Linux AX25 HOWTO

Terry Dawson (terry@perf.no.itg.telstra.com.au) und Gerd Röthig (roetg@medizin.uni-leipzig.de)
v1.6-2, 8. Juli 1999

Das Betriebssystem Linux ist vielleicht das einzige, das eine eingebaute Unterstützung für das im Packet Radio verwendete AX.25-Protokoll bietet. Dieser Text beschreibt, wie man diese Unterstützung installiert und einrichtet.

Inhaltsverzeichnis

1 Einführung

Dieses HOWTO beschreibt, wie man die AX.25-, NetROM- und ROSE-Unterstützung unter Linux installiert und einrichtet. Einige typische Konfigurationen sind beispielhaft aufgeführt und können als Ausgangsbasis dienen.

Die Implementation der Amateurfunk-Protokolle unter Linux ist sehr flexibel. Für Neulinge wird die Konfiguration reichlich kompliziert und verwirrend erscheinen. Es braucht ein wenig Zeit, bis man versteht, wie alles zusammenpaßt. Die Konfiguration wird sich als schwierig erweisen, wenn man sich nicht entsprechend vorbereitet und über Linux im allgemeinen informiert hat.

Zur Zeit ist mit der Einführung des Linux-Kernels 2.2.x auch das AX.25-Subsystem tiefgreifenden Veränderungen unterworfen. Dieses HOWTO wird darauf in kommenden Fassungen ausführlicher eingehen, wenn erste Erfahrungen mit der neuen Software vorliegen.

Die im folgenden beschriebenen Konfigurationshinweise und Beispiele beziehen sich daher, wenn nicht ausdrücklich anders vermerkt, auf ein Linux-System mit Kernel 2.0.37, libc5 und ax25-utilities-2.1.42a.

1.1 Wo bekommt man neue Versionen dieses Textes

Den englischen Originaltext findet man im Archiv des Linux-Dokumentation-Projektes. Die WWW-Fassung findet sich hier:

```
http://metalab.unc.edu/LDP/HOWTO/AX25-HOWTO.html
```

Man kann Terry Dawson (`terry@perf.no.itg.telstra.com.au`) auch direkt fragen; neue Versionen wird er jedoch sofort dem Koordinator des LDP schicken, so daß man immer erst bei metalab nachsehen sollte. Gibt es dort keine neue Version, so ist es wahrscheinlich, daß sie noch nicht fertig ist. Die deutsche Version findet man beim DLHP:

```
http://www.tu-harburg.de/dlhp/
```

1.2 Weitere Informationen zu diesem und verwandten Themen

Es gibt eine Menge weitere Dokumentation. Viele Texte befassen sich mit Netzwerken und Linux im allgemeinen und es wird wärmstens empfohlen, diese zu lesen, da sie bei eigenen Bemühungen helfen und Wege zu weiteren möglichen Konfigurationen aufzeigen.

Folgende Texte sind empfehlenswert:

- *HAM HOWTO*

- *NET-3 HOWTO*
- *Ethernet HOWTO*
- *Firewall HOWTO*

Allgemeinere Informationen zu Linux findet man in den anderen Linux-HOWTO-Dokumenten:

```
http://metalab.unc.edu/LDP/HOWTO/
```

2 Die Protokolle bei Packet Radio und Linux

Das AX.25-Protokoll gestattet den Betrieb sowohl mit als auch ohne laufende Verbindungen und wird entweder selbst zur Herstellung von Punkt-zu-Punkt-Verbindungen oder als Träger für andere Protokolle wie TCP/IP oder NET/ROM eingesetzt.

Es entspricht in seinem Aufbau dem X.25-Protokoll mit einigen amateurfunkspezifischen Erweiterungen. Das NetROM-Protokoll ist ein Versuch eines vollständigen Netzwerkprotokolls und nutzt AX.25 auf seiner niedrigsten Ebene als Protokoll zum Datenaustausch. Es stellt eine Netzwerkebene zur Verfügung, die eine angepaßte Form des AX.25 darstellt. Das NetROM-Protokoll bietet dynamisches Routing und Aliases für die einzelnen Nodes.

Das ROSE-Protokoll wurde von Tom Moulton (W2VY) entwickelt und implementiert. Es stellt eine Implementation des X.25-Packet-Layer-Protokolls dar, wurde für das Zusammenwirken mit AX.25 als Datalink entworfen und stellt ebenfalls eine Netzwerk-Ebene zur Verfügung.

ROSE-Adressen sind zehnstellige Nummern. Die ersten vier sind der sog. Data Network Identification Code (DNIC) und stammen aus dem Zusatz B der CCITT X.121-Empfehlung.

Mehr Informationen zum ROSE-Protokoll gibt es auf dem RATS (Radio Amateur Teleprinter Society) -Web-Server:

```
http://www.rats.org/
```

Das FPAC/Linux-Projekt als eine ROSE-Anwendung ist auf

```
http://www.qsl.net/fpac
```

zu finden.

Alan Cox entwickelte die frühe Kernel-Unterstützung für AX.25. Jonathan Naylor (`g4klx@g4klx.demon.co.uk`) entwickelte den Code weiter, fügte NetROM- und ROSE-Unterstützung hinzu und ist jetzt der Entwickler des AX.25-Codes für den Kernel.

Die Unterstützung für DAMA wurde von Joerg (DL1BKE, `jreuter@lykos.tng.oche.de`) entwickelt. Thomas Sailer (`sailer@ife.ee.ethz.ch`) fügte die Unterstützung für BayCom- und SoundModem hinzu. Die AX-25-Utilities werden von Craig Small (`csmall@debian.org`) betreut. Der Linux-Code unterstützt KISS-basierte TNCs (Terminal Node Controllers), die Ottawa PI-Card, die Gracilis PacketTwin-Karte und andere SCC-Karten auf der Basis des Z8530 mit dem allgemeinen SCC-Treiber sowie parallele und serielle BayCom-Modems.

Der SoundModem-Treiber von Thomas Sailer unterstützt den SoundBlaster und die Soundkarten auf der Basis des Crystal-Chipsatzes. Die Anwenderprogramme enthalten ein einfaches PMS (Personal Message System), eine Möglichkeit, Baken auszusenden, ein zeilenorientiertes Programm zum Verbindungsaufbau, 'listen', ein Beispiel, wie man alle AX.25-Pakete an einem Interface mitlesen kann und Programme zur Einrichtung des NetROM- Protokolls. Ebenso sind ein AX.25-Server-Programm zur Verwaltung eintreffender Verbindungen und ein NetROM-Daemon, der den Hauptanteil der NetROM- Funktionalität übernimmt, vorhanden.

2.1 Wie alles zusammenpaßt

Die AX.25-Implementation bei Linux ist noch sehr neu. Obwohl sie an vielen Stellen NOS, BPQ oder anderen Implementationen ähnelt, ist sie keine von diesen. Man kann sie so einrichten, daß sie sich fast wie andere Implementationen verhält, die Konfiguration ist jedoch vollkommen anders. Um beim Verständnis der Einrichtungsweise zu helfen, soll dieser Abschnitt einige strukturelle Eigenschaften beschreiben und darstellen, wie sie sich in die Systemstruktur von Linux einfügen.

Vereinfachte Zeichnung der Protokoll-Ebenen:

```
+----------+-----------+----------+----------+
| AF_AX25  | AF_NETROM | AF_INET  | AF_ROSE  |
|==========|===========|==========|==========|
|          |           |          |          |
|          |           | TCP/IP   |          |
|          |           +-------+  |          |
|          | NetROM            |  |          |
|          +-------------------+--+----------+
|      A  X  .  2  5                         |
+--------------------------------------------+
```

Diese Darstellung soll zeigen, daß NetROM, ROSE und TCP/IP alle direkt auf AX.25 aufgesetzt laufen, aber jedes Protokoll als einzelnes Protokoll an der Programmierschnittstelle behandelt wird. Die »AF_«-Namen sind die Namen für die Adreßfamilie jedes Protokolls, wie sie in den darauf aufsetzenden Programmen verwendet werden. Wichtig ist also die richtige Konfiguration der AX.25-Geräte, bevor man die NetROM-, TCP/IP- oder ROSE-Devices einrichten kann.

Darstellung der Linux-Netzwerk-Implementation:

```
---------+-----------+-------------------++----------+------------------------
Anwender | Programme | call     node     || Daemonen | ax25d   mheardd
         |           | pms      mheard   ||          | inetd   netromd
---------+-----------+-------------------++----------+------------------------
         | Sockets   | open(), close(), listen(), read(), write(), connect()
         |           +-----------+-----------+--------------+-----------------
         |           | AF_AX25   | AF_NETROM |  AF_ROSE     | AF_INET
         +-----------+-----------+-----------+--------------+-----------------
         | Protokolle| AX.25     | NetROM    |  ROSE        | TCP/IP/UDP
Kernel   +-----------+-----------+-----------+--------------+-----------------
         | Devices   | ax0, ax1  | nr0, nr1  | rose0, rose1 | eth0, ppp0...
         +-----------+-----------+-----------+--------------+-----------------
         | Treiber   | KISS, PI2, PacketTwin, SCC, BPQ,     | slip, ppp,
         |           |       SoundModem, BayCom             | ethernet
---------+-----------+--------------------------------------+-----------------
Hardware | PI2-Karte, PacketTwin-Karte, SCC-Karte, Serielle Schnittstelle,
         | SoundBlaster, Ethernet-Karte
---------+--------------------------------------------------------------------
```

Diese Zeichnung ist ein wenig allgemeiner als die erste. Sie soll den Zusammenhang zwischen Anwenderprogrammen, Kernel und Hardware und die Beziehungen von Programmierschnittstelle, Protokoll-Modulen, Kernel-Netzwerk-Devices und Gerätetreibern untereinander verdeutlichen. In dieser Darstellung ist jeder Teil von dem, was darunter steht, abhängig, man muß also von »unten« (Hardware) nach »oben« (Programme) alles der Reihe nach einrichten. Beispiel: Will man das Programm »call« starten, so muß man zunächst die Hardware einrichten, dann sicherstellen, daß der Kernel den passenden Treiber geladen hat, das zugehörige Netzwerk-Device erzeugt wurde und das zur Nutzung durch »call« gewünschte Protokoll im Kernel enthalten ist. Dieser Text wird sich im großen und ganzen an diese Reihenfolge halten.

3 Die Software zu AX.25/NetROM/ROSE

Die AX.25-Software besteht aus drei Komponenten:

- dem Kernel-Quelltext
- den Netzwerk-Konfigurationsprogrammen
- den Utilities.

In den Kernel-Versionen 2.0.35, 2.0.36 und 2.0.37 sind die Treiber für AX.25, NetROM, Z8530 SCC, PI-Karte und PacketTwin standardmäßig in einer aktuellen Version enthalten. Zum Kernel 2.2.x siehe Anmerkungen am Ende dieses Textes.

3.1 Woher bekommt man den passenden Kernel und die Software

3.1.1 Die Kernel-Quelltexte

Diese sind an der üblichen Stelle bei `ftp.kernel.org` zu finden. Empfohlen wird `linux-2.0.37.tar.gz`. Es gibt keinen Grund, eine andere als diese Kernel-Version aus der 2.0.x-Reihe zu verwenden. Wer seinen Kernel von älteren Versionen auf 2.0.37 aktualisieren will, sollte das komplette Quelltextarchiv herunterladen, da es mit Patches auf 2.0.37 zu Problemen kommt, wenn die vorherigen Kernel-Quellen nicht mehr »original« sind.

Im Hinblick auf die Kernel der 2.2.x-Reihe sind die Mirrors von `ftp.kernel.org` die erste Wahl. Die Adresse des zu bevorzugenden Mirrors erhält man, indem man den Landes-Suffix einfügt; aus `ftp.kernel.org` wird für Deutschland demnach `ftp.de.kernel.org` (für Nutzer kommerzieller Onlinedienste `ftp2.de.kernel.org`). Dort findet man die aktuellen Kernel-Archive unter `/pub/linux/kernel/v2.2/`.

3.1.2 Programme für die Netzwerkkonfiguration

Die Net-Tools Die aktuelle Version der Standard-Linux-Netzwerk-Tools unterstützt AX.25, NetROM und ROSE, sie findet sich auf

```
http://www.tazenda.demon.co.uk/phil/net-tools/
```

als `net-tools-1.52.tar.gz`.

Firewall-Konfiguration Die neueste Version des ipfwadm (IP Firewall Administrator) für den Kernel 2.0.37 kann hier bezogen werden:

```
ftp.xos.nl:/pub/linux/ipfwadm
```

Wer eine aktuelle Distribution einsetzt, sollte vor dem Download prüfen, ob diese Pakete nicht bereits installiert bzw. auf der Installations-CD enthalten sind. Wer auf seinem System eine Nettools-Version größer 1.33 vorfindet, braucht sie sich nicht nochmals zu installieren. Gleiches trifft auf ipfwadm zu. Hier ist 2.3.0 die aktuelle Version (wird mit `ipfwadm -h` angezeigt).

Für die Kernel der 2.2.x-Reihe gibt es an Stelle von ipfwadm das Programm ipchains. Zu finden ist es hier:

ftp.starshadow.com:/pub/rustcorp/ipchains/

Das Programm wird auf

http://www.rustcorp.com/linux/ipchains/

vorgestellt.

3.1.3 Die AX.25-Utilities Version 2.1.42a

Die empfohlene Version der AX.25-Utilities für den Kernel 2.0.37 auf libc5-basierten Systemen ist ax25-utils-2.1.42a. Zu finden ist sie hier:

ftp.funet.fi:/pub/ham/unix/Linux/packet/ax25/

Ältere Versionen sollten nicht eingesetzt werden. Binäre Pakete zur direkten Installation unter RedHat 5.2 oder S.u.S.E. 6.0 finden sich unter:

ftp.funet.fi:/pub/ham/unix/Linux/packet/ax25/packages/libc6/rpm

Wichtig ist hierbei, sowohl ax25-utils-2.1.42a-3.i386.rpm als auch ax25-utils-devel-2.1.42a-3.i386.rpm zu installieren, damit auf dem AX.25-Subsystem aufbauende Software kompiliert werden kann.

Dem Debian-Archiv ax25-utils_2.1.42a-6.deb fehlt auf Grund einer Fehlinformation der Verantwortlichen (man nahm Lizenzprobleme an) das Programm smdiag, so daß es nur für diejenigen geeignet ist, die weder BayCom- noch SoundModem verwenden möchten.

Neue Version 0.0.1 Für Kernel 2.2.10 gibt es eine komplett überarbeitete Fassung der AX.25-Utilities und der benötigten Bibliotheken:

ftp.hes.iki.fi:/pub/ham/linux/ax25

Da sich sowohl Software als auch Bibliotheken in einer Alpha-Version befinden und in nächster Zeit weiter entwickelt werden sollen, wurde das Archiv in die Bestandteile ax25-apps-0.0.2.tar.gz, ax25-tools-0.0.3.tar.gz und libax25-0.0.5.tar.gz aufgeteilt, die zur Installation dieser Version vollständig benötigt werden. Weitere Informationen liefert diese WWW-Seite:

http://www.eye-net.com.au/hamradio/

Vorausgesetzt wird ein installiertes XFree86, zumindest aber die Bibliothek libX11, da sich sonst das Programm smdiag nicht kompilieren läßt. Viele Dateien sind gespiegelt, d.h, sie befinden sich auf Servern, die schneller zu erreichen sind als die Originalserver. Am leichtesten findet man derartige Möglichkeiten mit FTPSearch.

4 Die Software zu AX.25/NetROM/ROSE installieren

Um die AX.25-Unterstützung erfolgreich nutzen zu können, muß man erst einen passenden Kernel und anschließend (unter dem neuen Kernel) die AX.25- Utilities installieren.

4.1 Den Kernel kompilieren

Wer sich mit dem Erstellen eines neuen Kernels auskennt, kann diesen Abschnitt überspringen. Wichtig ist die Auswahl der richtigen Optionen beim Kompilieren. Wer noch nicht weiß, wie es geht, sollte hier weiterlesen. Normalerweise befindet sich der Quelltext des Kernels in dem Verzeichnis `/usr/src/linux.` Will man nun einen neuen Kernel installieren, geht man so vor:

```
cd /usr/src
```

Falls man bereits ein Linux-Verzeichnis besitzt und dieses nicht löschen will, kann man es umbenennen:

```
mv linux linux.old
```

Da die ausgepackten Quelltexte etwa 20-30 MB auf der Platte belegen, sollte man sich überlegen, ob man das Verzeichnis nicht doch löscht:

```
rm -r linux
```

Die Quelltextarchive sind relativ zu `/usr/src` gepackt; zum Entpacken gibt man ein:

```
cd /usr/src
tar zxvf /tmp/linux-2.0.37.tar.gz
```

Bei diesem Beispiel haben wir angenommen, daß das Archiv sich in `/tmp` befindet.

Nachdem die Kernel-Quelltexte erfolgreich entpackt wurden, ruft man jetzt das Konfigurationsskript auf:

```
cd /usr/src/linux
make menuconfig
```

Unter X11 kann statt dessen auch

```
make xconfig
```

verwendet werden. Hierfür wird ein installiertes Tcl/Tk in einer aktuellen Version benötigt.

Etwas archaisch, aber bewährt und stabil:

```
make config
```

Im folgenden soll die Vorgehensweise bei `make menuconfig` beschrieben werden. Selbstverständlich lassen sich die beiden anderen Möglichkeiten auch nutzen, die Einstellungen sind entsprechend anzupassen. In jedem Fall werden viele Fragen gestellt, die man mit »y« (yes), »n« (no) oder »m« (Modul) beantworten kann. Einige Eigenschaften lassen sich nämlich auch als Module einrichten, um sie nur bei Bedarf nachzuladen und somit Speicher zu sparen. Der Einfachheit halber wird hier auf diese Möglichkeit nicht explizit eingegangen. Wer Module erstellen will, muß selbst die entsprechenden Veränderungen vornehmen. Weitere Informationen findet man im *Module HOWTO*, das zur Zeit leider nicht mehr betreut wird.

Folgende Optionen sind für AX.25 wichtig:

```
Code maturity level options

[*] Prompt for development and/or incomplete code/drivers

Loadable Module Support

[*] Enable loadable module support (wer Module vorsehen will)
...
[?] Kernel daemon support (e.g. autoload of modules)

General Setup

[*] Networking support

Networking options

[*] TCP/IP networking
[?] IP forwarding/gatewaying
...
[?] IP tunneling
...
[?] IP allow large windows (not recommended if < 16 MB of memory)
...
[*] Amateur Radio AX.25 Level 2
[?] Amateur Radio Net/ROM
[?] Amateur Radio X.25 PLP (ROSE)
```

Für Kernel 2.2.x zusätzlich:

```
[*] Packet socket (CONFIG_PACKET)

Network Device Support

[*] Network Device Support
...
[*] Radio network interfaces
[?] BAYCOM ser12 and par96 driver for AX.25
[?] Soundcard modem driver for AX.25
[?] Soundmodem support for Soundblaster and compatible cards
[?] Soundmodem support for WSS and Crystal cards
[?] Soundmodem support for 1200 baud AFSK modulation
[?] Soundmodem support for 2400 baud AFSK modulation (7.3728 MHz crystal)
[?] Soundmodem support for 2400 baud AFSK modulation (8 MHz crystal)
[?] Soundmodem support for 2666 baud AFSK modulation
[?] Soundmodem support for 4800 baud HAPN-1 modulation
[?] Soundmodem support for 4800 baud PSK modulation
[?] Soundmodem support for 9600 baud FSK G3RUH modulation
[?] Serial port KISS driver for AX.25
[?] BPQ Ethernet driver for AX.25
[?] Gracilis PacketTwin support for AX.25
[?] Ottawa PI and PI/2 support for AX.25
[?] Z8530 SCC KISS emulation driver for AX.25
...
```

Optionen, die mit einem [*] markiert sind, müssen eingeschaltet (mit »y« beantwortet) werden, die mit einem [?] markierten hängen von der zu verwendenden Hardware ab und welche weiteren Optionen einbezogen werden sollen. Einiges davon wird später noch genauer beschrieben, so daß es sinnvoll ist, bei Unklarheiten erst mal weiterzulesen und anschließend wieder zu diesem Abschnitt zurückzukehren. Nachdem die Konfiguration vervollständigt wurde, sollte es möglich sein, den Kernel problemlos zu kompilieren. Siehe dazu auch die Anmerkung zu KISS im Abschnitt 6.1.1. Wichtig ist es, in jedem Fall vor dem Kompilieren die Datei /usr/src/linux/Documentation/CHANGES zu lesen und sicherzustellen, daß die dort erwähnten Programmpakete mindestens in der angegebenen Version installiert sind. Kompiliert wird mit:

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

Der make-Befehl akzeptiert die einzelnen Optionen auf einer Zeile, so daß man auch

```
make dep clean zImage
```

eingeben kann. Wurde der Kernel erfolgreich kompiliert, kann man ihn dorthin bringen, wo er hin muß:

```
cat /usr/src/linux/arch/i386/boot/zImage > /vmlinuz
```

Wer Lilo installiert hat, muß diesen starten:

```
lilo
```

Auf diese Weise erkennt Lilo, daß ein neuer Kernel installiert wurde und bootet diesen beim nächsten Neustart. Ein

```
make zlilo
```

automatisiert die Schritte zImage, Kernelkopie und LILO-Installation. Da hierbei gleichzeitig die Datei System.map aktualisiert wird, ist dieser Befehl von Vorteil. Wer sich aber nicht sicher ist, ob der neue Kernel auch wirklich startet, sollte dann aber eine Bootdiskette bereithalten. Weitere Informationen zu diesem Thema gibt das *Kernel HOWTO*. Wer seinen Kernel mit einem Patch auf die aktuelle Version bringen will, sollte nach dem Patchen mittels

```
cd usr/src/linux
zcat /pfad/zum/patch-x.x.x.gz | patch -p1
```

das Quelltextverzeichnis mit einem

```
find /usr/src/linux -name *.orig | xargs rm
```

aufräumen. Ein

```
find /usr/src/linux -name *.rej -print
```

sollte ohne Ausgaben bleiben, sonst ist mit dem Patchen etwas schiefgelaufen.

4.1.1 Ein Wort zu den Kernel-Modulen

Einsteigern wird zwar nicht empfohlen, die Treiber als Module zu kompilieren, es ist jedoch in der Testphase manchmal recht hilfreich. Das korrekte Funktionieren der Module hängt von passenden Einträgen in der Datei `System.map` ab. Bekommt man beim Laden von Modulen Fehlermeldungen bezüglich »unresolved symbols«, so gibt es dafür zwei Ursachen. Das Problem kann sowohl durch eine nicht zum laufenden Kernel passende `System.map` als auch durch ein nicht aktualisiertes Modul-Verzeichnis hervogerufen werden. Im ersteren Fall hilft das Kopieren der passenden `System.map` nach `/`. In letzterem Fall sollte man das Verzeichnis `/lib/modules/{aktuelle Kernel-Version}` leeren und die Module noch einmal neu erstellen und installieren:

```
cd /usr/src/linux
make modules modules_install
```

Dieser Schritt ist selbstverständlich ebenfalls nach der Erstellung eines neuen Kernels fällig. Anschließend aktualisiert man die Abhängigkeiten der einzelnen Module untereinander mit:

```
depmod -a
```

Weiterhin ist es notwendig, einige Einträge in der Datei `/etc/conf.modules` hinzuzufügen, damit die Module im Bedarfsfall durch den `kerneld` automatisch geladen werden können. Der `kerneld` muß dafür gestartet sein, ein

```
ps fax | grep kerneld
```

sollte das auch anzeigen.

Hier also ein Beispiel:

```
#/etc/conf.modules
#
# Konfigurationsdatei für den kerneld
#
# WICHTIG: Alle genannten Treiber müssen als gleichnamige
# Module in /lib/modules/{Kernelversion} vorliegen!
#
alias net-pf3           ax25
#
# Suche nach einem Treiber unterdrücken:
# (Wichtig bei einigen Programmen der Debian-Distribution)
#
alias net-pf5           off
#
alias net-pf6           netrom
alias net-pf-11         rose
alias tty-ldisc-1       slip
alias tty-ldisc-3       ppp
alias tty-ldisc-5       mkiss
#
# Für jedes Device muß einzeln ein Alias-Eintrag erfolgen:
#
alias bc0               baycom
alias bc1               baycom

alias nr0               netrom
```

```
alias pi0a              pi2
alias pt0a              pt
#
# Für die nächste Zeile den jeweils passenden
# Treiber auswählen:
alias scc0              optoscc
#
alias sm0               soundmodem
alias tun10             newtunnel
#
# Wichtig für Anwernder der SuSE-Distribution, Version 6,
# mit Kernel 2.2.x:
#
alias char-major-4      serial
alias char-major-5      serial
alias char-major-6      lp
alias net-pf-17         af_packet
#
# ACHTUNG: Die folgenden Einträge werden nur für Kernel
# ab 2.2.0 benötigt!
alias bcsf0          baycom_ser_fdx
alias bcsh0          baycom_ser_hdx
alias bcp0           baycom_par
alias bce0           baycom_epp
#
# Wer den neuen 6pack-Treiber nutzen will, braucht:
alias tty-ldisc-7 6pack
alias sp0 6pack
```

Insbesondere diejenigen, die BayCom- oder SoundModems verwenden wollen, sollten die entsprechenden Treiber als Module vorsehen. Dazu folgen später noch weitere Erklärungen.

4.1.2 Was ist neu in den verschiedenen Kernel-Versionen?

In den Kerneln der Versionen ab 2.0.35 und 2.1.xx sind verbesserte Versionen von fast allen Protokollen und Treibern enthalten. Die wichtigsten Verbesserungen sind:

modularisiert

Alle Protokolle und Treiber liegen auch als Module vor, so daß man sie mit `insmod` laden und mit `rmmod` entfernen kann, wenn man es wünscht. Damit reduzieren sich die Speicheranforderungen des Kernels für selten genutzte Module, Entwicklung und Fehlersuche werden um einiges einfacher. Doch, wie bereits gesagt, die Konfiguration wird etwas schwieriger.

Alle Treiber sind jetzt Netzwerktreiber

Alle Devices wie PacketTwin, SCC und BayCom usw. bieten jetzt ein normales Netzwerkinterface an; sie erscheinen den Programmen wie der Ethernet-Treiber, nicht mehr wie KISS-TNCs. Wer will, kann mit dem neuen Utility net2kiss ein KISS-Interface auf diese Devices aufsetzen.

Fehler beseitigt

Viele Fehler wurden beseitigt, neue Eigenschaften wurden hinzugefügt. Eine wichtige Erweiterung ist das ROSE-Protokoll.

4.2 Die Tools zur Netzwerkkonfiguration

Nachdem der Kernel kompiliert wurde (und auch startet), sollten die neuen Netzwerk-Tools kompiliert werden. Sie erlauben Veränderungen an der Konfiguration der Netzwerk-Devices und das Hinzufügen von Routen zur Routing-Tabelle.

Informationen zur derzeit aktuellen Version (`net-tools-1.52.tar.gz`) finden sich hier:

```
http://www.tazenda.demon.co.uk/phil/net-tools/
```

Die Net-Tools werden von Phil Blundell (`pb@nexus.co.uk`) betreut, Bernd Eckenfels (`Bernd.Eckenfels@inka.de`) ist der Co-Maintainer". Seine Net-Tools-Page findet man dieser Adresse:

```
http://sites.inka.de/sites/lina/linux/NetTools/
```

4.2.1 Die Standardfassung der Net-Tools erstellen

Nicht vergessen: nach dem Entpacken die Datei `Release` lesen und den dort stehenden Anweisungen folgen. Mit folgenden Schritten installiert man die Net-Tools:

```
cd /usr/src
tar zxvf /tmp/net-tools-1.52.tar.gz
cd net-tools-1.52
make config
```

An dieser Stelle werden, ähnlich wie bei der Kernel-Konfiguration, einige Fragen gestellt. Man muß nun sicherstellen, alle Protokolle und Typen von Netzwerk-Devices einzubinden, die genutzt werden sollen.

Fragen, auf die man keine Antwort weiß, sollte man mit »y« beantworten. Wichtig: Die Protokolle müssen vorher auch bei der Kernel-Konfiguration eingeschaltet worden sein. Hat man zum Beispiel »AppleTalk« beim Kernel deaktiviert, so darf man die entsprechende Frage bei den Net-Tools nicht mit »y« beantworten. Beachtet man dies nicht, so können die Tools nicht kompiliert werden. Nach beendeter Kompilierung werden die Programme mit

```
make install
```

an passender Stelle installiert.

4.2.2 Ipfwadm und ipchains

Wer die IP-Firewall-Möglichkeiten nutzen will, benötigt das aktuelle Tool `ipfwadm` zur Administration der Firewalls. Das ältere Programm `ipfw` arbeitet mit den neueren Kernels nicht zusammen und wird daher durch `ipfwadm` ersetzt. Folgende Optionen beim Kernel-Kompilieren müssen gesetzt sein:

```
Networking options

[*] Firewalling
[*] IP Firewalling
```

Mit folgenden Befehlen wird `ipfwadm` kompiliert:

```
cd /usr/src
tar zxvf /tmp/ipfwadm-2.3.0.tar.gz
cd ipfwadm-2.3.0
make install
cp ipfwadm.8 /usr/man/man8
cp ipfw.4 /usr/man/man4
```

Nutzer eines Kernels der 2.2.x-Reihe benötigen an Stelle des `ipfwadm` das Programm `ipchains`. Wenn die Datei `/proc/net/ip_fwchains` existiert, dann ist der Kernel mit `ipchains` funktionsfähig. Andernfalls muß er neu übersetzt werden; siehe dazu auch das im `ipchains`-Archiv enthaltene HOWTO. Das Kompilieren ist auch hier recht einfach:

```
cd /usr/src
tar zxvf /tmp/ipchains-1.3.9.tar.gz
cd ipchains-1.3.9
make install
```

Wer diese Pakete bereits in seiner Distribution vorfindet, braucht sie natürlich nicht nochmals zu installieren.

4.3 Die Anwender- und Utilityprogramme zu AX.25

Nachdem der neue Kernel erfolgreich kompiliert und mit diesem neu gestartet wurde, müssen jetzt die Anwenderprogramme kompiliert werden. Das geht mit folgenden Befehlen:

```
cd /usr/src
tar zxvf /tmp/ax25-utils-2.1.42a.tar.gz
cd ax25-utils-2.1.42a
make config
make
make install
```

Debian-Nutzer müssen vorher die Verzeichnisse `/usr/include/asm` und `/usr/include/linux` umbenennen und an ihrer Stelle Links auf den Kernel-Quelltext legen:

```
cd /usr/include
mv linux linux.debian
mv asm asm.debian
ln -s /usr/src/linux/include/linux linux
ln -s /usr/src/linux/include/asm asm
```

Man sollte in diesem Zusammenhang kontrollieren, daß `/usr/src/linux/include/asm` ein Link auf `asm-i386` ist. Die Dateien werden im Verzeichnis `/usr` in die Unterverzeichnisse `bin`, `sbin` und `man` installiert. Werden die AX.25-Utilities zum allerersten Mal installiert, d.h., sie befanden sich noch nie vorher auf dem System, so verwendet man auch den Befehl

```
make installconf
```

der einige Beispiel-Konfigurationsdateien unter `/etc/ax25` installiert, die als Grundlage für die eigene Konfiguration dienen können. Erscheinen Bildschirmmeldungen wie

```
gcc-Wall -Wstrict-prototypes -o2 -I../lib -c call.c
call.c: In funktion '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)
```

sollte man doppelt überprüfen, ob das ncurses-Paket installiert ist. Das Konfigurationsskript versucht, ncurses an den üblichen Stellen zu finden, doch auf manchen Systemen ist ncurses nicht richtig installiert und es kann dann das Paket nicht finden. Wer Programme kompilieren will, die die AX.25-Bibliotheken nutzen, und dabei Fehlermeldungen erhält, kann die Bibliotheken mit

```
make installib
```

korrekt installieren.

4.3.1 Hinweis für die Benutzer libc6-basierter Systeme (S.u.S.E. 6.0, RedHat 5.2, Debian 2.0...)

Die im vorigen Abschnitt besprochenen AX-25-Utils lassen sich auf Systemen, die die libc6 bzw. glibc 2.x verwenden, nicht mehr kompilieren.

Unter folgenden Adressen

- `ftp.suse.com:/pub/projects/ham/`
- `ftp.funet.fi:/pub/ham/unix/Linux/packet/ax25/packages/libc6/rpm`

findet man die Pakete

- `ax25util-2.1.42a-0.i386.rpm(ftp.suse.com)`
- `ax25util-2.1.42a-3.i386.rpm(ftp.funet.fi)`
- `ax25util-devel-2.1.42a-0.i386.rpm(ftp.suse.com)`
- `ax25util-devel-2.1.42a-3.i386.rpm(ftp.funet.fi)`

die zur Verwendung auf den erwähnten Systemen gedacht sind. Auf

```
ftp.funet.fi:/pub/ham/unix/Linux/packet/ax25/packages/libc6/deb/
```

findet sich das Paket `ax25-utils_2.1.42a-6.deb` für die Nutzer der Debian-Distribution. Da die Pakete bereits vorkompilierte Versionen der AX-25-Utilities enthalten, ist ein Selbstkompilieren nicht mehr unbedingt notwendig. Wichtig ist jedoch, auf die korrekte Installation der Header-Dateien zu achten (`ax25util-devel-2.1.42a-0.i386.rpm`), da sich sonst einige AX.25-Anwendungen, wie PR-Terminalprogramme, nicht kompilieren lassen.

Dem Debian-Paket fehlt das Programm `smdiag`. Deshalb sollte es nur eingesetzt werden, wenn kein BayCom- oder SoundModem verwendet werden soll. Wer dieses Programm benötigt, sollte das RPM-Archiv der AX.25-Utilities ebenso wie das Development-Archiv mit dem Tool `alien` nach `*.deb` konvertieren und mit `dpkg` installieren.

Eine weitere Möglichkeit stellt das auf der Homepage von Patrick Ouellette (`pouellet@eng.utoledo.edu`) verfügbare Archiv `ax25-utils-2.1.42.kb8pym.tar.gz` dar, das Ergebnis von Patricks Modifikationen ist, die er in dem Patch `ax25-utils-2.1.42.kb8pym.diff` zusammengefaßt hat. Dieser Patch sollte mit dem Standardarchiv `ax25-utils-2.1.42a.tar.gz` funktionieren. Patrick weist jedoch ausdrücklich darauf hin, daß es sich hierbei um Testversionen handelt. Da zunächst nur Wert auf Kompilierbarkeit gelegt wurde, sind manche Programme evtl. nicht funktionsfähig.

4.3.2 Die aktuellen AX.25-Utilities 0.0.1

Die neue Version der AX.25-Utilities wird erstellt, indem die drei Archive

- `ax25-apps-0.0.2.tar.gz`
- `ax25-tools-0.0.3.tar.gz`
- `libax25-0.0.5.tar.gz`

in das Verzeichnis `/usr/src` entpackt und von dort aus installiert werden:

```
cd /usr/src
tar -zxvf archivname
cd {archivname ohne die Endung .tar.gz}
./configure
make install
```

5 Ein Hinweis zu Rufzeichen, Adressen und all diesen Dingen

Jeder AX.25- oder NetROM-Port muß ein eigenes Rufzeichen/SSID besitzen. Diese werden in den weiter unten beschriebenen Konfigurationsdateien eingestellt. Bei manchen AX.25-Implementationen wie NOS und BPQ kann man jedem AX.25- und NetROM-Port das gleiche Rufzeichen zuteilen.

Aus etwas komplizierten technischen Gründen ist das unter Linux nicht möglich. In der Praxis ist das nicht so ein großes Problem, wie es zunächst scheint. Das bedeutet, daß es einige Dinge gibt, die bei der Konfiguration beachtet werden müssen:

1. Jeder AX.25- und NetROM-Port muß sein eigenes Rufzeichen/SSID bekommen.
2. TCP/IP nutzt das Rufzeichen des Ports, über den es ausgesendet oder empfangen wird, d.h., das in Punkt 1. angegebene Rufzeichen.
3. NetROM nutzt das in seiner speziellen Konfigurationsdatei eingestellte Rufzeichen, dieses wird allerdings nur dann verwendet, wenn eine Verbindung zu einer anderen NetROM-Station besteht, es ist *nicht* das Rufzeichen, welches AX.25-Nutzer verwenden müssen, wenn sie den Node rufen wollen. Mehr dazu später.
4. ROSE nutzt standardmäßig das Rufzeichen des AX.25-Ports, es sei denn, es wurde mit dem `rsparms`-Befehl ein anderes Rufzeichen eingestellt. Wurde mit `rsparms` ein Rufzeichen vergeben, dann verwendet ROSE dieses auf allen (ROSE-)Ports.
5. Andere Programme, wie der `ax25d`, können zum Mithören jedes Rufzeichen verwenden, das sie wollen, und diese können auch für verschiedene Ports genutzt werden.
6. Wenn man das Routing sorgfältig einstellt, kann man allen Ports dieselbe IP-Adresse zuordnen.

5.1 Was bedeuten T1, T2, N2,...?

Nicht jede AX.25-Implementation ist ein TNC2. Linux verwendet eine Nomenklatur, die etwas anders ist als die von einem TNC gewohnte. In der folgenden Tabelle sind die einstellbaren Parameter und ihre Bedeutung aufgelistet, so daß man hier immer wieder nachschlagen kann, wenn sie im Text erwähnt werden.

Linux	TAPR TNC	TNC2	Beschreibung
T1	FRACK	F	(Frame Acknowledgement Timer) Gibt an, wie lange gewartet wird, bevor ein unbestätigtes Paket noch mal ausgesendet wird
T2	RESPTIME	@T2	Minimale Zeit, die auf ein weiteres Paket gewartet wird, bevor Empfangsbestätigung gesendet wird
T3	CHECK	@T3	Zeit, die gewartet wird, bevor der Link überprüft wird (Polling)
N2	RETRY	N	Zahl der Wiederholungen der Aussendung eines Paketes, bevor die Verbindung als zusammen- gebrochen angesehen wird
Idle			Zeit, die eine Verbindung unbenutzt sein darf, bis sie beendet wird (Link Timeout)
Window	MAXFRAME	O	Maximale Anzahl unbestätigter Pakete

5.2 Zur Laufzeit konfigurierbare Parameter

In den 2.1.xx-Kernels, den 2.0.xx-Kernels mit Module-xx-Patch und Kernels ab 2.0.35 lassen sich viele Parameter auch zur Laufzeit einstellen. Schaut man sich die Dateien unter /proc/sys/net an, so wird man viele Dateien mit selbsterklärenden Namen finden, die verschiedene Parameter der Netzwerkkonfiguration beschreiben. Jedes der Verzeichnisse unter /proc/sys/net/ax25 repräsentiert einen AX.25-Port, wobei dessen Name vom Portnamen abhängt. Die folgenden Dateien sind unter /proc/sys/net/ax25/<portname>/ zu finden:

Dateiname	Bedeutung	Wert	Voreinstellung
ip_default_mode	voreingestellter IP-Modus	0=DG 1=VC	0
ax25_default_mode	voreingestellter AX.25-Modus	0=Normal, 1=Erweitert	0
backoff_type	Backoff	0=Linear, 1=Exponentiell	1
connect_mode	Verbindungsstatus	0=nein, 1=ja	1
standard_window_size	Standard-Maxframe	1 <= O <= 7	2
extended_window_size	Erweitertes Maxframe	1 <= O <= 63	32
t1_timeout	T1-Timer	1s <=T1<= 30s	10 s
t2_timeout	T2-Timer	1s <=T2<= 20s	3 s
t3_timeout	T3-Timer	0s <=T3<= 3600s	300 s
idle_timeout	Link-Timeout	0min <=idle	20 min

```
maximum_retry_count   Anzahl Retries (N)       1 <= N <= 31      10
maximum_packet_length AX.25-Paketlänge         1 <=Länge<= 512   256
----------------------------------------------------------------------------
```

In dieser Tabelle sind die Werte für T1, T2 und T3 in Sekunden, für den idle-Timer (Link-Timeout) in Minuten angegeben, es muß aber beachtet werden, daß die Werte in dem sysctl-Interface in internen Einheiten gezählt werden. Diese entsprechen der Zeit in Sekunden * 10, so daß eine Schrittweite von 1/10 Sekunde möglich wird. Bei Zeitgebern, die einen Wert von 0 erlauben (z.B. T3 und idle), bedeutet 0, daß sie ausgeschaltet sind. In `/proc/sys/net/netrom` finden sich folgende Dateien:

```
Dateiname                                    Wert              Voreinstellung
----------------------------------------------------------------------------
default_path_quality                                           10
link_fails_count                                               2
network_ttl_initialiser                                        16
obsolescence_cont_initialiser                                  6
routing_control                                                1
transport_acknowledge_delay                                    50
transport_busy_delay                                           1800
transport_maximum_tries                                        3
transport_requested_window_size                                4
transport_timeout                                              1200
----------------------------------------------------------------------------
```

In `/proc/sys/net/rose` sieht die Struktur so aus:

```
Dateiname                                    Wert              Voreinstellung
----------------------------------------------------------------------------
acknowledge_hold_back_timeout                                  50
call_request_timeout                                           2000
clear_request_timeout                                          1800
link_fail_timeout                                              1200
maximum_virtual_circuits                                       50
reset_request_timeout                                          1800
restart_request_timeout                                        1800
routing_control                                                1
window_size                                                    3
----------------------------------------------------------------------------
```

Um einen Parameter einzustellen, muß man den gewünschten Wert in die entsprechende Datei schreiben, um zum Beispiel die Maxframe-Anzahl für ROSE zu prüfen und einzustellen, geht man so vor:

```
cat /proc/sys/net/rose/window_size

Bildschirmausgabe: 3
echo 4 > /proc/sys/net/rose/window_size
cat /proc/sys/net/rose/window_size

Bildschirmausgabe: 4
```

6 Einen AX.25-Port einrichten

Jede der AX.25-Anwendungen liest die Parametereinstellungen für die verschiedenen AX.25-Ports aus einer speziellen Konfigurationsdatei. Für reine AX.25-Ports ist dies die Datei `/etc/ax25/axports`. Sie muß für jeden auf dem

System verwendeten AX.25-Port einen Eintrag erhalten.

6.1 Das AX.25-Netzwerk-Device erstellen

Das Netzwerk-Device ist das, was aufgelistet wird, wenn man `ifconfig` startet. Es sind die Objekte, an die der Linux-Kernel Netzwerkdaten sendet und von denen er sie empfängt. Fast immer ist das Netzwerk-Device mit einem physikalischen Port verbunden, in manchen Situationen ist dies aber nicht notwendig. Das Netzwerk-Device steht in direkter Beziehung zu einem Gerätetreiber. Der Linux-AX.25-Code enthält einige Gerätetreiber. Der gebräuchlichste ist sicher der KISS-Treiber, weiterhin gibt es noch SCC-Treiber, den BayCom- Treiber und den SoundModem-Treiber.

Jeder dieser Treiber erzeugt ein Netzwerk-Device, wenn er gestartet wird. Wichtiger Hinweis für Nutzer eines 2.2.x-Kernels: Wird ein Kernel der 2.2.x-Reihe verwendet, so ist jedem AX.25-Netzwerk-Device eine IP-Adresse zuzuordnen, auch wenn damit kein TCP/IP gefahren werden soll. Im einfachsten Fall geschieht das mit `ifconfig {Devicename} {IP-Adresse}`:

```
ifconfig bcsf_0 44.136.8.5 netmask 255.255.255.0 up
```

6.1.1 KISS

Optionen bei der Kernel-Kompilierung:

```
General Setup

[*] Networking support

Network Device Support

[*] Network Device Support
...
[*] Radio network interfaces
...
[*] Serial port KISS driver for AX.25
```

Die häufigste Konfiguration ist sicher ein KISS-TNC an der seriellen Schnittstelle. Dieser muß vorkonfiguriert und an die Schnittstelle angeschlossen sein. Der TNC kann mit einem Programm wie Minicom oder Seyon in den KISS-Modus gebracht werden.

Um ein KISS-Device zu erzeugen, wird das Programm `kissattach` verwendet: (Annahmen: TNC hängt an `/dev/ttyS0` (COM1) und der vorgesehene Port in `/etc/ax25/axports` heißt radio)

```
/usr/sbin/kissattach /dev/ttyS0 radio
kissparms -p radio -t 100 -s 100 -r 25
```

Damit wird ein KISS-Netzwerk-Device erzeugt. Diese Devices erhalten die Namen `ax0` - `ax9`. Beim ersten Aufruf erzeugt `kissattach` `ax0`, beim zweiten `ax1` usw..

Jedes Kiss-Device hat eine zugehörige serielle Schnittstelle. Mit `kissparms` lassen sich verschiedene Parameter des Kiss-Device einstellen. Das oben dargestellte Beispiel erzeugt ein Kiss-Device, welches die erste serielle Schnittstelle und den Eintrag »radio« in der Datei `/etc/ax25/axports` nutzt. Anschließend wird es auf ein TXDelay und eine Slottime von 100 Millisekunden und einen Persistence-Wert von 25 eingestellt. Weitere Informationen geben die Hilfeseiten der einzelnen Programme.

Erscheint die Fehlermeldung

```
kissattach: TIOCSETD: Invalid argument
```

so sollte man nochmals prüfen, ob der Serial port KISS driver auch wirklich in den Kernel einkompiliert oder als Modul geladen wurde. Der Treiber kann fest einkompiliert werden (`CONFIG_MKISS=y` in `/usr/src/linux/config`) wenn die AX-25-Unterstützung (siehe Abschnitt 4.1) ebenfalls fest einkompiliert wurde. Andernfalls muß er als Modul kompiliert werden, oder die Erstellung des neuen Kernels würde fehlschlagen.

Einrichtung von Dual-Port-TNCs

Das `mkiss`-Utility, das bei den AX.25-Utilities dabei ist, erlaubt die Nutzung beider Modems an einem Dual-Port-TNC. Die Konfiguration ist recht einfach. Der Multiport-TNC wird an eine serielle Schnittstelle des Rechners angeschlossen. Die anschließende Konfiguration läßt ihn dann als mehrere einzelne seriell angeschlossene TNCs erscheinen.

Das Ganze muß *vor* der AX.25-Konfiguration durchgeführt werden. Die Geräte, die dann einzurichten sind, sind Pseudo-TTY-Interfaces, (`/dev/ttyq*`) und nicht die eigentliche serielle Schnittstelle. Mit Pseudo-TTY wird eine Art Röhre (Pipe) erzeugt, durch die Programme, die mit TTY-Geräten Daten austauschen, mit anderen Programmen, die ebenfalls für Datenaustausch mit TTY-Geräten entwickelt wurden, kommunizieren können.

Jede dieser Röhren hat ein »Master«- und ein »Slave«-Ende. Die »Master«-Enden heißen `/dev/ptyq*`, die »Slave«-Enden `/dev/ttyq*`. Jeder Master hat einen Slave, `/dev/ptyq0` ist also der Master einer Pipe, deren Slave `/dev/ttyq0` ist. Man muß das »Master«-Ende einer Pipe vor dem »Slave«-Ende öffnen. `mkiss` nutzt diesen Mechanismus, um ein einzelnes serielles Device auf mehrere virtuelle Devices aufzuteilen.

Hat man z.B. einen Dual-Port-TNC an eine serielle Schnittstelle `/dev/ttyS0` mit 9600 bps angeschlossen, erzeugen die Befehle

```
/usr/sbin/mkiss -s 9600 /dev/ttyS0 /dev/ptyq0 /dev/ptyq1
/usr/sbin/kissattach /dev/ttyq0 port1
/usr/sbin/kissattach /dev/ttyq1 port2
```

zwei Pseudo-TTY-Devices, die beide wie ein normaler Single-Port-TNC erscheinen.

Nun lassen sich `/dev/ttyq0` und `/dev/ttyq1` wie serielle Schnittstellen mit daran angeschlossenen konventionellen KISS-TNCs behandeln. Das heißt, man verwendet `kissattach` wie oben beschrieben, für die AX.25-Ports port1 und port2. Auf die serielle Schnittstelle selbst kann kein `kissattach` angewendet werden, da `mkiss` diese ja bereits nutzt. Der Befehl `mkiss` kennt einige Optionen:

-c

schaltet die Erzeugung einer Prüfsumme für jedes KISS-Paket ein. Die meisten KISS-Implementationen, außer dem G8BPG KISS-ROM, unterstützen dies jedoch nicht.

-s

<Baudrate> stellt die Baudrate der seriellen Schnittstelle ein.

-h

schaltet den Hardware-Handshake ein (Voreinstellung: Aus). Wird von den meisten KISS-Implementationen nicht unterstützt.

-l

schaltet eine Mitschrift (logging) in die syslog-Logdatei ein.

6.1.2 BayCom

Folgende Optionen zur Kernel-Kompilierung sind wichtig:

```
Code maturity level options

[*] Prompt for development and/or incomplete code/drivers

General Setup
[*] Networking support
...

Network Device Support

[*] Radio network interfaces
[*] BAYCOM ser12 and par96 driver for AX.25
```

Für erste Tests sollte der Treiber mit »m« als Modul kompiliert werden. Thomas Sailer (`sailer@ife.ee.ethz.ch`) entwickelte trotz des weitverbreiteten Glaubens, es würde nicht sonderlich gut funktionieren, eine BayCom-Unterstützung für Linux.

Sein Treiber unterstützt die seriellen Ser12, die parallelen Par96 und die verbesserten PicPar-Modems. Informationen über diese Modems erhält man auf der WWW-Seite des BayCom-Teams unter folgender URL:

```
http://www.baycom.de
```

Der erste Schritt ist, herauszufinden, welche I/O-Adresse und IRQ die Schnittstelle verwendet, an die das BayCom-Modem angeschlossen ist. Der BayCom-Treiber muß mit diesen Werten konfiguriert werden. Ist dies geschehen, erzeugt der Treiber Netzwerk-Devices mit den Namen `bc0`, `bc1`, `bc2` usw..

In den Kerneln der 2.2.x-Reihe wurden die Bezeichnungen für das Baycom-Modul und die von ihm erzeugten Devices geändert. Es gibt hier für Half- und Fullduplex getrennte Treiber:

```
Modul-Name           Funktion                                  Device
-----------------------------------------------------------------------------
baycom_ser_fdx       serielles BayCom-Modem, Fullduplex und    bcsf0..bcsf3
                     Halfduplex, umschaltbar,
                     wählbare Baudrate

baycom_ser_hdx       serielles BayCom-Modem, nur Halfduplex,   bcsh0..bcsh3
                     nur 1200 Baud

baycom_par           paralleles PicPar- und Par96-Modem,       bcp0..bcp3

baycom_epp           EPP-Modem                                 bce0..bce3
                     (Treiber noch in Entwicklung!)
```

Anstelle von

```
insmod baycom
```

schreibt man also

```
insmod baycom_ser_fdx
```

und konfiguriert `bcsf0` statt `bc0`.

Das sieht dann z.B. so aus:

```
insmod baycom_ser_fdx mode="ser12*" iobase=0x3f8 irq=4
sethdlc -i bcsf0 -p mode "ser12*" io 0x3f8 irq 4
```

Ausführlichere Informationen zu den neuen Treibern können in der Datei `/usr/src/linux/Documentation/networking/baycom.txt` nachgelesen werden.

Die Parameter der verwendeten Schnittstelle können mit dem Utility `sethdlc` eingestellt werden, hat man nur ein BayCom-Modem installiert, so kann man die Parameter auf der Kommandozeile für insmod angeben, wenn der als Modul eingerichtete Treiber geladen wird.

Als Beispiel eine einfache Konfiguration. Zunächst wird der normale serielle Treiber für die erste Schnittstelle (COM1) abgeschaltet, dann der BayCom-Treiber für ein serielles 1200-Baud-Modem an COM1 eingerichtet und die Software-DCD eingeschaltet:

```
setserial /dev/ttyS0 uart none
insmod hdlcdrv
insmod baycom mode="ser12*" iobase=0x3f8 irq=4
```

Wichtig: Einige Schnittstellenbausteine bereiten Probleme im Zusammenhang mit dem BayCom-Treiber. Dieser wird zwar geladen, kann aber nicht auf die Schnittstelle zugreifen. Dies betrifft insbesondere viele der neueren 16550A-UARTs, wie sie auf Pentium-Motherboards oft eingebaut sind. Benutzer eines Kernel 2.2.x können in einem solchen Fall an Stelle von `baycom_ser_fdx` `baycom_ser_hdx` probieren, da dieser Treiber in anderer Weise auf die Hardware zugreift.

Wer nun keine zusätzliche Schnittstellenkarte mit 8250 oder 16450 UART vorsehen will, der sollte vor die erste `setserial`-Zeile des Beispiels einfügen:

```
setserial /dev/ttyS0 uart 16550A skip_test
```

Weiterhin stört der Linux-Treiber für die parallele Schnittstelle die korrekte Funktion des BayCom-Treibers. Man sollte daher auf den »parallel printer support« im Kernel verzichten oder diesen als Modul kompilieren, damit er bei Notwendigkeit via `rmmod` entfernt werden kann:

```
rmmod lp
```

Das komplette Skript sieht dann etwa so aus:

```
#!/bin/sh
rmmod lp
setserial /dev/ttyS0 uart 16550A skip_test
sleep 3
setserial /dev/ttyS0 uart none
insmod hdlcdrv
insmod baycom mode="ser12*" iobase=0x3f8 irq=4
```

Damit sollte der Treiber funktionieren, was man mit `sethdlc -d` einfach nachprüfen kann. Der Wert hinter `dbg2` sollte etwa 2000-3000 sein und sich ständig ändern. Die Probleme mit dem Schnittstellenbaustein sind ausschließlich auf Schwierigkeiten des Linux-BayCom-Treibers bei der Hardwareinitialisierung zurückzuführen und deshalb von der eingesetzten Modemschaltung weitestgehend unabhängig. Für den Test mit `sethdlc` braucht das Modem nicht angeschlossen zu sein.

Ein Par96-Modem am Parallelport LPT1 mit Hardware-DCD richtet man so ein:

```
insmod hdlcdrv
insmod baycom mode="par96" iobase=0x378 irq=7 options=0
```

Dies ist aber nicht unbedingt der beste Weg. `sethdlc` arbeitet genau so gut mit einem Modem wie mit mehreren. In der Hilfeseite (`man sethdlc`) findet man alle Details, einige Beispiele sollen diesen Aspekt hier verdeutlichen. Es wird angenommen, das BayCom-Modul ist mit

```
insmod hdlcdrv
insmod baycom
```

bereits geladen oder als Treiber in den Kernel einkompiliert. Man kann das Netzwerk-Device bc0 nun einrichten:

- als Parallelport-Modem an LPT1 mit Software-DCD:

  ```
  sethdlc -p -i bc0 mode par96 io 0x378 irq 7
  ```

- als serielles Modem an COM1:

  ```
  sethdlc -p -i bc0 mode "ser12*" io 0x3f8 irq 4
  ```

AX.25-Kanalzugriffsparameter Die AX.25-Kanalzugriffsparameter entsprechen den Parametern ppersist, txdelay und slottime. Wiederum wird dazu `sethdlc` verwendet. Genaueres steht wiederum in der Hilfeseite, aber ein weiteres Beispiel kann nicht schaden. Wir setzen also das oben begonnene Skript fort, indem wir den BayCom-Treiber mit TXDelay 200 ms, SlotTime 100 ms PPersist 40 und Half-Duplex einrichten:

```
sethdlc -i bc0 -a txd 200 slot 100 ppersist 40 half
```

Alle Zeitwerte werden in Millisekunden angegeben.

Die AX.25-Unterstützung des Kernels für die Nutzung des BayCom-Device einrichten

Der BayCom-Treiber erzeugt Standard-Netzwerk-Devices, die der Kernel-AX.25-Code nutzen kann. Damit ist die Konfiguration fast dasselbe wie bei einer PI- oder PacketTwin-Karte. Zunächst gibt man dem BayCom-Device ein Rufzeichen:

```
/sbin/ifconfig bc0 hw ax25 VK2KTJ up
```

Einige Versionen von `ifconfig` unterstützen den eben angegebenen Weg nicht. Man kann dann so vorgehen:

```
/sbin/ifconfig bc0 up
axparms -setcall bc0 VK2KTJ up
```

Als nächstes wird in der Datei `/etc/ax25/axports` ein Eintrag für BayCom hinzugefügt. Die Verbindung des Eintrags zum entsprechenden Netzwerk-Device geschieht über das eingestellte Rufzeichen.

Verwendet ein Programm den Eintrag mit dem für BayCom vergebenen Rufzeichen, so wird das BayCom-Device angesprochen. Das neue AX.25-Device kann nun ganz normal verwendet werden, es läßt sich für TCP/IP einrichten, man kann es dem `ax25d` hinzufügen und NetROM oder ROSE darüber laufen lassen.

6.1.3 SoundModem

Folgende Optionen bei der Kernel-Kompilierung sind wichtig:

```
Code maturity level options

[*] Prompt for development and/or incomplete code/drivers

General Setup

[*] Networking support

Network Device Support

[*] Radio network interfaces
...
[*] Soundcard modem driver for AX.25
[?] Soundmodem support for Soundblaster and compatible cards
[?] Soundmodem support for WSS and Crystal cards
[?] Soundmodem support for 1200 baud AFSK modulation
[?] Soundmodem support for 2400 baud AFSK modulation (7.3728 MHz crystal)
[?] Soundmodem support for 2400 baud AFSK modulation (8 MHz crystal)
[?] Soundmodem support for 2666 baud AFSK modulation
[?] Soundmodem support for 4800 baud HAPN-1 modulation
[?] Soundmodem support for 4800 baud PSK modulation
[?] Soundmodem support for 9600 baud FSK G3RUH modulation
```

Thomas Sailer (`sailer@ife.ee.ethz.ch`) entwickelte einen neuen Treiber für den Kernel, mit dem man die Soundkarte als Modem nutzen kann. Schließt das Funkgerät an die Soundkarte an, um damit Packet zu spielen! Thomas empfiehlt mindestens einen 486DX2/66, wenn man diese Software verwenden will, da die gesamte digitale Signalverarbeitung von der CPU übernommen wird.

Der Treiber kann im Moment 1200 bps AFSK, 2400 bps AFSK, 4800 bps HAPN, 4800 bps PSK und 9600bps FSK (G3RUH-kompatibel) emulieren. Zur Zeit werden nur SoundBlaster- und Windows Sound System-kompatible Karten unterstützt. Da die Soundblaster-Emulation inzwischen selbst bei Billig-Karten recht gut geworden ist, lohnt sich ein Test in jedem Fall. Soundkarten, die vom Linux-Sound-Treiber nicht unterstützt werden, sollten unter DOS initialisiert werden. Dazu startet man ein Minimal-DOS, in dessen Konfigurationsdateien lediglich die Soundkartentreiber aufgerufen werden, und lädt anschließend Linux via LOADLIN. PCI-Soundkarten (z.B. SoundBlaster PCI64) werden derzeit noch *nicht* unterstützt.

Die Soundkarten benötigen eine kleine Zusatzschaltung zur Ansteuerung der PTT, Informationen dazu findet man auf Thomas Soundmodem-PTT-Seite:

```
<htmlurl url="http://www.ife.ee.ethz.ch/~sailer/pcf/ptt_circ/ptt.html"
         name="http://www.ife.ee.ethz.ch/~sailer/pcf/ptt_circ/ptt.html">
```

Es gibt einige Möglichkeiten für die PTT-Schaltung: die Soundausgabe von der Karte auswerten oder die Ausgabe von der seriellen, parallelen oder MIDI- Schnittstelle zu nutzen. Die Webseite bietet für jede Option Beispielschaltungen.

Wer die »2400 baud« Betriebsarten nutzen will, beachte eine kleine Besonderheit: Ursprünglich kam man auf 2400 baud, indem man den herkömmlichen 1200 baud-Modems auf der Basis des TCM3105 (siehe dazu auch `http://www.ardos.de/gerd/tcm3105.html`) einen Quarz mit höherer Frequenz verpaßte. Zwei Quarzfrequenzen sind üblich: 7.3728 und 8.0 MHz. Bevor man sich für eine davon entscheidet, sollte man in Erfahrung bringen, womit die Gegenstationen arbeiten, da die Wahl der falschen Frequenz die Verbindung stark beeinträchtigen bzw. unmöglich machen kann. Der SoundModem-Treiber erzeugt Netzwerk-Devices mit Namen `sm0`, `sm1`, `sm2` usw., wenn er eingerichtet wurde.

Der SoundModem-Treiber beansprucht die gleichen Ressourcen wie Linux-Sound- und Parallelport-Treiber. Wenn man also den SoundModem-Treiber verwenden möchte, dürfen sowohl der Linux-Sound-Treiber als auch der Treiber für die parallele Schnittstelle nicht geladen sein. Natürlich lassen sich all diese als Module kompilieren, so daß sie mit `insmod` und `rmmod` nach Belieben geladen und entfernt werden können.

Einige OMs laden zuerst den Linux-Sound-Treiber, um die Soundkarte zu initialisieren, enfernen diesen dann wieder und laden dann den SoundModem- Treiber:

```
(rmmod lp)
insmod sound (evtl. Optionen)
rmmod sound
insmod hdlcdrv
insmod soundmodem (evtl. Optionen, dazu später)
```

Die Soundkarte einrichten Der SoundModem-Treiber initialisiert die Soundkarte nicht. In den AX.25-Utilities ist zu diesem Zweck das Programm `setcrystal` enthalten, welches für Soundkarten mit dem Crystal-Chipset verwendet werden kann. Wer eine andere Karte hat, muß andere Software zum Initialisieren verwenden. Die Syntax von `setcrystal`:

```
setcrystal [-w wssio] [-s sbio] [-f synthio] [-i irq] [-d dma] [-c dma2]
```

Will man eine SoundBlaster auf I/O-Adresse 0x388, IRQ 10 und DMA 1 einrichten, so verwendet man:

```
setcrystal -s 0x388 -i 10 -d 1
```

Ein Windows-Sound System konfiguriert man so auf IO-Adresse 0x534, IRQ 5, DMA 3:

```
setcrystal -w 0x534 -i 5 -d 3
```

Mit »-f synthio« kann man die Adresse des Synthesizers einstellen, mit »-c dma2« richtet man den zweiten DMA-Kanal für Vollduplexbetrieb ein.

Den SoundModem-Treiber konfigurieren Nachdem die Soundkarte eingerichtet ist, muß man dem SoundModem-Treiber mitteilen, wo sich die Soundkarte befindet und welche Art von Modem emuliert werden soll. Diese Einstellungen können mit `sethdlc` vorgenommen werden, ebenso können die erforderlichen Parameter dem SoundModem-Modul auf der `insmod`-Kommandozeile mitgegeben werden.

Als Beispiel eine einfache Konfiguration für eine SoundBlaster, die ein 1200 bps-Modem emuliert:

```
insmod hdlcdrv
insmod soundmodem mode="sbc:afsk1200" iobase=0x220 irq=5 dma=1
```

Aber es geht auch genau so gut mit `sethdlc`, das sowohl mit einer Karte als auch mit mehreren funktioniert: Zunächst muß auch hier das Modul geladen

```
insmod hdlcdrv
insmod soundmodem
```

oder die SoundModem-Unterstützung in den Kernel einkompiliert sein. Wir richten damit beispielsweise das schon oben konfigurierte Windows Sound System ein, daß es ein 9600-bps-FSK-Modem nach G3RUH als Device `sm0` emuliert und einen Parallelport an 0x378 zur Ansteuerung der PTT nutzt:

```
sethdlc -p -i sm0 mode wss:fsk9600 io 0x534 \
        irq 5 dma 3 pario 0x378
```

Die in Punkt 6.1.3.1. erwähnte SoundBlaster einrichten, daß sie als Device `sm1` ein 4800 bps-HAPN-Modem emuliert und eine serielle Schnittstelle an 0x2f8 zur PTT-Ansteuerung nutzt:

```
sethdlc -p -i sm1 mode sbc:hapn4800 io 0x388 irq 10 dma 1 serio 0x2f8
```

...als 1200-bps-AFSK-Modem mit PTT über serielle Schnittstelle an 0x2f8:

```
sethdlc -p -i sm1 mode sbc:afsk1200 io 0x388 irq 10 dma 1 serio 0x2f8
```

Die Konfiguration der Kanalzugriffsparameter erfolgt analog dem bei BayCom Gesagten.

Das Device `sm0` soll ein TXDelay von 100ms, eine Slottime von 50ms, eine Ppersist (Persistence) von 128 und full-Duplex-Betrieb fahren:

```
sethdlc -i sm0 -a txd 100 slot 50 ppersist 120 full
```

Alle Werte sind auch hier in Millisekunden anzugeben.

Die Audiopegel einstellen und den Treiber feinabstimmen Es ist für die Funktion jedes Funkmodems sehr wichtig, daß die Audiopegel korrekt eingestellt sind. Dies gilt ebenso für das SoundModem. Thomas Sailer hat einige Utilities entwickelt, die diese Aufgabe erleichtern. Es sind die Programme `smdiag` und `smmixer`. `smdiag` bietet zwei Anzeigearten, einmal als Oszilloskop, zum zweiten als Augenmuster an. Mit `smmixer` kann man die Sende- und Empfangspegel abgleichen.

Um `smdiag` im »Augen-Modus« für das SoundModem-Device `sm0` zu starten:

```
smdiag -i sm0 -e
```

`smmixer` wird für `sm0` so gestartet:

```
smmixer -i sm0
```

Beide Programme zeigen die aktuellen Einstellungen für die Pegel an den Aus- und Eingängen der Karte an. Um diese zu verändern, gibt es zwei Wege:

1. (bei von Linux unterstützten Karten): Man besorge sich ein Mixerprogramm - es finden sich einige auf

   ```
   sunsite.unc.edu:/pub/Linux/apps/sound/mixers
   ```

   - und stelle damit die Werte ein. Günstig sind kommandozeilenorientierte Programme wie `cmix`, da sie in das Startskript eingebunden werden können und so immer die gleichen Einstellungen gegeben sind.

2. bei nicht von Linux unterstützten Soundkarten, die über DOS-Treiber initialisiert werden müssen, sollte man das der Karte meist beiliegende Mixer-Utility für DOS nutzen. Dies trifft auf einen Großteil der Onboard-Soundkarten zu. Siehe dazu auch das *Sound HOWTO*.

Die AX.25-Unterstützung des Kernels für die Nutzung des SoundModem-Device einrichten Der SoundModem-Treiber erzeugt Standard-Netzwerk-Devices, die der Kernel-AX.25-Code nutzen kann. Damit ist die Konfiguration mit der eines BayCom-Modems, einer PI- oder PacketTwin-Karte vergleichbar. Zunächst gibt man dem SoundModem-Device ein Rufzeichen:

```
/sbin/ifconfig sm0 hw ax25 VK2KTJ up
```

Einige Versionen von `ifconfig` unterstützen den eben angegebenen Weg nicht. Dann kann man so vorgehen:

```
/sbin/ifconfig sm0 up
axparms -setcall sm0 VK2KTJ up
```

Als nächstes wird in der Datei `/etc/ax25/axports` ein Eintrag für SoundModem hinzugefügt. Die Verbindung des Eintrags zum entsprechenden Netzwerk-Device geschieht über das eingestellte Rufzeichen. Verwendet ein Programm den Eintrag mit dem für SoundModem vergebenen Rufzeichen, so wird das SoundModem-Device angesprochen.

Das neue AX.25-Device kann nun ganz normal verwendet werden, es läßt sich für TCP/IP einrichten, man kann es dem `ax25d` hinzufügen und NetROM oder ROSE darüber laufen lassen.

Erscheinen Fehlermeldungen beim Aufruf von `ifconfig` wie diese

```
no such device
permission denied
```

sollte man die Konfigurationsoptionen (I/O-Adresse, IRQ, DMA) nochmals überprüfen.

6.1.4 PI-Karte

Folgende Optionen sind bei der Kernel-Kompilierung wichtig:

```
General Setup

[*] Networking support

Network Device Support
...
[*] Radio network interfaces
...
[*] Ottawa PI and PI/2 support for AX.25
```

Der Treiber erzeugt Netzwerk-Devices mit den Namen `pi0`, `pi1`, `pi2` usw., wobei die erste PI-Karte als `pi0` angesprochen wird, die zweite als `pi1` etc.. Wurde der Treiber in den Kernel kompiliert und hat er die Karte korrekt erkannt, läßt er sich einrichten:

```
/sbin/ifconfig pi0a hw ax25 VK2KTJ up
```

Damit wird die erste PI-Karte mit dem Rufzeichen VK2KTJ konfiguriert und aktiviert. Nun muß noch der entsprechende Eintrag in `/etc/ax25/axports` erfolgen, und es kann losgehen. Der PI-Karten-Treiber wurde von David Perry (`dp@hydra.carleton.edu`) geschrieben.

6.1.5 PacketTwin

Folgende Optionen beim Kernelkompilieren:

```
General Setup

[*] Networking support

Network Device Support

...
[*] Radio network interfaces
...
[*] Gracilis PacketTwin support for AX.25
```

Der Treiber erzeugt Netzwerk-Devices mit den Namen `pt0`, `pt1`, `pt2` usw., wobei die erste PacketTwin-Karte als `pt0` angesprochen wird, die zweite als `pt1` etc.. Wurde der Treiber in den Kernel kompiliert und hat er die Karte korrekt erkannt, läßt er sich einrichten:

```
/sbin/ifconfig pt0a hw ax25 VK2KTJ up
```

Damit wird die erste PacketTwin-Karte mit dem Rufzeichen VK2KTJ konfiguriert und aktiviert. Nun muß noch der entsprechende Eintrag in `/etc/ax25/axports erfolgen`, und es kann losgehen. Der PacketTwin-Treiber wurde von Craig Small (`csmall@triode.apana.org.au`) geschrieben.

6.1.6 SCC, allgemein

Wichtige Kernel-Kompilier-Optionen:

```
General Setup

[*] Networking support

Network Device Support

...
[*] Radio network interfaces
...
[*] Z8530 SCC KISS emulation driver for AX.25
```

Joerg Reuter, DL1BKE (`jreuter@lykos.tng.oche.de`) entwickelte die allgemeine Unterstützung für SCC-Karten. Sein Treiber ist für eine Vielzahl Karten konfigurierbar und stellt wie die anderen Netzwerk-Device zur Verfügung, so daß man die SCC-Karte wie eine Netzwerkkarte ansprechen kann.

Die Konfigurations-Tools finden und installieren Während der Kernel-Treiber in den Standard-Quelltexten enthalten ist, gibt es bei Joerg neuere Versionen seines Treibers und die dazu notwendigen Konfigurationsprogramme. Diese findet man hier:

- `ftp.tu-dresden.de:/pub/soft/hamradio/packet/tcpip/linux/`
- `ftp://ftp.ucsd.edu/hamradio/packet/tcpip/linux`

Es gibt verschiedene Versionen, man muß sich die für seinen Kernel passende heraussuchen.

Kernel 2.0.x:

`z8530drv-2.4a.dl1bke.tar.gz`

Kernel 2.1.6 oder neuer:

`z8530drv-utils-3.0.tar.gz`

Mit folgenden Befehlen läßt sich das Paket installieren:

```
cd /usr/src gzip -dc /tmp/z8530drv-2.4a.dl1bke.tar.gz | tar xvpofz -
cd z8530drv
make clean
make dep
make module (wenn der Treiber als Modul erstellt werden soll)
make for_kernel (wenn der Treiber in den Kernel einkompiliert werden soll)
make install
```

Nach dem erfolgreichen Kompilieren sollten sich drei neue Programme im Verzeichnis `/sbin` finden: `gencfg`, `sccinit` und `sccstat`. Diese Programme dienen zur Einrichtung des Treibers für die SCC-Karte. Der Treiber erzeugt Netzwerkdevices mit den Namenn `scc0` - `scc7`. Hat man vorhin `make for_kernel` eingegeben, so muß der Kernel neu kompiliert werden.

Die Option

```
[*] Z8530 SCC KISS emulation driver for AX.25
```

beim »Network Device Support« muß angegeben sein. Hat man sich entschieden, den Treiber als Modul zu kompilieren (`make module`), so wurde ein Modul namens `scc.o` in das entsprechende Verzeichnis `/lib/modules/{kernelversion}/net` kopiert, welches mit `insmod` geladen werden kann.

Den Treiber für die verwendete Karte einrichten Der Z8530-SCC-Treiber ist so flexibel entwickelt worden, daß er möglichst viele verschiedene SCC-Karten unterstützt. Der Preis dafür ist eine etwas kompliziertere Konfiguration. In dem Treiber-Archiv findet sich eine ausführliche Dokumentation, wer Probleme hat, sollte diese lesen.

Insbesondere `doc/scc_eng.doc` bzw. `doc/scc_ger.doc` bieten detailliertere Informationen, die nicht in diesem HOWTO enthalten sind. Das Programm `sccinit` liest die Datei `/etc/z8530drv.conf` als Haupt-Konfigurationsdatei aus. Sie ist in zwei große Abschnitte gegliedert, Hardware-Parameter und Kanal-Konfiguration. Nachdem diese Datei entsprechend editiert wurde, muß nur der Aufruf `sccinit` in das Skript, welches die Netzwerkkonfiguration während des Systemstarts vornimmt, eingetragen werden. Der Treiber läßt sich erst nach einem Aufruf von sccinit nutzen.

Konfiguration der Hardware-Parameter Der erste Abschnitt ist in Absätze unterteilt, von denen jeder einen Z8530-Chip repräsentiert. Jeder Absatz besteht aus einer Liste mit Schlüsselwörtern und den zugeordneten Werten. Standardmäßig lassen sich bis zu 4 SCC-Chips angeben. Wer mehr braucht, muß in der Datei `scc.c` die Zeile

```
#define MAXSCC 4
```

entsprechend anpassen. Erlaubte Schlüsselworte und Argumente:

chip

Wird verwendet, um die einzelnen Abschnitte voneinander zu trennen. Beliebige Argumente sind erlaubt, sie werden nicht verwendet.

data_a

Wird zur Angabe der Adresse des Datenports für den SCC-Kanal »A« verwendet. Argument ist eine Hexadezimalzahl, zum Beispiel 0x300.

ctrl_a

Wird zur Angabe der Adresse des Steuerports für den SCC-Kanal »A« verwendet. Argument ist eine Hexadezimalzahl, zum Beispiel 0x304.

data_b

Wird zur Angabe der Adresse des Datenports für den SCC-Kanal »B« verwendet. Argument ist eine Hexadezimalzahl, zum Beispiel 0x301.

ctrl_b

Wird zur Angabe der Adresse des Steuerports für den SCC-Kanal »B« verwendet. Argument ist eine Hexadezimalzahl, zum Beispiel 0x305.

irq

Gibt den IRQ an, den der in diesem Abschnitt einzustellende Chip verwendet. Argument ist eine Integerzahl, wie 5.

pclock

Gibt die am PCLK-Pin des Z8530 anliegende Taktfrequenz an. Als Argument wird ein Integerwert erwartet (Frequenz in Hz), Voreinstellung ist 4915200 Hz, wenn dieses Schlüsselwort nicht angegeben wird.

board

Der Typ der 8530-SCC-Karte. Folgende Werte sind erlaubt:

PA0HZP

die PA0HZP-SCC-Karte

EAGLE

die EAGLE-SCC-Karte

PRIMUS

die PRIMUS-PC (DG9BL-)Karte

BAYCOM

die BayCom-(U)SCC-Karte

escc

Optional, schaltet die Unterstützung für erweiterte SCC-Chips (ESCC) wie den 8580, 85180 oder 85280 ein. Als Argument steht entweder das Wort yes oder no. Voreinstellung ist no.

vector

Optional, gibt die Adresse des Vector-Latch (auch als Intack-Port bekannt) für die PA0HZP-Karten an. Es gibt nur ein Vector-Latch für alle Chips. Voreinstellung: 0.

special

Optional, gibt die Adresse eines speziellen Funktionsregisters für manche Karten an. Voreinstellung: 0.

Einige Beispielkonfigurationen:

```
BayCom USCC
chip    1
data_a  0x300
ctrl_a  0x304
data_b  0x301
ctrl_b  0x305
irq     5
board   BAYCOM
# # SCC Chip2 #
chip    2
data_a  0x302
ctrl_a  0x306
data_b  0x303
ctrl_b  0x307
board   BAYCOM

PA0HZP SCC-Karte
chip    1
data_a  0x153
data_b  0x151
ctrl_a  0x152
ctrl_b  0x150
irq    9
pclock  4915200
board   PA0HZP
vector 0x168
escc    no
# # SCC Chip2 #
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-Karte
chip    1
data_a  0x303
data_b  0x301
ctrl_a  0x302
ctrl_b  0x300
irq    7
pclock  4915200
board   DRSI
escc    no
```

Bei wem die Karte bereits unter NOS funktioniert, der kann die Treiber-Befehle des PE1CHL-NOS-Treibers mit dem Befehl `gencfg` in eine für die Konfigurationsdatei des Z8530-Treibers nutzbare Form bringen. `gencfg` wird genau so wie für den PE1CHL-Treiber von NOS aufgerufen: Zum Beispiel erstellt

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

eine Grundkonfiguration für die OptoSCC-Karte.

Kanal-Konfiguration Im Abschnitt Kanal-Konfiguration werden alle anderen für den jeweiligen Port relevanten Parameter eingestellt. Auch dieser Abschnitt ist in einzelne Absätze unterteilt. Jeder dieser Absätze steht für einen logischen Port, da jede SCC-Karte zwei Ports bereitstellt, gibt es für jeden Hardware-Absatz zwei solcher Absätze. Die dazu notwendigen Schlüsselworte und Werte müssen in der Datei `/etc/z8530drv.conf` immer *nach* dem Abschnitt mit den Hardware-Parametern stehen. Die Reihenfolge in diesem Abschnitt ist sehr wichtig, mit der hier vorgeschlagenen Reihenfolge sollte es funktionieren.

Folgende Schlüsselwörter und Werte gibt es hier:

device
: Muß in der ersten Zeile einer Port-Definition stehen und gibt den Namen der speziellen Gerätedatei an, auf die sich die weitere Konfiguration bezieht, z.B. `scc0`.

speed
: Gibt die Übertragungsrate in Bits pro Sekunde an, muß ganzzahlig sein, z.B. 1200.

clock
: Gibt an, aus welcher Quelle der Datentakt stammt.

Erlaubte Werte sind:

dpll
: normaler Halbduplexbetrieb

external
: Das Modem hat einen eigenen Sende-/Empfangstakt

divider
: verwendet den Fullduplex-Teiler, wenn installiert

mode
: Gibt die zu verwendende Datenkodierung an. Mögliche Werte sind nrz und nrzi.

rxbuffers
: Gibt die Anzahl der im Speicher zu reservierenden Empfangs- puffer vor. Der Wert ist ganzzahlig, z.B. 8.

txbuffers
: Gibt die Anzahl der im Speicher zu reservierenden Sende- puffer vor. Der Wert ist ganzzahlig, z.B. 8.

bufsize
: Gibt die Größe der Sende-/Empfangspuffer vor. Der Wert wird in Bytes angegeben und stellt die Gesamtlänge eines Paketes dar, es muß also die Länge der AX.25-Header zum Datenfeld hinzugerechnet werden. Dieses Schlüsselwort ist optional, die Voreinstellung 384.

txdelay
: Das von KISS bekannte TXDelay, der Wert ist ganzzahlig und wird in Millisekunden angegeben.

persist

Der Wert für die Persistence, ganzzahlig.

slot

KISS-Slottime, ganzzahlig, in Millisekunden.

tail

Der TXTail-Wert bei KISS, ganzzahlig, in Millisekunden.

fulldup

Das bei KISS verwendete Fullduplex-Flag, Wert ist entweder 1 für Vollduplex oder 0 für Halbduplex.

wait

Der Wait-Wert bei KISS, ganzzahlig, in Millisekunden.

min

Der Min-Wert bei KISS, ganzzahlig, in Sekunden.

maxkey

Die maximale Sendezeit bei KISS ganzzahlig, in Sekunden.

idle

Der Idle-Timer-Wert, ganzzahlig, in Sekunden.

maxdef

Der Maxdef-Wert bei Kiss, ganzzahlig.

group

Der group-Wert bei KISS, ganzzahlig.

txoff

Der txoff-Wert bei Kiss, ganzzahlig, in Millisekunden.

softdcd

Der Wert für SoftDCD (Software-Rauschsperre), ganzzahlig.

slip

Das Slip-Flag bei KISS, ganzzahlig.

Den Treiber verwenden Man verwendet die `scc*`-Geräte wie andere Netzwerk-Devices auch. Beispiel:

```
/sbin/ifconfig scc0 44.136.8.5 netmask 255.255.255.0
/sbin/ifconfig scc0 up
axparms -setcall scc0 VK2KTJ up
```

Die Programme sccstat und sccparam Bei der Fehlersuche kann das Programm `sccstat` helfen, indem man damit die aktuelle Konfiguration eines SCC-Device anzeigen lassen kann. Aufruf zum Beispiel mit:

```
sccstat scc0
```

Es werden viele Informationen zur Einstellung und Funktion des SCC-Ports `scc0` angezeigt.

Mit dem Programm `sccparam` kann man nach dem Booten die Konfiguration verändern. Die Syntax ist an den NOS-Befehl param angelehnt, zum Setzen des TXTail-Wertes auf 100 ms würde man eingeben:

```
sccparam scc0 txtail 0x8
```

6.1.7 BPQ-Ethernet

Folgende Optionen sind bei der Kernel-Kompilierung wichtig:

```
General Setup

[*] Networking support

Network Device Support

...
[*] Radio network interfaces
...
[*] BPQ Ethernet driver for AX.25
```

Linux bietet Kompatibilität mit BPQ-Ethernet. Damit kann man das AX.25-Protokoll über Ethernet im lokalen Netzwerk verwenden, um mit anderen BPQ-Maschinen im Netzwerk zusammenzuarbeiten. Die BPQ-Devices tragen die Namen `bpq1` bis `bpq9`. Das Device `bpq0` gehört zu `eth0`, `bpq1` zu `eth1` usw.. Die Konfiguration ist sehr offen.

Zunächst muß das Ethernet-Device eingerichtet sein. Das heißt, der Kernel muß mit Ethernet-Unterstützung kompiliert sein und diese muß auch funktionieren. Im *Ethernet HOWTO* findet man dazu weiterführende Informationen.

Um die BPQ-Unterstützung einzurichten, muß das Ethernet-Device mit einem Rufzeichen versehen werden:

```
/sbin/ifconfig bpq hw ax25 VK2KTJ up
```

Beachte, daß das Rufzeichen mit dem Rufzeichen in der Datei `/etc/ax25/axports` übereinstimmt, das für diesem Port gelten soll.

6.1.8 BPQ-Node mit Linux-AX.25-Unterstützung verbinden

BPQ verwendet normalerweise sogenannte Multicast-Adressen. Die Linux-Implementation macht das nicht, sie verwendet stattdessen die normale Ethernet Broadcast Address. Deshalb sollte die Datei `NET.CFG` für den BPQ-ODI-Treiber wie folgt geändert werden:

```
LINK SUPPORT
         MAX STACKS 1
         MAX BOARDS 1 LINK
DRIVER   NE2000     ; oder anderer Bezeichner, passend zur Karte
         INT 10     ; entsprechend den Einstellungen der
         PORT 300   ; Netzwerkkarte
```

```
FRAME    ETHERNET_II
PROTOCOL BPQ 8FF ETHERNET_II  ; für BPQ erforderlich - kann die PID
                              ; verändern
BPQPARAMS                     ; optional, nur gebraucht, wenn
                              ; die voreingestellte Zieladresse
                              ; überschrieben werden soll
ETH_ADDR FF:FF:FF:FF:FF:FF    ; Zieladresse
```

6.2 Die Datei /etc/ax25/axports

Diese Datei ist eine einfache Textdatei, die mit einem Texteditor erzeugt wird. Sie hat folgendes Format:

```
Portname    Rufzeichen    Baudrate  Paketlänge   Maxframe   Beschreibung
```

wobei gilt:

Portname
Bezeichner für den Port

Rufzeichen
Rufzeichen, welches dem Port zugeordnet werden soll

Baudrate
Baudrate zum TNC

Paketlänge
Länge des Datenfeldes eines Paketes in Bytes

Maxframe
maximale Anzahl unbestätigter Pakete (AX.25-Window)

Beschreibung
kurzer beschreibender Text

Beispieldatei von Terry Dawson:

```
radio  VK2KTJ-15       4800   256   2    4800 bps auf 144.800 MHz
ether  VK2KTJ-14   10000000   256   2    BPQ Ethernet-Device
```

Zur Erinnerung: Jeder AX.25-Port muß ein eigenes Rufzeichen/SSID bekommen. Jedes zu verwendende Device muß einen Eintrag in dieser Datei bekommen, dies betrifft KISS, BayCom, SCC, PI, PacketTwin und SoundModem-Ports. Jeder Eintrag beschreibt genau ein AX.25-Netzwerk-Device. Die Einträge in der Datei sind mit den Netzwerk-Devices über das Rufzeichen/SSID verbunden. Das ist nicht zuletzt ein Grund dafür, daß jeder Port ein eigenes Rufzeichen/SSID verlangt.

6.3 Das AX.25-Routing einrichten

Es ist sowohl für normale AX.25- als auch für IP-Verbindungen sinnvoll, voreingestellte Digipeaterpfade für spezielle Stationen zu erstellen. Dazu kann man das Programm `axparms` verwenden:

```
/usr/sbin/axparms -route add radio VK2XLZ VK2SUT
```

Mit diesem Befehl setzt man einen Digipeaterpfad für VK2XLZ via VK2SUT auf den AX.25-Port mit dem Namen radio. Weiterführende Informationen sind in der Hilfeseite zu `axparms` zu finden.

7 Ein AX.25-Interface für TCP/IP einrichten

Es ist sehr einfach, einen AX.25-Port für TCP/IP einzurichten. Für KISS-Interfaces gibt es zwei Möglichkeiten, eine IP-Adresse einzurichten. Die konventionellere Methode mit dem Befehl `ifconfig` funktioniert mit allen Interface-Typen.

Für einen an `/dev/ttyS0` angeschlossenen KISS-TNC gilt dieses Beispiel:

```
/usr/sbin/kissattach -i 44.136.8.5 -m 512 /dev/ttyS0 radio
/sbin/route add -net 44.136.8.0 netmask 255.255.255.0 ax0
/sbin/route add default ax0
```

Damit wird ein AX.25-Interface mit der IP-Adresse 44.136.8.5 und einer MTU (Maximum Transmit Unit, maximale Größe des ausgesendeten Datenpakets) von 512 Bytes erzeugt. Wenn notwendig, können mit dem `ifconfig`-Befehl weitere Parameter eingestellt werden.

Die anderen Interfaces können wie die Netzwerkkarte mit `ifconfig` auf IP-Adresse und Netzmaske eingestellt werden, ebenso wird die Route wie für eine Netzwerkkarte festgesetzt (`man route`). Das folgende Beispiel ist für eine PI-Karte gedacht, funktioniert aber auch mit jedem anderen AX.25-Netzwerk-Device, statt `pi0a` ist der jeweilige Device-Name einzusetzen:

```
/sbin/ifconfig pi0a 44.136.8.5 netmask 255.255.255.0 up
/sbin/ifconfig pi0a broadcast 44.136.8.255 mtu 512
/sbin/route add -net 14.136.8.0 netmask 255.255.255.0 pi0a
/sbin/route add default pi0a
```

Die hier aufgeführten Befehle sind typisch für diese Konfigurationen, vielen werden sie von NOS oder anderer TCP/IP-Software her bekannt sein. Beachten muß man, daß die Default-Route möglicherweise nicht gebraucht wird, wenn schon ein anderes Netzwerk-Device eingerichtet ist. Um das Ganze zu testen, versuche man ein ping oder telnet zum lokalen Host:

```
ping -i 5 44.136.8.58
```

Beachte die Option -i 5, die Ping veranlaßt, seine Pakete nur aller 5 Sekunden statt wie voreingestellt, aller Sekunden zu senden.

8 Einen NetROM-Port einrichten

NetROM basiert auf den vorher erstellten AX.25-Ports. Es setzt auf dem AX.25-Protokoll auf.

Um ein NetROM-Interface auf einem AX.25-Port einzurichten, müssen zwei Dateien angepaßt werden. Die eine Datei beschreibt die NetROM-Interfaces, und die andere, auf welche AX.25-Ports NetROM aufsetzt.

Man kann mehrere NetROM-Ports einrichten, für jeden ist die Vorgehensweise die gleiche.

8.1 Die Datei /etc/ax25/nrports

Die erste der beiden Dateien heißt `/etc/ax25/nrports`. Sie beschreibt die NetROM-Ports in etwa der gleichen Art wie `/etc/ax25/axports` die AX.25-Ports.

Jedes NetROM-Device braucht seinen Eintrag in `/etc/ax25/nrports`. Normalerweise wird es auf einer Linux-Maschine nur ein NetROM-Device geben, das eine definierte Anzahl von AX.25-Ports nutzt.

Manchmal will man aber einem besonderen Programm, wie einer Mailbox, ein separates NetROM-Alias vergeben. Dann kann man auch mehrere NetROM-Devices einrichten.

Die Datei `/etc/ax25/nrports` hat folgendes Format:

```
Name  Rufzeichen   Alias   Paketlänge  Beschreibung
```

Name

Der Bezeichner für den Port.

Rufzeichen

Das Rufzeichen, welches für den NetROM-Verkehr verwendet wird. Dies ist *nicht* das Rufzeichen, das andere Stationen als Nodecall verwenden können. Zu dem Node-Programm später. Es sollte nicht noch einmal in `/etc/ax25/axports` oder `/etc/ax25/nrports` auftauchen.

Alias

Der NetROM-Alias für diesen Port.

Paketlänge

Die maximale Größe der NetROM-Pakete.

Beschreibung

Eine kurze Beschreibung für den Port.

Ein Beispiel sähe so aus:

```
netrom  VK2KTJ-9    LINUX  236   Linux Packet Switch
```

Damit wird ein NetROM-Port erzeugt, der im übrigen NetROM-Netz als LINUX:VK2KTJ-9 erscheint. Programme wie `call` nutzen diese Datei.

8.2 Die Datei /etc/ax25/nrbroadcast

Die zweite der Dateien ist `/etc/ax25/nrbroadcast`. In dieser Datei können viele Einträge enthalten sein. Normalerweise gibt es für jeden AX.25-Port, über den NetROM-Verbindungen laufen sollen, einen Eintrag. Die Datei hat folgendes Format:

```
axport   min_obs  def_qual  worst_qual verbose
```

Wobei gilt:

axport

Der aus der Datei `/etc/ax25/axports` entnommene Portname. Steht kein Eintrag in `/etc/ax25/nrbroadcast`, so bedeutet das, daß kein NetROM-Routing durchgeführt wird und alle empfangenen NetROM-Broadcasts auf diesem Port ignoriert werden.

min_obs

Minimalwert für die Obsolescence.

def_qual

Voreingestellte Qualität für den Port.

worst_qual

Wert für die schlechteste Verbindungsqualität, Verbindungen mit schlechteren Werten werden ignoriert.

verbose

Legt fest, ob NetROM auf diesem Port Routing-Broadcasts aussendet oder nur auf seine Anwesenheit hinweist.

Ein Beispiel sähe so aus: radio 1 200 100 1

8.3 Das Netzwerk-Device für NetROM erstellen

Sind die beiden Konfigurationsdateien vervollständigt, dann muß das NetROM-Netzwerk-Device genau so wie die anderen AX.25-Devices erstellt werden. Diesmal wird dazu der Befehl `nrattach` verwendet. Dieser arbeitet wie `axattach`, im Unterschied zu diesem erzeugt er NetROM-Netzwerk-Devices mit den Namen `nr0` - `nr9`.

Beim ersten Aufruf erzeugt `nrattach` das Device `nr0`, beim zweiten `nr1` usw.. Um das Netzwerk-Device für den von uns definierten NetROM-Port zu erzeugen, geben wir ein:

```
nrattach netrom
```

Damit wird das NetROM-Device `nr0` mit dem Namen `netrom`, dessen Details in der Datei `/etc/ax25/nrports` festgelegt wurden, gestartet.

Wer einen Kernel der 2.2.x-Reihe verwendet, muß an dieser Stelle eine IP-Adresse angeben, auch wenn kein TCP/IP verwendet werden soll. Der Aufruf von `nrattach` sieht dann so aus:

```
nrattach -i 44.131.16.2 netrom
```

8.4 Den NetROM-Daemon starten

Der Linux-Kernel übernimmt alle mit dem NetROM-Protokoll und dem Switching verbundenen Aufgaben bis auf einige Funktionen. Der NetROM-Daemon verwaltet die NetROM-Routing-Tabellen und erzeugt die NetROM-Routing-Broadcasts. Er wird mit folgendem Befehl gestartet:

```
/usr/bin/netromd -i
```

Schon bald darauf sollte man sehen, wie sich die Datei `/proc/net/nr_neigh` mit den Namen der benachbarten NetROM-Stationen füllt:

```
cat /proc/net/nr_neigh
```

Man sollte den `netromd`-Aufruf in die während des Startens ausgeführten (rc-)Skripte einfügen, damit er automatisch beim Booten gestartet wird.

8.5 Das NetROM-Routing einrichten

Manchmal ist es wünschenswert, feste (statische) Routen für spezielle Rechner einzurichten. Dazu gibt es den Befehl `nrparms`. Eine vollständige Beschreibung kann in der Hilfeseite nachgelesen werden. Ein kleines Beispiel dazu:

```
/usr/sbin/nrparms -nodes VK2XLZ-10 + #MINTO 120 5 radio VK2SUT-9
```

Damit wird eine NetROM-Route zu #MINTO:VK2XLZ-10 über die benachbarte Station VK2SUT-9 auf dem AX.25-Port radio eingerichtet. Man kann damit auch manuell neue Einträge für benachbarte Stationen vornehmen:

```
/usr/sbin/nrparms -routes radio VK2SUT-9 +120
```

Damit wird VK2SUT-9 als benachbarte NetROM-Station mit einer fest eingestellten Qualität von 120 eingetragen, die nicht automatisch gelöscht bzw. geändert wird.

9 Ein NetROM-Interface für TCP/IP einrichten

Ein TCP/IP-Interface für NetROM wird fast genau so wie für AX.25 eingerichtet. Man kann entweder die IP-Adresse und MTU auf der Kommandozeile für `nrattach` angeben oder `ifconfig` und `route` benutzen. Es müssen jedoch per Hand die arp-Einträge für die Rechner, zu denen geroutet werden soll, ergänzt werden, da die Maschine nicht selbst herausbekommt, welche NetROM-Adresse sie verwenden muß, um einen bestimmten IP-Rechner zu erreichen. Will man also ein Device `nr0` mit der IP-Adresse 44.136.8.5, einer MTU von 512 und eingerichtet mit den Daten aus `/etc/ax25/nrports` für den NetROM-Port `netrom` erzeugen, gibt man ein:

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

Oder man benutzt die folgenden Befehle:

```
/usr/sbin/nrattach netrom ifconfig nr0 44.136.8.5 \
                   netmask 255.255.255.0 hw netrom VK2KTJ-9
route add 44.136.8.5 nr0
```

Anschließend müssen für jeden IP-Rechner, der über NetROM erreichbar sein soll, die Einträge für Route und ARP gesetzt werden. Um einen IP-Rechner mit der Adresse 44.136.80.4 auf NetROM-Adresse BBS:VK3BBS über die benachbarte Station VK2SUT-0 zu erreichen, gibt man folgendes ein:

```
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
```

Die Werte »120« und »6« beim `nrparms`-Befehl stehen für die NetROM-Qualität und die Obsolescence dieser Route.

10 Einen ROSE-Port einrichten

Die ROSE-Protokollebene entspricht der Ebene 3 der X.25-Spezifikation. Die im Kernel enthaltene ROSE-Unterstützung ist eine modifizierte Version des ROSE (RATS (Radio Amateur Teleprinter Society) Open System Environment) AX.25 Packet Switch (`http://www.rats.org/rose/`). Das ROSE-Protokoll setzt auf den vorher erstellten AX.25-Ports auf.

Um ROSE einzurichten, muß man eine Konfigurationsdatei erstellen, die die zu verwendenden ROSE-Ports definiert. Man kann mehrere ROSE-Ports erstellen, für jeden gilt die gleiche Vorgehensweise.

10.1 Die Datei /etc/ax25/rsports

Die ROSE-Ports werden in der Datei `/etc/ax25/rsports` eingerichtet. Sie beschreibt die ROSE-Ports ähnlich wie `/etc/ax25/axports` die AX.25-Ports. Die Datei hat folgendes Format:

```
Name   Adresse   Beschreibung
```

Wobei gilt:

Name
: Text-Bezeichner für den jeweiligen Port

Adresse
: 10stellige ROSE-Adresse für den Port

Beschreibung
: kurze, frei wählbare Beschreibung

Ein Beispiel:

```
rose     5050294760    ROSE-Port
```

Beachte, daß ROSE das für jeden AX.25-Port voreingestellte Rufzeichen verwendet, wenn nichts anderes angegeben wird. Um ein eigenes Rufzeichen/SSID für ROSE festzulegen, gibt man folgendes ein:

```
/usr/sbin/rsparms -call VK2KTJ-10
```

Damit wartet Linux auf ROSE-Rufe unter dem Rufzeichen/SSID VK2KTJ-10 auf allen eingerichteten AX.25-Ports und verwendet dieses auch für ROSE- Verbindungen.

10.2 Das ROSE-Netzwerk-Device einrichten

Wurde die Datei `/etc/ax25/rsports` erstellt, kann man die ROSE-Netzwerk-Devices genau so wie die AX.25-Netzwerk-Devices erstellen. Diesmal wird dazu der Befehl `rsattach` verwendet. Dieser arbeitet wie `axattach`, im Unterschied zu diesem erzeugt er ROSE-Netzwerk-Devices mit den Namen `rose0` - `rose5`. Beim ersten Aufruf erzeugt `rsattach` das Device `rose0`, beim zweiten `rose1` usw.. Beispiel:

```
rsattach rose
```

Damit wird das ROSE-Device `rose0` mit dem Namen `rose`, dessen Details in der Datei `/etc/ax25/rsports` festgelegt wurden, gestartet.

10.3 Das Routing für ROSE einrichten

Zur Zeit unterstützt das ROSE-Protokoll nur statisches Routing. Mit dem `rsparms`-Befehl kann die Routingtabelle eingerichtet werden:

```
rsparms -nodes add 5050295502 radio vk2xlz
```

Damit würde eine Route zum ROSE-Node 5050295502 über den AX.25-Port mit dem Bezeichner »radio« laut `/etc/ax25/axports` zu einer benachbarten Station mit dem Rufzeichen VK2XLZ hinzugefügt.

Es kann auch eine Maske angegeben werden, um mehrere ROSE-Zielrechner in einem Routing-Eintrag zu erfassen:

```
rsparms -nodes add 5050295502/4 radio vk2xlz
```

Damit werden alle Zieladressen erfaßt, die in den ersten 4 Stellen mit der angegebenen übereinstimmen, die also mit »5050« beginnen. Der Befehl kann auch in dieser, sicher eindeutigeren, Form eingegeben werden:

```
rsparms -nodes add 5050/4 radio vk1xlz
```

11 AX.25/NetROM/ROSE-Verbindungen

Jetzt, da alle AX.25-, NetROM- und ROSE-Devices eingerichtet und aktiviert sind, sollte es möglich sein, Test-Verbindungen zu starten. In den AX.25-Utilities ist das Programm `call` enthalten, das ein Terminal mit geteiltem Bildschirm für AX.25, NetROM und ROSE darstellt. Ein einfacher AX.25-Verbindungsaufbau sähe so aus:

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

Ein einfacher NetROM-Verbindungsaufbau zu einem Node mit dem Alias »SUNBBS« sähe so aus:

```
/usr/bin/call netrom SUNBBS
```

Ein einfacher ROSE-Verbindungsaufbau an HEARD auf dem Node 5050882960 sähe so aus:

```
/usr/bin/call rose HEARD 5050882960
```

Beachte: Man muß `call` mitteilen, auf welchem Port gerufen werden soll, da der Zielrechner möglicherweise auf allen eingerichteten Ports erreichbar ist. Das Programm `call` ist ein zeilenbasiertes Terminalprogramm, mit dem man AX.25-Stationen rufen kann.

Zeilen, die mit einem »~« beginnen, werden als Befehle interpretiert. Mit Eingabe von »~.« am Beginn einer neuen Zeile beendet man die Verbindung. Weiterführende Informationen findet man in der Hilfeseite zu `call`.

12 Linux für ankommende Packet-Verbindungen einrichten

Linux ist ein mächtiges Betriebssystem und bietet große Flexibilität bei der Konfiguration. Dadurch wird es etwas langwieriger, es so einzurichten, daß es das tut, was man will. Wenn man eine Linux-Maschine für die Entgegennahme von ankommenden AX.25-NetROM- und ROSE-Rufen einrichtet, muß man sich einige Fragen stellen. Die wichtigste davon ist: »Was sollen die Nutzer zu sehen bekommen, wenn sie verbunden sind?«

Es werden hübsche kleine Anwendungen entwickelt, die dazu verwendet werden können, den Anrufern bestimmte Dienste anzubieten. Ein einfaches Beispiel ist das in den AX.25-Utilities enthaltene Programm `pms`, etwas komplexer ist das ebenfalls dort vorhandene `node`-Programm. Als Alternative kann man den Nutzern einen Login-Prompt geben, so daß sie einen Shell-Account nutzen können, oder man hat sogar ein eigenes Programm, wie eine Datenbank oder ein Spiel, geschrieben, mit dem sich die Nutzer verbinden können. Was auch immer gewünscht wird, man muß es der AX.25-Software mitteilen, damit diese weiß, welches Programm bei einer hereinkommenden Verbindung gestartet werden soll.

Das Programm ax25d entspricht hierfür dem üblicherweise zur Entgegennahme von TCP/IP-Verbindungen auf UNIX-Maschinen eingesetzten inetd. Es wartet auf hereinkommende Verbindungen, wenn es eine erkennt, schaut es in einer Konfigurationsdatei nach, welches Programm zu starten und mit dieser Verbindung zu assoziieren ist.

Da der ax25d das Standardwerkzeug für die Entgegennahme von AX.25-, NetROM- und ROSE-Verbindungen ist, soll hier die Konfiguration erläutert werden.

12.1 Die Datei /etc/ax25/ax25d.conf

Sie ist die Konfigurationsdatei für den AX.25-Daemon ax25d, der die hereinkommenden AX.25-, NetROM- und ROSE-Verbindungen entsprechend handhabt. Die Datei sieht auf den ersten Blick etwas kryptisch aus, aber man wird bald sehen, daß sie in der Praxis recht einfach zu bearbeiten ist. Es gibt eine kleine Falle, die beachtet werden muß. Im allgemeinen hat die Datei /etc/ax25/ax25d.conf folgendes Format:

```
# Dieser Kommentar wird vom ax25d ignoriert.
[port_name] <port_name>
{port_name}
  window T1 T2 T3 idle N2
  window T1 T2 T3 idle N2
parameters window T1 T2 T3 idle N2    window T1 T2 T3
idle N2      ...
default  window T1 T2 T3 idle N2
```

Wobei gilt: Das Zeichen »#« am Anfang einer Zeile markiert einen Kommentar, die Zeile wird vom ax25d ignoriert.

port_name

Der Name des AX.25-, NetROM- oder ROSE-Ports, wie er in /etc/ax25/axports, /etc/ax25/nrports respektive /etc/ax25/rsports definiert ist.

peer

Das Rufzeichen der Station, auf die sich die Konfiguration bezieht. Wird hier keine SSID angegeben, so sind alle SSID gültig.

window

Der AX.25-Window-Parameter (Maxframe) für diese Konfiguration.

T1

Der Timer T1 (Zeit bis zum Wiederaussenden eines Paketes) in halben Sekunden.

T2

Der Timer T2 (Zeit, die auf eine weiteres Paket gewartet wird, bevor Empfangsbestätigung gesendet wird) in Sekunden.

T3

Zeit, die die Verbindung inaktiv sein darf, bevor sie getrennt wird in Sekunden.

idle

Der Idle-Timer-Wert in Sekunden

N2

Anzahl Versuche (retries), bevor eine Verbindung als gescheitert betrachtet wird.

mode

Legt einige allgemeine Zugriffsrechte fest. Die Modes werden ein-/ausgeschaltet, indem eine Reihe von Zeichen angegeben werden, von denen jedes für ein bestimmtes Zugriffsrecht steht. Die Buchstaben dürfen groß- oder klein geschrieben werden und dürfen nicht durch Leerzeichen getrennt werden.

Folgende Zeichen sind möglich:

u/U

UTMP - zur Zeit nicht unterstützt

v/V

Validiere alles - zur Zeit nicht unterstützt

q/Q

Quiet - schreibe Verbindung nicht mit

n/N

überprüfe NetROM-Nachbar - zur Zeit nicht unterstützt

d/D

Digipeater nicht erlaubt, Verbindungen müssen direkt erfolgen, nicht über Digipeater.

l/L

Aussperren (Lockout) - keine Verbindung erlaubt.

***/0**

Marker - Platzhalter, kein Mode gesetzt

uid

Die Nutzerkennung (User ID) unter der das für die Verbindung aufgerufene Programm laufen soll.

cmd

Der volle Pfadname des aufzurufenden Programms, ohne Kommandozeilenparameter

cmd_name

Text, der beim Aufrufen von ps (Anzeige des Prozeßstatus) erscheinen soll, normalerweise dasselbe wie cmd, aber ohne die Pfadangabe

arguments

Kommandozeilenparameter, die der in cmd angegebenen Anwendung beim Start übergeben werden. Folgende Kürzel können dazu verwendet werden:

%d

Name des Ports, auf dem die Verbindung eingegangen ist.

%U

AX.25-Rufzeichen der verbundenen Station ohne SSID, in Großbuchstaben.

%u

AX.25-Rufzeichen der verbundenen Station ohne SSID, in Kleinbuchstaben.

%S

AX.25-Rufzeichen der verbundenen Station mit SSID, in Großbuchstaben.

%s

AX.25-Rufzeichen der verbundenen Station mit SSID, in Kleinbuchstaben.

%P

AX.25-Rufzeichen des Nodes, von dem die Verbindung kam ohne SSID, in Großbuchstaben.

%p
AX.25-Rufzeichen des Nodes, von dem die Verbindung kam ohne SSID, in Kleinbuchstaben.

%R
AX.25-Rufzeichen des Nodes, von dem die Verbindung kam mit SSID, in Großbuchstaben.

%r
AX.25-Rufzeichen des Nodes, von dem die Verbindung kam mit SSID, in Kleinbuchstaben.

Für jedes AX.25-, NetROM- oder ROSE-Interface, auf dem Verbindungen entgegengenommen werden sollen, muß ein Abschnitt in diesem Format vorgesehen werden.

In jedem Abschnitt gibt es zwei besondere Zeilen, eine beginnt mit dem Wort »parameters«, die andere mit »default«. (Ja, das ist ein Unterschied.) Diese Zeilen dienen speziellen Funktionen.

Der Zweck der default-Zeile dürfte klar sein, diese Zeile enthält die Parameter, die auf alle Stationen zutreffen, für die keine speziellen Parameter definiert wurden. Wird keine default-Zeile angegeben, so werden alle Verbindungen, für die keine speziellen Voreinstellungen getroffen wurden, sofort wieder getrennt.

Die parameters-Zeile ist ein wenig kritischer, hier ist auch die vorhin erwähnte Falle. In jedem der Felder für alle aufgeführten Stationen kann das »*«-Zeichen benutzt werden, um den voreingestellten Wert zu übernehmen. Die parameters-Zeile setzt diese Voreinstellungen. Die Kernel-Software selbst hat einige Voreinstellungen, die dann verwendet werden, wenn die keine Voreinstellungen unter »parameters« angegeben sind.

Die Falle besteht nun darin, daß die Parameters-Werte nur für die darunterliegenden Zeilen gelten, nicht für die darüberstehenden. Man kann mehrere parameters-Zeilen pro Interface-Definition haben und somit Gruppen von Stationen mit gleichen Voreinstellungen einrichten.

12.2 Ein einfaches Beispiel für /etc/ax25/ax25d.conf

```
# ax25d.conf für VK2KTJ - 02/03/97
# Diese Konfiguration nutzt den vorher definierten AX.25-Port.
#  Win T1  T2  T3  idl N2    []


  [VK2KTJ-0 via radio]
  parameters 1    10  *  *  *   *   *
  VK2XLZ     *     *  *  *  *   *   *    root  /usr/sbin/axspawn axspawn %u +
  VK2DAY     *     *  *  *  *   *   *    root  /usr/sbin/axspawn axspawn %u +
  NOCALL     *     *  *  *  *   *   L
  default    1    10  5 100 180 5   *    root  /usr/sbin/pms pms -a -o vk2ktj

  [VK2KTJ-1 via radio]
  default    *     *    *   *   *   0    root /usr/sbin/node node


  <netrom>
  parameters 1    10  *  *  *   *   *
  NOCALL     *     *  *  *  *   *   L
  default    *     *  *  *  *   *   0    root /usr/sbin/node node

  {VK2KTJ-0 via rose}
  parameters 1    10  *  *  *   *   *
  VK2XLZ     *     *  *  *  *   *   *    root  /usr/sbin/axspawn axspawn %u +
  VK2DAY     *     *  *  *  *   *   *    root  /usr/sbin/axspawn axspawn %u +
  NOCALL     *     *  *  *  *   *   L
```

```
default    1    10  5 100 180 5   *    root  /usr/sbin/pms pms -a -o vk2ktj

{VK2KTJ-1 via rose}
default    *     *    *   *   *   0    root /usr/sbin/node node radio
```

In diesem Beispiel gelten für jede Station, die eine Verbindung mit dem Rufzeichen VK2KTJ-0 aufbauen will und auf dem AX.25-Port »radio« gehört wurde, die folgenden Einstellungen:

Jeder, dessen Rufzeichen auf »NOCALL« eingestellt ist, soll ausgesperrt bleiben, beachte die Verwendung des Mode »L«. Die parameters-Zeile ändert zwei der Kernel-Voreinstellungen (Window und T1) und startet das `axspawn`-Programm für die Nutzer. Alle auf diese Weise gestarteten `axspawn` erscheinen in der Ausgabe des Befehls `ps` als »axspawn« zum besseren Verständnis. Die nächsten beiden Zeilen definieren für zwei Stationen, daß für sie diese Einstellungen gelten.

Die defaults-Zeile dient als »Sammelbecken« für alle anderen Stationen (auch VK2XLZ und VK2DAY, wenn sie eine andere SSID als -1 verwenden). Hier werden alle Parameter implizit gesetzt und das Programm pms mit einer Kommandozeile aufgerufen, die anzeigt, daß es von einer AX.25-Verbindung aufgerufen wurde und das Rufzeichen des Eigentümers VK2KTJ ist. (Siehe den Abschnitt 15 für weitere Details.)

Die nächste Konfiguration nimmt Rufe an VK2KTJ-1 über den Port »radio« entgegen. Sie startet das Programm `node` für alle Stationen, die eine Verbindung zu VK1KTJ-1 aufbauen.

Anschließend folgt eine NetROM-Konfiguration, man beachte die Verwendung der »<>«-Klammern anstelle eckiger Klammern »[]«. Damit wird eine NetROM-Konfiguration markiert. Sie ist einfacher, da nur festgelegt wird, daß für jede Station, die eine Verbindung zu dem Port »netrom« aufgebaut hat, das Programm `node` gestartet wird. Stationen mit Rufzeichen NOCALL werden ausgesperrt.

Die letzten beiden Konfigurationen betreffen hereinkommende ROSE-Verbindungen. Die erste ist für Stationen, die VK2KTJ-0 gerufen haben, und die zweite für Verbindungen mit VK2KTJ-1, dem ROSE-Node. Sie funktionieren genau gleich. Man beachte die Verwendung von geschweiften Klammern »{}«, um die Konfiguration als ROSE-Konfiguration zu kennzeichnen.

Das Beispiel hier ist frei erfunden, doch es sollte die wichtigen Eigenschaften und Möglichkeiten der Syntax der Konfigurationsdatei aufzeigen. In der Hilfeseite zu `ax25d` wird die Konfigurationsdatei vollständig erklärt. Ein detaillierteres und sicher ebenfalls nützliches Beispiel ist in dem Archiv der AX.25-Utilities enthalten.

12.3 Den ax25d starten

Ist die Konfigurationsdatei vervollständigt, startet man den `ax25d`:

```
/usr/sbin/ax25d
```

Nun sollten andere Stationen Verbindungen zu unserer Maschine aufbauen können. Der Aufruf des `ax25d` sollte in die beim Systemstart ausgeführten Skripts eingefügt werden, damit dieser automatisch nach dem Booten zur Verfügung steht.

13 Die Node-Software einrichten

Die Node-Software wurde von Tomi Manninen (`tomi.manninen@hut.fi`) entwickelt und basiert auf dem Original-PMS-Programm. Es stellt eine vollständige und flexible Node-Funktion zur Verfügung, die man einfach einrichten kann. In den aktuellen AX.25-Utilities ist das Programm nicht mehr enthalten, es liegt vielmehr als separates Paket auf `ftp.hes.iki.fi`. Dieses Paket ist *nur* mit einem Kernel 2.2.x lauffähig. Das Programm erlaubt den verbundenen Nutzern, Telnet-, NetROM-, ROSE- und AX.25-Verbindungen aufzubauen und bestimmte Informationen zu erhalten, wie Finger, Nodes und Heard-Listen usw.

Der Node kann sehr einfach so eingestellt werden, daß er jedes gewünschte Linux-Kommando ausführen kann. Das Programm `node` wird normalerweise vom `ax25d` aufgerufen, kann aber auch vom `inetd` gestartet werden, um Nutzern Zugriff via Telnet zu ermöglichen. Auch ein Aufruf von der Kommandozeile ist möglich.

13.1 Die Datei /etc/ax25/node.conf

In dieser Datei wird die hauptsächliche Konfiguration des Programms `node` vorgenommen. Sie ist eine einfache Textdatei mit folgendem Format:

```
#/etc/ax25/node.conf
# Konfigurationsdatei für das Programm node.
#
# Zeilen, die mit einem '#' beginnen, sind Kommentare und werden vom
# Programm ignoriert.
# Hostname
# Gibt den Rechnernamen der Node-Maschine an
hostname     radio.gw.vk2ktj.ampr.org
# Local Network
# hier kann man einstellen, was als 'Lokal' gilt, wichtig für
# Zugriffsrechte in node.perms
localnet       44.136.8.96/29
# Hidden Ports
# Die hier angegebenen Ports sind für Nutzer unsichtbar. Sie werden
# vom Befehl (P)orts nicht ausgegeben.
hiddenports     rose netrom
# Callserver
# Wenn eingestellt, wird ein Callserver zur Verfügung gestellt
callserver      zone.oh7rba.ampr.org
# Node-Identifikation.
# Dieser Text erscheint im Node-Prompt
NodeId         LINUX:VK2KTJ-9
# NetRom port
# Name des für vom Node ausgehende
NetROM-Verbindungen verwendeten
# NetROM-Ports
NrPort         netrom
# Node Idle Timeout
# Gibt an, nach welcher Zeit der Inaktivität eine Verbindung vom Node
# beendet wird (in Sekunden)
idletimout      1800
# Connection Idle Timeout
# Gibt an, nach welcher Zeit der Inaktivität eine Verbindung, die über
# den Node läuft, beendet wird (in Sekunden)
conntimeout     1800
# Reconnect
# Gibt an, ob die Nutzer zum Node zurückverbunden werden sollen, wenn
# die abgehende Verbindung getrennt wurde (reconnect) oder ob sie
# vollständig vom Node getrennt werden sollen
reconnect       on
# Command Aliases
# Eine Möglichkeit, komplexe Befehle einfacher zu gestalten
```

```
alias          CONV    "telnet vk1xwt.ampr.org 3600"
alias          BBS     "connect radio vk2xsb"
# External Command Aliases
# Möglichkeit, externe Befehle unter node ausführen zu können
# extcmd
# Nur Flag == 1 ist implementiert
#  hat das gleiche Format wie in der Datei /etc/ax25/ax25d.conf
extcmd         PMS     1      root    /usr/sbin/pms pms -u %U -o VK2KTJ
# Logging
# Mitschriften ins System-Log.
# 3 - ausführlichste Form, 0 - ausgeschaltet
loglevel        3
```

13.2 Die Datei /etc/ax25/node.perms

Das Programm `node` erlaubt, einzelnen Nutzern Zugriffsrechte zuzuteilen. Diese erlauben festzulegen, welche Nutzer beispielsweise Befehle wie (T)elnet oder (C)onnect ausführen dürfen und welche nicht.

Diese Information wird in der Datei `/etc/ax25/node.perms` abgelegt, welche fünf Felder enthält. In allen Feldern ist der Stern »*« das Zeichen dafür, daß alle möglichen Optionen für das jeweilige Feld gelten sollen. Das ist nützlich, wenn man Standard-Festlegungen treffen will. Die einzelnen Felder:

user

Das Rufzeichen des Nutzers, für den die folgenden Regelungen gelten sollen. SSID-Werte werden ignoriert, man sollte also nur das Rufzeichen angeben.

method

Für jedes Protokoll oder Zugriffsmethode gibt es auch Zugriffsrechte. Beispielsweise kann man Nutzern, die über AX.25 oder NetROM verbunden sind, erlauben, die (C)onnect-Option zu verwenden, diese aber anderen, die z.B. über telnet eingeloggt sind, verwehren.

Man kann hier also festlegen, für welches Protokoll die Zugriffsrechte gelten sollen:

ampr

Nutzer ist über eine AMPR-Adresse via Telnet verbunden

ax25

Nutzer ist über AX.25 verbunden

host

Node wurde von der Kommandozeile gestartet

inet

Nutzer ist von einer nicht-lokalen, nicht- AMPR-Adresse aus verbunden

local

Nutzer ist von einem »lokalen« Rechner aus verbunden

netrom

Nutzer ist über NetROM verbunden

rose

Nutzer ist über ROSE verbunden

*****

Alle möglichen Verbindungsarten

port

Für AX.25-Nutzer lassen sich Zugriffsrechte auch auf der Basis des verwendeten Ports festlegen. Damit kann man definieren, was AX.25-Nutzer in Abhängigkeit von dem Port, über den sie verbunden sind, tun dürfen. Im dritten Feld steht bei dieser Möglichkeit dann der Portname. Das Ganze ist nur für AX.25-Verbindungen sinnvoll.

password

Optional kann man den Node so einstellen, daß er die Nutzer nach einem Paßwort fragt. Das kann sinnvoll sein, um Nutzer mit vielen Zugriffsrechten zu schützen. Das Paßwort steht dann im vierten Feld der Datei.

permissions

Zugriffsrechte. Dieses Feld ist das letzte bei jedem Eintrag in `/etc/ax25/node.perms`. Es ist als Bit-Feld codiert, wobei jede Option einem Bit entspricht, das diese einschaltet, wenn es gesetzt ist, und ausschaltet, wenn es nicht gesetzt ist. Hier eine Liste der steuerbaren Optionen und ihrer Bitwerte:

```
Wert         Beschreibung
------------------------------------------------------------
1            Login erlaubt
2            AX.25-(C)onnects erlaubt
4            NetROM-(C)onnects erlaubt
8            (T)elnet zu lokalen Rechnern erlaubt
16           (T)elnet zu AMPR-Net-Rechnern (44.0.0.0)
             erlaubt
32           (T)elnet zu nicht-lokalen, nicht-AMPR-Net-
             Rechnern erlaubt
64           Versteckte Ports für AX.25-(C)onnects erlaubt
128          ROSE-(C)onnects erlaubt
```

Um die Zugriffsrechte festzulegen, rechnet man die Werte für alle Rechte des Users zusammen und schreibt die Summe in das fünfte Feld.

Die Datei `node.perms` könnte also etwa so aussehen:

```
# etc/ax25/node.perms
#
# Der Nodebetreiber ist VK2KTJ, sein Paßort ist "secret"
# und er hat alle Zugriffsrechte bei allen Verbindungsarten
vk2ktj *      *       secret  255
# Folgende Nutzer dürfen keine Verbindungen aufbauen
NOCALL *      *       0
PK232  *      *       0
PMS    *      *       0
# INET-Nutzer dürfen keine Verbindungen aufbauen
*      inet   *       0
# AX.25-, NetROM, lokale, Host- und AMPR-Nutzer können (C)onnects
# und (T)elnet-Verbindungen zu anderen lokalen und AMPR-Rechnern,
# jedoch nicht zu anderen IP-Adressen aufbauen
*       ax25   *       *       159
*       netrom *       *       159
*       local  *       *       159
*       host   *       *       159
*       ampr   *       *       159
```

13.3 node vom ax25d aus starten

Das Programm `node` wird normalerweise vom `ax25d` aus gestartet. Dazu müssen entsprechende Einträge in der Datei `/etc/ax25/ax25d.conf` vorgenommen werden. In der Beispielkonfiguration haben die Nutzer die Wahl, ob sie eine Verbindung zum Node oder zu anderen Diensten aufbauen wollen. Bei `ax25d` gibt es zu diesem Zweck sogenannte Port-Aliases. Beispiel: Bei der weiter oben angegebenen Konfiguration für `ax25d` soll unter dem Rufzeichen VK2KTJ-1 der Node erreichbar sein. Dazu wird folgendes in der Datei `/etc/ax25/ax25d.conf` hinzugefügt:

```
[vk2ktj-1 via radio] default *  *    *    *    *  0    root /usr/sbin/node node
```

Damit wird festgelegt, daß der Linux-Kernel-Code alle Verbindungsanforderungen für das Rufzeichen VK2KTJ-1, die er auf dem AX.25-Port radio hört, beantwortet und das Programm `node` startet.

13.4 node vom inetd aus starten

Es ist leicht möglich, den Node auch von einer Telnet-Verbindung aus zu nutzen. Zunächst ist festzulegen, zu welchem Port die die Nutzer ihre Verbindung aufbauen sollen.

Im Beispiel wurde willkürlich Port 3694 gewählt, Informationen darüber, wie man den Telnet-Daemon durch `node` ersetzen kann, finden sich in Tomis Dokumentation.

Zwei Dateien sind anzupassen. Der Datei `/etc/services` fügt man hinzu:

```
node    3694/tcp        # OH2BNS's Node-Programm
```

und in `/etc/inetd.conf` kommt zusätzlich:

```
node    stream tcp     nowait  root    /usr/sbin/node node
```

Nach einem Neustart des `inetd` (vorher ein `kill -HUP 1`) bekommen alle Nutzer, die eine Telnet-Verbindung zum Port 3694 aufgebaut haben, eine Abfrage nach Loginname und Paßwort (wenn eingerichtet) und werden dann an `node` weitergegeben.

14 axspawn einrichten

`axspawn` ist ein einfaches Programm, das anrufenden AX.25-Stationen den Login auf die eigene Maschine erlaubt.

Es kann vom `ax25d` in gleicher Weise wie `node` gestartet werden. Um einem Nutzer ein Login auf die eigene Maschine zu erlauben, muß der Datei `/etc/ax25/ax25d.conf` eine Zeile ähnlich dieser hinzugefügt werden:

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

Endet diese Zeile mit einem »+«, so muß der jeweilige Nutzer Return drücken, bevor er sich einloggen darf. Voreingestellt ist, daß nicht auf das Return gewartet wird.

Für alle dieser Zeile folgenden Konfigurationen wird `axspawn` gestartet. `axspawn` prüft, ob das auf der Kommandozeile übergebene Rufzeichen gültig ist, entfernt dann eine etwaige SSID und schaut in `/etc/passwd` nach, ob der betreffende Nutzer einen Account besitzt. Gibt es einen Account , und das Paßwort ist entweder »« (nichts) oder »+«, dann ist der Nutzer eingeloggt, steht etwas im Paßwort-Feld, so wird der Nutzer nach einem Paßwort gefragt.

Gibt es keinen Account für den Nutzer, so kann `axspawn` so eingestellt werden, daß es automatisch einen einrichtet. Achtung: Bei Distributionen, die mit dem sogenannten Password-Shadowing arbeiten, bei denen das Paßwort also nicht in `/etc/passwd` steht, kann es mit dem automatischen Anlegen von Nutzer-Accounts Probleme geben.

In diesem Fall ist es günstiger, für alle nicht speziell definierten Nutzer eine Art Gast-Account vorzusehen, auf den jeder von ihnen zugreifen kann.

14.1 Die Datei /etc/ax25/axspawn.conf

Das Verhalten von `axspawn` kann mit der Datei `/etc/ax25/ax25spawn.conf` in verschiedener Weise beeinflußt werden. Diese Datei hat folgendes Format:

```
# /etc/ax25/axspawn.conf
#
# Automatische Erzeugung von Accounts für Nutzer?
create   yes
#
# Gastzugang, wenn oben 'no' eingestellt ist - oder nichts geht
# Gastzugang ausgeschaltet mit 'no'
guest    no
#
# Gruppen-ID oder Name für Auto-Account
group    ax25
#
# Erste zu verwendende Nutzerkennung (User-ID)
first_uid  2001
#
# Maximum für User-ID
max_uid    3000
#
# Home-Verzeichnis für die neuen Nutzer
home       home/ax25
#
# Shell für die neuen Nutzer
shell      /bin/bash
#
# User-ID mit Rufzeichen für ausgehende Verbindungen verknüpfen
associate  yes
```

Ein »#« in der Datei markiert einen Kommentar; der Rest der Zeile wird ignoriert.

Folgende acht Charakteristika von `axspawn` können eingestellt werden:

create

Wenn auf »yes« gesetzt, versucht `axspawn`, einen Account für alle Nutzer, die noch nicht in `/etc/passwd` aufgeführt sind, zu erzeugen.

guest

Dieses Feld bezeichnet den Loginnamen, der verwendet wird, wenn create auf »no« gestellt ist. Im Normalfall ist das »guest« oder »ax25«.

group

Bezeichnet den Gruppennamen, der für neu (create) angelegte Nutzer-Accounts verwendet wird.

first_uid

Nummer der ersten Benutzerkennung (User-ID), die für mit create neu angelegte Nutzer-Accounts verwendet werden soll.

max_uid

Höchste für Benutzerkennungen von mit create angelegten Accounts zu vergebende Zahl

home

Home-(Login-)Verzeichnis für neu angelegte Nutzer-Accounts

shell

Die für die neu angelegten Accounts zu verwendende Login-Shell

associate

Legt fest, ob abgehende Verbindungen unter dem Rufzeichen des ankommenden Nutzers oder dem der eigenen Station laufen sollen.

15 Das PMS (Personal Message System) einrichten

Das Programm `pms` ist eine Implementation eines einfachen Personal Message Systems. Es wurde von Alan Cox geschrieben und von Dave Brown, NR2JT (`dcb@vectorbd.com`) weiter verbessert.

Zur Zeit ist das Programm noch sehr einfach, es erlaubt nur, Mitteilungen an den Eigentümer des Systems zu richten und einige begrenzte Systeminformationen abzufragen. Dave arbeitet daran, die Möglichkeiten des Programms zu erweitern. Es gibt einige Dateien, die man erzeugen sollte, um den Nutzern einige Informationen über das eigene System zu geben.

Weiterhin müssen der Datei `/etc/ax25/ax25d.conf` die entsprechenden Einträge hinzugefügt werden, um das PMS den Nutzern zur Verfügung zu stellen.

15.1 Die Datei /etc/ax25/pms.motd

Diese Datei enthält den »Spruch des Tages« (Message Of The Day), den die Nutzer nach Aufbau der Verbindung und Empfang des üblichen BBS-ID-Headers lesen können.

Es ist eine einfache Textdatei, deren Inhalt an die Nutzer gesendet wird.

15.2 Die Datei /etc/ax25/pms.info

Diese ist ebenfalls eine einfache Textdatei, die detailliertere Informationen über die eigene Systemkonfiguration oder Station enthält. Sie wird als Antwort auf den info-Befehl ausgesendet, den die Nutzer am PMS> - Prompt eingeben können.

15.3 AX.25-Rufzeichen Systembenutzern zuordnen

Sendet ein mit dem System verbundener Nutzer eine Mail an ein AX.25-Rufzeichen, so erwartet `pms`, daß das Rufzeichen einem realen Systembenutzer auf der Maschine zugeordnet ist.

Dies wird in einem gesonderten Abschnitt beschrieben.

15.4 PMS in die Datei /etc/ax25/ax25d.conf einbauen

Dies ist sehr einfach. Eine Sache muß aber beachtet werden: Dave hat für `pms` Kommandozeilenparameter vorgesehen, damit dieses verschiedene Konventionen für die Zeilenendmarkierung verarbeiten kann. AX.25 und NetROM erwarten die Zeilenendmarkierung als Carriage Return+Linefeed (CR+LF), während unter Unix ein New Line Standard ist.

Will man also beispielsweise einen Eintrag in `/etc/ax25/ax25d.conf` vorsehen, der für jede auf einem AX.25-Port hereinkommende Verbindung das PMS startet, so fügt man etwa eine solche Zeile hinzu:

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

Damit wird das PMS gestartet und ihm mitgeteilt, daß es mit einer AX.25-Verbindung assoziiert ist und der Eigentümer des PMS VK2KTJ ist. Für andere Verbindungsarten gibt die Hilfeseite weiterführende Auskünfte.

15.5 Das PMS testen

Als Test für das PMS eignet sich z.B. folgender Befehl, der auf der Kommandozeile eingegeben wird:

```
/usr/sbin/pms -u vk2ktj -o vk2ktj
```

An Stelle von »vk2ktj« gibt man sein eigenes Rufzeichen an. PMS wird gestartet, es verwendet die unter UNIX übliche Zeilenendemarkierung und der eingeloggte Nutzer ist hier vk2ktj. Jetzt kann man alle Befehle, die den verbundenen Nutzern zur Verfügung stehen, ausprobieren.

Zusätzlich kann man eine andere Station bitten, sich mit der eigenen zu verbinden, um zu prüfen, ob die Einstellungen in `/etc/ax25/ax25d.conf` funktionieren.

16 Die user_call - Programme einrichten

Die user_call-Programme heißen in Wirklichkeit `ax25_call` und `netrom_call`. Es sind sehr einfache Programme, die für den Aufruf durch den `ax25d` vorgesehen sind und Netzwerkverbindungen zu entfernten Rechnern automatisieren sollen. Natürlich können sie auch von Shellskripts oder anderen Programmen, wie `node`, aufgerufen werden.

Die Programme sind ähnlich wie das Programm `call`: Sie verändern die Daten nicht, man muß sich also selbst darum kümmern, wie beispielsweise das Zeilenende behandelt werden soll.

Zunächst ein Beispiel für die Verwendung. Angenommen, man habe ein kleines Netzwerk zu Hause und man habe eine Linux-Maschine als Funk-Gateway und einen BPQ-Node, über Ethernet mit dieser verbunden.

Im Normalfall müßten Nutzer über Funk die Linux-Maschine als Digipeater verwenden oder eine Verbindung zum Node-Programm aufbauen und sich von da aus weiterverbinden lassen, wenn sie den BPQ-Node erreichen wollten. Dies kann durch das Programm `ax25_call` erleichtert werden, wenn es durch `ax25d` aufgerufen wird.

Angenommen, der BPQ-Node habe das Rufzeichen VK2KTJ-9 und die Linux-Maschine habe den AX.25/Ethernet-Port namens bpq sowie einen Funk-Port namens radio. Mit einem Eintrag in `/etc/ax25/ax25d.conf`

```
[VK2KTJ-1 via radio] default *  *  *  *  *  *  * root /usr/sbin/ax25_call ax25_call bpq %u vk
9
```

erlaubte man Nutzern eine direkte Verbindung zu VK2KTJ-1 und damit zum Linux-AX.25-Daemon, der sie dann automatisch auf eine AX.25-Verbindung zu VK2KTJ-9 über das Interface bpq schalten würde.

Es gibt eine Menge möglicher Konfigurationen. Die Programme `netrom_call` und `rose_call` funktionieren analog zu `ax25_call`.

Ein Funkamateur verwendete dieses Utility, um Verbindungen zu einer entfernten Mailbox leichter aufbauen zu können. Im Normalfall hatten die Nutzer eine lange Zeichenkette zum Aufbau der Verbindung einzugeben.

Deshalb erzeugte er einen Eintrag, der die Box als im lokalen Netzwerk befindlich erscheinen ließ und übertrug `ax25d` und `ax25_call` den weiteren Verbindungsaufbau (Proxy-Prinzip).

17 Die Befehle zum ROSE-Up- und -Downlink einrichten

Wer sich mit der ROM-basierten ROSE-Implementation auskennt, wird auch die Art des Verbindungsaufbaus über ein ROSE-Netzwerk kennen. Hat ein lokaler ROSE-Node eines AX.25-Nutzers das Rufzeichen VK2KTJ-5 und der Nutzer will eine Verbindung zu VK5XXX auf dem entfernten Node 5050882960, dann wird er folgenden Befehl eingeben:

```
c vk5xxx v vk2ktj-5 5050 882960
```

Die entfernte Station VK5XXX würde dann eine unter dem Rufzeichen des lokalen Nutzers über das Rufzeichen des entfernten ROSE-Nodes (via) hereinkommende Verbindung erkennen.

Diese Funktionen werden bei der ROSE-Implementation unter Linux nicht vom Kernel unterstützt, es gibt zwei Programme namens `rsuplnk` und `rsdownlnk` dafür.

17.1 Einen ROSE-Downlink einrichten

Damit die eigene Linux-Maschine hereinkommende ROSE-Verbindungen annimmt und entsprechend an das gewünschte Zielrufzeichen weiterleitet, muß der Datei `/etc/ax25/ax25d.conf` ein Eintrag hinzugefügt werden.

Im Normalfall wird dieser als Voreinstellung für alle hereinkommenden ROSE-Verbindungen eingerichtet. Beispielsweise kann man ROSE-Nutzer für HEARD-0 oder NODE-0 lokal arbeiten lassen, für alle anderen Zielrufzeichen `rsdownlink` aufrufen:

```
#
{* via rose}
NOCALL  *  * *  *  *  *  L default *  *  *
*  *  * - root /usr/sbin/rsdwnlnk rsdwnlnk 4800 vk2ktj-5
#
```

Bei dieser Einstellung wird jede über die ROSE-Node-Adresse hereinkommende Verbindung in eine AX.25-Verbindung auf dem AX.25-Port mit Namen 4800 mit dem Digipeater-Pfad VK2KTJ-5 umgewandelt, sofern kein gesondert zu behandelndes Zielrufzeichen angegeben wurde.

17.2 Einen ROSE-Uplink einrichten

Damit die eigene Linux-Maschine in gleicher Weise wie das ROM-basierte ROSE Verbindungen entgegennimmt, muß ein Eintrag in die Datei `/etc/ax25/ax25d.conf` eingefügt werden, etwa so:

```
#
[VK2KTJ-5* via 4800]
NOCALL *  *  *  *  *  *  L default *
*  *  *  *  * - root /usr/sbin/rsuplnk rsuplnk rose
#
```

Man beachte die spezielle Syntax für das lokale Rufzeichen. Das Zeichen »*« zeigt an, daß die Anwendung (hier `rsuplnk`) gestartet werden soll, wenn das Rufzeichen im Digipeater-Pfad einer Verbindung auftaucht. Mit dieser Einstellung würde man einem AX.25-Nutzer erlauben, ROSE-Verbindungen mit dem unter 18. einleitend angegebenen Befehl herzustellen. Jeder Nutzer, der den Digipeater über VK2KTJ-5 auf dem AX.25-Port 4800 nutzen will, wird über `rsuplnk` weiterverbunden.

18 AX.25-Rufzeichen Linux-Nutzern zuordnen

In vielen Situationen ist es wünschenswert, ein Rufzeichen einem Linux-Nutzer-Account zuzuordnen.

Beispielsweise könnten mehrere Funkamateure die gleiche Linux-Maschine nutzen und unter ihrem eigenen Rufzeichen Verbindungen aufbauen wollen. Oder Nutzer des PMS wollen einem bestimmten Linux-Nutzer eine Nachricht zukommen lassen.

Die AX.25-Software stellt eine Möglichkeit zur Verfügung, die Account-Namen von Linux-Nutzern mit Rufzeichen zu verbinden. Dies wurde im Abschnitt PMS bereits erwähnt, soll hier aber genauer dargestellt werden.

Die Verknüpfung wird mit dem Befehl `axparms` hergestellt, beispielsweise so:

```
axparms -assoc vk2ktj terry
```

Dieser Befehl verknüpft einen User-Account namens terry auf der Maschine mit dem Funk-Rufzeichen vk2ktj. Jede Nachricht vom PMS an vk2ktj wird jetzt an den Account terry weitergeleitet.

Nicht vergessen sollte man, diese Verknüpfungen in das beim Starten des Systems ausgeführte Skript einzubauen, damit sie bei jedem Neustart zur Verfügung stehen.

Beachte: Man sollte *nie* den root-Account mit einem Rufzeichen verknüpfen, da dies zu Konfigurationsproblemen in anderen Programmen führen kann. In den meisten Fällen funktionieren diese dann nicht wie erwartet.

19 Die Einträge im /proc-Dateisystem

In dem Dateisystem unter `/proc` finden sich eine Reihe für die AX.25- und NetROM-Kernel-Software spezifische Dateien. Sie werden normalerweise von den AX.25-Utilities genutzt, sind aber einfach formatiert, so daß es interessant sein kann, ihren Inhalt zu studieren. Da das Format so einfach ist, sollte keine spezielle Erklärung notwendig sein.

/proc/net/arp

Enthält die Liste der Address Resolution Protocol-Mappings zu Adressen auf MAC-Protokollebene. Diese können AX.25, Ethernet oder eine andere MAC-Protokollebene sein.

/proc/net/ax25

Enthält eine Liste der geöffneten AX.25-Sockets. Dies können aktive Sitzungen oder solche, die auf eine Verbindung warten, sein.

/proc/net/ax25_bpqether

Enthält die Mappings für die Rufzeichen für AX.25 über BPQ-Ethernet.

/proc/net/ax25_calls

Enthält die Zuordnungen von User-IDs zu Rufzeichen, wie sie mit `axparms -assoc` eingestellt wurden.

/proc/net/ax25_route

Enthält den AX.25-Digipeater-Pfad

/proc/net/nr

Enthält eine Liste der geöffneten NetROM-Sockets. Dies können aktive Sitzungen oder solche, die auf eine Verbindung warten, sein.

/proc/net/nr_neigh

Enthält Informationen über die der NetRom-Software bekannten benachbarten NetROM-Stationen.

/proc/net/nr_nodes

Enthält Informationen über die der NetRom-Software bekannten NetROM-Nodes.

/proc/net/rose

Enthält eine Liste der geöffneten ROSE-Sockets. Dies können aktive Sitzungen oder solche, die auf eine Verbindung warten, sein.

/proc/net/rose_neigh

Enthält Informationen über die der ROSE-Software bekannten benachbarten ROSE-Stationen.

/proc/net/rose_nodes

Enthält eine Zuordnung von ROSE-Zielen zu benachbarten ROSE-Stationen.

/proc/net/rose_routes

Enthält eine Liste aller bestehenden ROSE-Verbindungen

20 AX.25-, NetROM- und ROSE-Programmierung

Der vielleicht größte Vorteil bei der Nutzung der kernelbasierten Implementationen der Packet-Radio-Protokolle besteht in der Leichtigkeit, mit der Programme und Anwendungen zu ihrer Nutzung entwickelt werden können. Das Thema Netzwerk-Programmierung unter UNIX würde den Rahmen dieses Textes sprengen; hier sollen die elementaren Details der Nutzung der AX.25-, NetROM- und ROSE-Protokolle in eigener Software besprochen werden.

20.1 Adreßfamilien

Die Netzwerkprogrammierung für AX.25, NetROM und ROSE ist der TCP/IP-Programmierung unter Linux sehr ähnlich.

Die hauptsächlichen Unterschiede ergeben sich aus den verwendenden Adreßfamilien und -strukturen, die entsprechend angeordnet werden müssen. Die Adreßfamilien tragen die Namen AF_AX25, AF_NETROM und AF_ROSE für die jeweiligen Protokolle.

20.2 Headerdateien

In eigenen Quelltexten muß immer ein »include«-Statement für `ax25.h`, `netrom.h` oder `rose.h` vorgesehen werden, wenn man mit diesen Protokollen arbeitet.

Für AX.25:

```
#include <ax25.h>
int s, addrlen = sizeof(struct full_sockaddr_ax25);
struct full_sockaddr_ax25 sockaddr;
sockaddr.fsa_ax25.sax25_family = AF_AX25
```

Für NetRom:

```
#include <ax25.h>
#include <netrom.h>
int s, addrlen = sizeof(struct full_sockaddr_ax25);
struct full_sockaddr_ax25 sockaddr;
sockaddr.fsa_ax25.sax25_family = AF_NETROM;
```

Für ROSE:

```
#include <ax25.h>
#include <rose.h>
int s, addrlen = sizeof(struct sockaddr_rose);
struct sockaddr_rose sockaddr;
sockaddr.srose_family = AF_ROSE;
```

20.3 Rufzeichenhandhabung und Beispiele

Die Rufzeichenumwandlungen werden von Routinen erledigt, welche in der zu den AX.25-Utilities gehörenden Bibliothek `/lib/ax25.a` stehen.

Natürlich kann man sich auch seine eigenen schreiben. Die User_call-Programme sind ein gutes Beispiel, von dem man starten kann. Ihre Quelltexte sind im Paket der AX.25-Utilities mit enthalten.

Wenn man sich ein wenig damit befaßt, wird man feststellen, daß 90% der Arbeit in der Vorbereitung zum Öffnen des Sockets bestehen. Die Herstellung einer Verbindung ist einfach, nur die Vorbereitung braucht Zeit.

Diese Beispiele sollten so einfach sein, daß sie nicht zu Verwirrungen führen. Wer Fragen hat, sollte sie an die Linux-Hams-Mailingliste schicken. Es wird sich dann sicherlich jemand finden, der weiterhilft.

21 Einige Beispielkonfigurationen

Im folgenden werden Beispiele für einige der üblichsten Konfigurationen vorgestellt. Sie sollen als Anregung für die eigene Konfiguration dienen.

21.1 Kleines Ethernet-LAN mit Linux als Router auf Funk-LAN

```
---                                    .
 | Netzwerk       /---------\      .    Netzwerk
 | 44.136.8.96/29 |         |     .     44.136.8/24        \ | /
```

```
  |                 | Linux    |    .                          \|/
  |                 |          |   .                            |
  |           eth0  | Router   |   .  /-----\    /----------\   |
  |-----------------|          |-----| TNC |----| Radio    |---/
  |    44.136.8.97  |  und     |   .  \-----/    \----------/
  |                 |          | sl0
  |                 | Server   | 44.136.8.5
  |                 |          |     .
  |                 |          |      .
  |                 \__________/        .
---                                         .      .   .    .    . .

#!/bin/sh
# /etc/rc.net
# Diese Konfiguration stellt einen KISS basierten AX.25-Port und
# ein Ethernet-Device zur Verfügung.
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: "
kissattach -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
```

/etc/ax25/axports

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

/etc/ax25/nrports

```
# name
callsign       alias   paclen  description netrom
VK2KTJ-9       LINUX   235     Linux Switch Port
```

/etc/ax25/nrbroadcast

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

- IP_FORWARDING muß im Kernel eingeschaltet sein.

- Die AX.25-Konfigurationsdateien entsprechen weitestgehend den weiter oben aufgeführten Beispielen, siehe auch dort.
- Für den Funkport wurde eine IP-Adresse gewählt, die nicht im Adreßblock des eigenen Netzwerkes enthalten ist. Das ist nicht unbedingt so notwendig, genausogut hätte man auch 44.136.8.97 für diesen Port vorsehen können.
- 44.136.8.68 ist hier der lokale IPIP-gekapselte Gateway - daher wird die voreingestellte Route dorthin gesetzt.
- Jede der Maschinen im Ethernet-Netzwerk hat eine Route:

  ```
  route add -net 44.0.0.0 netmask 255.0.0.0 \
        gw 44.136.8.97 window 512 mss 512 eth0
  ```

  Mit der Verwendung der MSS und Window-Parameter erhält man optimale Performance sowohl im Ethernet als auch bei den Funkverbindungen.
- Auf dem Router laufen ebenso die Daemonen smail, http, ftp und andere, so daß sie die einzige Maschine ist, die anderen Nutzern Dienste anbietet.
- Der Router ist ein langsamer 386DX20 mit 20MB-Festplatte und minimaler Linux-Konfiguration.

21.2 IPIP-gekapselter Gateway

Linux wird oft auch für TCP/IP-gekapselte Gateways verwendet. Der neue Tunnel-Treiber unterstützt mehrere gekapselte Routen und löst den veralteten ipip-Daemon ab. Eine typische Konfiguration würde etwa so aussehen:

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

Einige zu beachtende Punkte:

- Der neue Tunnel-Treiber nutzt das GW-Feld in der Routing-Tabelle an Stelle des pointopoint-Parameters, zur Angabe der Adresse des entfernten IPIP-Gateway. Auf diese Weise unterstützt er nun mehrere Routen pro Interface.
- Man kann zwei Netzwerk-Devices mit gleicher Adresse einrichten. In diesem Beispiel wurden die Devices sl0 und tunl0 mit der IP-Adresse des Funkports konfiguriert. Auf diese Weise erkennt der entfernte Gateway die korrekte Adresse des eigenen Gateways in den gekapselten Paketen, die an ihn gesendet wurden.
- Die Route-Befehle zur Angabe der gekapselten Routen können automatisch mit einer modifizierten Version des Munge-Skripts erzeugt werden, die weiter unten abgedruckt ist. Die Route-Befehle werden dann in eine separate Datei geschrieben und mit dem Bash-Befehl

```
source /etc/ipip.routes
```

(angenommen, die Datei mit den Route-Befehlen heiße `/etc/ipip.routes`) wie dargestellt eingelesen. Die Quelldatei muß das NOS-Format für Route-Befehle haben.

- Beachte die Verwendung des Window-Arguments bei dem Route-Befehl. Der richtige Wert für diesen Parameter erhöht die Performance der Funkverbindung.

Das neue Tunnel-munge-Script:

```
#!/bin/sh
#
#  Von: Ron Atkinson
#
#  Dieses Skript basiert auf dem ursprünglich für den IPIP-
#  Daemon geschriebenen »munge«-Skript von Bdale N3EUA,
#  es wurde von Ron Atkinson N8FOW modifiziert.
#  Es dient zur Umwandlung einer Gateway-Route-Datei im KA9Q NOS-Format
#  (üblicherweise »encap.txt«) in das Format der Linux-Routing-Tabelle
#  für den IP -Tunnel-Treiber.
#
#  Aufruf: Gateway-Datei auf stdin, Linux -Routingtabelle auf stdout.
#          z.B.  tunnel-munge < encap.txt > ampr-routes
#
#  ACHTUNG: Bevor das Skript nutzbar ist, muß folgendes überprüft werden:
#
#     1) Trage für »Local routes« und »Misc user routes« die im eigenen
#        Gebiet gültigen Routen ein und entferne die Beispiele!
#     2) Auf der »fgrep«-Zeile muß die EIGENE Internet-Gateway-Adresse
#        eingetragen werden. Wird dies nicht getan, so entstehen
#        ernsthafte Probleme durch Routing-Schleifen.
#     3) Der voreingestellte Name des Interfaces ist »tunl0«.
#        Es ist sicherzustellen, daß dies auf dem eigene System zutrifft.
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"
```

Für den Kernel 2.2.x mit aktuellem AX.25-Subsystem gilt ein neues Skript:

```
#!/usr/bin/perl
#
#  Von: Ron Atkinson
#
#  Dieses Skript basiert auf dem ursprünglich für den IPIP-
#  Daemon geschriebenen »munge«-Skript von Bdale N3EUA,
#  es wurde von Ron Atkinson N8FOW modifiziert.
#  Es dient zur Umwandlung einer Gateway-Route-Datei im KA9Q NOS-Format
#  (üblicherweise »encap.txt«) in das Format der Linux-Routing-Tabelle
#  für den IP -Tunnel-Treiber.
#
# Verbesserte Perl-Version by Heikki Hannikainen
```

```
#
# Versionen:
#
#       2.1     hessu   Thu Jul  8 19:41:18 EEST 1999
#               - Modifiziert für /sbin/ip
#
#  Aufruf: z.B.    munge encap.txt rc.tun-routes
#
#  ACHTUNG: Bevor das Skript nutzbar ist, muß folgendes
#  überprüft werden:
#
#     1) Die @my_addresses-Liste muß STATT DER in diesem
#        Text stehenden Adressen die IP-Adresse des eigenen
#        Gateways (und Adressen, für die manuell eingetragene
#        statische Routen existieren) enthalten.
#
# verwendetes Tunnel-Device
$tunnel_device = "tunl0";
# Adressen lokaler Gateways (deren Routen werden übergangen)
@my_addresses = ("193.167.178.112", "193.166.55.160", "195.148.207.30"); #
route binary $routebin = "/sbin/ip"; # tcp window to set $window = 840;

if ($#ARGV != 1) {
 print "Aufruf: $0 encap.txt rc.tun-routes\n";
 exit 1;
}

open(inf, @ARGV[0]) || die "Kann nicht oeffnen: @ARGV[0]: $!";
open(upf, ">@ARGV[1]") or die "Kann nicht oeffnen: @ARGV[1]: $!";;

$hdr = "#\n# IP tunnel route table\n#\n#\n";
print upf $hdr;

LOOP: while ($line = ) {

 @fields = ();
 @abytes = ();
 $bits = "";

 @fields = split(' ', $line);
 if (@fields[0] ne "route") { next LOOP; }

 $gw = @fields[4];
 $net = @fields[2];
 ($addr, $bits) = split('/', $net);
 if (!$bits) { $bits = 32; }
 @abytes = split('\.', $addr);

 for ($i = 0; $i < 4; $i++) {
  if (@abytes[$i] eq "") { @abytes[$i] = "0"; }
 }
 $addr = join('.', @abytes);

 foreach $my (@my_addresses) {
  if ($gw eq $my) {
```

```
       print "$addr/$bits > $gw blocked, local gw\n";
       next LOOP;
      }
     }

     if ($addr !~ /^44\./) {
      print "$addr/$bits > $gw blocked, non-amprnet address!\n";
      next LOOP;
     }

     if ($gw =~ /^44\./) {
      print "$addr/$bits > $gw blocked, inside amprnet!\n";
      next LOOP;
     }

     if ($bits < 16) {
      print "$addr/$bits > $gw  blocked, mask smaller than 16 bits!\n";
      next LOOP;
     }

     print upf "$routebin route add $addr/$bits via $gw dev $tunnel_device
     onlink 2>/dev/null\n";
    }

    print upf "#\n# the end\n#\n";

    close(inf);
    close(upf);
```

Zum Schluß ist das Tunnel-Device mit

```
ifconfig tunl0 44.139.39.66 mtu 250 up
```

zu aktivieren, wobei für 44.139.39.66 die eigene IP-Adresse einzusetzen ist.

21.3 Die Einrichtung des AXIP-encapsulated Gateway

Auf vielen Internet-Gateways des Amateurfunks werden AX.25, NetRom und ROSE zusätzlich zu TCP/IP gekapselt.

Die Kapselung von AX.25-Frames innerhalb von IP-Datagrammen wird in RFC-1226 von Brian Kantor beschrieben.

Mike Westerhof schrieb 1991 einen Daemon für AX.25-Kapselung für UNIX. Die AX-25-Utils enthalten eine leicht verbesserte Version für Linux. Ein Programm zur AXIP-Kapselung nimmt an einem Ende AX.25-Frames entgegen, prüft die AX.25-Empfängeradresse, um festzulegen, an welche IP-Adresse das gekapselte Paket weitergeleitet werden muß, kapselt es sodann in ein TCP/IP-Datagramm und sendet es schließlich an das passende Ziel.

In umgekehrter Richtung nimmt es in TCP/IP-Datagrammen enthaltene AX.25-Frames entgegen, packt diese aus und behandelt sie weiter, als ob sie direkt von einem AX.25-Port empfangen worden wären.

Um TCP/IP-Datagramme, die AX.25-Frames enthalten, von anderen ohne solchen Inhalt unterscheiden zu können, werden AXIP-Datagramme mit einer Protokoll-Identifizierung von 4 (oder früher 94) versehen.

Dies wird in RFC-1226 beschrieben. Das in den AX-25-Utils enthaltene Programm `ax25ipd` stellt sich als KISS-Interface, über das AX.25-Frames übertragen werden, und als Interface zu TCP/IP dar. Es wird mit einer einzigen Konfigurationsdatei, `/etc/ax25/ax25ipd.conf`, eingerichtet.

21.3.1 Einstellungsoptionen für AXIP

Das Programm `ax25ipd` kennt zwei hauptsächliche Betriebsarten, »Digipeater«- und »TNC«-Modus.

Im TNC-Modus wird der Daemon wie ein KISS-TNC verwendet, man schickt KISS-Pakete an das Programm und es sendet diese weiter - dies ist die übliche Konfiguration.

Im »Digipeater«-Modus wird der Daemon wie ein AX.25-Digipeater behandelt. Es gibt einige feine Unterschiede zwischen diesen Betriebsarten. In der Konfigurationsdatei definiert man »routes« oder mappings zwischen den AX.25-Ziel-Rufzeichen und den IP-Adressen der Rechner, denen man die Pakete schicken will.

Jede Route besitzt Optionen, die später beschrieben werden. Weiterhin kann man hier folgendes einrichten:

- die tty-Schnittstelle, die vom `ax25ipd` geöffnet werden soll und deren Geschwindigkeit (üblicherweise ein Ende einer Pipe)
- das Rufzeichen, welches für den »Digipeater«-Modus verwendet werden soll
- Bakenintervall und -text
- ob die AX.25-Frames in IP-Datagramme oder in UDP/IP-Datagramme gekapselt werden sollen.

So gut wie alle AXIP-Gateways verwenden IP-Kapselung, doch einige befinden sich hinter Firewalls, die keine IP-Pakete mit der Protokoll-ID des AXIP durchlassen, und sind daher gezwungen, UDP/IP zu verwenden. Die hier getroffene Einstellung muß mit der des TCP/IP-Zielrechners übereinstimmen.

21.3.2 Eine typische Datei /etc/ax25/ax25ipd.conf

```
#
# ax25ipd -Konfigurationsdatei für die Station floyd.vk5xxx.ampr.org
#
# Datentransportart wählen. »ip« für die Kompatibilität
# mit den meisten anderen Gateways.
socket ip
#
# Betriebsart des ax25ipd einstellen. (digi oder tnc)
#
mode tnc
#
# Wurde digi eingestellt, muß ein Rufzeichen angegeben werden.
# Im tnc -Modus ist das rufzeichen zur Zeit optional, doch das kann
# sich in Zukunft ändern! (2 Rufzeichen bei dual port kiss)
#
#mycall vk5xxx-4
#mycall2 vk5xxx-5
#
# In digi -Modus können Aliases genutzt werden. (2 für dual port)
#
#myalias svwdns
#myalias2 svwdn2
#
# Aussenden einer Indentifikation aller 540 Sekunden ...
#
beacon after 540
#
btext ax25ip -- tncmode rob/vk5xxx -- Experimenteller AXIP Gateway
```

```
#
# Serielle Schnittstelle oder mit einem kissattach verbundene Pipe
# in diesem Fall:
device /dev/ttyq0
#
# Transferrate einstellen:
#
speed 9600
#
# loglevel 0 - keine Ausgabe
# loglevel 1 - nur Konfigurationsinformationen
# loglevel 2 - Schwerwiegende Ereignisse und Fehler
# loglevel 3 - Schwerwiegende Ereignisse und Fehler, AX.25-Paketmitschnitt
# loglevel 4 - Alle Ereignisse
#
loglevel 2
#
# im Digi-Modus mit einem echten TNC verwendet man param zur Einstellung
# der TNC-Parameter...
#
#param 1 20
#
# Definition der Broadcast-Adresse . Jede der aufgeführten Adressen
# wird an jede als broadcastfähig markierte Route weitergeleitet
#
broadcast QST-0 NODES-0
#
# Definition der AX.25-Routen, soviele einstellbar, wie benötigt.
# Format: Route (Rufzeichen/Wildcard) (Adresse des IP-Zielrechners)
# ssid of 0 routes all ssid's
#
# route   [flags]
#
# Gültige Flags:
#         b - Broadcasts dürfen über diese Route
#             geschickt werden
#         d - diese Route ist die standardmäß voreingestellte
#   route vk2sut-0 44.136.8.68 b
route vk5xxx 44.136.188.221 b   route vk2abc 44.1.1.1
#
#
```

21.3.3 ax25ipd starten

Zunächst wird der Datei `/etc/ax25/axports` ein Eintrag hinzugefügt:

```
# /etc/ax25/axports
#
axip   VK2KTJ-13      9600    256     AXIP port
#
```

Dann wird `kissattach` gestartet, um diesen Port zu erzeugen:

```
/usr/sbin/kissattach /dev/ptyq0 axip
```

Schließlich startet man `ax25ipd`:

```
/usr/sbin/ax25ipd &
```

Die AXIP-Verbindung testet man mit:

```
call axip vk5xxx
```

21.3.4 Einige Bemerkungen zu Routen und Flags

Mit dem »route«-Befehl gibt man an, wo die AX.25-Pakete gekapselt hingeschickt werden sollen. Empfängt `ax25ipd` ein Paket von seinem Interface, so vergleicht das Programm das Zielrufzeichen mit jedem Rufzeichen in seiner Routing-Tabelle. Bei einer Übereinstimmung wird das AX.25-Paket in ein IP-Datagramm gekapselt und an den Rechner mit der entsprechend angegebenen IP-Adresse geschickt. Zwei Flags können jedem »route« Befehl in der Datei `/etc/ax25ipd.conf` hinzugefügt werden. Es sind folgende:

b

Pakete mit einer Zieladresse, die einer der mit dem Schlüsselwort »Broadcast« definierten Adressen entspricht, sollen über diese Route weitergeleitet werden.

d

Pakete, die mit keiner festgelegten Route übereinstimmen, sollen über diesen Weg weitergeleitet werden.

Das Flag »b« (Broadcast) ist sehr nützlich, da es ermöglicht, Informationen, die normalerweise für alle Stationen gedacht waren, auch einer bestimmten Anzahl AXIP-Stationen zukommen zu lassen. Normalerweise sind AXIP-Routen Punkt-zu-Punkt-Verbindungen und können nichts mit »Broadcast«-Paketen anfangen.

21.4 Wie verbindet man NOS und die Linux-Kernel-Netzwerk-Software

Viele Leute verwenden eine NOS-Version unter Linux, da es alle von ihnen gewünschten Eigenschaften und Möglichkeiten bietet. Die meisten von ihnen würden gern das NOS mit dem Linux-Kernel verbinden, so daß einige der Linux-Möglichkeiten für Funk-Nutzer zur Verfügung stünden.

21.4.1 NOS und Linux mit einer Pipe verbinden

Von Brandon S. Allbery, KF8NH, wurden die folgenden Informationen beigesteuert, die erklären, wie man das NOS mit dem Linux-Kernel-Code mittels einer Linux-Pipe verbindet.

Da sowohl Linux als auch NOS das SLIP-Protokoll unterstützen, kann man beide über SLIP miteinander verbinden. Man könnte dazu zwei serielle Schnittstellen mit einem Loopback-Kabel verwenden, doch das wäre langsam und kostspielig. Linux stellt wie viele andere UNIX-ähnliche Betriebssysteme sogenannte Pipes zur Verfügung. Dabei handelt es sich um Pseudo-Devices, die für die Software wie TTY-Devices erscheinen, aber in Wirklichkeit eine Schleife (Loopback) auf ein anderes Pipe-Device darstellen.

Um diese Pipes zu verwenden, muß das erste Programm das »master«-Ende und das zweite das »slave«-Ende der Pipe öffnen. Siehe dazu auch den Absatz »Dual-Port-TNCs einrichten« unter dem Abschnitt »7.1. KISS«.

Sind beide Enden eröffnet, so können die beiden Programme miteinander Daten austauschen, indem sie Zeichen in die Pipe wie in ein normales TTY-Device schreiben.

Damit das Ganze genutzt werden kann, um den Linux-Kernel und NOS oder irgendein anderes Programm miteinander zu verbinden, muß zunächst das zu verwendende Pipe-Device aus dem `/dev`-Verzeichnis ausgewählt werden. Die »master«-Enden der Pipes tragen die Namen `ptyq1` bis `ptyqf`, die »slave«-Enden `ttyq1` bis `ttyqf`. Zu jedem »master« gehört immer ein »slave«, so daß man `ttyqf` als »slave« verwenden muß, wenn `ptyqf` als »master« ausgewählt wurde.

Wenn man nun ein solches Device-Paar gewählt hat, sollte man das »master«-Ende mit dem Linux-Kernel und das »slave«-Ende mit NOS verbinden, da der Linux-Kernel als erstes startet und das »master«-Ende immer vor dem »slave«-Ende geöffnet werden muß.

Ebenso muß der Linux-Kernel eine andere IP-Adresse als NOS haben, so daß man diesem eine eigene IP-Adresse zuweisen sollte, wenn das nicht bereits geschehen ist.

Die Pipe wird wie ein serielles Device eingerichtet, um eine SLIP-Verbindung vom Linux-Kernel aus zu erstellen, kann man etwa folgende Befehle verwenden:

```
/sbin/slattach -s 38400 -p slip /dev/ptyqf &
/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 netmask 255.0.0.0 \
            gw 44.70.248.67
```

In diesem Beispiel hat der Linux-Kernel die IP-Adresse 44.70.4.88 und NOS die IP-Adresse 44.70.248.67.

Der route-Befehl in der letzten Zeile veranlaßt den Linux-Kernel, alle Datagramme (Pakete) für das AMPRNet (44.0.0.0) über die SLIP-Verbindung, die mit `slattach` vorher erzeugt wurde, weiterzuleiten.

Im Normalfall kommen diese Befehle in das Skript `/etc/rc.d/rc.inet2` (Slackware) nach die übrige Netzwerkkonfiguration, damit die SLIP-Verbindung beim Neustart automatisch wiederhergestellt wird. Man beachte, daß die Verwendung von CSLIP (compressed SLIP) hier keinen Vorteil bringt, da die Verbindung nur virtuell ist und durch das Komprimieren ein Performanceverlust eintreten würde.

Dadurch würden die Pakete mit komprimiertem Header langsamer übertragen als die unkomprimierten Pakete. Um das NOS-seitige Ende der SLIP-Verbindung einzurichten, kann man diese Befehle verwenden:

```
# Das Interface kann einen beliebigen Namen tragen; hier wird
# wegen der Eindeutigkeit »linux« verwendet
attach asy ttyqf - slip linux 1024 1024 38400 route \
       addprivate 44.70.4.88 linux
```

Damit wird ein SLIP-Port mit Namen linux über das »slave«-Ende der Pipe zum Linux-Kernel erzeugt. Der Route-Befehl aktiviert diesen dann.

Wenn NOS gestartet wurde, sollte man jetzt in der Lage sein, ping und telnet von NOS nach Linux und umgekehrt ausführen zu können. Funktioniert dies nicht, so muß man nochmals überprüfen, ob kein Fehler, besonders beim Konfigurieren der Adressen, aufgetreten ist und die Pipe-Devices ordnungsgemäß verfügbar sind.

22 Wo finde ich Informationen zu ...?

Da dieser Text einige Erfahrung mit Packet Radio voraussetzt und dies nicht immer der Fall ist, hat Terry einige Verweise zu nützlichen Informationsquellen zusammengestellt.

22.1 Packet Radio

Allgemeine Informationen zu Packet Radio findet man hier:

American Radio Relay League

http://www.arrl.org/

Radio Amateur Teleprinter Society

http://www.rats.org

Tucson Amateur Packet Radio Group

http://www.tapr.org/

22.2 Protokolldokumentation

Jonathan Naylor hat eine Reihe von Dokumenten gesammelt, die sich mit den im Packet Radio verwendeten Protokollen beschäftigen. Sie wurden gepackt und sind hier zu finden:

```
ftp.pspt.fi:/pub/ham/linux/ax25/ax25-doc-1.0.tar.gz
```

22.3 Hardware-Dokumentation

Informationen zur PI2-Karte werden von der Ottawa Packet Radio Group, zur Verfügung gestellt. Informationen zur Baycom-Hardware ist auf der Baycom-WWW-Seite

```
http://www.baycom.de/
```

zu finden. Eine Seite mit vielen Links zum Thema Amateurfunk und Packet Radio findet man hier:

```
http://www.ardos.de/gerd/ham.html
```

23 Diskussionsforen zu Amateurfunk und Linux

Diskussionen zu Amateurfunk und Linux finden an vielen Stellen statt. Sowohl in den Newsgroups der comp.os.linux.*-Hierarchie, als auch in der HAM-Liste auf vger.rutgers.edu kann man an ihnen teilnehmen. Ebenso gibt es noch die tcp-group-Mailingliste auf ucsd.edu als Heimat für die Diskussionen über TCP/IP im Amateurfunk und ebenso den Channel #linpeople auf IRC. Die Mailingliste für FPAC erreicht man unter fpac@qth.net.

Um der Mailingliste linux-hams beizutreten, sendet man eine E-Mail an Majordomo@vger.rutgers.edu, die den Text

```
subscribe linux-hams
```

enthält.

Die »Subject«-Zeile wird ignoriert (freilassen). Diese Mailingliste wird hier archiviert:

- http://hes.iki.fi/archive/

- `http://zone.oh7rba.ampr.org/archive/linux-hams`

Für den Anfang sollte man diese Archive verwenden, da viele häufige Fragen hier bereits beantwortet werden.

Der tcp-Gruppe tritt man so bei: E-mail an `listserver@ucsd.edu` mit der Zeile

```
subscribe tcp-group
```

als Text schicken.

Beachte: Die TCP/IP-Gruppe ist primär für Diskussionen über verbesserte Protokolle, wie TCP/IP, im Amateurfunk gedacht. Linux-spezifische Fragen sollten möglichst nicht dort gestellt werden.

24 Danksagung

Folgende Leute haben auf die eine oder andere Weise, wissentlich oder unwissentlich, zum Entstehen dieses Textes beigetragen.

In ungeordneter Reihenfolge sind dies Jonathan Naylor, Thomas Sailer, Joerg Reuter, Ron Atkinson, Alan Cox, Craig Small, John Tanner, Brandon Allbery - und nicht zuletzt der Autor des englischen Originaltextes, Terry Dawson, dem ich an dieser Stelle für seine geduldigen und kompetenten Antworten in der Linux-Ham-Mailingliste danken will.

Bitte Fragen zur deutschen Version *nicht* an Terry Dawson oder in die Linux-Ham-Mailingliste sonder an Gerd Röthig (`roetg@medizin.uni-leipzig.de`) schicken. Ich würde mich freuen, wenn ich über evtl. notwendige Veränderungen, Umformulierungen oder Korrekturen in Kenntnis gesetzt würde. Viel Freude also mit Linux und Packet Radio, 55 & 73.

25 Copyright

26 ANHANG: Einige aus dem Englischen übernommene Fachbegriffe und deren Erklärung

Broadcast

Aussendungen, die nicht an eine bestimmte Adresse, sondern an alle gerichtet sind und daher unprotokolliert gesendet werden (Beispiel: Aussendung der Routing- Informationen durch NetROM, auch Baken können in diesem Sinne als Broadcast betrachtet werden)

Device

hier im Sinne eines sogenannten »virtuellen Geräts« gebraucht, d.h., ein Treiber (Driver) stellt ein Software-Interface zur Verfügung, das ähnlich wie eine Gerätedatei angesprochen werden kann. Auf diese Weise wird die

Unterscheidung zwischen den Protokollen realisiert - jedem dieser Devices ist genau ein Übertragungsprotokoll zugeordnet. Sie können auf weiteren Devices aufsetzen, also deren Geladensein voraussetzen.

Digipeater

Station, die die eigenen Pakete, so wie sie sie empfängt, wiederholt (DIgital rePEATER), es erfolgt kein direkter Verbindungsaufbau zu diesem Rechner, sondern man baut einen Link "über"(via) ihn auf

Encapsulation

svw. Kapselung (hier auch so übersetzt), bezeichnet einen Weg der Übertragung über mit TCP/IP arbeitende Netzwerke, indem AX.25-Pakete in spezielle IP-Datagramme hineingeschrieben werden. Die Protokoll-Informationen des IP wirken dann wie eine »Kapsel« um das AX.25-Frame herum, die dessen unbeschadeten Transport und Identifikation ermöglicht

Heard, MHeard

Ausgabe, welche Stationen auf einem bestimmten Kanal gehört (aufgenommen) werden konnten

Gateway

hier: mit mehreren Netzen gleichzeitig verbundener Rechner, an den alle Pakete gehen, die innerhalb eines Netzes nicht zugestellt werden können - diese Pakete werden dann in eines der anderen erreichbaren Netze weitergeleitet

Nodes

meist TNCs mit spezieller Node-Software ohne daran angeschlossenen Host-Rechner, aber auch PCs mit entsprechender Software (z.B. node oder BPQ), zu denen man im Unterschied zu den Digipeatern eine direkte Verbindung aufbaut und ihnen dann den Befehl zum Weiterverbinden gibt

Obsolescence

(svw. Alterung) NetROM-Begriff, der die Aktualität einer Verbindung angibt, Zeitspanne, die seit der letzten gehörten Aktivität der betreffenden Station vergangen ist, zur Beurteilung der Qualität einer Funkverbindung hilfreich

Routing

Weiterleiten von Paketen/Daten auf einem vorgegebenen Weg/an einen in der Routing-Tabelle (routing table) festgelegten Rechner, evtl. auf einem dort definierten Weg (Routing-Path).

Switching

svw. Umschalten: Aussenden/Weiterleiten von Daten über einen anderen Kanal/Port, netzwerktechnisch Schalten einer Punkt-zu-Punkt-Verbindung zwischen Sender und Empfänger für die Zeitdauer der Übertragung des entsprechenden Paketes

Ein Wort zu Kernel 2.2 und dessen derzeitigem Entwicklungsstand Aktuelle Mitteilung: Das AX.25-Subsystem ist im Kernel 2.2.x noch unvollständig. Wenn man beispielsweise ein AX.25-Netzwerk-Device bei einem noch laufenden Connect herunterfährt, bekommt man einen Systemabsturz (»Oops«). Um das System wieder nutzbar zu machen, ist anschließend ein Neustart fällig.

Ich wurde gefragt, weshalb der aktuelle Kernel 2.2 momentan in diesem HOWTO keine ausführlichere Erwähnung findet. Meine persönliche Meinung dazu gibt folgendes Statement wieder:

Zum derzeitigen Zeitpunkt ist ein Upgrade auf den Kernel 2.2 keine gute Wahl. Der Kernel verwendet in vielen Bereichen eine andere Treiberarchitektur, was zur Folge hat, daß bestimmte Hardware neu eingerichtet werden muß. Was auf die Hardware-Treiber zutrifft, ist auch beim Kernel-AX.25 der Fall - eine völlige Neueinrichtung wird fällig.

Da die Systemschnittstelle auch im AX.25-Bereich abgewandelt wurde, werden ältere Binärversionen der AX.25-Programme nicht mehr vollständig funktionieren. Es kann nur geraten werden, abzuwarten, bis das neue AX.25-Subsystem und die Programmierschnittstelle fertiggestellt ist.

Zu deren Neuübersetzung siehe Bemerkung am Ende dieses Textes. Fakt ist weiterhin, daß der Kernel 2.2 höhere Speicheranforderungen als die 2.0.x-Reihe stellt. Da aber als PR-Rechner meistens ältere und weniger gut ausgestattete Systeme zum Einsatz kommen, ist ein Upgrade auch aus diesem Grunde nicht ratsam.

Hinzu kommt noch die Tatsache, daß laut Alan Cox auch die Kernel der 2.0.x-Reihe weiterentwickelt werden sollen. Man kann sich also die Upgrade-Orgie noch etwas aufsparen (neben den neuen Kernelsourcen wird die Aktualisierung einiger anderer Programmpakete fällig) und abwarten, bis es beispielsweise Version 2.2.30 gibt. Nicht ohne Grund beobachtet man derzeit eine sehr rasche Folge neuer Versionen in der 2.2.x-Reihe.