Linux IPX−HOWTO

Table of Contents

Table of Contents

Linux IPX−HOWTO

Kevin Thorpe, `kevin@pricetrak.com`

v2.3, 06 May 1998

This document aims to describe how to obtain, install and configure various tools available for the the Linux operating system that use the Linux kernel IPX protocol support.

7.Greg Pages IPX tools.

8.Configuring your Linux machine as an IPX router.

9.Configuring your Linux machine as an NCP client.

10.Configuring your Linux machine as an NCP server.

11.Configuring your Linux machine as a Novell Print Client.

12.Configuring your Linux machine as a Novell Print Server.

13.An overview of the *ncpfs* user and adminstration commands

14.Configuring PPP for IPX support.

15.IPX tunnel over IP

16.Commercial IPX support for Linux.

17.Some Frequently Asked Questions

18.Copyright Message.

19.Miscellaneous and Acknowledgements.

1.Introduction.

This is the Linux IPX−HOWTO. You should read the Linux NET−3−HOWTO in conjunction with this document.

1.1 Changes from the previous release.

```
Change of author:
        Many thanks to Terry Dawson for passing on this document and
        congratulations on becoming a father :-).

Additions:
        Addition of a brief explanation of IPX. This is in response to
```

```
        many baffled queries on the discussion lists.

Corrections/Updates:
        New version of ncpfs which now supports NDS logins. This is early
        beta test and may be prohibited in your country due to the use of
        patented technology.

        Addition of support for trustee rights in mars_nwe. This is still
        in beta test.
```

1.2 Introduction.

The Linux Kernel has a completely new network implementation as compared to other Unix like operating systems. The ability to take a fresh approach to developing the kernel networking software has led to the Linux kernel having support for a range of non tcp/ip protocols being built. The IPX protocol is one of those that have been included.

The Linux kernel supports the IPX protocol only. It does not yet support protocols such as IPX/RIP, SAP or NCP, these are supported by other software such as that documented elsewhere in this document.

The IPX support was originally developed by Alan Cox `<alan@lxorguk.ukuu.org.uk>` and has been significantly enhanced by Greg Page `<greg@caldera.com>`.

2.Disclaimer.

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

In no way do I or the authors of the software in this document offer protection against your own actions. If you configure this software, even as described in this document and it causes problems on your network then you alone must carry the responsibility. I include this warning because IPX network design and configuration is not always a simple matter and sometimes undesirable interaction with other routers and fileservers can result if you do not design or configure your network carefully. I also include this warning because I was asked to by someone unfortunate enough to have discovered this lesson the hard way.

3.Related Documentation.

This document presumes you understand how to build a Linux kernel with the appropriate networking options selected and that you understand how to use the basic network tools such as *ifconfig* and *route*. If you do not, then you should read the NET−3−HOWTO in conjunction with this document as it describes these.

Other Linux HOWTO documents that might be useful are:

The Ethernet−HOWTO, which describes the details of configuring an Ethernet device for Linux.

The PPP−HOWTO as IPX support is available for version 2.2.0d and later of the Linux PPP implementation.

3.1 New versions of this document.

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

`ftp`:/sunsite.unc.edu/pub/Linux/docs/HOWTO/IPX−HOWTO>/ or:
`ftp`:/sunsite.unc.edu/pub/Linux/docs/HOWTO/other−formats/IPX−HOWTO{−html.tar,ps,dvi}.gz>/ via the World Wide Web from the Linux Documentation Project Web Server, at page: IPX−HOWTO or directly from me, `<kevin@pricetrak.com>`. It may also be posted to the newsgroups: `comp.os.linux.networking`, `comp.os.linux.answers` and `news.answers` from time to time.

3.2 Feedback.

Please send any comments, updates, or suggestions to me, `<kevin@pricetrak.com>`. The sooner I get feedback, the sooner I can update and correct this document. If you find any problems with it, please mail me directly as I can miss info posted to the newsgroups.

3.3 Mailing list support.

There is a mailing list established for discussion of the various Linux IPX software packages described in this document. You can subscribe to it by sending a mail message to `` `listserv@sh.cvut.cz' `` with `` `add linware' `` in the body of the message. To post to the list your send your mail to `` `linware@sh.cvut.cz' ``. I regularly watch this list.

The mailing list is archived at www.kin.vslib.cz.

4.Some of the terms used in this document.

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

client

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

server

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

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

Other terms you will see are:

Bindery

The *bindery* is a specialised database storing network configuration information on a Novell fileserver. Netware clients may query the *bindery* to obtain information on available services, routing and user information.

Frame Type

is a term used to describe that actual protocol used to carry the IPX (and IP) datagrams across your ethernet style network segments. There are four common ones. They are:

Ethernet_II

This is a refined version of the original DIX ethernet standard. Novell has been allocated a formal protocol id and this means that both IPX and IP can coexist happily in an Ethernet_II environment quite happily. This is commonly used in Novell environments and is a good choice.

802.3

This is an I.E.E.E. protocol defining a Carrier Sense Multiple Access with Collision Detection (CSMA/CD) mechanism. It was based on the original DIX Ethernet standard, with an important modification, the type (protocol id) field was converted into a length field instead. It is for this reason that IPX really shouldn't be run here. IEEE 802.3 was designed to carry IEEE 802.2 frames **only** but there are implementations that use it to carry IPX frames directly and remarkably it does work. Avoid it unless you are trying to interwork with a network already configured to use it.

802.2

This is an I.E.E.E. protocol that defines a set of Logical Link Control procedures. It provides a simplistic way of allowing different protocols to coexist, but is quite limited in this respect. Novell uses an unofficial Service Address Point (like a protocol id) but since everyone else uses it as well, that hasn't yet presented too much of a problem.

SNAP

SNAP is the Sub Network Access Protocol. This protocol is designed ride on top of 802.3

and 802.2. It expands the multiprotocol capability of 802.2 and provides some measure of compatability with existing Ethernet and Ethernet_II frame types.

IPX

Internet Packet eXchange is a protocol used by the Novell corporation to provide internetworking support for their NetWare(tm) product. IPX is similar in functionality to the IP protocol used by the tcp/ip community.

IPX network address

This is a number which uniquely identifies a particular IPX network. The usual notation for this address is in hexadecimal. An example might look like: `0x23a91002`.

IPX Internal network

This is a virtual IPX network. It is virtual because it does not correspond to a physical network. This is used to provide a means of uniquely identifying and addressing a particular IPX host. This is generally only useful to IPX hosts that exist on more than one physical IPX network such as fileservers. The address is coded in the same form as for a physical IPX network.

RIP

Routing Information Protocol is a protocol used to automatically propagate network routes in an IPX network. It is functionally similar to the RIP used within the tcp/ip community.

NCP

NetWare Core Protocol is a networked filesystem protocol designed by the Novell Corporation for their NetWare(tm) product. NCP is functionally similar to the NFS used in the tcp/ip community.

SAP

Service Advertisement Protocol is a protocol designed by the Novell Corporation that is used to advertise network services in a NetWare(tm) environment.

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*. An Ethernet address is generally coded as six hexadecimal values separated by colon characters eg. `00:60:8C:C3:3C:0F`

route

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

5.A brief discussion of IPX network topology

This is a much simplified explanation for people new to IPX. Large networks will probably break lots of the rules explained here. In complex IPX networks the administrator should always be consulted.

IPX networking revolves around a scheme of numbered *networks* unlike IP which places more emphasis on the *interface* addresses. A network is a collection of equipment connected to the same LAN segment and *using the same frame type*. Different frame types on the same LAN segment are treated as seperate networks.

Each network must be allocated a number which is unique across the entire internetwork. This is usually performed by a NetWare(tm) server, but can easily be performed by Linux. IPX clients are given this number by the server when starting, they only require to know the correct frame type.

Routing between networks is usually performed by putting two network cards in a server. This server then runs the RIP protocol which holds a routing table for the internetwork. Periodic broadcasts of this routing table are exchanged between servers. Within a short time each server 'discovers' the topology of the internetwork.

If you only wish to use the services of an existing NetWare server, you can use `ipx_configure` (section 7.1) to automatically define the IPX interfaces by using broadcast queries to look for a server. If this fails, or you wish to provide IPX services, you will need to define the interfaces manually using `ipx_interface` or `mars_nwe`.

6.The IPX related files in the `/proc` filesystem.

There are a number of files related to the Linux IPX support that are located within the `/proc` filesystem. They are:

/proc/net/ipx_interface

This file contains information about the IPX interfaces configured on your machine. These may have been configured manually by command or automatically detected and configured.

/proc/net/ipx_route

This file contains a list of the routes that exist in the IPX routing table. These routes may have been added manually by command or automatically by an IPX routing daemon.

/proc/net/ipx

This file is a list of the IPX sockets that are currently open for use on the machine.

7.Greg Pages IPX tools.

Greg Page <`greg@caldera.com` of Caldera Incorporated has written a suite of IPX configuration tools and enhanced the Linux IPX kernel support.

The kernel enhancements allow linux to be configured as a fully featured IPX bridge or router. The enhanced IPX support has already been fed back into the mainstream kernel distribution so you will probably already have it.

The network configuration tools provide you with the capability to configure your network devices to support IPX and allow you to configure IPX routing and other facilities under Linux. The Linux IPX network tools are available from: sunsite.unc.edu.

7.1 The IPX tools in more detail.

ipx_interface

This command is used to manually add, delete or check ipx capability to an existing network device. Normally the network device would be an Ethernet device such at `eth0`. At least one IPX interface must be designated the *primary* interface and the *−p* flag to this command does this. For example to enable Ethernet device `eth0` for IPX capability as the primary IPX interface using the IEEE 802.2 frame type and IPX network address `39ab0222` you would use:

```
# ipx_interface add −p eth0 802.2 0x39ab0222
```

If the frame type differs from NetWare(tm) servers on this network, they will studiously ignore you. If the frame type is correct but the network number differs, they will still ignore you but complain frequently on the NetWare server console. The latter is guaranteed to gain you flames from your NetWare administrator and may disrupt existing NetWare clients.
If you get an error while running this program and you happen to not have already configured tcp/ip, then you will find that you need to manually start the `eth0` interface using the command:

```
# ifconfig eth0 up
```

ipx_configure

This command enables or disables the automatic setting of the interface configuration and primary interface settings.

`−−auto_interface`

allows you to select whether new network devices should be automatically configured as IPX devices or not.

`−−auto_primary`

allows you to select whether the IPX software should automatically select a primary interface or not. Problems have been noted using this with Windows 95 clients on the network.

A typical example would be to enable both automatic interface configuration and automatic primary interface setting with the following command:

```
# ipx_configure --auto_interface=on --auto_primary=on
```

ipx_internal_net

This command allows you to configure or deconfigure an internal network address. An internal network address is optional, but when it is configured it will always be the primary interface. To configure an IPX network address of `ab000000` on IPX node `1` you would use:

```
# ipx_internal_net add 0xab000000 1
```

ipx_route

The command allows you to manually modify the IPX routing table. For example to add a route to IPX network `39ab0222` via a router with node number `00608CC33C0F` on IPX network `39ab0108`:

```
# ipx_route add 0x39ab0222 0x39ab0108 0x00608CC33C0F
```

8.Configuring your Linux machine as an IPX router.

If you have a number of IPX segments that you wish to internetwork you need the services of a router. In the Novell environment there are two pieces of information which are necessary to be propagated around the network. They are the network routing information propagated using Novell RIP, and the service advertisement information propagated using Novell SAP. Any router must support both of these protocols to be useful in most situations.

Linux has support for both of these protocols and can be fairly easily made to function as a fully Novell compliant router.

The Linux kernel IPX support actually manages the IPX packet forwarding across interfaces, but it does this according to the rules coded into the IPX routing table. Linux needs a program to implement the Novell RIP and SAP to ensure that the IPX routing table is built correctly and updated periodically to reflect changes in the network status.

Volker Lendecke `<lendecke@namu01.gwdg.de>` has developed a routing daemon *ipxripd* that will do this for you. The *mars_nwe* package mentioned later includes an alternative routing daemon.

You can find *ipxripd* at:

sunsite.unc.edu

or at Volkers home site at:

ftp.gwdg.de

Configuring your Linux machine to act as a router is very straightforward. The steps you must take are:

1. Build your kernel with IPX, Ethernet and `/proc` support.
2. Obtain, compile and install the *ipxd* daemon program.
3. Boot the new kernel and ensure that each of the Ethernet cards has been properly detected and there are no hardware conflicts.
4. Enable the IPX protocol on each of the interfaces using the *ipx_interface* command described above.
5. Start the *ipxd* daemon program.

Consider the following simple network:

```
 IPX Addr: 0x01000000  802.2
|--------------------------|
              |
              \__________________________
                                         \          Linux Router
 IPX Addr: 0x02000000  802.2              \
|--------------------------|               \   eth0/-----------\
              |                             \--====|           |
              \__________________________          | IPX route |
                                         \     eth1|  Table    |
 IPX Addr: 0x03000000  etherII            \----====|    ^      |
|--------------------------|                       |    |      |
              |                                eth2|  IPXd     |
              \_______________________________/====|           |
                                                   |  SAPd     |
 IPX Addr: 0x04000000  etherII                 eth3|           |
|--------------------------|                   /====|           |
              |                               |    \___________/
              \_______________________________/
```

The configuration for the above network would look like:

```
# ipx_interface add eth0 802.2 0x0100000000
# ipx_interface add eth1 802.2 0x0200000000
# ipx_interface add eth2 etherii 0x0300000000
# ipx_interface add eth3 etherii 0x0400000000
# ipxd
```

You should then wait a moment or two and check your `/proc/net/ipx_route` file and you should see it populated with the IPX routes relevant to your configuration and any learned from any other routers in the network.

8.1 Do I need to configure an internal network ?

Novell has a feature called an internal network, which it uses to simplify routing in situations where a host has more than one network device connected. This is useful in the case of a fileserver connected to multiple networks as it means that only one route needs to be advertised to reach the server regardless of which network you are attempting from.

In the case of a configuration where you are not running a fileserver and your machine acting only as an IPX router the question is not as simple to answer. It has been reported that configuring for IPX/PPP works `better' if you also configure an internal network.

In any case it is easy to do, but may require a rebuild of your kernel. When you are working through the kernel `make config` you must answer `y` when asked `Full internal IPX network` as illustrated:

```
         ...
         ...
        Full internal IPX network (CONFIG_IPX_INTERN) [N/y/?] y
         ...
         ...
```

To configure the internal network interface, use the *ipx_internal_net* command described earlier in the IPX tools section. The main precaution to take is to ensure that they IPX network address you assign is unique on your network and that no other machine or network is using it.

9.Configuring your Linux machine as an NCP client.

If you are a user of a mixed technology network that comprises both IP and IPX protocols it is likely that at some time or another you have wanted to have your Linux machine access data stored on a Novell fileserver on your network. Novell have long offered an NFS server package for their fileservers that would allow this, but if you are a small site or have only a small number of people interested in doing this it is difficult to justify the cost of the commercial package.

Volker Lendecke `<lendecke@namu01.gwdg.de>` has written a Linux filesystem kernel module that supports a subset of the Novell NCP that will allow you to mount Novell volumes into your Linux filesystem without requiring any additional products for your fileserver. Volker has called the package *ncpfs* and derived the necessary information mainly from the book "Netzwerkprogrammierung in C" by Manfred Hill and Ralf Zessin (further details of the book are contained within the README file in the *ncpfs* package).

The software causes Linux to emulate a normal Novell workstation for file services. It also includes a small print utility that allows you to print to Novell print queues (This is documented in the Print Client section later). The *ncpfs* package will work with Novell fileservers of version 3.x and later, it will not work the Novell 2.x. The *ncpfs* client will also work with close Novell compatible products, but unfortunately some products that claim to be compatible aren't compatible enough. To use *ncpfs* with Novell 4.x fileservers, it is preferred to use the Novell server in *bindery* emulation mode. The NDS support is a very recent early beta addition to *ncpfs* and additionally its use may be prohibited in your country due to the inclusion of patented technology.

9.1 Obtaining *ncpfs*.

The latest *ncpfs* package was designed to be built against the version `1.2.13` kernel or kernels later than `1.3.71` (this includes 2.x.x). If you not using a kernel in either of these categories then you will have to upgrade your kernel. The Kernel−HOWTO describes how to do this in detail.

You can obtain the *ncpfs* package by anonymous ftp from Volker's home site at: ftp.gwdg.de or sunsite.unc.edu or mirror sites. The current version at the time of writing was:

`ncpfs−2.0.11.tgz` or `ncpfs−2.2.0.tgz` which adds the NDS support.

9.2 Building *ncpfs* for kernel 1.2.13.

Build a kernel with Ethernet and IPX support

The first thing you need to do is ensure that your kernel has been built with IPX support enabled. In the `1.2.13` version kernel you need only ensure that you have answered `Y` to the question: '`The IPX protocol`' as illustrated:

```
 ...  ... Assume subnets are local (CONFIG_INET_SNARL) [y]  Disable NAGLE algorithm (r
```

will also need to ensure that you include an appropriate driver for your Ethernet card. If you do not know how to do this then you should read the Ethernet−HOWTO.
You can then proceed to build your kernel. Make sure you remember to run *lilo* to install it when you have finished.

Untar the ncpfs software

```
 # cd /usr/src # tar xvfz ncpfs-2.0.11.tgz # cd ncpfs
```

Check the Makefile

If you intend to use *kerneld* to autoload the *ncpfs* kernel module then you must uncomment the line in the `Makefile` that refers to: `KERNELD`. If you are unsure what this means then you should read the Kernel−HOWTO to familiarise yourself with kernel module configuration.

Make the ncpfs software

The software should compile cleanly with no other configuration necessary:

```
 # make
```

Copy the IPX tools somewhere useful if you don't already have them.

After the *make* has completed you should find all of the tools you need in the `ncpfs/bin` directory. You can use:

```
# make install
```

to install the tools in Volkers choice of directories. If you are running on an ELF based system then you will need to rerun `` `ldconfig -v' `` to ensure that the shared library is able to be found.

Copy the ncpfs.o module somewhere useful if necessary.

If you are compiling for a `1.2.*` kernel then you will find a file called `ncpfs.o` in the `ncpfs/bin` directory after the *make* has completed. This is the *ncpfs* kernel module. You should copy this somewhere useful. On my *debian* system I have copied it to the `/lib/modules/1.2.13/fs` directory and added `ncpfs` to the `/etc/modules` file so that it will be automatically started at boot time. If you are using some other distribution you should find where it keeps its modules and copy it there, or just copy it to your `/etc` directory. To load the modules manually you need to use the command:

```
# insmod ncpfs.o
```

9.3 Building *ncpfs* for kernels 1.3.71++/2.0.*.

For the latest version of *ncpfs* you must use kernel `1.3.71` or newer, this includes the `2.0.*` kernels.

If you intend using a kernel that is version `1.3.71` or newer then the *ncpfs* kernel code has been included in the standard kernel distribution. You need only answer `Y` to:

```
Networking options  --->
    ...
    ...
    <*> The IPX protocol
    ...
Filesystems  --->
    ...
    ...
    <*> NCP filesystem support (to mount NetWare volumes)
    ...
```

You will still need to follow the instructions for building for kernels `1.2.*` so that you can build the tools but there will not be a module file for you to install.

9.4 Configuring and using *ncpfs*.

Configure the IPX network software

There are two ways of configuring the IPX network software. You can manually configure all of your IPX network information or you can choose to let the software determine for itself some reasonable settings using the command:

```
# ipx_configure --auto_interface=on --auto_primary=on
```

This should be reasonable in most circumstances, but if it doesn't work for you then read the 'IPX tools' section above to configure your software manually. Problems have been noted using this on networks containing Windows '95 clients.

Test the configuration

After your IPX network is configured you should be able to use the *slist* command to see a list of all of the Novell fileserver on your network:

```
# slist
```

If the slist command displays a message like: `ncp_connect: Invalid argument` then your kernel probably does not support IPX. Check that you have actually booted off the appropriate kernel. When you boot you should see messages about '`IPX`' and '`ncpfs`' in the system startup messages. If the *slist* command does not list all of your fileservers then you may need to use the manual network configuration method.

Mount a Novell(tm) server or volume.

If your IPX network software is working ok you should now be able to mount a Novell fileserver or volume into your Linux filesystem. The *ncpmount* command is used for this purpose and requires that you specify at least the following information:

The fileserver name
(optionally) The fileserver directory to mount
The fileserver login id. If it has a password you will also need that.
The mount point ie. where you want the mount to go. This will be an existing directory on your machine.

There is an equivalent *ncpumount* command to unmount a mounted NCP filesystem. The NCP filesystems will be unmounted cleanly if you shutdown your machine normally, so you needn't worry about *ncpumount*ing your filesystems manually before a *halt* or *shutdown*.
An example command to mount fileserver `ACCT_FS01`, with a login id of `guest` with no password, under the `/mnt/Accounts` directory might look like the following:

```
# ncpmount -S ACCT_FS01 /mnt/Accounts -U guest -n
```

Note the use of the `-n` option to indicate that no password is required for the login. The same login specifying a password of `secret` would look like:

```
# ncpmount -S ACCT_FS01 /mnt/Accounts -U guest -P secret
```

If you don't specify either the `-n` or the `-P` options you will be prompted for a password.

Check the mount

If the mount is successful you will find all the volumes accessible to the userid used for login listed as directories under the mount point. You should then also be able to traverse the directory structure to find other files. You may alternatively use the `-V` option to mount a single volume.
NCP does not provide uid or gid ownership of files. All the files will have the permission and ownership assigned to the mount point directory restricted by trustee permissions on the Novell server. Bear this in mind when sharing mounts between Linux users.

Configure mounts to be automatically performed.

If you have some need to permanently have an ncp mount then you will want to configure the commands above into your *rc* files so that they occur automatically at boot time. If your distribution doesn't already provide some way of configuring IPX like debian then I recommend you place them in your `/etc/rc.local` file if you have one. You might use something like:

```
# # Start the ncp filesystem  /sbin/insmod /lib/modules/1.2.13/fs/ncpfs.o  # configure
```

is another means of configuring NCP mounts and that is by building a `$HOME/.nwclient` file. This file contains details of temporary or user specific NCP mounts that would be performed regularly. It allows you to store the details of mounts so that you can recreate them without having to specify all of the detail each time.

Its format is quite straightforward:

```
# The first entry is the 'preferred server' entry and is # used whenever you do not sp
```

activate these mounts you could use:

```
$ ncpmount /home/terry/docs
```

to mount: DOCS_FS01 with a login of TERRY under the /home/terry/docs directory. Note that this entry was chosen because no fileserver was specified in the mount command. If the following command were used:

```
$ ncpmount −S ACCT_FS01 /home/terry/docs
```

then a GUEST login to ACCT_FS01 would be mounted there instead.

Note: for this mechanism to work the permissions of the `$HOME/.nwclient` file must be `0600` so you would need to use the command:

```
$ chmod 0600 $HOME/.nwclient
```

If non−root users are to be allowed to use this mechanism then the *ncpmount* command must be Set Userid Root, so you would need to give it permissions:

```
# chmod 4755 ncpmount
```

Try out the nsend utility

a utility to send messages to Novell users is also included in the package, it is called *nsend* and is used as follows:

```
# nsend rod hello there
```

would send the message "hello there" to a logged in user "rod" on your "primary" fileserver (the first one appearing in your `.nwclient` file. You can specify another fileserver with the same syntax as for the *ncpmount* command.

10.Configuring your Linux machine as an NCP server.

There are two packages available that allow Linux to provide the functions of a Novell Fileserver. They both allow you to share files on your linux machine with users using Novell NetWare client software. Users can attach and map filesystems to appear as local drives on their machines just as they would to a real Novell fileserver. You may want to try both to see which best serves your intended purpose.

10.1 The *mars_nwe* package.

Martin Stover <`mstover@freeway.de`> developed *mars_nwe* to enable linux to provide both file and print services for NetWare clients.

In case you are wondering about the name: *mars_nwe* is Martin Stovers Netware Emulator.

Capability of *mars_nwe*.

mars_nwe implements a subset of the full Novell NCP for file services, disk based bindery and also print services. It is likely to contain bugs but there are many people using it now and the number of bugs is steadily decreasing as new versions are released.

Obtaining *mars_nwe*.

You can obtain *mars_nwe* from ftp.gwdg.de or from ftp://sunsite.unc.edu/pub/Linux/system/filesystems/ncpfs/.

The version current at the time of writing was: `mars_nwe-0.99.pl10.tgz`.

Building the *mars_nwe* package.

Build a kernel with Ethernet and IPX Support

In the `1.2.13` version kernel you need only ensure that you have answered `Y` to the question: '`The IPX protocol`' and `N` to the question: `` `Full internal IPX network`` ' as illustrated:

```
  ...  ... The IPX protocol (CONFIG_IPX) [n] y  ...  ... Full internal IPX network (CON
```

newer kernels a similar process is adopted but the actual text of the prompt may have changed slightly.
You will also need to ensure that you include an appropriate driver for your Ethernet card. If you do not know how to do this then you should read the Ethernet−HOWTO.
You can then proceed to build your kernel. Make sure you remember to run *lilo* to install it when you have finished.

Untar the mars_nwe package.

```
 # cd /usr/src # tar xvfz mars_nwe-0.99.pl10.tgz
```

Make mars_nwe.

To make the package is very simple. The first step is to simply run `make`, this will create a `config.h` file for you. Next you should look at and edit the `config.h` file if necessary. It allows you to configure items such as the installation directories that will be used and the maximum number of sessions and volumes that the server will support. The really important entries to look at are:

```
 FILENAME_NW_INI       the location of the initialisation file PATHNAME_PROGS        wh
```

The defaults will probably be ok but you should check anyway.
When this is done:

```
 # make # make install
```

will build the servers and install them in the appropriate directory. The installation script also installs the configuration file `/etc/nwserv.conf`.

Configure the server.

Configuration is fairly simple. You need to edit the `/etc/nwserv.conf` file. The format of this file may at first look a little cryptic, but it is fairly straightforward. The file contains a number of single line configuration items. Each line is whitespace delimited and begins with a number that indicates the contents of the line. All characters following a '#' character are considered a comment and ignored. Martin supplies an example configuration file in the package, but I'll present what I consider to be a simplified example to offer an alternative for you.

```
# VOLUMES (max. 5) # Only the SYS volume is compulsory. The directory containing the S
```

Start the server

If you've configured the server to expect external programs to configure your network and/or provide the routing function then you should start those before starting the server. Presuming you have configured the server so that it will configure your interfaces for you and provide the routing services you need only issue the command:

```
# nwserv
```

Test the server

To test the server you should first try to attach and login from a NetWare client on your network. You then set a `CAPTURE` from the client and attempt a print. If both of these are successful then the server is working.

10.2 The *lwared* package.

Ales Dryak `<A.Dryak@sh.cvut.cz>` developed *lwared* to allow Linux to function as an NCP based fileserver.

Ales has called the package *lwared*, an abbreviation for *LinWare Daemon*.

Capability of *lwared*.

The *lwared* server is capable of providing a subset of the full function of the Novell NCP. It incorporates messaging but it does not provide any printing facilities at all. It does not currently work very well with either Windows95 or Windows NT clients. The *lwared* server relies on external programs to build and update the IPX routing and SAP tables. Misbehaving clients can cause the server to crash. Importantly, filename translation facilities have not been included.

The server does work for NETX and VLM NetWare shells.

Obtaining *lwared*

The *lwared* package can be built for any kernel newer than `1.2.0`, I recommend you use version `1.2.13` as no kernel patches are required if you do. Some of the IPX functionality has changed with the version `1.3.*` kernels and this means that patches are now required to make it work properly. Appropriate patches are included for the new kernels, so if you must use an alpha kernel you should still be able to get *lwared* to work properly for you.

You can obtain the *lwared* package by anonymous ftp from: klokan.sh.cvut.cz

or from:

sunsite.unc.edu or mirror sites. The current version at the time of writing was: `lwared−0.95.tar.gz`

Building *lwared*

Untar the lwaredpackage

Something like:

```
# cd /usr/src # tar xvpfz lwared−0.95.tar.gz
```

Build a kernel with Ethernet and IPX support

If you are using an alpha `1.3.*` kernel then you should try and use kernel version `1.3.17` or newer because the supplied patches were built against it. `1.3.*` kernels older than `1.3.17` will require hand patching to install. (*some information on how to do this is included in the `INSTALL` file in the package.*). To install the patches against a `1.3.17` kernel or newer you should try:

```
# make patch
```

After applying the patches if necessary, the next thing you need to do is ensure that your kernel has been built with IPX support enabled. In the `1.2.13` version kernel you need only ensure that you have answered `Y` to the question: '`The IPX protocol`' as illustrated:

```
 ...  ... Assume subnets are local (CONFIG_INET_SNARL) [y] Disable NAGLE algorithm (nc
```

newer kernels a similar process is adopted by the actual text of the prompt may have changed slightly.

You will also need to ensure that you include an appropriate driver for your Ethernet card. If you do not know how to do this then you should read the Ethernet−HOWTO.

You can then proceed to build your kernel. Make sure you remember to run *lilo* to install it when you have finished.

Compile and install lwared.

To compile *lwared* you should first check, edit if necessary, the `server/config.h` file. This file contains various settings that will govern the way your server will behave when it is running. The defaults are reasonable, though you might want to check that the directories specified for the log files and configuration files suit your system.

```
# make depend # make # make install
```

I found that the '`make depend`' complained about not finding the `float.h` file on my system but appeared to work anyway. I also found that when I tried compiling with gcc `2.6.3` I found I had to change the line:

`#include <net/route.h>` to

`#include <net/if_route.h>` in `lib/ipxkern.c` as this file changed name sometime.

The '`make install`' will attempt to install the server and routing daemon programs into your `/usr/sbin` directory, the *lwpasswd* program into your `/usr/bin` directory, the IPX utility programs will be installed into your `/sbin` directory and last but not least the manual pages will go into the `/usr/man` directory structure. If any of these locations are not suitable for your system then you should edit the relevant `Makefile` and change the target directories to suit.

Configuring and using *lwared*

Now the fun bit!

Configuring the IPX network

The first thing you must do is configure your Ethernet interfaces to support the IPX networks your server will support. To do this you will need to know the IPX network addresses for each of your LAN segments, which Ethernet device (`eth0`, `eth1` etc.) is on which segment, what frame type (`802.3`, `EtherII` etc.) each LAN segment uses and what Internal Network address your server should use (this is really needed if your server will service more than one LAN segment). A configuration for a server that is on two dis−similar segments with IPX network addresses `23a91300` and `23a91301` and internal network address `bdefaced` might look like:

```
# ipx_internal_net add BDEFACED 1 # ipx_interface add eth0 802.3 23a91300 # ipx_interf
```

Start the routing daemons

The kernel software itself actually does the IPX packet forwarding as it does for IP, but the kernel requires additional programs to manage the routing table updates. In the case of IPX two daemons are needed and both are supplied with *lwared*: *ipxripd* manages the IPX routing information and *ipxsapd* manages the SAP information. To start the daemons you need only specify the location of where they should write their log messages:

```
# ipxripd /var/adm/ipxrip # ipxsapd /var/adm/ipxsap
```

Configure the lwared server

There are two files that you must manually configure to allow user login to your *lwared* server. They are:

/etc/lwpasswd

This is where LinWare user account information is kept. The *lwpasswd* program is to keep it up to date. In its simplest form the `/etc/lwpasswd` file looks like:

`ales: terryd: guest:` Its format is a simple list of login id followed by a ':' character and then the encrypted version of the login passwd. A couple of important caveats here: No encrypted password means no password, LinWare users must have Linux accounts, that is any user you place in `/etc/lwpasswd` must also appear in `/etc/passwd` and `root` is the only account that can change the password of another LinWare user. If you are logged in as `root` you can change the password of a LinWare user as this transcript demonstrates:

```
# lwpasswd rodg Changing password for RODG Enter new password:  Re-type new password:
```

/etc/lwvtab

This is the LinWare volume tables and it stores information about what directories should be made available to LinWare users (this file is similar in nature to the NFS

`/etc/exports` file). A simple example of its format is as follows:

```
SYS              /lwfs/sys DATA             /lwfs/data HOME             /home
```

The format is simple: Volume name followed by whitespace followed by Linux directory to export. You must have at **least** an entry for the `SYS` volume for the server to start. If you intend your DOS based users to be able use your LinWare server as their primary server then you must install a standard `SYS` volume directory structure underneath the directory you export as your `SYS` volume. Since these files are proprietary and copyright to the Novell corporation you should have a license for these. If you users will be using a Novell fileserver as their primary server then this will not be necessary.

Start the lwared server.

tada!

```
# lwared
```

It is almost an anticlimax isn't it ? Ok so you've got a question, right? What is the fileserver name that is being advertised ? If you started the server as shown then the LinWare server name being advertised will be based on what is returned by the Linux *hostname*. If you'd like it to be something else then you can give the server the name when you start it, for example:

```
# lwared −nlinux00
```

would start the server with the name `linux00`.

Test the lwared server.

The very first thing to test is that your LinWare server appears in an *slist* from a DOS client on your network. The *slist* program is stored on the `SYS` volume of a Novell fileserver so you must do this from a machine that is already logged in somewhere. If this is not successful then check that *ipxsapd* and *lwared* are both running. If the *slist* is successful then you should try attaching to the server and mapping a volume:

```
C:> attach linux00/ales
 ...
 ...
C:> map l:=linux00/data:
C:> l:
```

You should then be able to treat the new map just like any other map. The file permissions you will have will be based on those allowed to the *linux* account that parallels your LinWare login.

11.Configuring your Linux machine as a Novell Print Client.

The *ncpfs* package includes two small programs that allow you to handle printing from you Linux machine to a printer attached to a Novell print server. The *nprint* command allows you to print to a file to a NetWare print queue. The *pqlist* command allows you the list the available print queues on a NetWare server.

To obtain and install these commands just follow the instructions relating to the NCP client described earlier.

Both commands require that you supply username and password so you might normally consider building some shell scripts to make the task of printing easier.

An example might look like:

```
# pqlist −S ACCT_FS01 −U guest −n
# nprint −S ACCT_FS01 −q LASER −U guest −n filename.txt
```

The login syntax is similar to the *ncpmount* command. The examples above assume that fileserver `ACCT_FS01` has a `guest` account with no password, that a print queue called `LASER` exists and that `guest` is allowed to print to it.

On my Linux boxen I have a short shell script for each Novell printer. This can then be used as a print filter to allow printing using the standard Linux spooler.

12.Configuring your Linux machine as a Novell Print Server.

A program to allow your Linux machine to act as a print server on a Netware network is included in the *ncpfs* package. For instructions on how to obtain and build, it follow the directions in the `Netware client' section above. Alternatively, support is included in the *mars_nwe* package.

12.1 Prerequisites

Configuration is quite straightforward but relies on you already having your printer configuration completed and working under Linux. This is covered in the Printing−HOWTO in some depth.

12.2 Configuration

When you have a working printer configuration, and you have built and installed the *pserver* utility then you need to add commands to start it into your `rc` files.

Exactly what command will use will depend on depend on exactly how you want it to operate, but in its simplest form something like the following will work:

```
# pserver −S ACCT_01 −U LASER −P secret −q LASERJET
```

This example asks the *pserver* utility to login in to the `ACCT_01` fileserver with username `LASER` and password `secret` and to take jobs from the `LASERJET` print queue. When an incoming print job is received it will use the default print command of *lpr* to feed the print job to the Linux print daemon. The print queue must already be defined on the fileserver and the username must have server priveliges for the queue.

You could if you wished use any Linux command to accept and print the print job. The `−c` argument allows you to specify the exact print command. For example:

```
# pserver −S ACCT_01 −U LASER −P secret −q LASERJET −c "lpr −Plaserjet"
```

would do exactly the same as the previous example except it would send the job to the `laserjet`*printcap* configuration instead of the default one.

13.An overview of the *ncpfs* user and adminstration commands

Recent versions of Volker's *ncpfs* package include a range of user and administration commands that you might want to use. The tools are built and installed as part of the *ncpfs* installation process, so if you haven't already, follow the instructions supplied in the Novell Client section above to build and install them.

Detailed information is available in the supplied *man* pages but a brief summary of the commands is as follows;

13.1 User commands.

ncopy

Network Copy − allows efficient file copies to be performed by using a Netware function rather than a copy across the network.

nprint

Network Print − allows you to print a file to a Netware print queue on a Netware server.

nsend

Network Send − allows you to send messages to other users on a Netware server.

nwbols

List Bindery Objects − allows you to list the bindery contents of a Netware server.

nwboprops

List Properties of a Bindery Object − allows you to the properties of a Netware bindery object.

nwbpset

Set Bindery Property − allows you to set the properties of a Netware bindery object.

nwbpvalues

Print Netware Bindery Objects Property Contents − allows you to print the contents of a Netware bindery property.

nwfsinfo

Fileserver Information − prints some summary information about a Netware server.

nwpasswd

Netware Password − allows you to change a Netware users password.

nwrights

Netware Rights − displays the rights associated with a particular file or directory.

nwuserlist

Userlist − lists the users currently logged into a Netware fileserver.

pqlist

Print Queue List − displays the contents of a Netware print queue.

slist

Server List − displays a list of know Netware fileserver.

13.2 Administration tools.

nwbocreate

Create a Bindery Object − allows you to create a Netware bindery object.

nwborm

Remove Bindery Object − allows you to delete a Netware bindery object.

nwbpadd

Add Bindery Property − allows you to set the value of an existing property of a Netware bindery object.

nwbpcreate

Create Bindery Property − allows you to create a new property for an existing Netware bindery object.

nwbprm

Remove Bindery Property − allows you to remove a property from a Netware bindery object.

nwgrant

Grant Trustee Rights − allows you to assign trustee rights to a directory on a Netware fileserver.

nwrevoke

Revoke Trustee Rights − allows you to remove trustee rights from a directory on a Netware fileserver.

14.Configuring PPP for IPX support.

New versions of the *pppd* PPP daemon for Linux have support that allows you to carry IPX packets across a PPP serial link. You need at least version `ppp−2.2.0d` of the daemon. See the PPP−HOWTO for details on where to find it. When you compile *pppd* you must ensure you enable the IPX support by adding the following two lines:

```
IPX_CHANGE = 1
USE_MS_DNS = 1
```

to: `/usr/src/linux/pppd−2.2.0f/pppd/Makefile.linux`.

The `IPX_CHANGE` is what configures the IPX support into PPP. The `USE_MS_DNS` define allows Microsoft Windows95 machines to do Name Lookups.

The real trick to getting it to work in knowing how to configure it.

There are many ways of doing this, but I'm only going to describe the two that I've received any information on. I've tried neither yet, so consider this section experimental, and if you get something to work, please let me know.

14.1 Configuring an IPX/PPP server.

The first thing you need to do is configure your Linux machine as an IP/PPP server. Don't panic! This isn't difficult. Again, follow the instructions in the PPP−HOWTO and you should be pretty much ok. When you have this done there are a couple of simple modifications you need to make to get IPX working over the same configuration.

First steps.

One of the first steps you must take is to configure your linux machine as an IPX router as described in the appropriate section earlier in this document. You won't need to use the *ipx_route* command for the `ppp` interface because *pppd* will configure these for you as it does for IP. When you have the *ipxd* daemon running it will automatically detect any new IPX interfaces and propogates routes for them. In this way your dialup hosts will be seen by other machines automatically when they connect.

Design.

When you are running as a server it will normally be your responsibility to assign network address to each of the PPP links when they are established. This is an important point, each PPP link will be an IPX network and will have a unique IPX network address. This means that you must decide how you will allocate addresses and what what they will be. A simple convention is to allocate one IPX network address to each serial device that will support IPX/PPP. You could allocate IPX network addresses based on the login id of the connecting user, but I don't see any particularly good reason to do so.

I will assume that this is what you have done, and that there are two serial devices (modems) that we will use. The addresses I've assigned in this contrived example are:

```
device IPX Network Address
------ -------------------
ttyS0  0xABCDEF00
ttyS1  0xABCDEF01
```

Configure *pppd*.

Configure your `/etc/ppp/options.ttyS0` file as follows:

```
ipx-network 0xABCDEF00
ipx-node 2:0
ipxcp-accept-remote
```

and your `/etc/ppp/options.ttyS1` file as:

```
ipx−network 0xABCDEF01
ipx−node 3:0
ipxcp−accept−remote
```

These will ask *pppd* to allocate the appropriate IPX network addresses to the link when the link is established, set the local node number to `2` or `3` and will let the remote node overwrite what the remote node number with what it thinks it is. Note that each of the addresses are hexadecimal numbers and that `0x` is required at the start of the network address, but not required at the start of the node address.

There are other places this information could be configured. If you have only one dialin modem then an entry could go into the `/etc/ppp/options` file. Alternatively this information can be passed on the command line to *pppd*.

Test the server configuration.

To test the configuration you will need to have a client configuration that is known to work. When the caller dials in, logs in and *pppd* starts it will assign the network address, advise the client of the servers node number and negotiate the clients node number. When this has completed, and after *ipxd* has detected the new interface the client should be able to establish IPX connections to remote hosts.

14.2 Configuring an IPX/PPP client.

In a client configuration, whether or not you configure your Linux machine as an IPX router depends on whether you have a local LAN that you wish to act as an IPX router for. If you are a standalone machine connecting to an IPX/PPP dialin server then you won't need to run *ipxd*, but if you have a LAN and wish all of the machines on the LAN to make use of the IPX/PPP route then you must configure and run *ipxd* as described. This configuration is much simpler because you do not have multiple serial devices to configure.

Configuring *pppd*

The simplest configuration is one that allows the server to supply all of the IPX network configuration information. This configuration would be compatible with the server configuration described above.

Again you need to add some options to your `/etc/ppp/options` file, they are:

```
ipxcp−accept−network
ipxcp−accept−remote
ipxcp−accept−local
```

These options tell *pppd* to act completely passively and accept all of the configuration details from the server.

You could supply default values here for servers that don't supply details by adding `ipx−network` and `ipx−node` entries similar to the server configuration.

Testing the IPX/PPP client.

To test the client you will need a known working server to dial into. After you have dialled in and pppd has run you should see the IPX details configured on your `ppp0` device when you run the *ifconfig* command and you should be able to use *ncpmount*.

I'm not sure whether you will have to manually add IPX routes so that you can reach distant fileserver or not. This seems likely. If anyone running this configuration could tell me I'd be grateful.

15.IPX tunnel over IP

Many of you will be in a situation where you have two Novell Local Area Netorks with only an IP connection between them. How do you play multiplayer deathmatch DOOM for DOS via this arrangement you might ask ? Andreas Godzina `<ag@agsc.han.de>` has an answer for you in the form of *ipxtunnel*.

ipxtunnel provides a bridge−like facility for IPX by allowing IPX packets to be encapsulated with tcp/ip datagrams so that they can be carried by a tcp/ip connection. It listens for IPX packets and when it hears one it wraps it within a tcp/ip datagram and routes it to a remote IP address that you specify. For this to work of course the machine that you route the encapsulated IPX must also be running a copy of the same version of *ipxtunnel* as you.

15.1 Obtaining *ipxtunnel*

You can obtain *ipxtunnel* from sunsite.unc.edu or mirror sites.

15.2 Building *ipxtunnel*

ipxtunnel built cleanly for me using the following commands:

```
# cd /usr/src
# tar xvfz .../ipxtunnel.tgz
# cd ipxtunnel
# make
```

15.3 Configuring *ipxtunnel*

Configuration for *ipxtunnel* is easy. Lets say that your friends machine is `gau.somewhere.com` and your machine is called `gim.sw.edu`. *ipxtunnel* uses a configuration file called `/etc/ipxtunnel.conf`. This file allows you to specify the default UDP port to use for the tcp/ip connection, where to send the encapsulated data and which of your local interfaces *ipxtunnel* should listen on and deliver IPX packets to.

A simple configuration file would look like the following:

```
#
# /etc/ipxtunnel.conf for gim.sw.edu
#
# The UDP port to use:                       (default 7666)
port 7777
#
# The remote machine to send IPX packets to: (no default)
remote gau.somewhere.com
#
# The local interfaces to listen for IPX on: (default eth0)
interface eth0
interface eth1
```

Obviously the other machine would have a similar configuration file specifying this machine as a `remote` host.

15.4 Testing and using *ipxtunnel*

ipxtunnel acts **like** an IPX bridge, so the IPX networks at either end of the link should probably be the same. Andreas has never tested the *ipxtunnel* in an environment that actually supports Novell file servers so if you do try this in a real environment let Andreas know if it works or not.

If the *ipxtunnel* is working you should be able to start your DOOM machines up at each end of the link running IPX mode and they should see each other.

Andreas has only used this code over good high speed lines and he makes no claim as to its performance when your link is low speed. Again, let him know what works for you and what doesn't.

16.Commercial IPX support for Linux.

16.1 Caldera'a Network Desktop

Caldera Inc., produce a Linux distribution that features a range of commercially supported enhancements including fully functional Novell NetWare client support. The base distribution is the well respected Red Hat Linux Distribution and Caldera have added their "Network Desktop" products to this. The NetWare support provides a fully featured Novell NetWare client built on technology licensed from Novell Corporation. The client provides full client access to Novell 3.x and 4.x fileservers and includes features such as NetWare Directory Service (NDS) and RSA encryption.

You can obtain much more information and ordering details from the: Caldera Inc Web Server.

If you work within a Netware 4.x and/or NDS environment then the Caldera Netware Client is the only solution available.

If you have a business critical application for Novell support for Linux then the Caldera product should be something you take a close look at.

17.Some Frequently Asked Questions

Where can I find commercially supported IPX software for Linux ?

The Caldera Corporation offers a fully licensed and fully supported Netware 3.x and 4.x client. You can obtain information about it from the Caldera Inc Web Server.

Does the IPX software work with Arcnet/Token Ring/etc. ?

The Linux IPX software does work with ArcNet and Token Ring interfaces. I haven't heard of anyone trying it with AX.25 yet. Configuration is the same as for configuring for ethernet except you will have to substitute appropriate device names in place of 'eth0' and appopriate hardware addresses where necessary.

How do I configure more than one IPX interface ?

If you have more than one interface in your machine you should use the *ipx_interface* command to manually configure each one, you should not use the `plug n play' configuration.

How do I choose IPX addresses ?

IPX networking is similar, but not identical to, IP networking. A major difference is the way that addresses are used. IPX does not use the concept of subnetworking and so the sort of associations that you have between network addresses and networks is different. The rules are fairly simple:
Every IPX network address must be unique on a wide area network. This includes Internal Network Addresses. Many organisations using IPX over a wide area network will have some sort of addressing standard that you should follow.

Every Host address on an individual network must be unique. This means that every host on each IPX network must have a uniquely assigned address. In the case of ethernet network this isn't difficult as the cards each have a unique address. In the case of IPX/PPP this means you must ensure that you allocate unique addresses to all hosts on the network, irrespective of which end of the link(s) they are connected. Host address do not need to be unique across a wide area network as the network address is used in combination with the host address to uniquely identify a host.

What are frame types, which should I use ?

There are a variety of frame types in use over which you can run IPX. The most common of these are described in the 'common terms' section of this document (under the `` `Frame Type `` entry').
If you are installing your machine on an existing network then you must use whatever is already in use to allow you to interwork with the other hosts on the network, but if the installation is a brand new network you can use any of a range of protocols to carry your IPX traffic. My recommendation if you are configuring a brand new network and you need to carry both IPX and IP traffic is to use the `Ethernet_II` frame type.

My Windows95 machines mess up my frame type autodetection ?

Apparently they can, yeah. I could make nasty comments, but instead I'll just suggest that you use the manual frame type configuration instead of the automatic one. It is probably the better way anyway.

Why do I get the message `invalid argument' when I configure IPX ?

You are probably not running a kernel that supports IPX, either recompile your kernel so it does, or double check that you have actually used lilo to install and run the new kernel.

Why do I get the message `package not installed' when I configure IPX ?

You are probably not running a kernel that supports IPX, either recompile your kernel so it does, or double check that you have actually used lilo to install and run the new kernel.

Why do I get the message `IPX support not in kernel' from pppd ?

You've probably compiled IPX as a module and not ensured that it was loaded before started *pppd*.

How do I NFS export a mounted NCP filesystem ?

To use NFS to export an NCP filesystem you must mount it using the *ncpmount*`−V` option. This option allows you to mount only one volume of a fileserver instead of the usual mounting of all of them. When you do this your NFS daemon will allow you to export that filesystem in the usual way.

Why doesn't slist work when I have an internel network with mars_nwe ?

You must have the get nearest server enabled. That is, entry 401 in /etc/nwserv.conf should be 0 unless you have a reason for not responding to get nearest servers. If you just want slist to work and not respond to every get nearest server request, include your internal network and node number in /etc/nwserv.stations and set entry 401 in /etc/nwserv.conf to 2.

Does ncpfs package work with mars_nwe ?

Martin and Volker's code is slowly beginning to converge. Recent versions of *mars_nwe* have an option to enable it to work with *ncpfs*. You must enable the `WITH_NAME_SPACE_CALLS` in the *mars_nwe*`config.h` file.

Is there any free DOS software to work with mars_nwe ?

A contrived question deserves a contrived answer. I'm glad you asked, Martin has a package that he distributes alongside his *mars_nwe* package that offers free DOS client support for the *mars_nwe* server. You can find it at the same sites as the server, and it will be called `mars_dosutils-0.01.tgz`. It includes C source code for programs such as *slist.exe*, *login.exe*, *map.exe* etc. The source is compilable with Borland(tm) C.

18.Copyright Message.

19.Miscellaneous and Acknowledgements.

Terry Dawson `<terry@perf.no.itg.telstra.com.au>` for the original document

David E. Storey `<dave@tamos.gmu.edu>` and Volker Lendecke `<lendecke@namu01.gwdg.de>` both assisted greatly by supplying me with information for this document. Gilbert Callaghan `<gilbert@pokey.inviso.com>`, David Higgins `<dave@infra.com>` and Chad Robinson `<chadr@brtgate.brttech.com>` each contributed information on configuring IPX/PPP. Bennie Venter

<bjv@Gil−galad.paradigm−sa.com> contributed some useful information relating to frame types. Christopher Wall <vergil@idir.net contributed some useful suggestions to improve the readability and layout of the document. Axel Boldt <boldt@math.ucsb.edu> contributed some useful suggestions and feedback. Erik D. Olson <eriko@wrq.com> provided some useful feedback and information on configuring PPP for IPX. Brian King <root@brian.library.dal.ca> contributed a question for the FAQ section.

regards Kevin Thorpe.

<kevin@pricetrak.com>