Linux AX25-HOWTO, Amateur Radio.

Terry Dawson, VK2KTJ, terry@perf.no.itg.telecom.com.au v1.1, 14 June 1996

The Linux Operating System is perhaps the only operating system in the world that can boast native and standard support for the AX.25 packet radio protocol utilised by Amateur Radio Operators worldwide. This document aims to describe how to install and configure this support.

Contents

1 Introduction.

This document was originally an appendix to the HAM-HOWTO, but grew too large to be reasonably managed in that fashion. This document describes how to install and configure the native AX.25 and NetRom support for Linux. A few typical configurations are described that could be used as models to work from.

1.1 Changes from the previous version

```
Additions:
        Baycom support - ye gads!

Corrections/Updates:
        Included SLIP support in kernel config - oops.
        2.0.0 kernel and ax25-tools update.
        Updated net-tools to 1.32-alpha version.
        Removed Jonathons 'special' net-tools.
```

```
A bit of shuffling about to make more sense.
Updated the example configurations.
A minor copyright ooops.
```

1.2 Other related documentation.

There is a lot of related documentation. There are many documents that relate to Linux networking in more general ways and I strongly recommend you also read these as they will assist you in your efforts and provide you with stronger insight into other possible configurations.

They are:

The HAM-HOWTO `<http://sunsite.unc.edu/mdw/HOWTO/HAM-HOWTO.html>`

the Net-2-HOWTO `<http://sunsite.unc.edu/mdw/HOWTO/NET-2-HOWTO.html>`

the Ethernet-HOWTO `<http://sunsite.unc.edu/mdw/HOWTO/Ethernet-HOWTO.html>`

the IPX-HOWTO `<http://sunsite.unc.edu/mdw/HOWTO/IPX-HOWTO.html>`

and

the Firewall-HOWTO `<http://sunsite.unc.edu/mdw/HOWTO/Firewall-HOWTO.html>`

2 Where to obtain new versions of this document.

The best place to obtain the latest version of this document is from a Linux Documentation Project archive. The Linux Documentation Project runs a Web Server and this document appears there as *The AX25-HOWTO* `<http://sunsite.unc.edu/mdw/HOWTO/AX25-HOWTO.html>`. You can always contact me, but I pass new versions of the document directly to the LDP HOWTO coordinator, so if it isn't there then chances are I haven't finished it.

3 Linux and the AX.25 and NetRom Packet Radio Protocols.

The *AX.25* protocol offers both connected and connectionless modes of operation, and is used either by itself for point-point links, or to carry other protocols such as TCP/IP and NetRom.

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

The NetRom protocol is an attempt at a full network protocol and uses AX.25 at its lowest layer as a datalink protocol. It provides a network layer that is an adapted form of AX.25.

Alan Cox developed some early kernel based AX.25 software support for Linux. Jonathon Naylor <`g4klx@g4klx.demon.co.uk`> has taken up ongoing development of the code, has added NetRom support and is now the developer of the AX.25 related kernel code and utilities. DAMA support was developed by Joerg, DL1BKE. Finally Baycom support was added by Thomas Sailer, <`sailer@ife.ee.ethz.ch`>.

The Linux code supports KISS based TNC's (Terminal Node Controllers), the Ottawa PI card, the Gracilis PacketTwin card and other Z8530 SCC based cards with the generic SCC driver and both the Parallel and Serial port Baycom modems.

The User programs contain a simple PMS (Personal Message System), a beacon facility, a line mode connect program, 'listen' an example of how to capture all AX.25 frames at raw interface level and programs to configure the NetRom protocol. Included also are an AX.25 server style program to handle and despatch incoming AX.25 connections and a NetRom daemon which does most of the hard work for NetRom support.

4 The AX.25/NetRom software components.

The AX.25 software is comprised of three components, the kernel source, the network configuration tools and the utility programs.

The version 1.3.xx Linux kernels include the AX.25, NetRom, Z8530 SCC, PI card and PacketTwin drivers by default. I recommend you obtain and use the version 1.3.xx kernel source.

These are three reasons I'm recommending you use the alpha version software even though most everyone else will recommend you don't use it:

- it makes the process of building and configuring the software a whole lot simpler. There are no patches to apply it is a simple matter of configuring and compiling a kernel with the appropriate options enabled.
- it is the most recent software and is most likely to provide the most satisfactory results.
- like all new software, it needs testing and bug reports. The more people that take the software and attempt to use it in their own configurations the more likely it will be that bugs are uncovered and the easier it will be to get the software debugged. Join the team, contribute bug reports and help everybody.

Be warned though, this software is **alpha** and may have problem that you wouldn't otherwise encounter. Please take all the usual precautions and be sure to keep a working kernel as backup when testing new kernels. It is wise to test a new kernel by booting from it from floppy before actually installing it with *lilo*, though I haven't bothered for about 90 kernels revisions so I'm fairly confident you can pretty safely run with alpha kernels.

4.1 Finding the kernel, tools and utility packages.

4.1.1 The kernel source:

The kernel source can be found in its usual place at:

ftp.funet.fi

`/pub/Linux/PEOPLE/Linus/v2.0/`

4.1.2 The network tools:

The latest alpha release of the standard Linux network tools support AX.25 and NetRom and can be found at:

ftp.inka.de

```
/pub/comp/Linux/networking/net-tools/net-tools-1.32-alpha.tar.gz
```

or:

ftp.linux.org.uk

```
/pub/linux/Networking/PROGRAMS/NetTools/net-tools-1.32-alpha.tar.gz
```

The latest ipfwadm package can be found at:

ftp.xos.nl

```
/pub/linux/ipfwadm/
```

4.1.3 The AX25 utilities:

The AX.25 utility programs can be found at:

sunsite.unc.edu

```
/pub/Linux/apps/ham/ax25-utils-2.0.0.tar.gz
```

or:

ftp.ucsd.edu

```
/hamradio/packet/tcpip/incoming/ax25-utils-2.0.0.tar.gz
```

5 Installing the AX.25/NetRom software.

To successfully install AX.25 support on your linux system you must configure and install an appropriate kernel and then install the AX.25 utilities.

5.1 Compiling the kernel.

If you are already familiar with the process of compiling the Linux Kernel then you can skip this section, just be sure to select the appropriate options when compiling the kernel. If you are not, then read on.

The normal place for the kernel source to be unpacked to is the `/usr/src` directory into a subdirectory called `linux`. To do this you should be logged in as `root` and execute a series of commands similar to the following:

```
# mv linux linux.old
# cd /usr/src
# gzip -dc linux-2.0.0.tar.gz | tar xvofp -
# cd linux
```

After you have unpacked the kernel source into place you need to run the configuration script and choose the options that suit your hardware configuration and the options that you wish built into your kernel. You do this by using the command:

```
# make config
```

You might also try:

```
# make menuconfig
```

if you prefer a full screen menu based method. I'm going to describe the original method, but you use whichever you are most comfortable with.

In either case you will be offered a range of options at which you must answer 'Y' or 'N'. (Note you may also answer 'M' if you are using modules. For the sake of simplicity I will assume you are not, please make appropriate modifications if you are).

The options most relevant to an AX.25 configuration are:

```
Networking support (CONFIG_NET) [Y/n/?]                            Y
Loopback device support (CONFIG_BLK_DEV_LOOP) [N/y/m/?]            Y
Network firewalls (CONFIG_FIREWALL) [N/y/?]
TCP/IP networking (CONFIG_INET) [Y/n/?]                            Y
IP: forwarding/gatewaying (CONFIG_IP_FORWARD) [N/y/?]
IP: firewalling (CONFIG_IP_FIREWALL) [N/y/?] (NEW)
IP: tunneling (CONFIG_NET_IPIP) [N/y/m/?] (NEW)
Amateur Radio AX.25 Level 2 (CONFIG_AX25) [N/y/?]                  Y
AX.25 over Ethernet (CONFIG_BPQETHER) [N/y/?] (NEW)
Amateur Radio NET/ROM (CONFIG_NETROM) [N/y/?] (NEW)
Kernel/User network link driver(ALPHA) (CONFIG_NETLINK) [N/y/?]
Network device support (CONFIG_NETDEVICES) [Y/n/?]                 Y
SLIP (serial line) support (CONFIG_SLIP) [N/y/m/?]                 Y
Radio network interfaces (CONFIG_NET_RADIO) [N/y/?]
BAYCOM ser12 and par96 kiss emulation driver for AX.25 (CONFIG_BAYCOM) [N/y/m/?]
Z8530 SCC kiss emulation driver for AX.25 (CONFIG_SCC) [N/y/m/?]
Other ISA cards (CONFIG_NET_ISA) [N/y/?]
Ottawa PI and PI/2 support (CONFIG_PI) [N/y/?] (NEW)
Gracilis PackeTwin support (CONFIG_PT) [N/y/?] (NEW)
Standard/generic serial support (CONFIG_SERIAL) [Y/n/?]
```

The options I have flagged with a 'Y' are those that you **must** must answer 'Y' to. The rest are dependent on what hardware you have and what options you want to include. Some of these options are described in more detail later on, so if you don't know what you want yet, then read ahead and come back to this step later.

After you have completed the kernel configuration you should be able to cleanly compile your new kernel:

```
# make dep
# make clean
# make zImage
```

Make sure you move your `arch/i386/boot/zImage` file wherever you want it and then run *lilo* to ensure that you actually boot from it.

5.2 The network configuration tools.

After you have compiled the kernel you should compile the new network configuration tools. These tools allow you to modify the configuration of network devices and to add routes to the routing table.

The new alpha release of the standard `net-tools` package includes support for AX.25 and NetRom support. I've tested this and it seems to work well for me.

5.2.1 Building the standard net-tools release.

Don't forget to read the `Release` file and follow any instructions there. The steps I used to compile the tools were:

```
# cd /usr/src
# tar xvfz net-tools-1.32-alpha.tar.gz
# cd net-tools-1.32-alpha
# make config
```

At this stage you will be presented with a series of configuration questions, similar to the kernel configuration questions. Be sure to include support for all of the protocols and network devices types that you intend to use. If you do not know how to answer a particular question then answer 'Y'.

The tools compiled cleanly against a `2.0.0` kernel without error.

When the compilation is complete, you should need only use the:

```
# make install
```

command to install the programs in their proper place.

If you wish to use the IP firewall facilities then you will need the latest firewall administration tool `ipfwadm`. This tool replaces the older `ipfw` tool which will not work with new kernels.

I compiled the `ipfwadm` utility with the following commands:

```
# cd /usr/src
# tar xvfz ipfwadm-2.0beta2.tar.gz
# cd ipfwadm-2.0beta2
# make install
# cp ipfwadm.8 /usr/man/man8
# cp ipfw.4 /usr/man/man4
```

5.3 The AX.25 user programs.

After you have successfully compiled and booted your new kernel, you need to compile the user programs. To compile and install the user programs you should use a series of commands similar to the following:

```
# cd /usr/local
# gzip -dc ax25-utils-2.0.0.tar.gz | tar xvvof -
# cd ax25-utils-2.0
# make install
```

The files will be installed under the `/usr/local` directory by default in subdirectories: `bin, sbin, etc` and `man`.

If you get messages something like:

```
gcc -Wall -Wstrict-prototypes -O2 -I../lib -c call.c
call.c: In function ‘statline’:
call.c:268: warning: implicit declaration of function ‘attron’
call.c:268: ‘A_REVERSE’ undeclared (first use this function)
call.c:268: (Each undeclared identifier is reported only once
call.c:268: for each function it appears in.)
```

then you should read the `README.ncurses` file in the top level directory. You can do as Jonathon suggests, or you can try replacing:

```
#include <curses.h>
```

with:

```
#include <ncurses/curses.h>
```

in the:

```
call/call.c
call/menu.c
```

files.

6 Configuring a RAW AX.25 connected mode interface.

The first stage to configuration of an AX.25 interface is to configure it to work as a 'vanilla' AX.25 interface with no TCP/IP. The following configuration will get you to the point of being able to make AX.25 calls from your Linux machine to other AX.25 nodes.

The AX.25 software has been designed to work with a TNC in *kiss* mode or with other cards such as the Ottawa PI2 card, PacketTwin and other SCC cards via special drivers that emulate a *kiss* tnc.

For real KISS TNC's there are two steps to complete in order to create an AX.25 port ready to use to make outgoing calls. For other types of hardware you need only complete the first step.

6.1 Creating the `/usr/local/etc/axports` file.

The AX.25 ports have a configuration file that is read by many programs that want to find information about an AX.25 port. This file is called the:

```
/usr/local/etc/axports
```

file. The format of the file is as follows:

```
portname  callsign  baudrate  paclen  window  description
```

where:

portname
is a text name that you will refer to the port by.

callsign
is the AX.25 callsign you want to assign to the port.

baudrate
is the speed at which you wish the port to communicate with your TNC.

paclen
is the maximum packet length you want to configure the port to use for AX.25 connected mode connections.

window
is the AX.25 window (K) parameter. This is the same as the `MAXFRAME` setting of many tnc's.

description
is a textual description of the port.

In my case, mine looks like:

```
radio    VK2KTJ-0        4800    256     2       144.800 MHz
```

6.2 Creating the AX.25 device

If you are using an SCC card like the PI2 or PacketTwin then you do not need to create the network device, as the kernel driver will automatically do this for you. If you are using a KISS TNC then you will need to create the AX.25 interface as it will not already exist. Creating an AX.25 port is very similar to creating a slip device.

You will need to have the TNC preconfigured and connected to your serial port. You can use a communications program like *minicom* or *seyon* to configure the TNC into kiss mode if you wish.

You use the *axattach* program in much the same way as you would use the *slattach* program (refer to the NET-2-HOWTO for more information on slattach and the other network software for Linux). For example:

```
# /usr/local/sbin/axattach /dev/ttyS0 radio
```

would configure your `/dev/ttyS1` serial device to be a *kiss* interface configured as per the details for the line beginning with the portname "radio" in the `/usr/local/etc/axports` file.

All this step has done is to actually activate the device in the kernel, you need to run other programs before you an actually make use of the port.

6.2.1 Setting the callsign for SCC cards.

If you are using a PI or PacketTwin card then you should use the *axparms -setcall* command to change the callsign of the appropriate port to that which you intend to use. Refer to the PI/PacketTwin section for the names of the network devices to use.

6.2.2 Testing the interface.

You now should be able to make outgoing AX.25 connections. To test AX.25 connected mode you could use the *call* program as demonstrated:

```
/usr/local/bin/call radio VK2DAY via VK2SUT
```

Note: you must tell call which AX.25 port you wish to make the call on, as the same AX.25 node might be reachable on any of the ports you have configured.

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

Please refer to the man pages in `/usr/local/man` and the `README` file in `ax25-utils-2.0.0` distribution for more information.

7 Configuring an AX.25 interface for tcp/ip.

It is very simple to configure an AX.25 port to carry tcp/ip. Jonathon has modified the *axattach* command so that you can include the IP address that you want the interface to be configured with on the command line. For example, to modify the example presented earlier, I could use the command:

```
# /usr/local/sbin/axattach -i 44.136.8.5 -m 512 /dev/ttyS0 radio
# /sbin/route add -net 44.136.8.0 sl0
# /sbin/route add default sl0
```

to create the AX.25 interface with an IP address of `44.136.8.6` and an `MTU` of `512` bytes. You should still use the *ifconfig* to configure the other parameters if necessary.

You do not need to do it this way, you can of course just use the *ifconfig* program to configure the ip address and netmask details for the port and add a route via the port, just as you would for any other tcp/ip interface.

```
# /sbin/ifconfig sl0 44.136.8.5
# /sbin/ifconfig sl0 netmask 255.255.255.0
# /sbin/ifconfig sl0 broadcast 44.136.8.255
# /sbin/ifconfig sl0 arp mtu 256 up
# /sbin/route add -net 44.136.8.0 sl0
# /sbin/route add default sl0
```

The commands listed above are typical of the sort of configuration many of you would be familiar with if you have used NOS or any of its derivatives or any other tcp/ip software. Note that the default route might not be required in your configuration if you have some other network device configured.

To test it out, try a ping or a telnet to a local host.

```
# ping -i 5 44.136.8.58
```

Note the use of the '`-i 5`' arguments to *ping* to tell it to send pings every 5 seconds instead of its default of 1 second.

8 Configuring an AX.25 interface for NetRom.

To configure NetRom on an AX.25 interface you must configure two files.

8.1 Configuring `/usr/local/etc/nrports`

The first is the `/usr/local/etc/nrports` file. This file describes the NetRom port in much the same way as the `/usr/local/etc/axports` file describes the AX.25 ports.

This file is formatted as follows:

```
name callsign  alias  paclen   description
```

Where:

name

is the text name that you wish to refer to the port by.

callsign

is the callsign that the NetRom traffic from this port will use.

alias

is the NetRom alias this port will have assigned to it.

paclen

is the maximum size of NetRom frames transmitted by this port.

description

is a free text description of the port.

An example would look something like the following:

```
netrom  VK2KTJ-1        LINUX   235     Linux Switch Port
```

Note that the ports are referred to by name by programs such as *call*.

8.2 Configuring /usr/local/etc/nrbroadcast

The second file is the `/usr/local/etc/nrbroadcast` file.

This file is formatted as follows:

```
axport min_obs def_qual worst_qual verbose
```

Where:

axport

is the port name obtained from the `/usr/local/etc/axports` file. If you do not have an entry in `/usr/local/etc/nrbroadcasts` for a port then this means that no NetRom routing will occur and any received NetRom broadcasts will be ignored for that port.

min_obs

is the minimum obselesence value for the port.

def_qual

is the default quality for the port.

worst_qual

is the worst quality value for the port, any routes under this quality will be ignored.

verbose

is a flag determining whether full NetRom routing broadcasts will occur from this port or only a routing broadcast advertising the node itself.

An example would look something like the following:

```
radio    1        200       10          1
```

8.3 Creating the NetRom device

When you have the two configuration files completed you must create the NetRom device in much the same way as you did for the AX.25 devices. This time you use the *nrattach* command:

```
# nrattach netrom
```

This command would start the NetRom device (`nr0`) named `netrom` configured with the details specified in the `/usr/local/etc/nrports` file.

8.4 Starting the NetRom daemon

The NetRom daemon manages the NetRom routing tables and generates the NetRom routing broadcasts. You start NetRom with the command:

```
# /usr/local/sbin/netromd
```

You should soon see the `/proc/net/nr_neigh` file filling up with information about your NetRom neighbours.

Remember to put the `/usr/local/sbin/netromd` command in your *rc* files so that it is started automatically each time you reboot.

9 Configuring IP on a NetRom interface.

Configuring a NetRom interface for tcp/ip is almost identical to configuring an AX.25 interface for tcp/ip.

Again you can either specify the ip address and mtu on the *nrattach* command line, or use the *ifconfig* and *route* commands, but you need to manually add *arp* entries for hosts you wish to route to because there is no mechanism available for your machine to learn what NetRom address it should use to reach a particular IP host.

So, to create an `nr0` device with an IP address of `44.136.8.5` and an mtu of `512` and configured with the details from the `/usr/local/etc/nrports` file for a NetRom port named `netrom` you would use:

```
# /usr/local/sbin/nrattach -i 44.136.8.5 -m 512 netrom
# route add 44.136.8.5 nr0
```

or you could use something like the following commands manually:

```
# ifconfig nr0 44.136.8.5 hw netrom VK2KTJ-1
# route add 44.136.8.5 nr0
```

Then for each IP host you wish to reach via NetRom you need to set route and arp entries. To reach a destination host with an IP address of `44.136.80.4` at NetRom address `BBS:VK3BBS` via a NetRom neighbour with callsign `VK2SUT-0` you would use commands as follows:

```
# route add 44.136.80.4 nr0
# arp -t netrom -s 44.136.80.4 vk2sut-0
# nrparms -nodes vk3bbs + BBS 120 6 sl0 vk2sut-0
```

The '`120`' and '`6`' arguments to the *nrparms* command are the NetRom *quality* and *obsolescence count* values for the route.

10 Configuring an Ethernet device for AX.25 ala BPQ

Linux support BPQ Ethernet compatibility. This enables you to run the AX.25 protocol over your Ethernet LAN and to interwork your linux machine with other BPQ machines on the LAN.

Configuration is quite straightforward. You firstly must have configured a standard Ethernet device. This means you will have compiled your kernel to support your Ethernet card, and executed configuration commands similar to the following:

```
# ifconfig eth0 44.136.8.5 netmask 255.255.255.248 up
# ifconfig eth0 broadcast 44.136.8.103
# route add -net 44.136.8.96 netmask 255.255.255.248 eth0
```

To configure the BPQ support you need to associate an AX.25 callsign with your Ethernet hardware address. You should use the following:

```
# axparms -dev eth0 vk2ktj-9
```

This says that AX.25 callsign `vk2ktj-9` is associated with the `eth0` device.

BPQ Ethernet normally uses a multicast address. The Linux implementation does not, and instead it uses the normal Ethernet broadcast address. The NET.CFG file for the BPQ ODI driver should therefore look similar to this:

```
LINK SUPPORT

        MAX STACKS 1
        MAX BOARDS 1
```

```
LINK DRIVER E2000                   ; or other MLID to suit your card

        INT 10                      ;
        PORT 300                    ; to suit your card

        FRAME ETHERNET_II

        PROTOCOL BPQ 8FF ETHERNET_II ; required for BPQ - can change PID

BPQPARMS                            ; optional - only needed if you want
                                    ; to override the default target addr

        ETH_ADDR  FF:FF:FF:FF:FF:FF  ; Target address
```

11 Ottawa PI/PI2 card driver and Gracilis PacketTwin driver.

The Ottawa PI and the Gracilis PacketTwin are Z8530 SCC based card for IBM PC type machines that are in common usage by Amateur Radio operators worldwide. The PI card supports a high speed half duplex (single DMA channel) port, and a low speed (<9k6bps interrupt driven) half duplex port. The PI2 is a new version of the card that supports an on board radio modem and improved hardware design. The PacketTwin card supports two high speed port and is capable of accepting on board modems.

The PI card driver was written by `David Perry, <dp@hydra.carleton.edu>` The PacketTwin card driver was written by `Craig Small VK2XLZ, <csmall@triode.apana.org.au>`. Both drivers are a standard feature of the 1.3.xx kernel.

Please refer to the AX.25 section above for details on how to configure the kernel to include the PI card or the PacketTwin card drivers.

Once you have the appropriate driver configured into your kernel then you should reboot and when you do, you should see mention of the PI or PacketTwin card driver in the messages that appear on the screen while the kernel is booting. The drivers test each of the I/O port addresses that the card might be configured for and reports any that it finds. You should then look at the `/proc/net/dev` file and you should see reference to devices called `pi0a` and `pi0b` or `pt0a` and `pt0b` for the PI and PacketTwin cards respectively.

If the devices don't appear then recheck that you have configured and compiled your kernel correctly and that you are in fact actually running your new kernel. If you still don't then this suggests that your PI or PacketTwin card was not detected by the kernel while it booted and may indicate that you have some sort of hardware conflicting with your PI/PT card that prevented it being detected.

If all of the above went as planned then you will need to configure your PI/PT card with an AX.25 address and for IP. The configuration of the PI/PT cards is virtually identical to that of any other IP interface. Something like the following should work ok for you:

```
# /usr/local/sbin/axparms -setcall pi0a VK2KTJ-2
# /sbin/ifconfig pi0a 44.136.8.5
# /sbin/ifconfig pi0a netmask 255.255.255.0
# /sbin/ifconfig pi0a broadcast 44.136.8.255
```

```
# /sbin/ifconfig pi0a arp mtu 256 up
# /sbin/route add -net 44.136.8.0 pi0a
# /sbin/route add default pi0a
```

Note that `pi0a` refers to the ‘a’ port on the first PI card found, and that `pi0b` would therefore refer to the ‘b’ port on the first PI card found. `pt0a` would therefore be the first port on the first PacketTwin card detected.

Note also the use of the *axparms -setcall* command to set the AX.25 callsign of the port.

As usual, when this has been done you can test the interface with the *ping* or telnet command to ensure it is working.

12 Z8530 SCC Kiss Emulation

Joerg Reuter, DL1BKE has developed generic support for Z8530 SCC based cards. His driver is configurable to support a range of different types of cards and present an interface that looks like a KISS TNC so you can treat it as though it were a KISS TNC.

12.1 Obtaining and building the configuration tool package.

The kernel driver is included in the standard kernel distribution, but there are a suite of configuration tools that you will need to obtain as well.

To compile the kernel support you must be sure to answer ‘`Y`’ to: ‘**`Z8530 SCC kiss emulation driver for AX.25`**’ when asked during a kernel ‘**`make config`**’.

You can obtain the configuration tools package from:

ftp.ucsd.edu

```
/hamradio/packet/tcpip/incoming/z8530drv-2.0.dl1bke.real.tar.gz
```

The following command were what I used to compile and install the package for kernel version 1.3.93:

```
# cd /usr/src
# gzip -dc z8530drv-2.0.dl1bke.real.tar.gz | tar xvpofz -
# cd z8530drv
# rm gencfg sccinit sccstat
# make dep
# make utils
# make install
```

Note: I’ve not followed the instructions included because the version of driver in the 2.0.0 kernel is actually newer than that included in the package and the normal installation process would have overwritten the new kernel source with the older version.

After the above is complete you should have three new programs installed in your `/sbin` directory: *gencfg*, *sccinit* and *sccstat*. It is these programs that you will use to configure the driver for your card.

You will also have a group of new special device files created in your `/dev` called `scc0` .. `scc7`. These will be used later and will be the 'KISS' devices you will end up using.

12.2 Configuring the driver for your card.

The z8530 SCC driver has been designed to be as flexible as possible so as to support as many different types of cards as possible. With this flexibility has come some cost in configuration.

There is more comprehensive documentation in the package and you should read this if you have any problems. You should particularly look at `doc/scc_eng.doc` or `doc/scc_ger.doc` for more detailed information. I've paraphrased the important details, but as a result there is a lot of lower level detail that I have not included.

The main configuration file is read by the *sccinit* program and is called `/etc/z8530drv.rc`. This file is broken into two main stages: Configuration of the hardware parameters and channel configuration. After you have configured this file you need only add:

```
# sccinit
```

into the `rc` file that configures your network and the driver will be initialised according to the contents of the configuration file. You must do this before you attempt to use the driver.

12.2.1 Configuration of the hardware parameters.

The first section is broken into stanzas, each stanza representing an 8530 chip. Each stanza is a list of keywords with arguments. You may specify up to four SCC chips in this file by default. The `#define MAXSCC 4` in `scc.c` can be increased if you require support for more.

The allowable keywords and arguments are:

chip

the `chip` keyword is used to separate stanzas. It will take anything as an argument. The arguments are not used.

data_a

this keyword is used to specify the address of the data port for the z8530 channel 'A'. The argument is a hexadecimal number e.g. 0x300

ctrl_a

this keyword is used to specify the address of the control port for the z8530 channel 'A'. The arguments is a hexadecimal number e.g. 0x304

data_b

this keyword is used to specify the address of the data port for the z8530 channel 'B'. The argument is a hexadecimal number e.g. 0x301

ctrl_b

this keyword is used to specify the address of the control port for the z8530 channel 'B'. The arguments is a hexadecimal number e.g. 0x305

irq

this keyword is used to specify the IRQ used by the 8530 SCC described in this stanza. The argument is an integer e.g. 5

pclock

this keyword is used to specify the frequency of the clock at the PCLK pin of the 8530. The argument is an integer frequency in Hz which defaults to 4915200 if the keyword is not supplied.

board

the type of board supporting this 8530 SCC. The argument is a character string. The allowed values are:

PA0HZP

the PA0HZP SCC Card

EAGLE

the Eagle card

PC100

the DRSI PC100 SCC card

PRIMUS

the PRIMUS-PC (DG9BL) card

BAYCOM

BayCom (U)SCC card

escc

this keyword is optional and is used to enable support for the Extended SCC chips (ESCC) such as the 8580, 85180, or the 85280. The argument is a character string with allowed values of 'yes' or 'no'. The default is 'no'.

vector

this keyword is optional and specifies the address of the vector latch (also known as "intack port") for PA0HZP cards. There can be only one vector latch for all chips. The default is 0.

special

this keyword is optional and specifies the address of the special function register on several cards. The default is 0.

option

this keyword is optional and defaults to 0.

Some example configurations for the more popular cards are as follows:

BayCom USCC

```
chip    1
data_a  0x300
ctrl_a  0x304
data_b  0x301
ctrl_b  0x305
irq     5
board   BAYCOM
#
# SCC chip 2
#
chip    2
data_a  0x302
ctrl_a  0x306
data_b  0x303
ctrl_b  0x307
board   BAYCOM
```

PA0HZP SCC card

```
chip 1
data_a 0x153
data_b 0x151
ctrl_a 0x152
ctrl_b 0x150
irq 9
pclock 4915200
board PA0HZP
vector 0x168
escc no
#
#
#
chip 2
data_a 0x157
data_b 0x155
ctrl_a 0x156
ctrl_b 0x154
irq 9
pclock 4915200
board PA0HZP
vector 0x168
escc no
```

DRSI SCC card

```
chip 1
data_a 0x303
```

```
data_b 0x301
ctrl_a 0x302
ctrl_b 0x300
irq 7
pclock 4915200
board DRSI
escc no
```

If you already have a working configuration for your card under NOS, then you can use the *gencfg* command to convert the PE1CHL NOS driver commands into a form suitable for use in the z8530 driver configuration file.

To use *gencfg* you simply invoke it with the same parameters as you used for the PE1CHL driver in NET/NOS. For example:

```
# gencfg 2 0x150 4 2 0 1 0x168 9 4915200
```

will generate a skeleton configuration for the OptoSCC card.

12.2.2 Channel Configuration

The Channel Configuration section is where you specify all of the other parameters associated with the port you are configuring. Again this section is broken into stanzas. One stanza represents one logical port, and therefore there would be two of these for each one of the hardware parameters stanzas as each 8530 SCC supports two ports.

These keywords and arguments are also written to the `/etc/z8530drv.rc` file and must appear **after** the hardware parameters section.

Sequence is very important in this section, but if you stick with the suggested sequence it should work ok. The keywords and arguments are:

device

this keyword must be the first line of a port definition and specifies the name of the special device file that the rest of the configuration applies to. e.g. `/dev/scc0`

speed

this keyword specifies the speed in bits per second of the interface. The argument is an integer: e.g. `1200`

clock

this keyword specifies where the clock for the data will be sourced. Allowable values are:

dpll

normal halfduplex operation

external

MODEM supplies its own Rx/Tx clock

divider

use fullduplex divider if installed.

mode

this keyword specifies the data coding to be used. Allowable arguments are: `nrzi` or `nrz`

rxbuffers

this keyword specifies the number of receive buffers to allocate memory for. The argument is an integer, e.g. 8.

txbuffers

this keyword specifies the number of transmit buffers to allocate memory for. The argument is an integer, e.g. 8.

bufsize

this keyword specifies the size of the receive and transmit buffers. The arguments is in bytes and represents the total length of the frame, so it must also take into account the AX.25 headers and not just the length of the data field. This keyword is optional and default to `384`

txdelay

the KISS transmit delay value, the argument is an integer in mS.

persist

the KISS persist value, the argument is an integer.

slot

the KISS slot time value, the argument is an integer in mS.

tail

the KISS transmit tail value, the argument is an integer in mS.

fulldup

the KISS full duplex flag, the argument is an integer. `1`==Full Duplex, `0`==Half Duplex.

wait

the KISS wait value, the argument is an integer in mS.

min

the KISS min value, the argument is an integer in S.

maxkey

the KISS maximum keyup time, the argument is an integer in S.

idle

the KISS idle timer value, the argument is an integer in S.

maxdef

the KISS maxdef value, the argument is an integer.

group

the KISS group value, the argument is an integer.

txoff

the KISS txoff value, the argument is an integer in mS.

softdcd

the KISS softdcd value, the argument is an integer.

slip

the KISS slip flag, the argument is an integer.

12.3 Using the driver.

To use the driver you simply treat the `/dev/scc*` devices just as you would a serial tty device with a KISS TNC connected to it. For example, to configure Linux Kernel networking to use your SCC card you could use something like:

```
# axattach -s 4800 /dev/scc0 VK2KTJ
```

You can also use NOS to attach to it in precisely the same way. From JNOS for example you would use something like:

```
attach asy scc0 0 ax25 scc0 256 256 4800
```

12.4 The *sccstat* and *sccparam* tools.

To assist in the diagnosis of problems you can use the *sccstat* program to display the current configuration of an SCC device. To use it try:

```
# sccstat /dev/scc0
```

you will displayed a very large amount of information relating to the configuration and health of the `/dev/scc0` SCC port.

The *sccparam* command allows you to change or modify a configuration after you have booted. Its syntax is very similar to the NOS `param` command, so to set the `txtail` setting of a device to 100mS you would use:

```
# sccparam /dev/scc0 txtail 0x8
```

13 Configuring an interface for a Baycom modem.

Thomas Sailer, <sailer@ife.ee.ethz.ch>, despite the popularly held belief that it would not work very well, has developed Linux support for Baycom modems. His driver supports the `Ser12` serial port, `Par96` and the enhanced `Par97` parallel port modems.

When the driver is running it follows the standard of looking like a KISS interface, so you treat it in much the same way as you would the PI card, PacketTwin or SCC driver ports.

13.1 Compiling the kernel

The driver is only available as a module, so you must have compiled module support into your kernel, and also selected the appopriate options to support the Baycom driver during the `make config` stage when compiling your kernel.

The relevant options that you must answer 'Y' for are:

```
Radio network interfaces (CONFIG_NET_RADIO) [N/y/?]
BAYCOM ser12 and par96 kiss emulation driver for AX.25 (CONFIG_BAYCOM) [N/y/m/?]
```

13.2 Creating the /dev device files.

The Baycom driver uses special device files in the `/dev` directory. You need to create these manually. The following commands should work ok for you:

```
# mknod /dev/bc0 c 60 0
# mknod /dev/bc1 c 60 1
# mknod /dev/bc2 c 60 2
# mknod /dev/bc3 c 60 3
```

You will use these later.

13.3 Loading the module.

When the kernel compilation is complete and you have installed the module `baycom.o` wherever you normally keep your modules (conventionally they are stored in `/lib/modules/2.0.0/`), you must start the module. The module takes configuration paramters from the *insmod* command line, so you must start it manually from an *rc* file or if you use the *kerneld* program you must configure it with the relevant paramters so that it supplies them when it loads the module.

If you use *kerneld* you will need to add the following line to your `/etc/conf.modules` file:

```
alias char-major-60 baycom
```

There are four main items to specify when loading the module, they are:

modem

the baycom modem type you are configuring it for.

1. Ser12 type modem.
2. Par96 or Par97 type modem.

iobase

the i/o address of the serial or parallel port you will use.

0x3f8

COM1:

0x2f8

COM2:

0x378

LPT1:

0x278

LPT2:

irq

the Interrupt Request (IRQ) line your port will use.

4

COM1:

3

COM2:

7

LPT1:

5

LPT2:

options

some baycom specific options.

0

use hardware generated DCD signal.

1

use software DCD signal.

13.4 A small trap to be aware of.

Because the Baycom driver uses the serial and parallel ports of your machines, and because it is a low level driver for these, it competes with the existing serial and parallel drivers in the kernel. There are some ways of working around this problem. The first way is to make sure that you load the Baycom module first so that it takes the hardware it needs before the serial or parallel device driver modules load and search for their own. With the serial driver you have the luxury of being able to disable it for a single port using the *setserial* command, this will be demonstrated in the examples. If you are using the parallel port modems then beware of the `lp.o` Line Printer and `plip.o` Parallel Line IP modules.

13.5 Some example configurations.

These examples are for the two most common configurations.

Disable the serial driver for COM1: then configure the Baycom driver for a Ser12 serial port modem on COM1: with the software DCD option enabled:

```
# setserial /dev/ttyS0 uart none
# insmod baycom modem=1 iobase=0x3f8 irq=4 options=1
```

Par96 parallel port type modem on LPT1: using hardware DCD detection:

```
# insmod baycom modem=2 iobase=0x378 irq=7 options=0
```

The first device created is `/dev/bc0`, the second is `/dev/bc1` etc.

13.6 Configuring multiple devices.

Using the *insmod* program and its command line arguments you can only configure one Baycom modem. There is a *setbaycom* utility which I have not yet located that will allow you to configure others. If you happen to find it please let me know where it is kept so that I can include details for it here.

13.7 Actually using the devices.

This device driver when loaded looks like a serial tty driver with a KISS TNC attached. This means that configuration of the AX.25 software for this driver is identical to the configuration for a KISS TNC presented earlier. You need to use the *axattach* program and add the port to your `/usr/local/etc/axports` file. The only difference is that you would use the device `/dev/bc0` instead of `/dev/ttyS0`.

14 Configuring *ax25d* to accept incoming AX.25/Netrom connections.

Linux is a powerful operating system and offers a great deal of flexibility in how it is configured. With this flexibility comes a cost in configuring it to do what you want. When configuring your Linux machine to accept incoming AX.25 or NetRom connections there are a number of questions you need to ask yourself. The most important of which is: "What do I want users to see when they connect?". As detailed in the HAM-HOWTO there are BBS systems that are being developed that you might want users to see when they connect, alternatively you might want to give users a login prompt so that they can make use of a shell account, or you might even have written your own program, such as a customised database or a game, that you want people to connect to. Whatever you choose, you must tell the AX.25 software about this so that it knows what software to run when it accepts an incoming AX.25 connection.

14.1 Creating the `/usr/local/etc/ax25d.conf` file.

This file is the configuration file for the *ax25d* AX.25 daemon which handles incoming AX.25 and NetRom connections.

The file is a little cryptic looking at first, but you'll soon discover it is very simple in practice, with a small trap for you to be wary of.

The format of the **ax25d.conf** file is as follows:

```
# This is a comment and is ignored by the ax25d program.
[<interface_call>]
<peer1>  window T1 T2 T3 N2 <mode> <uid> <cmd> <cmd-name> <arguments>
<peer2>  window T1 T2 T3 N2 <mode> <uid> <cmd> <cmd-name> <arguments>
defaults window T1 T2 T3 N2 <mode> <uid> <cmd> <cmd-name> <arguments>
<peer3>  window T1 T2 T3 N2 <mode> <uid> <cmd> <cmd-name> <arguments>
...
default  window T1 T2 T3 N2 <mode> <uid> <cmd> <cmd-name> <arguments>
```

Where:

#

at the start of a line marks a comment and is completely ignored by the *ax25d* program.

<interface_call>

is the AX.25 callsign of the AX.25 or NetRom interface that this particular paragraph is for.

<peer>

is the callsign of the peer node that this particular configuration applies to. If you don't specify an SSID here then any SSID will match.

window

is the AX.25 Window parameter (K) or MAXFRAME parameter for this configuration.

T1

is the Frame retransmission (T1) timer in half second units.

T2

is the amount of time the AX.25 software will wait for another incoming frame before preparing a response in 1 second units.

T3

is the amount of time of inactivity before the AX.25 software will disconnect the session in 1 second units.

N2

is the number of consecutive retransmissions that will occur before the connection is closed.

<mode>

provides a mechanism for determining certain types of general permissions. The modes are enabled or disabled by setting or resetting bits in an 8 bit map. You set that map by setting <mode> to the sum of the following values:

1

Setup utmp, and run on a ptty (planned).

2

UNUSED.

4

UNUSED.

8

UNUSED.

16

UNUSED.

32

UNUSED.

64

Disallow digipeated uplinks, only allow direct connections.

128

Lockout, Barred, immediate disconnect.

<uid>

is the userid that the program to be run to support the connection should be run as.

<cmd>

is the full pathname of the command to be run, with no arguments specified.

<cmd-name>

is the text that should appear in a *ps* as the command name running (normally the same as <cmd> except without the directory path information.

<arguments>

are the command line argument to be passed to the <:cmd> when it is run. You pass useful information into these arguments by use of the following tokens:

%U

AX.25 callsign of the connected party without the SSID, in uppercase.

%u

AX.25 callsign of the connected party without the SSID, in lowercase.

%S

AX.25 callsign of the connected party with the SSID, in uppercase.

%s

AX.25 callsign of the connected party with the SSID, in lowercase.

You need one section in the above format for each AX.25 or NetRom interface you want to accept incoming AX.25 or NetRom connections on.

There are two special lines in the paragraph, one starts with the string '`defaults`' and other starts with the string '`default`' (yes there is a difference). These lines serve special functions.

The '`default`' lines purpose should be obvious, this line acts as a catch-all, so that any incoming connection on the <interface_call> interface that doesn't have a specific rule will match the '`default`' rule. If you don't have a '`default`' rule, then any connections not matching any specific rule will be disconnected immediately without notice.

The '`defaults`' line is a little more subtle, and here is the trap I mentioned earlier. In any of the fields for any definition for a peer you can use the '*' character to say 'use the default value'. The '`default`' line is what sets those default values. The kernel software itself has some defaults which will be used if you don't specify any. The trap is that the these defaults apply **only** to those rules **below** the '`default`' line not to those above. You may have more than one '`default`' rule per interface definition, and in this way you may create groups of default configurations.

14.2 A simple example `ax25d.conf` file.

Ok, an illustrative example:

```
# ax25d.conf for VK2KTJ - 21/11/95
# This configuration uses the AX.25 port defined earlier.
[VK2KTJ-1]
NOCALL     *     *  *  *  *   0    guest /usr/games/fortune
defaults   1    10  *  *  *   0    root  /usr/local/sbin/axspawn axspawn %u +
VK2XLZ-1   *     *  *  *  *   *      *     *         *
VK2DAY-1   *     *  *  *  *   *      *     *         *
default    1    10  5 100 5   0    root  /usr/local/bin/pms pms -a -o vk2ktj
#
```

This example says that anybody connecting to the interface on my machine with the callsign `VK2KTJ-1` will have the following rules applied:

Anyone whose callsign is set to 'NOCALL' should use the kernel default parameters and have the *fortune* program run as `guest` for them.

The `defaults` line changes two parameters from the kernel defaults (Window and T1) and will run the */usr/local/sbin/axspawn* program for them. Any copies of */usr/local/sbin/axspawn* run this way will appear as *axspawn* in a *ps* listing for convenience. The next two lines definitions for two stations who will receive those permissions.

The last line in the paragraph is the 'catch all' definition that everybody else will get (including VK2XLZ and VK2DAY using any other SSID other than -1). This definition sets all of the parameters implicitly and will cause the *pms* program to be run with a command line argument indicating that it is being run for an AX.25 connection (assuming `VK2KTJ-1` to be an AX.25 interface) and that the owner callsign is `VK2KTJ`. (See the 'Configuring the PMS' section below for more details).

If you wanted to define rules for another interface, a NetRom interface for example, then you would add another paragraph coded in the same way with rules and callsign appropriate to it.

This example is a contrived one but I think it illustrates clearly the important features of the syntax of the configuration file. Jonathon includes a much longer and more detailed example in the `ax25-utils` package that you should look at if you need more information.

14.3 Starting *ax25d*

When you have the two configuration files completed you start *ax25d* with the command:

```
# /usr/local/sbin/ax25d
```

When this is run people should be able to make AX.25 connections to your Linux machine. Remember to put the `ax25d` command in your *rc* files so that it is started automatically when you reboot each time.

14.4 Configuring *axspawn*.

The *axspawn* program is a simple program that allows AX.25 stations who connect to be log in to your machine. It is invoked from the *ax25d* program as described above. To allow a user to log in to your machine you should add a line similar to the following into your `/usr/local/etc/ax25d.conf` file:

```
default * * * * * 1 root /usr/local/sbin/axspawn axspawn %u
```

If the line ends in the `+` character then the connecting user must hit return before they will be allowed to login. The default is to not wait. Any individual host configurations that follow this line will have the *axspawn* program run when they connect. When *axspawn* is run it first checks that the command line argument it is supplied is a legal callsign, strips the SSID, then it checks that `/etc/passwd` file to see if that user has an account configured. If there is an account, and the password is either `""` (null) or `+` then the user is logged in, if there is anything in the password field the user is prompted to enter a password. If there is not an existing account in the `/etc/passwd` file then *axspawn* may be configured to automatically create one.

14.4.1 Creating the /usr/local/etc/axspawn.conf file.

You can alter the behaviour of *axspawn* in various ways by use of the `/usr/local/etc/axspawn.conf` file. This file is formatted as follows:

```
# /usr/local/etc/axspawn.conf
#
# allow automatic creation of user accounts
create    yes
#
# guest user if above is 'no' or everything else fails. Disable with "no"
guest     no
#
# group id or name for autoaccount
group     ax25
#
# first user id to use
first_uid 2001
#
# maximum user id
max_uid   3000
#
# where to add the home directory for the new users
home      /home/ax25
#
# user shell
shell     /bin/bash
#
# bind user id to callsign for outgoing connects.
associate yes
```

The eight configurable characteristics of *axspawn* are as follows:

#

indicates a comment.

create

if this field is set to `yes` then *axspawn* will attempt to automatically create a user account for any user who connects and does not already have an entry in the `/etc/passwd` file.

guest

this field names the login name of the account that will be used for people who connect who do not already have accounts if *create* is set to `no`. This is usually `ax25` or `guest`.

group

this field names the group name that will be used for any users who connect and do not already have an entry in the `/etc/passwd` file.

first_uid

this is the number of the first userid that will be automatically created for new users.

max_uid

this is the maximum number that will be used for the userid of new users.

home

this is the home (login) directory of new users.

shell

this is the login shell of any new users.

associate

this flag indicates whether outgoing AX.25 connections made by this user after they login will use their own callsign, or your stations callsign.

14.5 Configuring the *pms*

The *pms* program is an implementation of a simple personal message system. It was originally written by Alan Cox. Dave Brown, N2RJT, `<dcb@vectorbd.com>` has taken on further development of it. At present it is still very simple, supporting only the ability to send mail to the owner of the system and to obtain some limited system information but Dave is working to expand its capability to make it more useful.

Configuring the *pms* program is still a little clumsy at the moment because it is new but bare with it while Dave works on it. The only difficult part is modifying the source code to support your local mail delivery agent and whether you use the *ttylinkd* program or the *talk-ax25* program.

After that is done there are a couple of simple files that you should create that give users some information about the system and then you need to add appropriate entries into the `ax25d.conf` file so that connected users are presented with the PMS.

14.5.1 Customising `pms.c`

If you take a look at the `pms.c` file in the `ax25-utils` package you will see that there a few items that you can configure. Perhaps the most important to configure is the one that allows you to specify what local mail delivery agent you use. In most installations the *deliver* program may well work, but in the minimal installation on my gateway machine I needed to modify the line that previously said:

```
#define MAIL_DELIVERY_AGENT     "deliver %s"
```

to:

```
#define MAIL_DELIVERY_AGENT     "/bin/mail.local %s"
```

There is no easy way of knowing what local delivery agent your machine is configured to use, so perhaps you could simply look for either of these files in the obvious places and choose whichever seems to be there.

I also have the *ttylinkd* program installed, so I needed to modify the section relating to it. I keep my copy of the *ttylinkd* program in my `/usr/sbin` directory, so I modified the line that previously said:

```
/* #define MY_TALK              "/usr/local/sbin/ttylinkd" */
```

to:

```
#define MY_TALK              "/usr/sbin/ttylinkd"
```

If you use the *talk-ax25* program then you would remove the `/*` and `*/` from around that line instead.

There is one last definition that might be useful to you, and that is if you have a default email gateway. This line is the line containing the string `MY_GATEWAY`. If you have an email gateway that you use, then you should remove the comments from this line and change the address to that of your gateway.

14.5.2 Create the `/usr/local/pms/motd` file.

The `/usr/local/pms/motd` file contains the 'message of the day' that users will be presented with after they connect and receive the usual BBS id header. The file is a simple text file, any text you include in this file will be sent to users.

14.5.3 Create the `/usr/local/pms/info` file.

The `/usr/local/pms/info` file is also a simple text file in which you would put more detailed information about your station or configuration. This file is presented to users in response to their issuing of the `Info` command from the `PMS>` prompt.

14.5.4 Associate AX.25 callsigns with system users.

When a connected user sends mail to an AX.25 callsign, the *pms* expects that callsign to be mapped, or associated with a real system user on your machine.

You make this association with the *axparms* command. An example looks like:

```
# axparms -assoc vk2ktj terry
```

This command associates that AX.25 callsign `vk2ktj` with the user `terry` on the machine, so any mail for `vk2ktj` on the *pms* will be sent to `terry`.

Remember to put these associations into your *rc* file so that they are available each time your reboot.

14.5.5 Add the PMS to the /usr/local/etc/ax25d.conf file.

Adding the *pms* to your `ax25d.conf` file is very simple. There is one small thing you need to think about though. Dave has added command line arguments to the PMS to allow it to handle a number of different text end-of-line conventions. AX.25 and NetRom by convention expect the end-of-line to be *carriage return, linefeed* while the standard unix end-of-line is just *newline*. So, for example, if you wanted to add an entry that meant that the default action for a connection received on an AX.25 port is to start the PMS then you would add a line that looked something like:

```
default  1  10 5 100 5   0    root  /usr/local/bin/pms pms -a -o vk2ktj
```

This simply runs the *pms* program, telling it that it is an AX.25 connection it is connected to and that the PMS owner is `vk2ktj`. Check the *man* page for what you should specify for other connection methods.

14.5.6 Test the PMS.

To test the PMS, you can try the following command from the command line:

```
# /usr/local/bin/pms -u vk2ktj -o vk2ktj
```

Substitute your own callsign for mine and this will run the pms, telling it that it is to use the unix end-of-line convention, and that user logging in is `vk2ktj`. You can do all the things connected users can.

Additionally you might try getting some other node to connect to you to confirm that your `ax25d.conf` configuration works.

15 HOWTO link NOS and the Linux kernel networking software

Many people like to run some version of NOS under Linux because it has all of the features and facilities they are used to. Most of those people would also like to have the NOS running on their machine capable of talking to the Linux kernel so that they can offer some of the linux capabilities to radio users.

15.1 Linking NOS and Linux using a pipe device

Brandon S. Allbery, KF8NH, contributed the following information to explain how to interconnect the NOS running on a Linux machine with the kernel code using the Linux pipe device.

Since both Linux and NOS support the slip protocol it is possible to link the two together by creating a slip link. You could do this by using two serial ports with a loopback cable between them, but this would be slow and costly. Linux provides a feature that many other Unix-like operating systems provide called ‘pipes’. These are special pseudo devices that look like a standard tty device to software but in fact loopback to another pipe device. To use these pipes the first program must open the **master** end of the pipe, and the open then the second program can open the **slave** end of the pipe. When both ends are open the programs can communicate with each other simply by writing characters to the pipes in the way they would if they were terminal devices.

To use this feature to connect the Linux Kernel and a copy of NOS, or some other program you first must choose a pipe device to use. You can find one by looking in your `/dev` directory. The master end of the pipes are named: `ptyp[1-f]` and the slave end of the pipes are known as: `ttyp[1-f]`. Remember they come in pairs, so if you select `/dev/ptypf` as your master end then you must use `/dev/ttypf` as the slave end.

Once you have chosen a pipe device pair to use you should allocate the master end to you linux kernel and the slave end to the NOS program, as the Linux kernel starts first and the master end of the pipe must be opened first. You must also remember that your Linux kernel must have a different IP address to your NOS, so you will need to allocate a unique address for it if you haven't already.

You configure the pipe just as if it were a serial device, so to create the slip link from your linux kernel you can use commands similar to the following:

```
# /sbin/slattach -s 38400 -p slip /dev/ptypf &
# /sbin/ifconfig sl0 broadcast 44.255.255.255 pointopoint 44.70.248.67 /
        mtu 1536 44.70.4.88
# /sbin/route add 44.70.248.67 sl0
# /sbin/route add -net 44.0.0.0 gw 44.70.248.67
```

In this example the Linux kernel has been given IP address `44.70.4.88` and the NOS program is using IP address `44.70.248.67`. The *route* command in the last line simply tells your linux kernel to route all datagrams for the amprnet via the slip link created by the *slattach* command. Normally you would put these commands into your `/etc/rc.d/rc.inet2` file after all your other network configuration is complete so that the slip link is created automatically when you reboot. Note: there is no advantage in using *cslip* instead of *slip* as it actually reduces performance because the link is only a virtual one and occurs fast enough that having to compress the headers first takes longer than transmitting the uncompressed datagram.

To configure the NOS end of the link you could try the following:

```
# you can call the interface anything you want; I use "linux" for convenience.
attach asy ttypf - slip linux 1024 1024 38400
route addprivate 44.70.4.88 linux
```

These commands will create a slip port named 'linux' via the slave end of the pipe device pair to your linux kernel, and a route to it to make it work. When you have started NOS you should be able to ping and telnet to your NOS from your Linux machine and vice versa. If not, double check that you have made no mistakes especially that you have the addresses configured properly and have the pipe devices around the right way.

16 Some sample configurations.

Following are examples of the most common types of configurations. These are guides only as there are as many ways of configuring your network as there are networks to configure, but they may give you a start.

16.1 Small Ethernet LAN with Linux as a router to Radio LAN

Many of you may have small local area networks at home and want to connect the machines on that network to your local radio LAN. This is the type of configuration I use at home. I arranged to have a suitable block of addresses allocated to me that I could capture in a single route for convenience and I use these on my Ethernet LAN. Your local IP coordinator will assist you in doing this if you want to try it as well. The addresses for the Ethernet LAN form a subset of the radio LAN addresses. The following configuration is the actual one for my linux router on my network at home:

```
                                            .      .   .    .    . .
     ---                                .
      | Network       /---------\     .    Network
      | 44.136.8.96/29|         |    .     44.136.8/24        \ | /
      |               | Linux   |   .                          \|/
      |               |         |  .                            |
      |          eth0 | Router  |  .  /-----\    /----------\   |
      |---------------|         |-----| TNC |----| Radio    |---/
      |   44.136.8.97 |  and    |  .  \-----/    \----------/
      |               |         | sl0
      |               | Server  | 44.136.8.5
      |               |         |    .
      |               |         |     .
      |               \_________/       .
     ---                                    .      .   .    .    . .
```

```
#!/bin/sh
# /etc/rc.net
# This configuration provides one KISS based AX.25 port and one
# Ethernet device.

echo "/etc/rc.net"
echo "  Configuring:"

echo -n "    loopback:"
/sbin/ifconfig lo 127.0.0.1
/sbin/route add 127.0.0.1
echo " done."

echo -n "    ethernet:"
/sbin/ifconfig eth0 44.136.8.97 netmask 255.255.255.248 \
                broadcast 44.136.8.103 up
/sbin/route add 44.136.8.97 eth0
/sbin/route add -net 44.136.8.96 netmask 255.255.255.248 eth0
echo " done."

echo -n "    AX.25: "
axattach -i 44.136.8.5 -m 512 /dev/ttyS1 4800
ifconfig sl0 netmask 255.255.255.0 broadcast 44.136.8.255
```

```
route add -host 44.136.8.5 sl0
route add -net 44.136.8.0 window 1024 sl0

echo -n "    Netrom: "
nrattach -i 44.136.8.5 netrom

echo "  Routing:"
/sbin/route add default gw 44.136.8.68 window 1024 sl0
echo "    default route."
echo done.

# end
```

/usr/local/etc/axports

```
# name  callsign        speed   paclen  window  description
4800    VK2KTJ-0        4800    256     2       144.800 MHz
```

/usr/local/etc/nrports

```
# name  callsign        alias   paclen  description
netrom  VK2KTJ-1        LINUX   235     Linux Switch Port
```

/usr/local/etc/nrbroadcast

```
# ax25_name     min_obs def_qual        worst_qual      verbose
4800            1       120             10              1
```

- You must have IP_FORWARDING enabled in your kernel.
- The AX.25 configuration files are pretty much those used as examples in the earlier sections, refer to those where necessary.
- I've chosen to use an IP address for my radio port that is not within my home network block. I needn't have done so, I could have easily used `44.136.8.97` for that port too.
- `44.136.8.68` is my local IPIP encapsulated gateway and hence is where I point my default route.
- Each of the machines on my Ethernet network have a route:

  ```
  route add -net 44.0.0.0 gw 44.136.8.97 window 512 mss 512 eth0
  ```

  The use of the *mss* and *window* parameters means that I can get optimum performance from both local Ethernet and radio based connections.
- I also run my sendmail, http, ftp and other daemons on the router machine so that it needs to be the only machine to provide others with facilities.
- The router machine is a lowly 386DX20 with a 20Mb harddrive and a very minimal linux configuration.

16.2 IPIP encapsulated gateway configuration.

Linux is now very commonly used for tcp/ip encapsulated gateways around the world. The new tunnel driver supports multiple encapsulated routes and makes the older *ipip* daemon obsolete.

A typical configuration would look similar to the following.

```
                                          .      .   .    .    . .
  ---                                .
   | Network       /---------\     .    Network
   | 154.27.3/24   |         |    .     44.136.16/24       \ | /
   |               | Linux   |   .                          \|/
   |               |         |  .                            |
   |          eth0 | IPIP    |  .  /-----\    /----------\   |
---|---------------|         |-----| TNC |----| Radio    |---/
   |   154.27.3.20 | Gateway |  .  \-----/    \----------/
   |               |         | sl0
   |               |         | 44.136.16.1
   |               |         |    .
   |               |         |     .
   |               \_________/       .
  ---                                     .      .   .    .    . .
```

The configuration files of interest are:

```
# /etc/rc.net
# This file is a simple configuration that provides one KISS AX.25
# radio port, one Ethernet device, and utilises the kernel tunnel driver
# to perform the IPIP encapsulation/decapsulation
#
echo "/etc/rc.net"
echo "  Configuring:"
#
echo -n "    loopback:"
/sbin/ifconfig lo 127.0.0.1
/sbin/route add 127.0.0.1
echo " done."
#
echo -n "    ethernet:"
/sbin/ifconfig eth0 154.27.3.20 netmask 255.255.255.0 \
               broadcast 154.27.3.255 up
/sbin/route add 154.27.3.20 eth0
/sbin/route add -net 154.27.3.0 netmask 255.255.255.0 eth0
echo " done."
#
echo -n "    AX.25: "
axattach -i 44.136.16.1 -m 512 /dev/ttyS1 4800
/sbin/ifconfig sl0 netmask 255.255.255.0 broadcast 44.136.16.255
```

```
/sbin/route add -host 44.136.16.1 sl0
/sbin/route add -net 44.136.16.0 netmask 255.255.255.0 window 1024 sl0
#
echo -n "    tunnel:"
/sbin/ifconfig tunl0 44.136.16.1 mtu 512 up
#
echo done.
#
echo -n "Routing ... "
source /etc/ipip.routes
echo done.
#
# end.
```

and:

```
# /etc/ipip.routes
# This file is generated using the munge script
#
/sbin/route add 44.134.8.0 netmask 255.255.255.0 tunl0 gw 134.43.26.1
/sbin/route add 44.34.9.0 netmask 255.255.255.0 tunl0 gw 174.84.6.17
/sbin/route add 44.13.28.0 netmask 255.255.255.0 tunl0 gw 212.37.126.3
   ...
   ...
   ...
```

`/usr/local/etc/axports`

```
# name  callsign        speed   paclen  window  description
4800    VK2KTJ-0        4800    256     2       144.800 MHz
```

Some points to note here are:

- The new tunnel driver uses the *gw* field in the routing table in place of the *pointopoint* parameter to specify the address of the remote IPIP gateway. This is why it now supports multiple routes per interface.
- You **can** configure two network devices with the same address. In this example both the `sl0` and the `tunl0` devices have been configured with the IP address of the radio port. This is done so that the remote gateway sees the correct address from your gateway in encapsulated datagrams sent to it.
- The route commands used to specify the encapsulated routes can be automatically generated by a modified version of the *munge* script. This is included below. The route commands would then be written to a separate file and read in using the *bash* `source /etc/ipip.routes` command (assuming you called the file with the routing commands `/etc/ipip.routes`) as illustrated. The source file must be in the NOS route command format.

- Note the use of the *window* argument on the *route* command. Setting this parameter to an appropriate value improves the performance of your radio link.

The new tunnel-munge script:

```
#!/bin/sh
#
# From: Ron Atkinson <n8fow@hamgate.cc.wayne.edu>
#
#  This script is basically the 'munge' script written by Bdale N3EUA
#  for the IPIP daemon and is modified by Ron Atkinson N8FOW. It's
#  purpose is to convert a KA9Q NOS format gateways route file
#  (usually called 'encap.txt') into a Linux routing table format
#  for the IP tunnel driver.
#
#        Usage: Gateway file on stdin, Linux route format file on stdout.
#               eg.  tunnel-munge < encap.txt > ampr-routes
#
# NOTE: Before you use this script be sure to check or change the
#       following items:
#
#     1) Change the 'Local routes' and 'Misc user routes' sections
#        to routes that apply to your own area (remove mine please!)
#     2) On the fgrep line be sure to change the IP address to YOUR
#        gateway Internet address. Failure to do so will cause serious
#        routing loops.
#     3) The default interface name is 'tunl0'. Make sure this is
#        correct for your system.

echo "#"
echo "# IP tunnel route table built by $LOGNAME on `date`"
echo "# by tunnel-munge script v960307."
echo "#"
echo "# Local routes"
echo "route add -net 44.xxx.xxx.xxx netmask 255.mmm.mmm.mmm dev sl0"
echo "#"
echo "# Misc user routes"
echo "#"
echo "# remote routes"

fgrep encap | grep "^route" | grep -v " XXX.XXX.XXX.XXX" | \
awk '{
        split($3, s, "/")
        split(s[1], n,".")
        if      (n[1] == "")    n[1]="0"
        if      (n[2] == "")    n[2]="0"
        if      (n[3] == "")    n[3]="0"
        if      (n[4] == "")    n[4]="0"
```

```
        if      (s[2] == "1")   mask="128.0.0.0"
        else if (s[2] == "2")   mask="192.0.0.0"
        else if (s[2] == "3")   mask="224.0.0.0"
        else if (s[2] == "4")   mask="240.0.0.0"
        else if (s[2] == "5")   mask="248.0.0.0"
        else if (s[2] == "6")   mask="252.0.0.0"
        else if (s[2] == "7")   mask="254.0.0.0"
        else if (s[2] == "8")   mask="255.0.0.0"
        else if (s[2] == "9")   mask="255.128.0.0"
        else if (s[2] == "10")  mask="255.192.0.0"
        else if (s[2] == "11")  mask="255.224.0.0"
        else if (s[2] == "12")  mask="255.240.0.0"
        else if (s[2] == "13")  mask="255.248.0.0"
        else if (s[2] == "14")  mask="255.252.0.0"
        else if (s[2] == "15")  mask="255.254.0.0"
        else if (s[2] == "16")  mask="255.255.0.0"
        else if (s[2] == "17")  mask="255.255.128.0"
        else if (s[2] == "18")  mask="255.255.192.0"
        else if (s[2] == "19")  mask="255.255.224.0"
        else if (s[2] == "20")  mask="255.255.240.0"
        else if (s[2] == "21")  mask="255.255.248.0"
        else if (s[2] == "22")  mask="255.255.252.0"
        else if (s[2] == "23")  mask="255.255.254.0"
        else if (s[2] == "24")  mask="255.255.255.0"
        else if (s[2] == "25")  mask="255.255.255.128"
        else if (s[2] == "26")  mask="255.255.255.192"
        else if (s[2] == "27")  mask="255.255.255.224"
        else if (s[2] == "28")  mask="255.255.255.240"
        else if (s[2] == "29")  mask="255.255.255.248"
        else if (s[2] == "30")  mask="255.255.255.252"
        else if (s[2] == "31")  mask="255.255.255.254"
        else                    mask="255.255.255.255"

if (mask == "255.255.255.255")
        printf "route add -host %s.%s.%s.%s gw %s dev tunl0\n"\
                ,n[1],n[2],n[3],n[4],$5
else
        printf "route add -net %s.%s.%s.%s gw %s netmask %s dev tunl0\n"\
                ,n[1],n[2],n[3],n[4],$5,mask
 }'

echo "#"
echo "# default the rest of amprnet via mirrorshades.ucsd.edu"
echo "route add -net 44.0.0.0 gw 128.54.16.18 netmask 255.0.0.0 dev tunl0"
echo "#"
echo "# the end"
```

17 Discussion relating to Amateur Radio and Linux.

There are various places that discussion relating to Amateur Radio and Linux take place. They take place in the `comp.os.linux.*` newsgroups, they also take place on the `HAMS` list on `vger.rutgers.edu`. Other places where they are held include the `tcp-group` mailing list at `ucsd.edu` (the home of amateur radio tcp/ip discussions), and you might also try the `#linpeople` channel on the `linuxnet` irc network.

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

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

with the line:

```
subscribe linux-hams
```

in the message body. The subject line is ignored.

To join the `tcp-group` send mail to:

```
listserver@ucsd.edu
```

with the line:

```
subscribe tcp-group
```

in the body of the text.

Note: Please remember that the `tcp-group` is primarily for discussion of the use of advanced protocols, of which tcp/ip is one, in Amateur Radio. *Linux specific questions should not ordinarily go there.*

18 Copyright.