enable arc0s (ARCnet RFC1051 packet format)
```

Once you have your kernel properly built to support your ethernet card then configuration of the card is easy.

Typically you would use something like:

```
        root# ifconfig arc0e 192.168.0.1 netmask 255.255.255.0 up
        root# route add -net 192.168.0.0 netmask 255.255.255.0 arc0e
```

Please refer to the /usr/src/linux/Documentation/networking/arcnet.txt and /usr/src/linux/Documentation/networking/arcnet-hardware.txt files for further information.

ARCNet support was developed by Avery Pennarun, apenwarr@foxnet.net.

11.2. Appletalk (AF_APPLETALK)

The Appletalk support has no special device names as it uses existing network devices.

Kernel Compile Options:

```
        Networking options  --->
            <*> Appletalk DDP
```

Appletalk support allows your Linux machine to interwork with Apple networks. An important use for this is to share resources such as printers and disks between both your Linux and Apple computers. Additional software is required, this is called *netatalk*. Wesley Craig `netatalk@umich.edu` represents a team called the `Research Systems Unix Group' at the University of Michigan and they have produced the *netatalk* package which provides software that implements the Appletalk protocol stack and some useful utilities. The *netatalk* package will either have been supplied with your Linux distribution, or you will have to ftp it from its home site at the University of Michigan

To build and install the package do something like:

```
        user% tar xvfz .../netatalk-1.4b2.tar.Z
        user% make
        root# make install
```

You may want to edit the `Makefile' before calling *make* to actually compile the software. Specifically, you might want to change the DESTDIR variable which defines where the files will be installed later. The default of /usr/local/atalk is fairly safe.

11.2.1. Configuring the Appletalk software.

The first thing you need to do to make it all work is to ensure that the appropriate entries in the `/etc/services` file are present. The entries you need are:

```
  rtmp  1/ddp   # Routing Table Maintenance Protocol
  nbp   2/ddp   # Name Binding Protocol
  echo  4/ddp   # AppleTalk Echo Protocol
  zip   6/ddp   # Zone Information Protocol
```

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

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

```
eth0
```

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

11.2.2. Exporting a Linux filesystems via Appletalk.

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

To do this you must configure the `/usr/local/atalk/etc/AppleVolumes.system` file. There is another configuration file called `/usr/local/atalk/etc/AppleVolumes.default` which has exactly the same format and describes which filesystems users connecting with guest privileges will receive.

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

A simple example might look like:

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

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

11.2.3. Sharing your Linux printer across Appletalk.

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

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

A sample configuration might look like:

```
TricWriter:\
```

```
    :pr=lp:op=cg:
```

Which would make a printer named `TricWriter' available to your Appletalk network and all accepted jobs would be printed to the linux printer ``lp`' (as defined in the `/etc/printcap` file) using *lpd*. The entry ``op=cg`' says that the linux user ``cg`' is the operator of the printer.

11.2.4. Starting the appletalk software.

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

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

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

11.2.5. Testing the appletalk software.

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

11.2.6. Caveats of the appletalk software.

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

information is available in the `/proc/net/` directory if you need it.

11.2.7. More information

For a much more detailed description of how to configure Appletalk for Linux refer to Anders Brownworth *Linux Netatalk−HOWTO* page at thehamptons.com.

11.3. ATM

Werner Almesberger <`werner.almesberger@lrc.di.epfl.ch`> is managing a project to provide Asynchronous Transfer Mode support for Linux. Current information on the status of the project may be obtained from: lrcwww.epfl.ch.

11.4. AX25 (`AF_AX25`)

AX.25 device names are \``sl0`', \``sl1`', etc. in `2.0.*` kernels or \``ax0`', \``ax1`', etc. in `2.1.*` kernels.

Kernel Compile Options:

```
        Networking options  --->
            [*] Amateur Radio AX.25 Level 2
```

The AX25, Netrom and Rose protocols are covered by the AX25−HOWTO. These protocols are used by Amateur Radio Operators world wide in packet radio experimentation.

Most of the work for implementation of these protocols has been done by Jonathon Naylor, `jsn@cs.nott.ac.uk`.

11.5. DECNet

Support for DECNet is currently being worked on. You should expect it to appear in a late `2.1.*` kernel.

11.6. FDDI

FDDI device names are \``fddi0`', \``fddi1`', \``fddi2`' etc. The first card detected by the kernel is assigned \``fddi0`' and the rest are assigned sequentially in the order they are detected.

Larry Stefani, `lstefani@ultranet.com`, has developed a driver for the Digital Equipment Corporation FDDI EISA and PCI cards.

Kernel Compile Options:

```
        Network device support  --->
            [*] FDDI driver support
            [*] Digital DEFEA and DEFPA adapter support
```

When you have your kernel built to support the FDDI driver and installed, configuration of the FDDI interface is almost identical to that of an ethernet interface. You just specify the appropriate FDDI interface name in the *ifconfig* and *route* commands.

11.7. Frame Relay

The Frame Relay device names are \``dlci00`', \``dlci01`' etc for the DLCI encapsulation devices and \``sdla0`', \``sdla1`' etc for the FRAD(s).

Frame Relay is a new networking technology that is designed to suit data communications traffic that is of a \`bursty' or intermittent nature. You connect to a Frame Relay network using a Frame Relay Access Device (FRAD). The Linux Frame Relay supports IP over Frame Relay as described in RFC−1490.

Kernel Compile Options:

```
        Network device support  --->
            <*> Frame relay DLCI support (EXPERIMENTAL)
            (24)   Max open DLCI
            (8)   Max DLCI per device
            <*>   SDLA (Sangoma S502/S508) support
```

11.8. IPX (`AF_IPX`)

The IPX protocol is most commonly utilized in Novell NetWare(tm) local area network environments. Linux includes support for this protocol and may be configured to act as a network endpoint, or as a router for IPX.

Kernel Compile Options:

```
        Networking options  --->
            [*] The IPX protocol
            [ ] Full internal IPX network
```

The IPX protocol and the NCPFS are covered in greater depth in the IPX−HOWTO.

11.9. NetRom (AF_NETROM)

NetRom device names are `nr0', `nr1', etc.

Kernel Compile Options:

```
        Networking options  --->
            [*] Amateur Radio AX.25 Level 2
            [*] Amateur Radio NET/ROM
```

The AX25, Netrom and Rose protocols are covered by the AX25−HOWTO. These protocols are used by Amateur Radio Operators world wide in packet radio experimentation.

Most of the work for implementation of these protocols has been done by Jonathon Naylor, jsn@cs.nott.ac.uk.

11.10. Rose protocol (AF_ROSE)

Rose device names are `rs0', `rs1', etc. in 2.1.* kernels. Rose is available in the 2.1.* kernels.

Kernel Compile Options:

```
        Networking options  --->
            [*] Amateur Radio AX.25 Level 2
            <*> Amateur Radio X.25 PLP (Rose)
```

The AX25, Netrom and Rose protocols are covered by the AX25−HOWTO. These protocols are used by Amateur Radio Operators world wide in packet radio experimentation.

Most of the work for implementation of these protocols has been done by Jonathon Naylor, jsn@cs.nott.ac.uk.

11.11. SAMBA − `NetBEUI', `NetBios', `CIFS' support.

SAMBA is an implementation of the Session Management Block protocol. Samba allows Microsoft and other systems to mount and use your disks and printers.

SAMBA and its configuration are covered in detail in the SMB−HOWTO.

11.12. STRIP support (Starmode Radio IP)

STRIP device names are `st0', `st1', etc.

Kernel Compile Options:

```
        Network device support  --->
                [*] Network device support
                ....
                [*] Radio network interfaces
                < > STRIP (Metricom starmode radio IP)
```

STRIP is a protocol designed specifically for a range of Metricom radio modems for a research project being conducted by Stanford University called the MosquitoNet Project. There is a lot of interesting reading here, even if you aren't directly interested in the project.

The Metricom radios connect to a serial port, employ spread spectrum technology and are typically capable of about 100kbps. Information on the Metricom radios is available from the: Metricom Web Server.

At present the standard network tools and utilities do not support the STRIP driver, so you will have to download some customized tools from the MosquitoNet web server. Details on what software you need is available at the: MosquitoNet STRIP Page.

A summary of configuration is that you use a modified *slattach* program to set the line discipline of a serial tty device to STRIP and then configure the resulting `st[0−9]' device as you would for ethernet with one important exception, for technical reasons STRIP does not support the ARP protocol, so you must manually configure the ARP entries for each of the hosts on your subnet. This shouldn't prove too onerous.

11.13. Token Ring

Token ring device names are `tr0', `tr1' etc. Token Ring is an IBM standard LAN protocol that avoids collisions by providing a mechanism that allows only one station on the LAN the right to transmit at a time. A `token' is held by one station at a time and the station holding the token is the only station allowed to transmit. When it has transmitted its data it passes the token onto the next station. The token loops amongst all active stations, hence the name `Token Ring'.

Kernel Compile Options:

```
        Network device support  --->
                [*] Network device support
                ....
                [*] Token Ring driver support
                < > IBM Tropic chipset based adaptor support
```

Configuration of token ring is identical to that of ethernet with the exception of the network device name to configure.

11.14. X.25

X.25 is a circuit based packet switching protocol defined by the C.C.I.T.T. (a standards body recognized by Telecommunications companies in most parts of the world). An implementation of X.25 and LAPB are being worked on and recent 2.1.* kernels include the work in progress.

Jonathon Naylor jsn@cs.nott.ac.uk is leading the development and a mailing list has been established to discuss Linux X.25 related matters. To subscribe send a message to: majordomo@vger.rutgers.edu with the text "subscribe linux−x25" in the body of the message.

Early versions of the configuration tools may be obtained from Jonathon's ftp site at ftp.cs.nott.ac.uk.

11.15. WaveLan Card

Wavelan device names are `eth0', `eth1', etc.

Kernel Compile Options:

```
Network device support  --->
        [*] Network device support
        ....
        [*] Radio network interfaces
        ....
        <*> WaveLAN support
```

The WaveLAN card is a spread spectrum wireless lan card. The card looks very like an ethernet card in practice and is configured in much the same way.

You can get information on the Wavelan card from Wavelan.com.

Chapter 12. Cables and Cabling

Those of you handy with a soldering iron may want to build your own cables to interconnect two linux machines. The following cabling diagrams should assist you in this.

12.1. Serial NULL Modem cable

Not all NULL modem cables are alike. Many null modem cables do little more than trick your computer into thinking all the appropriate signals are present and swap transmit and receive data. This is ok but means that you must use software flow control (XON/XOFF) which is less efficient than hardware flow control. The following cable provides the best possible signalling between machines and allows you to use hardware (RTS/CTS) flow control.

```
Pin Name  Pin                              Pin
Tx Data    2  -----------------------------  3
Rx Data    3  -----------------------------  2
RTS        4  -----------------------------  5
CTS        5  -----------------------------  4
Ground     7  -----------------------------  7
DTR        20 -\---------------------------  8
DSR        6  -/
RLSD/DCD   8  ---------------------------/-  20
                                         \-  6
```

12.2. Parallel port cable (PLIP cable)

If you intend to use the PLIP protocol between two machines then this cable will work for you irrespective of what sort of parallel ports you have installed.

```
Pin Name    pin            pin
STROBE      1*
D0->ERROR   2  ----------- 15
D1->SLCT    3  ----------- 13
D2->PAPOUT  4  ----------- 12
D3->ACK     5  ----------- 10
D4->BUSY    6  ----------- 11
D5          7*
D6          8*
D7          9*
ACK->D3     10 ----------- 5
BUSY->D4    11 ----------- 6
PAPOUT->D2  12 ----------- 4
SLCT->D1    13 ----------- 3
FEED        14*
ERROR->D0   15 ----------- 2
INIT        16*
SLCTIN      17*
GROUND      25 ----------- 25
```

Notes:

- Do not connect the pins marked with an asterisk `*'.
- Extra grounds are 18,19,20,21,22,23 and 24.
- If the cable you are using has a metallic shield, it should be connected to the metallic DB−25 shell at *one end only*.

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

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

12.3. 10base2 (thin coax) Ethernet Cabling

10base2 is an ethernet cabling standard that specifies the use of 50 ohm coaxial cable with a diameter of about 5 millimeters. There are a couple of important rules to remember when interconnecting machines with 10base2 cabling. The first is that you must use terminators at *both ends* of the cabling. A terminator is a 50 ohm resistor that helps to ensure that the signal is absorbed and not reflected when it reaches the end of the cable. Without a terminator at each end of the cabling you may find that the ethernet is unreliable or doesn't work at all. Normally you'd use `T pieces' to interconnect the machines, so that you end up with something that looks like:

```
13; |==========T=============T=============T=========T==========|
           |             |             |         |
           |             |             |         |
         -----         -----         -----     -----
         |   |         |   |         |   |     |   |
         -----         -----         -----     -----13;
```

where the `|' at either end represents a terminator, the `======' represents a length of coaxial cable with BNC plugs at either end and the `T' represents a `T piece' connector. You should keep the length of cable between the `T piece' and the actual ethernet card in the PC as short as possible, ideally the `T piece' will be plugged directly into the ethernet card.

12.4. Twisted Pair Ethernet Cable

If you have only two twisted pair ethernet cards and you wish to connect them you do not require a hub. You can cable the two cards directly together. A diagram showing how to do this is included in the Ethernet−HOWTO

Chapter 13. Glossary of Terms used in this document.

The following is a list of some of the most important terms used in this document.

ARP

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

ATM

This is an acronym for *Asynchronous Transfer Mode*. An ATM network packages data into standard size blocks which it can convey efficiently from point to point. ATM is a circuit switched packet network technology.

Client

This is usually the piece of software at the end of a system where the user is. There are exceptions to this, for example, in the X11 window system it is actually the server with the user and the client runs on the remote machine. The client is the program or end of a system that is receiving the service provided by the server. In the case of *peer to peer* systems such as *slip* or *ppp* the client is taken to be the end that initiates the connection and the remote end, being called, is taken to be the server.

Datagram

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

DLCI

The DLCI is the Data Link Connection Identifier and is used to identify a unique virtual point to point connection via a Frame Relay network. The DLCI's are normally assigned by the Frame Relay network provider.

Frame Relay

Frame Relay is a network technology ideally suited to carrying traffic that is of bursty or sporadic nature. Network costs are reduced by having many Frame Relay customer sharing the same network capacity and relying on them wanting to make use of the network at slightly different times.

Hardware address

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

ISDN

This is an acronym for *Integrated Services Digital Network*. ISDN provides a standardized means by which Telecommunications companies may deliver either voice or data information to a customers premises. Technically ISDN is a circuit switched data network.

ISP

This is an acronym of Internet Service Provider. These are organizations or companies that provide people with network connectivity to the Internet.

IP address

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

MSS

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

MTU

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

Route

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

Server

This is usually the piece of software or end of a system remote from the user. The server provides some service to one or many clients. Examples of servers include *ftp*, *Networked File System*, or *Domain Name Server*. In the case of *peer to peer* systems such as *slip* or *ppp* the server is taken to be the end of the link that is called and the end calling is taken to be the client.

Window

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

Chapter 14. Authors

14.1. Current

Joshua D. Drake

14.2. Past

Terry Dawson Alessandro Rubini

Chapter 15. Copyright.

Copyright Information