Linux AX.25-HOWTO, Amateur Radio.

Terry Dawson, VK2KTJ, terry@perf.no.itg.telstra.com.au, traduit par François Romieu, romieu@ensta.fr
v1.5, 17 Octobre 1997

Le système d'exploitation Linux est sûrement le seul au monde à pouvoir se vanter d'offrir un support natif du protocole de transmission de données AX.25 employé par les radioamateurs à travers le monde. Le présent document se veut un guide d'installation et de configuration de cette prise en charge.

Contents

1 Introduction.

Le document trouve son origine dans une annexe du HAM-HOWTO. L'importance de son développement devint cependant incompatible avec une telle organisation. L'installation et la prise en charge intégrée d'AX.25, la gestion NetRom et Rose sous Linux sont décrites. Quelques exemples de configurations typiques fournissent une base de travail.

La mise en oeuvre des protocoles radioamateurs sous Linux est très souple. Les personnes peu familières du système d'exploitation Linux trouveront peut-être la configuration un peu obscure. Il vous faudra un certain temps pour maîtriser l'interaction des différents éléments. Attendez-vous à une configuration pénible si vous ne vous êtes pas auparavant familiarisé avec Linux. N'espérez pas passer à Linux depuis un autre environnement en faisant l'économie de tout apprentissage.

1.1 Modifications par rapport à la version précédente

```
Ajouts:
        Page ouaibe de Joerg Reuters
        Section "Informations supplémentaires"
        configuration d'ax25ipd.

Corrections/Mises à jour:
        Prévention des conflits dus aux pty
        Nouvelles versions du module et des ax25-utils

A faire:
        Mettre au point la section SCC qui est sûrement erronée
        Étoffer la section touchant à la programmation
```

1.2 Nouvelles versions du document

Les archives du Projet de Documentation Linux (LDP ou Linux Documentation Project) constituent le meilleur emplacement où trouver la dernière mouture de ce texte. Le LDP tient à jour un site ouaibe dans lequel figure l'AX.25 HOWTO :

AX.25-HOWTO `http://sunsite.unc.edu/LDP/HOWTO/AX.25-HOWTO.html` . Le texte est disponible sous différents formats à l'adresse suivante :

archive ftp sunsite.unc.edu `ftp://sunsite.unc.edu/pub/Linux/docs/howto/` . La version française est accessible via :

archive Traduc.org `ftp://ftp.traduc.org/pub/HOWTO/FR` .

Vous pouvez me contacter mais comme je transmets directement les nouvelles versions au coordinateur LDP des HOWTO, l'absence d'une nouvelle version indique sûrement que je ne l'ai pas terminée.

1.3 Autres documents

La documentation sur les sujets apparentés ne manque pas. Bon nombre de textes traitent de l'utilisation générale de Linux en réseau et je vous conseille vivement de les lire : ils vous guideront dans vos efforts et offrent une vision élargie à d'autres configurations envisageables.

Par exemple :

HAM-HOWTO `http://sunsite.unc.edu/LDP/HOWTO/HAM-HOWTO.html` ,

NET-3-HOWTO `http://sunsite.unc.edu/LDP/HOWTO/NET-3-HOWTO.html` ,

Ethernet-HOWTO `http://sunsite.unc.edu/LDP/HOWTO/Ethernet-HOWTO.html` ,

et :

le Firewall-HOWTO `http://sunsite.unc.edu/LDP/HOWTO/Firewall-HOWTO.html`

Des informations plus générales sur Linux sont disponibles :

Linux HOWTO `http://sunsite.unc.edu/LDP/HOWTO/`

2 Les protocoles de paquets par radio et Linux

Le protocole *AX.25* fonctionne aussi bien en mode connecté que non-connecté et s'emploie tel quel pour des liaisons point-à-point ou pour encapsuler d'autres protocoles tels qu'IP ou NetRom.

Sa structure se rapproche de celle du niveau 2 d'X25 avec des extensions qui l'adaptent à l'environnement radioamateur.

Le protocole NetRom a pour objectif de fournir un protocole réseau complet. Il repose sur AX.25 au niveau liaison de données et procure une couche réseau dérivée d'AX.25. Le protocole NetRom autorise le routage dynamique et la création d'alias pour les noeuds.

Le protocole Rose a été initialement conçu et réalisé par Tom Moulton alias W2VY. Il constitue une mise en oeuvre du protocole par paquets X25 et peut inter-opérer avec AX.25 au niveau liaison. Il fournit des services de couche réseau. Les adresses Roses comportent 10 digits. Les quatre premiers constituent le code d'identification du réseau de données (DNIC ou Data Network Identification Code) et sont référencés dans l'Appendice B de la recommandation X121 du CCITT. Des informations supplémentaires sur le protocole Rose sont disponibles sur le site suivant :

Serveur Web RATS `http://www.rats.org/` .

Alan Cox a créé les toutes premières versions de support noyau pour AX.25. Jonathon Naylor <`g4klx@g4klx.demon.co.uk`> a poursuivi le développement, ajouté la gestion de NetRom et de Rose et assure à présent officiellement la maintenance du code noyau relatif à AX.25. La prise en compte de DAMA est l'oeuvre de Joerg, DL1BKE, `jreuter@poboxes.com`. Thomas Sailer, <`sailer@ife.ee.ethz.ch`> s'est chargé des matériels Baycom et SoundModem. J'assure pour ma part le suivi des utilitaires AX.25.

Linux gère les TNC (Terminal Node Controllers) KISS, les cartes Ottawa PI, les PacketTwin Gracilis et autres cartes à base de SCC Z8530 via le pilote SCC générique ainsi que les modems sur ports série et parallèle de Baycom. Le nouveau pilote pour modems à base de carte son de Thomas accepte les Soundblaster et les cartes à base de composants Crystal.

Le paquetage de programmes applicatifs comprend une messagerie individuelle (PMS ou Personal Message System), une balise, un programme de connexion en mode texte, un exemple de récupération des trames AX.25 au niveau de l'interface et des utilitaires de configuration du protocole NetRom. Il comprend également un serveur de type AX.25 qui gère les demandes de connexions AX.25 et un démon qui se charge de l'essentiel du travail pour le protocole NetRom.

2.1 Fonctionnement

La mise en oeuvre d'AX.25 sous Linux lui est propre de A à Z. Bien qu'elle puisse ressembler à NOS, à BPQ ou à d'autres versions d'AX.25 sur certains points, elle ne se confond avec aucune d'entre elles. La version Linux peut être configurée pour se comporter de façon voisine aux autres mais le processus n'en reste pas moins radicalement différent.

Pour vous aider à comprendre la démarche intellectuelle à suivre lors de la configuration, cette section décrit les fonctionnalités structurelles d'AX.25 et son adaptation au contexte Linux.

Diagramme simplifié des couches protocolaires

```
+----------+-----------+-------------+---------+
| AF_AX.25 | AF_NETROM |   AF_INET   | AF_ROSE |
+=========+==========+============+=========+
|          |           |             |         |
|          |           |    TCP/IP   |         |
|          |           +--------+    |         |
|          |  NetRom            |    | Rose    |
|          +--------------------+----+---------+
|            AX.25                             |
+----------------------------------------------+
```

Le diagramme précédent illustre simplement le fait que Rose, NetRom, AX.25 et TCP/IP reposent tous sur AX.25 mais que chacun est traité comme un protocole différent au niveau de l'interface de programmation. Les noms '`AF_`' correspondent aux noms donnés aux '*Familles d'Adresses*' de chacun du point de vue du programmeur. On notera ici l'obligation de configurer la pile AX.25 avant toute configuration des protocoles NetRom, Rose ou TCP/IP.

Diagramme des modules logiciels de la pile réseau de Linux

```
---------------+-----------+-----------------------++----------+---------------
  Utilisateur  |Programmes |   call        node    ||  Démons  | ax25d  mheardd
               |           |   pms         mheard  ||          | inetd  netromd
---------------+-----------+-----------------------++----------+---------------
               |Sockets    | open(), close(), listen(), read(), write(), connect()
               |           +----------------------+-------------------+----------
               |           |    AF_AX.25   |  AF_NETROM |   AF_ROSE   |  AF_INET
               +-----------+--------------+-------+-----+-------------+----------
Noyau          |Protocoles |    AX.25     |   NetRom    |     Rose    | IP/TCP/UDP
               +-----------+--------------+-------------+-------------+----------
               |Périph.    |    ax0,ax1   |  nr0,nr1    | rose0,rose1 | eth0,ppp0
               +-----------+--------------+-------------+-------------+----------
               |Pilotes    |  Kiss   PI2   PacketTwin   SCC   BPQ     | slip ppp
               |           |   modems type son   Baycom               | ethernet
---------------+-----------+------------------------------------------+-----
Matériel | Cartes PI2, PacketTwin, SCC, Série, Ethernet
---------------------------------------------------------------------------
```

Ce diagramme est plus général que le précédent. Il montre les relations entre les applications, le noyau et le matériel ainsi qu'entre l'interface de programmation des sockets, les modules de protocoles, les périphériques réseau et leurs pilotes. Chaque niveau dépend de celui sur lequel il repose et, de façon générale, la configuration doit se faire de bas en haut. Par exemple, si vous souhaitez exécuter le programme *call*, vous devez configurer le matériel, vérifier que le pilote adéquat est inclus dans le noyau, créer les périphériques noyaux correspondants et inclure le protocole requis par le programme *call*. J'ai essayé d'organiser le présent document de cette façon.

3 Composants logiciels de la suite AX.25/NetRom/Rose

Le paquetage AX.25 comprend trois volets : les sources du noyau, les outils de configuration réseau et les applications utilisateur.

Les version 2.0.xx des noyaux Linux incluent les gestionnaires AX.25, NetRom, SCC Z8530, PacketTwin et ceux des cartes PI. Les noyaux 2.1.* les améliorent substantiellement. L'emploi d'un noyau 2.1.* dans un système de production est vivement déconseillé. Pour y remédier, Jonathon Naylor propose un ensemble de patches pour mettre à niveau la gestion du protocole radio amateur dans un noyau 2.0.28. L'application des patches est très simple et apporte une palette de fonctionnalités autrement absentes du noyau tel le support Rose. L'emploi d'un noyau 2.2.x est également envisageable.

3.1 Récupération du noyau, des outils et utilitaires

3.1.1 Sources du noyau

Les sources du noyau sont disponibles via le réseau de miroirs de ftp.kernel.org : **ftp.xx.kernel.org** où xx désigne un code pays tel fr, uk, de, us, etc... Les différentes version du noyau se trouvent en :

```
/pub/linux/kernel/
```

Version courante de mise à jour d'AX.25 : **ftp.pspt.fi**

```
/pub/linux/ham/ax25/ax25-module-14e.tar.gz
```

3.1.2 Les outils réseau

Dernière version alpha des outils réseau standard pour Linux gérant AX.25 et NetRom : **ftp.inka.de**

```
/pub/comp/Linux/networking/net-tools/net-tools-1.33.tar.gz
```

Paquetage ipfwadm : **ftp.xos.nl**

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

En 2.2.x, le paquetage *ipchains* remplace ipfwadm devenu obsolète.

3.1.3 Les utilitaires AX.25

Il existe deux familles distinctes d'outils AX.25. L'une dédiée aux noyaux `2.0.*` et l'autre destinée aussi bien aux version `2.1.*` qu'aux noyaux `2.0.*` patchés. Le numéro de version de ax25-utils indique la version du noyau la plus ancienne à partir de laquelle les outils fonctionneront. A vous de choisir une version des ax25-utils appropriée. Les combinaisons suivantes fonctionnent, **utilisez les** .

```
Noyau Linux              Utilitaires AX.25
----------------------   ------------------------
linux-2.0.29             ax25-utils-2.0.12c.tar.gz **
linux-2.0.28+module12    ax25-utils-2.1.22b.tar.gz **
```

```
linux-2.0.30+module14c   ax25-utils-2.1.42a.tar.gz
linux-2.0.31+module14d   ax25-utils-2.1.42a.tar.gz
linux-2.1.22 ++          ax25-utils-2.1.22b.tar.gz
linux-2.1.42 ++          ax25-utils-2.1.42a.tar.gz
```

Note: les versions `ax25-utils-2.0.*` identifiées ci-dessus avec le symbole '**' sont à présent obsolètes. Le document couvre l'emploi des logiciels conseillés dans les tables. Bien que les paquetages diffèrent, la plus grande partie des informations reste valable pour les versions suivantes.

Utilitaires AX.25 :

ftp.pspt.fi `ftp://ftp.pspt.fi/pub/linux/ham/ax25/`

ou :

sunsite.unc.edu `ftp://sunsite.unc.edu/pub/Linux/apps/ham/`

4 Installation des logiciels AX.25/NetRom/Rose

Une mise en oeuvre correcte d'AX.25 dans votre système Linux nécessite l'installation et la configuration d'un noyau approprié ainsi que des utilitaires AX.25.

4.1 Compilation du noyau

Si vous êtes un habitué de la compilation du noyau Linux, contentez-vous de vérifier que vous avez activé les options adéquates et sautez cette section. Si ce n'est pas le cas, lisez ce qui suit.

En principe, les sources du noyau sont décompactées au niveau du répertoire `/usr/src` dans un sous-répertoire nommé `linux`. Pour ce faire, prenez l'identité du super-utilisateur `root` et exécutez les commandes ci-dessous :

```
# mv linux linux.old
# cd /usr/src
# tar xvfz linux-2.0.31.tar.gz
# tar xvfz /pub/net/ax25/ax25-module-14e.tar.gz
# patch -p0 </usr/src/ax25-module-14/ax25-2.0.31-2.1.47-2.diff
# cd linux
```

Une fois les sources du noyau décompactées et la mise à jour appliquée, lancez le script de configuration et activez les options qui correspondent à la configuration matérielle dont vous souhaitez disposer. Vous utiliserez la commande :

```
# make menuconfig
```

Si vous êtes bête^H^H^H^Hcourageux, vous pouvez essayer

```
# make config
```

Les claviophobes se serviront de :

```
# make xconfig
```

Je vais décrire la méthode plein-écran (menuconfig) dont j'apprécie la facilité de déplacement mais vous êtes libre d'en utiliser une autre.

Dans tous les cas, vous devrez choisir parmi une série d'options auxquelles il faudra répondre par 'Y' ou 'N' (voire 'M' si vous avez recours aux modules, ce sur quoi je fais l'impasse pour simplifier).

Options importantes pour la configuration d'AX.25 :

```
Code maturity level options  --->
    ...
    [*] Prompt for development and/or incomplete code/drivers
    ...
General setup  --->
    ...
    [*] Networking support
    ...
Networking options  --->
    ...
    [*] TCP/IP networking
    [?] IP: forwarding/gatewaying
    ...
    [?] IP: tunneling
    ...
    [?] IP: Allow large windows (not recommended if <16Mb of memory)
    ...
    [*] Amateur Radio AX.25 Level 2
    [?] Amateur Radio NET/ROM
    [?] Amateur Radio X.25 PLP (Rose)
    ...
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 4800 baud HAPN-1 modulation
    [?] Soundmodem support for 9600 baud FSK G3RUH modulation
    [?] Serial port KISS driver for AX.25
    [?] BPQ Ethernet driver for AX.25
    [?] Gracilis PackeTwin support for AX.25
    [?] Ottawa PI and PI/2 support for AX.25
    [?] Z8530 SCC KISS emulation driver for AX.25
    ...
```

Vous **devez** répondre 'Y' aux options marquées d'un *'. Le reste dépend de votre configuration matérielle et d'options laissées à votre choix. Certaines de ces options sont décrites un peu plus loin. Si vous ne voyez pas ce dont il retourne, continuez la lecture et revenez à cette section ultérieurement.

Une fois la configuration du noyau achevée, vous devriez pouvoir compiler proprement un nouveau noyau :

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

Déplacez ensuite le fichier `arch/i386/boot/zImage` et éditez le fichier `/etc/lilo.conf` en conséquence avant de relancer *lilo* pour être sûr que vous démarrerez bien sur le bon noyau.

4.1.1 Un mot sur les modules

Je vous recommande de ne **pas** compiler quelque pilote que ce soit en tant que module. Dans presque toutes les installations, vous n'y gagnez rien sinon une complexité accrue. De nombreuses personnes ont des problèmes avec les modules, non par la faute du code, mais parce que les modules sont plus compliqués à installer et à configurer. [NdT:manifestement nous ne faisons pas le même arbitrage complexité/souplesse]

Si vous avez choisi de compiler certains composants en tant que modules, vous devrez également utiliser :

```
# make modules
# make modules_install
```

afin d'installer vos modules à l'emplacement adéquat.

Certains ajouts au fichier `/etc/conf.modules` sont nécessaires afin que *kerneld* sache gérer l'interface d'accès aux fonctions modularisées. Les entrées suivantes doivent être présentes :

```
alias net-pf-3     ax25
alias net-pf-6     netrom
alias net-pf-11    rose
alias tty-ldisc-1  slip
alias tty-ldisc-3  ppp
alias tty-ldisc-5  mkiss
alias bc0          baycom
alias nr0          netrom
alias pi0a         pi2
alias pt0a         pt
alias scc0         optoscc    (or one of the other scc drivers)
alias sm0          soundmodem
alias tunl0        newtunnel
alias char-major-4 serial
alias char-major-5 serial
alias char-major-6 lp
```

```
# modprobe -c
```

vous renverra la configuration courante.

4.1.2 Qu'y a-t-il de nouveau dans les noyaux 2.0.x patchés et les 2.1.y ?

Les noyaux `2.1.*` présentent des améliorations au niveau de quasiment tous les pilotes et protocoles. Citons les plus significatives :

Modularisation

tous les protocoles et gestionnaires ont été modularisés de façon à être gérés via *insmod* et *rmmod*. La mémoire demandée par le noyau diminue dans le cas de modules employés par intermittence. Le développement et la mise au point des gestionnaires devient également plus facile. Cela étant, la configuration devient légèrement plus compliquée.

Uniformisation des pilotes

l'accès aux périphériques tels les Baycom, SCC, PI, PacketTwin et autres a maintenant lieu via une interface réseau usuelle semblable à celle du gestionnaire ethernet. Ils n'apparaissent désormais plus comme des TNC KISS. L'utilitaire *net2kiss* permet de créer une interface KISS pour ces périphériques si on le souhaite.

bugs

il y a eu de nombreuses corrections et des fonctionnalités ont été ajoutées tel le protocole Rose.

4.2 Les outils de configuration du réseau

A présent que le noyau est compilé, vous devez faire de même avec les nouveaux outils de configuration du réseau. Ces outils permettent de modifier la configuration des périphériques réseau et des tables de routage.

Le nouveau paquetage alpha des `net-tools` standard gère AX.25 et NetRom. Je l'ai essayé et il semble fonctionner correctement chez moi.

4.2.1 Patch correctif incluant la gestion Rose

Le paquetage standard net-tools-1.33.tar.gz comporte certains bugs qui affectent AX.25 et NetRom. J'ai produit un correctif qui supporte aussi Rose.

Le patch est disponible à l'adresse suivante :

zone.pspt.fi `ftp://zone.pspt.fi/pub/linux/ham/ax25/net-tools-1.33.rose.tjd.diff.gz` .

4.2.2 Compilation des net-tools standard

Lisez le fichier `Release` et suivez les indications qui y sont données. Je suis passé par les étapes ci-dessous :

```
# cd /usr/src
# tar xvfz net-tools-1.33.tar.gz
# zcat net-tools-1.33.rose.tjd.diff.gz | patch -p0
# cd net-tools-1.33
# make config
```

Arrivés à ce point, vous devrez répondre à une série de questions de configuration d'une façon similaire à ce qui se fait pour le noyau. N'oubliez pas d'inclure tous les protocoles et gestionnaires de périphériques dont vous souhaitez vous servir ultérieurement. Dans le doute, répondez par l'affirmative ("Y").

Une fois la compilation effectuée :

```
# make install
```

installera les programmes à leur place définitive.

Pour disposer des fonctionnalités de type pare-feu IP (firewall), vous aurez besoin des derniers outils d'administration `ipfwadm`. Ils remplacent `ipfw` qui ne fonctionne à présent plus.

Pour la compilation d'`ipfwadm` :

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

4.3 Utilitaires et applications AX.25

Une fois les étapes de compilation et de redémarrage du noyau menées à leur terme avec succès, il vous reste à compiler les applications AX.25. Les commandes devraient ressembler à ce qui suit :

```
# cd /usr/src
# tax xvfz ax25-utils-2.1.42a.tar.gz
# cd ax25-utils-2.1.42a
# make config
# make
# make install
```

Les fichiers sont installés par défaut dans les sous-répertoires `bin`, `sbin`, `etc` et `man` du répertoire `/usr`.

S'il s'agit de la première installation des utilitaires AX.25 sur votre système, vous devrez installer quelques fichiers de configuration type dans le répertoire `/etc/ax25/` via :

```
# make installconf
```

En cas de messages du type :

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

vérifiez encore une fois que les *ncurses* sont correctement installées. Le script de configuration tente de localiser les ncurses à certains emplacements usuels mais sur des installations faisant n'importe quoi avec les ncurses, le script échoue à cette étape.

5 Numéros d'identification, adresses et préliminaires divers

Chaque port AX.25 et NetRom sur votre système doit se voir allouer un numéro d'identification (callsign/ssid). Il se configure dans les fichiers dont il va être à présent question.

Certaines mises en oeuvre d'AX.25 telles NOS et BPQ permettent l'emploi d'un ssid commun sur un même port AX.25 et NetRom. Pour des raisons techniques assez compliquées, Linux l'interdit. En pratique, ça ne s'avère pas un problème aussi important qu'on pourrait le croire.

Cela signifie que vous devez garder présents à l'esprit certains éléments lorsque vous configurez votre système.

1. Chaque port AX.25 et NetRom doit disposer d'un ssid unique.

2. TCP/IP utilise le ssid du port AX.25 par lequel il émet ou reçoit (celui dont il est question juste au-dessus).
3. NetRom emploie le ssid spécifié dans son fichier de configuration mais seulement lorsqu'il dialogue avec un autre NetRom. Il ne s'agit **pas** du ssid que les clients AX.25 de votre noeud NetRom vont employer. Davantage de détails sur ce point tout à l'heure.
4. Rose utilise par défaut le ssid du port AX.25 à moins qu'on ne lui en spécifie explicitement un autre grâce à la commande '*rsparms*' qui forcera le même ssid sur **tous** les ports.
5. Les autres programmes tels '*ax25d*' écoutent via un ssid quelconque qui n'est soumis à aucune contrainte d'unicité entre ports différents.
6. Si le routage est fait avec attention, vous pouvez affecter la même adresse IP à tous les ports.

5.1 Que sont les T1, T2, N2 ?

Toutes les piles AX.25 ne sont pas de type TNC2. La nomenclature Linux diffère sur certains points de celle du monde des TNC. Le tableau ci-dessous vous aidera à établir les correspondances entre les différents concepts.

```
-------+----------+-----------------------------------------------
Linux  | TAPR TNC | Description
-------+----------+-----------------------------------------------
T1     | FRACK    | Temps d'attente avant retransmission d'une
       |          | trame privée d'accusé de réception.
-------+----------+-----------------------------------------------
T2     | RESPTIME | Temps minimum d'attente entre trames avant
       |          | émission d'un acquittement.
-------+----------+-----------------------------------------------
T3     | CHECK    | Périodicité d'émission d'un paquet de
       |          | vérification de l'état de la connexion.
-------+----------+-----------------------------------------------
N2     | RETRY    | Nombre de tentatives de retransmission avant
       |          | de signaler un échec.
-------+----------+-----------------------------------------------
Idle   |          | Durée d'inactivité d'une connexion avant sa
       |          | fermeture.
-------+----------+-----------------------------------------------
Window | MAXFRAME | Nombre maximal de trames transmises sans
       |          | acquittement.
-------+----------+-----------------------------------------------
```

5.2 Paramètres configurables dynamiquement

Les noyaux `2.1.*` et `2.0.* +moduleXX` permettent la modification à la volée de paramètres auparavant statiques. Un examen attentif de la structure du répertoire `/proc/sys/net/` révèle de nombreux fichiers dont les noms correspondent à ceux de paramètres réseau. Les fichiers dans le répertoire `/proc/sys/net/ax25/` représentent chacun un port AX.25 configuré. Le nom du fichier reflète celui du port. La structure des fichiers dans `/proc/sys/net/ax25/<portname>/` est la suivante :

```
Fichier             Signification       Valeur                 Défaut
ip_default_mode     Mode IP par défaut  0=DG 1=VC              0
```

```
ax25_default_mode     Mode AX.25 par défaut 0=normal 1=étendu       0
backoff_type          Backoff               0=Linéaire 1=exponentiel 1
connect_mode          Mode connecté         0=non 1=oui             1
standard_window_size  Fenètre standard      1  <= N <= 7            2
extended_window_size  Fenètre étendue       1  <= N <= 63           32
t1_timeout            Délai maximal T1      1s <= N <= 30s          10s
t2_timeout            Délai maximal T2      1s <= N <= 20s          3s
t3_timeout            Délai maximal T3      0s <= N <= 3600s        300s
idle_timeout          Attente d'inactivité  0m <= N                 20m
maximum_retry_count   N2                    1  <= N <= 31           10
maximum_packet_length Trame AX.25           1  <= N <= 512          256
```

T1, T2, T3 sont donnés en secondes tandis que la durée d'inactivité est en minutes. Notez que les valeurs employées dans l'interface sysctl s'expriment dans une unité interne multiple par 10 du temps en secondes. La résolution atteint donc le dixième de seconde. Dans le cas d'une alarme qui peut être nulle, c'est à dire pour T3 et pour la durée d'inactivité, une valeur nulle équivaut à une désactivation.

La structure des fichiers dans `/proc/sys/net/netrom/` est la suivante :

```
Fichier                           Valeur par défaut
default_path_quality              10
link_fails_count                  2
network_ttl_initialiser           16
obsolescence_count_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
```

La structure des fichiers dans `/proc/sys/net/rose/` est la suivante :

```
Fichier                           Valeur par défaut
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
```

Le positionnement d'un paramètre se fait simplement en l'écrivant dans le fichier. Par exemple, pour vérifier puis modifier la taille de fenêtre Rose, vous pourriez exécuter :

```
# cat /proc/sys/net/rose/window_size
3
# echo 4 >/proc/sys/net/rose/window_size
# cat /proc/sys/net/rose/window_size
4
```

6 Configuration d'un port AX.25

Chaque application AX.25 nécessite un fichier de configuration spécifique pour obtenir les paramètres des ports AX.25 définis sur votre système. Pour les ports AX.25, il s'agit du fichier `/etc/ax25/axport`. Chaque port dont vous souhaitez vous servir doit être répertorié dans ce fichier.

6.1 Création des périphériques AX.25

Le périphérique réseau correspond à ce qui apparaît lorsque vous entrez la commande '*ifconfig*'. Il s'agit de l'abstraction logicielle par le biais de laquelle le noyau Linux émet et reçoit des données réseau. Presque tous les périphériques réseau sont associés à une entité matérielle mais il y a certaines exceptions. Le périphérique réseau se rattache directement à un gestionnaire de périphérique.

Le code AX.25 de Linux inclut un grand nombre de gestionnaires de périphériques. Le pilote KISS est sûrement le plus courant mais on peut également citer les pilotes SCC, Baycom et modem-son.

Chacun de ces pilotes crée un périphérique lors de son invocation.

6.1.1 Création des périphériques KISS

Options de configuration du noyau :

```
General setup  --->
    [*] Networking support
Network device support  --->
    [*] Network device support
    ...
    [*] Radio network interfaces
    [*] Serial port KISS driver for AX.25
```

Le TNC KISS sur un port série constitue sûrement la configuration la plus courante. À vous de préconfigurer et de connecter le TNC à un port série. Un programme de communication tel *minicom* ou *seyon* vous permettra de configurer le TNC en kiss.

Servez-vous du programme *kissattach* pour créer les périphériques KISS. Par exemple :

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

Les périphériques KISS se retrouvent sous la dénomination '`ax[0-9]`'. Au premier appel de *kissattach*, '`ax0`' est créé ; au second, '`ax1`', etc ... Chaque périphérique KISS est associé à un port série.

kissparms permet de positionner divers paramètres sur un périphérique KISS.

De façon précise, l'exemple précédent créerait un périphérique KISS reposant sur le périphérique série '`/dev/ttyS0`' et le port '`radio`' du fichier `/etc/ax25/axports`. Il positionne ensuite *txdelay* et *slottime* à 100 ms et *ppersist* à 25.

Reportez vous aux pages de *man* pour davantage d'informations.

Configuration des TNC Dual Port L'utilitaire *mkiss* inclus dans le paquetage ax25-utils permet l'emploi des modems d'un TNC à doubles ports. La configuration est simple. Elle consiste à prendre le contrôle du périphérique série connecté au TNC multiports et à le faire ressembler à une collection de périphériques chacun connecté à un TNC monoport. Vous devrez le faire *avant* toute autre configuration

AX.25. Les périphériques que vous configurerez correspondent à des pseudo-TTY (`/dev/ttyq*`) et non aux ports série. Les pseudo-TTY mettent en place un équivalent de tuyau via lequel des programmes prévus pour dialoguer avec des périphériques de type tty peuvent communiquer. Chaque tuyau possède une extrémité maître ('`/dev/ptyq*`') et une esclave ('`/dev/ttyq*`'). Les extrémités sont en relation telles que si `/dev/ptyq0` est l'extrémité maître d'un tuyau, alors `/dev/ttyq0` est son extrémité esclave. Le côté maître doit être ouvert avant le côté esclave. *mkiss* divise un périphérique série grâce à ce mécanisme.

Par exemple, pour un TNC double-port connecté au port série `/dev/ttyS0` en 9600 bps, les commandes suivantes créeront deux pseudo-tty qui se comporteront comme des ports séries munis de TNC usuels :

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

`/dev/ttyq0` et `/dev/ttyq1` se manipulent ensuite avec *kissattach* comme décrit précédemment dans l'exemple relatif à `port1` et `port2`. N'utilisez pas directement *kissattach* sur le port série car *mkiss* y accède.

mkiss accepte de nombreux arguments optionnels. En voici un résumé :

-c

provoque l'ajout d'un octet de contrôle à chaque trame KISS. La plupart des mises en oeuvre de KISS ne le gèrent pas. La rom KISS G8BPG en est capable.

-s <speed>

fixe le débit du port série.

-h

active la négociation matérielle sur le port série (inactive par défaut). La plupart des mises en oeuvre KISS ne la gèrent pas.

-l

déclenche l'émission de messages à destination de *syslog*.

6.1.2 Création d'un périphérique Baycom

Options de compilation du noyau :

```
Code maturity level options  --->
    [*] Prompt for development and/or incomplete code/drivers
General setup  --->
    [*] Networking support
Network device support  --->
    [*] Network device support
    ...
    [*] Radio network interfaces
    [*] BAYCOM ser12 and par96 driver for AX.25
```

Malgré l'opinion suivant laquelle les modems Baycom ne fonctionneraient pas très bien sous Linux, Thomas Sailer(<`sailer@ife.ee.ethz.ch`>) en a développé le gestionnaire. Son pilote gère les ports série `Ser12` et `Par96` ainsi que les modems parallèles `PicPar`. Vous trouverez davantage d'informations concernant les modems à l'adresse :

Baycom Web site `http://www.baycom.de/` .

La première étape consiste à déterminer les ports d'entrée/sortie et les adresses des ports série ou parallèle auxquels se connecte(nt) le(s) modem(s).

Les périphériques BayCom se retrouvent sous la dénomination `bc0`, `bc1`, `bc2` etc...

L'utilitaire *sethdlc* permet de configurer le pilote avec les paramètres précédents. Si votre système n'est muni que d'un seul modem, vous pouvez également les passer en argument lors du chargement du module avec *insmod*.

Un exemple. Désactivation du gestionnaire du port série COM1: puis configuration du pilote BayCom pour un modem série Ser12 sur ce même port avec activation de l'option logicielle DCD :

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

Un modem parallèle de type Par96 sur le port LPT1: utilisant la détection DCD matérielle :

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

Ce n'est pas la meilleure façon de faire. L'utilitaire *sethdlc* fonctionne également avec plusieurs périphériques.

La page de *man* d'*sethdlc* est très détaillée mais quelques exemples mettront en lumière les aspects les plus importants de la configuration. On suppose que le module BayCom a déjà été chargé avec :

```
# insmod hdlcdrv
# insmod baycom
```

Vous pouvez également avoir incorporé le gestionnaire en dur dans le noyau.

Configuration de `bc0` pour un modem parallèle BayCom sur LPT1 avec détection DCD logicielle :

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

Configuration de `bc1` pour un modem série sur COM1 :

```
# sethdlc -p -i bc1 mode "ser12*" io 0x3f8 irq 4
```

6.1.3 Configuration des paramètres d'accès au canal AX.25

Ces paramètres équivalent à ppersist, txdelay et slottime pour KISS. Ici aussi, vous utiliserez *sethdlc*.

La page de man relative à *sethdlc* reste la source d'informations la plus complète mais un ou deux autres exemples ne feront pas de mal.

Configuration de `bc0` avec TxDelay égal à 200 ms, SlotTime à 100 ms, PPersist à 40, en half duplex :

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

Notez que les paramètres de durée sont donnés en millisecondes.

Configuration d'AX.25 avec le pilote BayCom Le pilote BayCom crée des périphériques réseau standard dont la configuration pour AX.25 est voisine de celle liée à l'emploi des cartes PI ou PacketTwin.

Tout d'abord il faut donner un numéro d'identification AX.25 au périphérique. *ifconfig* le fait très bien :

```
# /sbin/ifconfig bc0 hw ax25 VK2KTJ-15 up
```

La commande précédente affecte l'identité AX.25 `VK2KTJ-15` au périphérique `bc0`. Vous disposez également de *axparms* mais vous aurez de toute façon besoin d'*ifconfig* pour activer le périphérique :

```
# ifconfig bc0 up
# axparms -setcall bc0 vk2ktj-15
```

L'étape suivante consiste à ajouter une entrée dans le fichier `/etc/ax25/axports` comme vous le feriez pour tout autre périphérique. Les données du fichier `axports` étant associées aux périphériques réseau par l'intermédiaire du numéro d'identification, la ligne que vous rajouterez devra comprendre celui de votre BayCom.

La nouvelle interface AX.25 se comporte à présent comme les autres. Vous pouvez la configurer pour IP, la gérer via ax25d et l'utiliser pour NetRom ou Rose si bon vous semble.

6.1.4 Création d'un périphérique modem-son

Options de compilation du noyau :

```
Code maturity level options  --->
    [*] Prompt for development and/or incomplete code/drivers
General setup  --->
    [*] Networking support
Network device 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 4800 baud HAPN-1 modulation
    [?] Soundmodem support for 9600 baud FSK G3RUH modulation
```

Thomas Sailer a développé un nouveau pilote noyau qui traite une carte son comme un modem : connectez votre dispositif radio directement sur votre carte son pour émettre des paquets ! Thomas conseille au moins un 486DX2 à 66 MHz pour exploiter le logiciel ; tout le traitement numérique est effectué par le microprocesseur.

Actuellement, le pilote émule les modems AFSK à 1200 bps, HAPN à 4880 et FSK à 9600 (compatible avec G3RUH). Seules les cartes son compatibles SoundBlaster et WindowsSoundSystem sont supportées. Un soupçon d'électronique est nécessaire pour aider la carte son à alimenter le dispositif radio. Des informations sur ce sujet se trouvent sur la page suivante :

Thomas's SoundModem PTT circuit web page `http://www.ife.ee.ethz.ch/~sailer/pcf/ptt_circ/ptt.html` . Les possibilités sont nombreuses : récupération à la sortie de la carte son, traitement sur les ports parallèle, série ou midi. Des exemples de schémas illustrent tout ces cas sur le site de Thomas.

Les périphériques modem-son se retrouvent sous la dénomination `sm0`, `sm1`, `sm2`, etc...

Remarque: le pilote SoundModem et le sous-système de gestion du son entrent en compétition sous Linux. Assurez-vous que le son est désactivé avant d'utiliser le pilote SoundModem. Vous pouvez bien sûr compiler les deux en tant que modules, les insérer et les ôter en fonction de vos besoins.

Configuration de la carte son Le pilote SoundModem n'initialise pas la carte réseau. Le paquetage ax25-utils comprend l'utilitaire '*setcrystal*' pour le faire sur les cartes son à base de composants Crystal. Si vous avez un autre modèle de carte, servez-vous d'un autre logiciel pour l'initialiser. L'emploi de setcrystal est fort simple :

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

Par exemple, pour une carte SoundBlaster à l'adresse 0x388 employant l'interruption 10 et la canal DMA 1, vous entreriez :

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

Pour une carte WindowSoundSystem à l'adresse 0x534 employant l'interruption 5 et la canal DMA 3 :

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

Le paramètre `[-f synthio]` correspond à l'adresse du synthétiseur. Le paramètre `[-c dma2]` détermine le second canal DMA pour un fonctionnement simultané dans les deux sens (full-duplex).

Configuration des périphériques modem-son Une fois la carte son configurée, vous devez spécifier au pilote où la trouver et quelle type de modem il lui faut émuler.

L'utilitaire *sethdlc* vous permet de passer ces paramètres. Si vous n'avez qu'une seule carte installée, vous pouvez les passer en arguments à l'insertion du module SoundModem.

Par exemple, avec une seule carte de type SoundBlaster configurée comme ci-dessus, émulant un modem 1200 bps :

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

Ce n'est pas la meilleure façon de faire. L'utilitaire *sethdlc* fonctionne également avec plusieurs périphériques.

La page de *man* d'*sethdlc* est très détaillée mais quelques exemples mettront ici encore en lumière les aspects les plus importants de la configuration. On suppose que le module modem-son a déjà été chargé avec :

```
# insmod hdlcdrv
# insmod soundmodem
```

Vous pouvez également avoir incorporé le gestionnaire en dur dans le noyau.

Configuration du pilote pour émuler un modem G3RUH 9600 sur le périphérique `sm0` avec la carte WindowsSoundSystem précédente et le port parallèle en 0x378 pour alimenter l'émetteur :

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

Configuration du pilote pour émuler un modem HAPN 4800 sur le périphérique `sm1` avec la carte SoundBlaster précédente et le port série en 0x2f8 pour alimenter l'émetteur :

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

Configuration du pilote pour émuler un modem AFS 1200 sur le périphérique `sm1` avec la carte SoundBlaster précédente et le port série en 0x2f8 pour alimenter l'émetteur :

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

Configuration des paramètres d'accès au canal AX.25 Ces paramètres équivalent à ppersist, txdelay et slottime pour KISS. Ici aussi, vous utiliserez *sethdlc*.

La page de man relative à *sethdlc* reste la source d'informations la plus complète mais un ou deux autres exemples ne feront toujours pas de mal.

Configuration de `sm0` avec TxDelay égal à 100 ms, SlotTime à 50 ms, PPersist à 128 en full duplex :

```
# sethdlc -i sm0 -a txd 100 slot 50 ppersist 128 full
```

Notez que les paramètres de durée sont donnés en millisecondes.

Choix du volume et ajustement du pilote Il est *très* important que les niveaux audio soient correctement ajustés pour qu'un modem-radio fonctionne correctement. Les modem-son n'échappent pas à la règle. Thomas a mis au point des utilitaires pour faciliter cette tâche : *smdiag* et *smmixer*.

smdiag

fournit deux type d'affichage : soit un écran de type oscilloscope, soit un visuel normal.

smmixer

permet l'ajustement des niveaux audio de transmission et de réception.

smdiag en mode 'visuel' avec un périphérique SoundModem en `sm0` :

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

smmixer avec un périphérique SoundModem en `sm0` :

```
# smmixer -i sm0
```

Configuration d'AX.25 avec le pilote SoundModem Le pilote soundmodem crée des périphériques réseau standard dont la configuration pour AX.25 est voisine de celle liée à l'emploi des cartes PI ou PacketTwin.

Tout d'abord il faut donner un numéro d'identification AX.25 au périphérique. *ifconfig* le fait très bien :

```
# /sbin/ifconfig sm0 hw ax25 VK2KTJ-15 up
```

La commande précédente affecte l'identité AX.25 `VK2KTJ-15` au périphérique `sm0`. Vous disposez également de *axparms* mais vous aurez de toute façon besoin d'*ifconfig* pour activer le périphérique :

```
# ifconfig sm0 up
# axparms -setcall sm0 vk2ktj-15
```

L'étape suivante consiste à ajouter une entrée dans le fichier `/etc/ax25/axports` comme vous le feriez pour tout autre périphérique. Les données du fichier `axports` étant associées aux périphériques réseau par l'intermédiaire du numéro d'identification, la ligne que vous rajouterez devra comprendre celui de votre modem-son.

La nouvelle interface AX.25 se comporte à présent comme les autres. Vous pouvez la configurer pour IP, la gérer via ax25d et l'utiliser pour NetRom ou Rose si bon vous semble.

6.1.5 Création d'un périphérique à base de carte PI

Options de compilation du noyau :

```
General setup  --->
    [*] Networking support
Network device support  --->
    [*] Network device support
    ...
    [*] Radio network interfaces
    [*] Ottawa PI and PI/2 support for AX.25
```

Les périphériques PI se retrouvent sous la dénomination '`pi[0-9][ab]`' où la première carte détectée se verra allouer '`pi0`', la seconde '`pi1`', etc... '`a`' et '`b`' se rapportent à la première et à la seconde interface physique des cartes PI. Si vous avez inclus le pilote de cartes PI dans votre noyau et que la détection s'est effectuée correctement, vous pouvez configurer le périphérique :

```
# /sbin/ifconfig pi0a hw ax25 VK2KTJ-15 up
```

La commande précédente affecte l'identité AX.25 `VK2KTJ-15` au premier port de la carte PI et l'active. Pour utiliser le périphérique, il vous reste à ajouter au fichier `/etc/ax25/axports` l'entrée correspondant à son identité AX.25.

Le gestionnaire de cartes PI a été écrit par : `David Perry, <dp@hydra.carleton.edu>`

6.1.6 Création d'un périphérique PacketTwin

Options de compilation du noyau :

```
General setup  --->
    [*] Networking support
Network device support  --->
    [*] Network device support
    ...
    [*] Radio network interfaces
    [*] Gracilis PackeTwin support for AX.25
```

Les périphériques PacketTwin se retrouvent sous la dénomination '`pt[0-9][ab]`' où la première carte détectée se verra allouer '`pt0`', la seconde '`pt1`', etc. '`a`' et '`b`' se rapportent à la première et à la seconde interfaces physiques des cartes PacketTwin. Si vous avez inclus le pilote de cartes PI dans votre noyau et que la détection s'est effectuée correctement, vous pouvez configurer le périphérique :

```
# /sbin/ifconfig pt0a hw ax25 VK2KTJ-15 up
```

La commande précédente affecte l'identité AX.25 `VK2KTJ-15` au premier port de la carte PacketTwin et l'active. Pour utiliser le périphérique, il vous reste à ajouter au fichier `/etc/ax25/axports` l'entrée correspondant à son identité AX.25.

Le gestionnaire de cartes PacketTwin a été écrit par : `Craig Small VK2XLZ, <csmall@triode.apana.org.au>`.

6.1.7 Création d'un périphérique SCC générique

Options de compilation du noyau :

```
General setup  --->
    [*] Networking support
Network device support  --->
    [*] Network device support
    ...
    [*] Radio network interfaces
    [*] Z8530 SCC KISS emulation driver for AX.25
```

Joerg Reuter, DL1BKE, `jreuter@poboxes.com` a écrit le module générique de gestion des cartes à base de SCC Z8530. Son pilote supporte une large gamme de cartes différentes et offre une interface similaire à un TNC KISS que vous pouvez traiter comme telle.

Récupération et compilation des outils de configuration Bien que le pilote soit inclus dans les arborescences standard du noyau, Joerg accompagne le paquetage de configuration dont vous aurez besoin des versions les plus récentes.

Vous trouverez le paquetage des outils de configuration à une des adresses suivantes :

Joerg's web page `http://www.rat.de/jr/`

db0bm.automation.fh-aachen.de

```
/incoming/dl1bke/
```

insl1.etec.uni-karlsruhe.de

```
/pub/hamradio/linux/z8530/
```

ftp.ucsd.edu

```
/hamradio/packet/tcpip/linux
/hamradio/packet/tcpip/incoming/
```

Différentes versions s'offrent à vous. Choisissez la plus adaptée à votre noyau :

```
z8530drv-2.4a.dl1bke.tar.gz   2.0.*
z8530drv-utils-3.0.tar.gz    2.1.6 et au delà
```

Voici les commandes que j'ai employées lors de la compilation et de l'installation du paquetage pour mon noyau 2.0.30 :

```
# cd /usr/src
# gzip -dc z8530drv-2.4a.dl1bke.tar.gz | tar xvpofz -
# cd z8530drv
# make clean
# make dep
# make module         # Si vous souhaitez modulariser le pilote
# make for_kernel     # Si vous préférez un pilote inclus dans le noyau
# make install
```

Au terme de ces opérations, trois nouveaux exécutables devraient s'être installés dans votre répertoire `/sbin` : *gencfg*, *sccinit* et *sccstat*. Ces programmes vont vous servir à configurer le pilote pour votre carte.

De nouveaux périphériques apparaîtront également dans votre répertoire `/dev` sous les noms `scc0-scc7`. Ils joueront plus tard le rôle de périphériques KISS que vous pourrez employer.

Si vous lancez 'make for_kernel', vous devrez également recompiler votre noyau. Afin que le pilote z8530 soit inclus, vérifiez que vous avez bien répondu '`Y`' à : '`Z8530 SCC kiss emulation driver for AX.25`' durant le '`make config`'.

Si vous avez choisi 'make module', le module `scc.o` sera installé dans le sous-répertoire adéquat de `/lib/modules` et il ne vous sera pas nécessaire de recompiler tout le noyau. N'oubliez pas d'exécuter un *insmod* afin de charger le module avant d'essayer de le configurer.

Configurer le pilote pour sa carte La conception du pilote SCC z8530 vise une flexibilité maximale ainsi que la gestion du plus grand nombre de cartes possible. Le prix à payer se retrouve au niveau de la configuration.

Le paquetage comprend une documentation plus détaillée et vous aurez tout intérêt à vous y reporter si vous rencontrez le moindre problème. Intéressez-vous plus particulièrement à `doc/scc_eng.doc` et à `doc/scc_ger.doc`. J'ai repris les points les plus importants mais de nombreux détails sont passés sous silence.

Le fichier de configuration principal, lu par le programme *sccinit*, se trouve en `/etc/z8530drv.conf`. Il se divise en deux parties : configuration des paramètres matériels et configuration du canal. Une fois ce fichier au point, vous n'aurez plus qu'à ajouter :

```
# sccinit
```

au fichier `rc` chargé de la configuration du réseau et le périphérique sera initialisé conformément au contenu du fichier de configuration. Effectuez ces opérations avant d'utiliser le gestionnaire.

Configuration des paramètres matériels La première partie se divise en strophes, chacune correspondant à un composant 8530. Une strophe comprend une liste de mots clefs et d'arguments. Le fichier peut décrire jusqu'à quatre composants SCC par défaut. Si vous avez besoin d'aller au-delà, modifiez la ligne `#define MAXSCC 4` dans le fichier `scc.c`.

Liste des mots-clefs et des arguments :

chip

le terme `chip` sert à séparer les strophes. Il ne nécessite pas d'arguments et ceux-ci sont de toute façon ignorés.

data_a

adresse du port de données pour le canal 'A' du z8530. Un nombre hexadécimal est attendu en argument (par exemple 0x300).

ctrl_a

adresse du port de contrôle pour le canal 'A' du z8530. Un nombre hexadécimal est attendu en argument (par exemple 0x304).

data_b

adresse du port de données pour le canal 'B' du z8530. Un nombre hexadécimal est attendu en argument (par exemple 0x301).

ctrl_b

adresse du port de contrôle pour le canal 'B' du z8530. Un nombre hexadécimal est attendu en argument (par exemple 0x305).

irq

interruption (IRQ) utilisée par le SCC 8530. Un entier, 5 par exemple, est attendu.

pclock

fréquence du signal d'horloge sur la broche PCLK du 8530. L'argument est donné en Hz par un nombre entier (4915200 par défaut).

board

modèle de la munie du 8530 : <<====== ne manque-t-il pas un mot ?

PA0HZP

carte SCC PA0HZP

EAGLE

carte Eagle

PC100

carte SCC PC100 DRSI

PRIMUS

carte PRIMUS-PC (DG9BL)

BAYCOM

carte (U)SCC BayCom

escc

optionnel, active la gestion des cartes SCC étendues (ESCC) telles la 8580, la 85180 ou la 85280. L'argument est une chaîne de caractères qui peut prendre les valeurs 'yes' ou 'no' ('no' par défaut).

vector

optionnel, donne l'adresse du vecteur d'acquittement pour les cartes PA0HZP. Il est commun à l'ensemble des composants et prend par défaut la valeur nulle.

special

optionnel, donne l'adresse du registre spécial sur diverses cartes. Nul par défaut.

option

optionnel. Nul par défaut.

Quelques exemples de configuration des cartes les plus courantes :

BayCom USCC

```
chip    1
data_a  0x300
ctrl_a  0x304
data_b  0x301
ctrl_b  0x305
irq     5
board   BAYCOM
#
```

```
# SCC chip 2
#
chip    2
data_a  0x302
ctrl_a  0x306
data_b  0x303
ctrl_b  0x307
board   BAYCOM
```

PA0HZP SCC card

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

DRSI SCC card

```
chip 1
data_a 0x303
data_b 0x301
ctrl_a 0x302
ctrl_b 0x300
irq 7
pclock 4915200
board DRSI
escc no
```

Si vous disposez déjà d'une configuration qui fonctionne avec votre carte sous NOS, la commande *gencfg* permet de convertir les commandes du pilote NOS PE1CHL en quelque chose d'utilisable pour le pilote z8530.

gencfg s'invoque simplement avec les mêmes paramètres que ceux employés pour le pilote PE1CHL avec NET/NOS. Par exemple, pour obtenir une ébauche de fichier de configuration pour une carte OptopSCC :

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

Configuration du canal Vous préciserez tous les autres paramètres relatifs au port que vous configurez dans la section spécifique au canal. Cette section se divise également en strophes. Une strophe correspond à un port logique et il y aura donc deux strophes de canal pour une strophe de paramètres matériels puisque chaque SCC 8530 inclut deux ports.

Les mots-clefs et leurs arguments s'inscrivent également dans le fichier `/etc/z8530drv.conf`, **à la suite** de la section des paramètres matériels.

L'ordre est très important dans cette section mais tout devrait marcher même si vous vous écartez de celui proposé.

device

en première position, spécifie le nom du périphérique auquel le reste de la configuration s'applique (par exemple `/dev/scc0`)

speed

débit de l'interface en bits par seconde. Un nombre entier est attendu (par exemple `1200`)

clock

origine de l'horloge de synchronisation des données. Les valeurs possibles sont :

dpll

fonctionnement normal monodirectionnel (half-duplex) ;

external

le modem dispose de sa propre horloge Rx/Tx ;

divider

utilisation du diviseur bidirectionnel (si disponible).

mode

type de codage des données. À choisir entre `nrzi` et `nrz`

rxbuffers

nombre de tampons de réception à allouer en mémoire. Un nombre entier est attendu (8 par exemple)

txbuffers

nombre de tampons d'émission à allouer en mémoire. Un nombre entier est attendu (8 par exemple)

bufsize

taille des tampons d'émission et de réception. La valeur est donnée en octets et correspond à la longueur totale d'une trame. Elle doit donc prendre en compte aussi bien les données que l'en-tête. Cet argument est optionnel et prend par défaut la valeur `384`

txdelay

délai d'attente de la transmission KISS. Un nombre entier de ms est attendu

persist

paramètre persist (KISS). Argument de type entier

slot

slot time (KISS). Argument de type entier en ms

tail

the KISS transmit tail value. Argument entier en ms

fulldup

indicateur de fonctionnement bidirectionnel (KISS), à choisir entre `1` pour le bidirectionnel et `0` pour le monodirectionnel

wait

paramètre d'attente (KISS). Argument de type entier en ms

min

paramètre min (KISS). Argument de type entier en secondes

maxkey

temps de keyup (?) maximal (KISS). Argument de type entier en secondes

idle

délai d'attente sur inactivité (KISS). Argument de type entier en secondes

maxdef

paramètre maxdef (KISS). Argument de type entier

group

paramètre group (KISS). Argument de type entier

txoff

valeur de txoff (KISS). Argument de type entier en ms

softdcd

valeur de softdcd (KISS). Argument de type entier

slip

indicateur slip (KISS). Argument de type entier

Utilisation du pilote Il suffit d'employer les périphériques `/dev/scc*` comme on le ferait avec n'importe quel tty série connecté à un TNC KISS. Par exemple, avec une carte SCC, vous exécuteriez quelque chose du style :

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

NOS permet également d'attacher le périphérique de la même façon. Avec JNOS, vous entreriez une commande du style :

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

Les outils *sccstat* et *sccparam* Afin de diagnostiquer les problèmes, *sccstat* affiche la configuration courante de n'importe quel périphérique SCC. Essayez :

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

Vous devriez récupérer une quantité impressionnante d'informations touchant à la configuration et à l'état du port SCC `/dev/scc0`.

sccparam sert à modifier la configuration après l'initialisation du noyau. La syntaxe est similaire à celle de la commande `param` de NOS. Pour positionner `txtail` à 100 ms sur un port :

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

6.1.8 Création d'un périphérique BPQ

Options de configuration du noyau :

```
General setup  --->
    [*] Networking support
Network device support  --->
    [*] Network device support
    ...
    [*] Radio network interfaces
    [*] BPQ Ethernet driver for AX.25
```

Linux gère le BPQ compatible Ethernet. Vous pouvez ainsi dialoguer en AX.25 via un réseau Ethernet local et interconnecter votre poste Linux avec d'autres machines BPQ sur réseau local.

Les périphériques BPQ se retrouvent sous la dénomination '`bpq[0-9]`'. '`bpq0`' est associé à '`eth0`', '`bpq1`' à '`eth1`' etc.

La configuration est simple. Mettez d'abord en place un périphérique Ethernet standard. Pour cela, vous aurez pris soin d'inclure dans le noyau la gestion de votre adaptateur Ethernet. Pour plus de détails, reportez vous à :

Ethernet-HOWTO `Ethernet-HOWTO.html` .

Avant d'activer la gestion BPQ, le périphérique Ethernet doit s'être vu affecter un numéro d'identification AX.25. Par exemple :

```
# /sbin/ifconfig bpq0 hw ax25 vk2ktj-14 up
```

Vérifiez bien que l'identifiant correspond à celui qui figure dans le fichier `/etc/ax25/axports` pour ce port.

6.1.9 Configuration d'un noeud BPQ pour le dialogue avec la couche AX.25 de Linux

Souvent, l'Ethernet BPQ repose sur des adresses de type multicast. Ce n'est pas le cas dans la mise en oeuvre sous Linux qui recourt aux adresses générales (broadcast) usuelles sur Ethernet. Le fichier NET.CFG du gestionnaire ODI BPQ doit donc être modifié pour ressembler à ce qui suit :

```
LINK SUPPORT

        MAX STACKS 1
        MAX BOARDS 1
```

```
LINK DRIVER E2000                   ; ou tout autre MLID adapté à votre carte

        INT 10                      ;
        PORT 300                    ; selon votre carte

        FRAME ETHERNET_II

        PROTOCOL BPQ 8FF ETHERNET_II ; requis pour BPQ - peut jouer sur PID

BPQPARAMS                           ; optionnel - requis seulement pour
                                    ; modifier la cible par défaut

        ETH_ADDR  FF:FF:FF:FF:FF:FF  ; adresse de la cible
```

6.2 Mise au point du fichier /etc/ax25/axports

`/etc/ax25/axports` est un fichier texte standard que vous créerez avec n'importe quel éditeur. Son format est le suivant :

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

avec :

portname

nom affecté au port

callsign

identifiant AX.25

baudrate

vitesse de communication avec le TNC

paclen

longueur de paquet maximale applicable au port pour les communications AX.25 en mode connecté

window

paramètre de fenêtre (K) AX.25. Il s'agit de la même chose que le paramètre `MAXFRAME` de nombreux TNC.

description

champ de commentaire

Chez moi, le fichier ressemble à ça :

```
radio    VK2KTJ-15      4800        256     2       4800bps 144.800 MHz
ether    VK2KTJ-14      10000000    256     2       BPQ/ethernet device
```

Rappelez-vous que vous devez affecter un numéro d'identification (ssid) unique à chaque port AX.25 que vous créez. Ajoutez une ligne pour chaque périphérique que vous emploierez ; cela concerne les ports KISS, BayCom, SCC, PI, PT et modem-son. Les entrées dans le fichier sont associées aux périphériques réseau par le biais de l'identificateur AX.25 : au moins une bonne raison de les prendre différents.

6.3 Routage AX.25

Vous pouvez décider de mettre en place des routes par défaut spécifiques à certains hôtes, par exemple pour des connexions AX.25 courantes ou des connexions IP. L'utilitaire *axparms* effectue cette tâche. Sa page de *man* en donne une description exhaustive. À titre d'exemple :

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

Cette commande établit une entrée pour `VK2XLZ` via `VK2SUT` sur le port AX.25 nommé `radio`.

7 TCP/IP et l'interface AX.25

Si vous disposez d'interfaces KISS, deux méthodes s'offrent à vous pour configurer une adresse IP : soit la commande *kissattach*, soit le recours conventionnel à *ifconfig*.

Modifiant l'exemple KISS précédent de façon à créer une interface AX.25 avec une adresse IP égale à `44.136.8.5` et un `MTU` de `512` octets :

```
# /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
```

Au besoin, vous emploierez *ifconfig* pour configurer les autres paramètres.

Si vous disposez d'autre interfaces, utilisez *ifconfig* pour configurer l'adresse IP et le masque de réseau du port et ajoutez une route vers le port comme vous le feriez avec n'importe quelle autre interface IP. L'exemple suivant s'appuie sur une carte PI mais fonctionnerait de façon similaire avec un périphérique AX.25 quelconque :

```
# /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 44.136.8.0 netmask 255.255.255.0 pi0a
# /sbin/route add default pi0a
```

Les commandes précédentes correspondent à une configuration familière aux utilisateurs de NOS et de ses variantes ou de toute autre logiciel IP. Notez que la route par défaut n'est pas nécessaire si un autre périphérique réseau la met lui-même en place.

Pour tester votre configuration, lancez un ping ou un telnet vers votre machine :

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

L'argument '`-i 5`' force *ping* à envoyer ses requêtes ICMP toutes les 5 secondes et non chaque seconde.

8 Configuration d'un port NetRom

Le protocole NetRom s'appuye sur les ports AX.25 que vous créerez. Sa configuration s'effectue par l'intermédiaire de deux fichiers. L'un décrit les interfaces NetRom et l'autre les ports AX.25 sous-jacents. La procédure détaillée ci-dessous s'appliquera à toutes les interfaces NetRom que vous souhaiterez définir.

8.1 Le fichier `/etc/ax25/nrports`

Ce fichier est l'analogue pour les ports NetRom du fichier `/etc/ax25/axports` pour les ports AX.25. Tous les périphériques NetRom que vous souhaitez employer doivent figurer dans le fichier `/etc/ax25/nrports`. Le plus souvent, une station Linux ne comprendra qu'un seul port NetRom qui utilisera certains des périphériques AX.25. Pour certains services tels un BBS, le besoin de définir plusieurs alias NetRom peut se manifester ; on ajoute alors des périphériques NetRom en conséquence.

Le format du fichier est le suivant :

```
name callsign  alias  paclen   description
```

Avec :

name
: nom affecté au port.

callsign
: identifiant pour le trafic NetRom transitant par ce port. Attention, il ne s'agit **pas** de l'adresse à laquelle les clients doivent se connecter pour disposer d'une interface de type *noeud* (ce mode sera décrit un peu plus loin). L'identifiant doit être unique et ne réapparaître nulle part dans les fichiers `/etc/ax25/axports` et `/etc/ax25/nrports`.

alias
: alias NetRom du port.

paclen
: taille maximale des trames NetRom transmises par le port.

description
: commentaire.

Par exemple, pour créer un port NetRom connu du reste du réseau NetRom sous l'identité '`LINUX:VK2KTJ-9`' :

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

Des programmes tels *call* se servent du fichier `nrports`.

8.2 Le fichier `/etc/ax25/nrbroadcast`

Ce second fichier peut contenir une nombre d'entrées variable, normalement une pour chaque port AX.25 convoyant du trafic NetRom.

Le format du fichier est le suivant :

```
axport min_obs def_qual worst_qual verbose
```

Avec :

axport
: nom du port tiré du fichier `/etc/ax25/axports`. En l'absence d'entrée dans le fichier `/etc/ax25/nrbroadcasts` pour un port AX.25, aucun routage NetRom n'aura lieu via ce port et toute diffusion NetRom sera ignorée.

min_obs

paramètre d'obsolescence minimale du port.

def_qual

qualité par défaut.

worst_qual

qualité minimale admissible. Toute route de qualité moindre sera ignorée.

verbose

activation de la diffusion des informations de routage globales ou seulement relatives au noeud.

Par exemple :

```
radio    1       200       100          1
```

8.3 Création des périphériques réseau NetRom

Une fois les deux fichiers mis au point, il faut créer les périphériques NetRom. La démarche est proche du cas AX.25 à ceci près que l'on se sert à présent de la commande *nrattach*. Elle constitue un pendant à la commande *axattach* et crée des périphériques NetRom qui se retrouvent sous la dénomination '`nr[0-9]`' (la première invocation produit '`nr0`', la seconde '`nr1`' etc.) Pour associer un périphérique NetRom au port défini précédemment, on utilise :

```
# nrattach netrom
```

Cette commande active le périphérique NetRom (`nr0`) nommé `netrom` configuré conformément au contenu du fichier `/etc/ax25/nrports`.

8.4 Lancement du démon NetRom

Le noyau Linux gère le protocole NetRom et assure la commutation mais il ne prend pas en charge certaines fonctions. Le démon NetRom maintient les tables de routage NetRom et diffuse les messages de routage NetRom. Il se lance via :

```
# /usr/sbin/netromd -i
```

Le fichier `/proc/net/nr_neigh` devrait progressivement se remplir d'informations concernant vos voisins NetRom.

N'oubliez pas d'inclure la commande `/usr/sbin/netromd` dans vos scripts de démarrage ou d'en créer un dédié à l'automatisation du processus.

8.5 Routage NetRom

Peut-être voudrez-vous mettre en place des routes statiques pour certains hôtes particuliers. La commande *nrparms* dispose d'une telle fonction. Reportez-vous à la page de *man* pour une description complète. A titre d'exemple, pour indiquer sur mon port AX.25 '`radio`' une route NetRom vers le `#MINTO:VK2XLZ-10` en passant par mon voisin `VK2SUT-9` :

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

nrparms permet également de créer manuellement de nouveau voisins. La commande suivante crée un voisin NetRom `VK2SUT-9` d'une qualité de `120` qui ne sera pas supprimé automatiquement.

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

9 TCP/IP sur une interface NetRom

La configuration ressemble à celle d'AX.25 pour TCP/IP.

Soit vous précisez l'adresse IP et le MTU avec *nrattach*, soit vous utilisez les commandes *ifconfig* et *route*. Il vous faudra ajouter à la main les caractéristiques *arp* des hôtes concernés par votre routage puisque votre machine ne dispose d'aucun mécanisme pour déterminer une adresse NetRom utilisable afin d'atteindre une interface IP particulière.

Pour créer une interface `nr0` d'adresse IP `44.136.8.5`, de MTU `512` et configuré conformément aux spécifications du fichier `/etc/ax25/nrports` relatives au port NetRom appelé `netrom` :

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

Autre méthode :

```
# /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
```

En ce qui concerne le volet arp et le routage, pour joindre l'interface IP `44.136.80.4` à l'adresse NetRom `BBS:VK3BBS` via un voisin NetRom d'identifiant `VK2SUT-0`, on exécuterait :

```
# 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
```

Les arguments '`120`' et '`6`' passés à la *nrparms* fixent les paramètres de qualité et d'obsolescence NetRom pour la route.

10 Configuration des ports Rose

Le protocole de transmission de paquets Rose est semblable à la couche trois des spécifications X.25. La gestion Rose du noyau est une version **modifiée** de

FPAC Rose implementation `http://fpac.lmi.ecp.fr/f1oat/f1oat.html` .

La couche Rose s'appuie sur les ports AX.25 que vous définissez. La procédure détaillée ci-dessous s'appliquera à toutes les interfaces NetRom que vous souhaiterez définir.

10.1 Le fichier `/etc/ax25/rsports`

Ce fichier est l'analogue pour les ports Rose du fichier `/etc/ax25/axports` pour les ports AX.25.

Le format du fichier est le suivant :

```
name  addresss  description
```

Avec :

name

nom affecté au port.

address

adresse Rose sur 10 digits.

description

commentaire.

Par exemple :

```
rose  5050294760  Rose Port
```

Notez que Rose emploie par défaut l'identifiant/ssid du port AX.25.

La commande *rsparms* permet de modifier l'identifiant Rose. Par exemple, pour que Linux se serve de l'identifiant `VK2KTJ-10` pour le trafic Rose sur tous les ports AX.25 .

```
# /usr/sbin/rsprams -call VK2KTJ-10
```

10.2 Création des périphériques réseau Rose

Une fois le fichier `/etc/ax25/rsports` mis au point, vous pouvez créer les périphériques Rose en reprenant la démarche AX.25. Vous emploierez la commande *rsattach* qui crée des périphériques sous l'appellation '`rose[0-5]`' (la première invocation produit '`rose0`', la seconde '`rose1`' etc...). Par exemple :

```
# rsattach rose
```

Cette commande active le périphérique Rose (`rose0`) nommé '`rose`' configuré conformément au contenu du fichier `/etc/ax25/rsports`.

10.3 Routage Rose

Le protocole Rose ne gère pour l'instant que le routage statique. Il se définit par le biais de la commande *rsparms*.

Par exemple, pour indiquer une route vers le noeud Rose `5050295502` via un port AX.25 nommé '`radio`' dans le fichier `/etc/ax25/axports` en passant par le voisin d'identificateur `VK2XLZ` :

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

Un masque vous permettra éventuellement de regrouper différentes destinations Rose sur une seule route. Par exemple :

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

On retrouve l'exemple précédent à ceci près que toute adresse de destination dont les quatre premiers digits correspondent (toute adresse commençant par `5050` donc) sera routée. La variante suivante s'avère sûrement la moins ambiguë :

```
# rsparms -nodes add 5050/4 radio vk2xlz
```

11 Communications AX.25/NetRom/Rose

Maintenant que vos interfaces AX.25, NetRom et Rose sont activées, vous devriez être capable de procéder à des essais.

Le paquetage des utilitaires AX.25 comprend le programme '*call*' qui sert d'intermédiaire pour AX.25, NetRom et Rose.

Un appel AX.25 :

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

Un appel NetRom vers un noeud d'alias `SUNBBS` :

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

Un appel Rose pour `HEARD` au noeud `5050882960` :

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

Remarque : vous devez préciser à *call* le port à employer, vu que le même noeud de destination peut être joignable via n'importe lequel des ports que vous aurez configurés.

call fournit un terminal de contrôle en mode ligne de commande pour les appels AX.25. Les lignes commençant par '`~`' sont identifiées comme des commandes. La commande '`~.`' coupe la communication.

Reportez-vous à la page de man sous `/usr/man` pour davantage d'informations.

12 Configurer Linux pour accepter les connexions

Linux est un système d'exploitation puissant qui présente beaucoup de flexibilité dans sa configuration. Le coût de cette flexibilité se retrouve dans la mise au point de la configuration souhaitée. Avant d'être en mesure d'accepter les connexions AX.25, NetRom ou Rose, vous devez vous poser un certain nombre de questions. La plus importante : "Que vais-je laisser de visible aux utilisateurs une fois connectés ?" Des gens ont mis au point de sympathiques petites applications qui fournissent des services aux appelants tels *pms* ou, plus évolué, *node* (tous deux sont compris dans le paquetage des utilitaires AX.25). Vous pouvez également souhaiter offrir une invite d'identification afin que les utilisateurs disposent d'un shell ou même écrire vos propres programmes tels une base de données maison ou un jeu. Quoi que vous fassiez, il faut spécifier à AX.25 le programme à exécuter quand une connexion s'établit.

Le démon *ax25d* joue un rôle similaire à celui rempli par *inetd* pour les connexion TCP/IP entre machines UNIX. Il se met à l'écoute des connexions entrantes et lorsqu'il en détecte une, il examine par l'intermédiaire d'un fichier de configuration le programme à lancer auquel il transmet la connexion. Puisqu'il s'agit d'un outil standard de gestion des appels AX.25, NetRom et Rose, je vais à présent décrire les étapes de sa configuration.

12.1 Le fichier `/etc/ax25/ax25d.conf`

Ce fichier contient la configuration du démon *ax25d* en charge des connexions AX.25, NetRom et Rose.

Bien que le fichier paraisse un peu cryptique au premier abord, il s'avère rapidement des plus simples à l'usage, avec quelques pièges à éviter.

Le format général du fichier est le suivant :

```
# Je suis un commentaire qu'ax25d ignorera
[nom de port] || <nom de port> || {nom de port}
<interlocuteur1> window T1 T2 T3 idle N2 <mode> <uid> <cmd> <commande> <args>
<interlocuteur2> window T1 T2 T3 idle N2 <mode> <uid> <cmd> <commande> <args>
parametres window T1 T2 T3 idle N2 <mode>
<interlocuteur3> window T1 T2 T3 idle N2 <mode> <uid> <cmd> <commande> <args>
   ...
default    window T1 T2 T3 idle N2 <mode> <uid> <cmd> <commande> <args>
```

Avec :

#

en début de ligne pour indiquer un commentaire ignoré du programme *ax25d*

<port_name>

nom du port AX.25, NetRom ou Rose tel que spécifié dans un des fichiers `/etc/ax25/axports`, `/etc/ax25/nrports` ou `/etc/ax25/rsports`. Le nom du port est entouré par '`[]`' s'il s'agit d'un port AX.25, '<>' si c'est un port NetRom ou '{}' pour un port Rose. Ce champ admet une variante qui précède le nom du port par '`callsign/ssid via`' pour indiquer que vous voulez accepter les appels vers l'identificateur cité par l'intermédiaire de cette interface. Un exemple l'illustrera.

<peer>

est l'identifiant du noeud auquel la configuration s'applique. Si vous ne spécifiez pas de ssid, tous seront considérés comme valables.

window

paramètre de fenêtre AX.25 (K) ou valeur de MAXFRAMDE pour cette configuration.

T1

délai de retransmission de trame (T1) exprimé en demi-secondes.

T2

délai d'attente par le logiciel AX.25 d'une seconde trame avant de préparer une réponse. S'exprime en secondes.

T3

délai d'inactivité avant qu'une connexion inactive ne soit coupée. S'exprime en secondes.

idle

période d'inactivité en secondes.

N2

nombre d'essais de retransmission avant qu'une connexion ne soit coupée.

<mode>

procure un mécanisme d'établissement de certains types de permissions. Les modes sont activés ou inhibés grâce à une combinaison de caractères représentant chacun un droit. L'accentuation ne joue pas et les caractères doivent former un bloc ininterrompu.

u/U

UTMP - non-supporté

v/V

Validate call - non-supporté

q/Q

Quiet - pas d'enregistrement des connexions

n/N

check NetRom Neighbour - non-supporté

d/D

Disallow Digipeaters - les connexions doivent être directes

l/L

Lockout - connexion interdite

***/0**

marker - marqueur, pas de mode spécifique

<uid>

userid sous laquelle le programme maintenant la connexion sera exécuté.

<cmd>

nom complet de la commande à lancer, sans arguments.

<cmd-name>

texte qui apparaîtra à l'invocation de *ps* comme commande du programme (en général la même chose que <cmd> mais sans le chemin d'accès).

<arguments>

arguments de ligne de commande passés à <:cmd> lorsqu'il est lancé. Les éléments suivants permettent de passer des informations utilisées :

%d

nom du port recevant la connexion

%U

identificateur AX.25 de l'extrémité connectée, sans ssid, en majuscules

%u

identificateur AX.25 de l'extrémité connectée, sans ssid, en minuscules

%S

identificateur AX.25 de l'extrémité connectée, avec ssid, en majuscules

%s

identificateur AX.25 de l'extrémité connectée, avec ssid, en minuscules

%P

identificateur AX.25 du noeud distant initiateur de la connexion, sans ssid, en majuscules

%p

identificateur AX.25 du noeud distant initiateur de la connexion, sans ssid, en minuscules

%R

identificateur AX.25 du noeud distant initiateur de la connexion, avec ssid, en majuscules

%r

identificateur AX.25 du noeud distant initiateur de la connexion, avec ssid, en minuscules

Ue section au format précédent est requise pour chaque interface AX.25, NetRom ou Rose que vous voulez voir accepter des connexions.

Le paragraphe comprend deux lignes particulières, l'une commençant par la chaîne '`parameters`' et l'autre par la chaîne '`default`' (il y a une différence).

'`default`' couvre tous les cas qui ne sont pas spécifiés ailleurs. Ainsi, tous les appels sur l'interface <interface_call> ne disposant pas d'une règle spécifique se retrouvent dans la rubrique '`default`'. En l'absence d'une telle section, toutes les connexions hors règle sont immédiatement interrompues sans autre forme de procès.

'`parameters`' est plus subtil et dissimule le piège mentionné précédemment. Si le caractère '*' est présent dans un champ, une valeur par défaut issue de la section '`parameters`' est employée. Le noyau possède d'ailleurs une liste de valeurs utilisées en l'absence de '`parameters`'. Le danger réside en ce que les entrées spécifiées via '`parameters`' ne s'appliquent qu'aux règles qui les suivent. Une même interface peut comporter plusieurs entrées '`parameters`'. Notez que les règles '`parameters`' ne permettent pas de positionner les champs '`uid`' et '`command`'.

12.2 Un exemple de fichier `ax25d.conf`

```
# ax25d.conf pour VK2KTJ - 02/03/97
# Ce fichier de configuration utilise le port AX.25 défini plus haut.

# <peer> Win T1  T2  T3  idl N2 <mode> <uid> <exec> <argv[0]>[<args....>]

[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
```

Dans cet exemple, toute personne réclamant une connexion via l'identificateur '`VK2KTJ-0`' du port AX.25 '`radio`' se verra appliquer les règles suivantes :

Tout appel depuis un identifiant 'NOCALL' se verra rejeté. Notez l'emploi du mode 'L'.

La ligne `parameters` modifie deux paramètres par défaut du noyau (Window et T1) et exécutera */usr/sbin/axspawn*. Les instances de */usr/sbin/axspawn* appelées ainsi apparaîtront en tant que *axspawn* dans un affichage issu de *ps*. Les deux lignes qui suivent définissent deux stations auxquelles s'appliqueront les permissions.

La dernière ligne de la section est la règle fourre-tout appliquée au reste des connexions (VK2XLZ et VK2DAY inclus dès lors qu'ils ont recours à un ssid différent de -1). Tous les paramètres prennent leurs valeurs par défaut et le programme *pms* sera lancé avec un argument de ligne de commande spécifiant une connexion AX.25 d'identifiant `VK2KTJ` (reportez-vous à la partie 'Configuration du PMS' pour davantage de détails).

La configuration suivante accepte les appels à `VK2KTJ-1` via le port `radio`. Le programme *node* est exécuté à chaque connexion.

Vient ensuite une spécification de connexions NetRom (notez l'emploi des signes inférieur et supérieur à la place des crochets). Toute personne se connectant via le port '`netrom`' déclenche le programme *node* si son identifiant est différent de '`NOCALL`'. Dans le cas contraire, tout accès est refusé.

Les deux dernières configurations concernent des connexions entrantes Rose, la première pour ceux qui appellent le '`VK2KTJ-0`' sur notre noeud Rose et la seconde pour ceux qui emploient le '`VK2KTJ-1`'. Elles fonctionnent de la même façon. Notez l'emploi des accolades qui indiquent des ports Rose.

L'exemple manque un peu de naturel mais je crois qu'il illustre clairement les propriétés importantes de la syntaxe du fichier de configuration. La page de *man* explique dans son intégralité le contenu du fichier `ax25d.conf`. Le paquetage `ax25-utils` inclut un exemple plus détaillé qui pourrait également vous être utile.

12.3 Lancer *ax25d*

Une fois les deux fichiers de configuration mis au point, lancez la commande :

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

À présent, les gens devraient pouvoir se connecter en AX.25 à votre machine. N'oubliez pas de modifier les fichiers de commande de démarrage du système de façon que `ax25d` soit invoqué automatiquement à chaque réinitialisation de la station.

13 Le logiciel *node*

Le logiciel *node* a été développé par Tomi Manninen <`tomi.manninen@hut.fi`>. Il a été conçu à partir du programme PMS et offre une fonctionnalité de noeud facilement configurable. Une fois les utilisateurs connectés, il leur permet de se servir de telnet, de NetRom, de Rose et de AX.25 vers l'extérieur ainsi que d'obtenir diverses informations telles finger, la liste des noeuds et des écoutes etc. Le noeud peut être configuré assez simplement pour exécuter n'importe quelle commande Linux.

Normalement, le noeud sera invoqué par *ax25d*, bien qu'il puisse également être appelé par le démon IP *inetd* pour permettre aux utilisateurs d'obtenir un accès telnet à votre machine. Le lancement depuis la ligne de commande est également possible.

13.1 Le fichier `/etc/ax25/node.conf`

`node.conf` est un fichier texte qui spécifie la configuration du noeud. Son format est le suivant :

```
# /etc/ax25/node.conf
# Fichier de configuration du programme node(8)
#
# Un '#' indique une ligne de commentaire qui sera ignorée.

# Nom d'hôte de la machine noeud
hostname        radio.gw.vk2ktj.ampr.org

# Réseau local
# définit ce qui doit être considéré comme 'local' du point de vue de la
# vérification des permissions grâce à nodes.perms.
localnet        44.136.8.96/29

# Ports cachés
# rend certains ports invisibles aux utilisateurs. Les ports n'apparaîtront pas
# via la commande Ports.
hiddenports     rose netrom

# Identification du noeud
# apparaîtra à l'invite du noeud
NodeId          LINUX:VK2KTJ-9

# Port NetRom
# nom du port NetRom qui employé pour les connexions NetRom sortant du noeud
NrPort          netrom

# Délai d'inactivité du noeud
# en secondes
idletimout      1800

# Délai d'inactivité des connexions
# en secondes
conntimeout     1800

# Reprise de connexion
# indique si les utilisateurs doivent être reconnectés spontanément en cas
# de rupture de la connexion distante ou bien s'ils doivent être complètement
# déconnectés.
reconnect       on

# Alias
alias           CONV    "telnet vk1xwt.ampr.org 3600"
alias           BBS     "connect radio vk2xsb"

# Alias (commandes externes)
# exécution de commandes externes au noeud
# extcmd <cmd> <flag> <userid> <commande>
# Flag == 1 pour l'instant
# <commande> a le même format que dans ax25d.conf
extcmd          PMS     1       root    /usr/sbin/pms pms -u %U -o VK2KTJ

# Enregistrement
# le niveau 3 est le plus détaillé, 0 désactive l'enregistrement
loglevel        3
```

```
# Caractère de contrôle
# 20 = (Control-T)
EscapeChar      20
```

13.2 Le fichier `/etc/ax25/node.perms`

node affecte des permissions aux utilisateurs. On décide ainsi des utilisateurs qui ont le droit ou non d'employer des commandes telles (T)elnet ou (C)onnect. `node.perms` contient cinq champs. Le caractère '*' dans l'un d'eux indique une absence de contraintes pour son application. On construit ainsi facilement des règles applicables par défaut.

user

Le premier champ indique l'identifiant d'appel concerné par les permissions. Une éventuelle partie ssid sera ignorée.

method

Chaque protocole et chaque méthode d'accès disposent également de permissions. Par exemple, les utilisateurs connectés via AX.25 ou NetRom peuvent être autorisés à se servir de (C)onnect tandis que ceux issus d'une session telnet depuis un noeud non-local s'en verront refuser l'accès. Le deuxième champ spécifie donc à quelle méthode d'accès les permissions s'appliquent. Voici les classes de méthodes d'accès :

```
method  description
------  ----------------------------------------------------------
ampr    session telnet depuis une adresse amprnet (44.0.0.0)
ax25    connexion AX.25
host    node invoqué depuis la ligne de commande
inet    session telnet depuis une adresse non locale, de type non amprnet
local   session telent depuis un hôte 'local'
netrom  connexion NetRom
rose    connexion Rose
*       n'importe quelle connexion
```

port

Vous pouvez également contrôler les permissions pour les utilisateurs AX.25 sur la base des ports employés. Le troisième champ contient un nom de port si vous décidez d'employer cette possibilité (disponible seulement pour les connexions AX.25).

password

Un mot de passe peut également être demandé lors de l'établissement de la connexion. Cela s'avère pratique pour protéger des comptes utilisateurs spécifiques disposant de privilèges particulièrement élevés. Si le quatrième champ est positionné, il correspond au mot de passe attendu.

permissions

Les permissions sont fixées par le biais du dernier champ de chaque ligne. L'information est codée au niveau du bit, chaque service disposant d'un bit qui indique s'il est ou non activé. Ci-suit la liste des services et la position du champ avec le bit positionné :

```
valeur  description
------  ------------------------------------------------
 1      Login
 2      (C)onnect AX.25
```

```
4       (C)onnect NetRom
8       (T)elnet vers les hôtes locaux
16      (T)elnet vers amprnet (44.0.0.0)
32      (T)elnet vers les hôtes non-locaux, de type non-amprnet
64      (C)onnect AX.25 pour les ports cachés
128     (C)onnect Rose
```

On additionne ensuite les puissances de deux associées aux bits des permissions activées. Le résultat va dans le cinquième champ.

Un exemple de fichier `nodes.perms` :

```
# /etc/ax25/node.perms
#
# L'opérateur a pour identité VK2KTJ, s'identifie par le mot de passe 'secret'
# et dispose de toutes les permissions pour toutes les méthodes de connexion.
vk2ktj  *       *       secret  255

# Les utilisateurs suivants sont exclus
NOCALL  *       *       *       0
PK232   *       *       *       0
PMS     *       *       *       0

# Les utilisateur d'INET n'ont pas le droit de se connecter
*       inet    *       *       0

# Les utilisateurs AX.25, NetRom, locaux, liés à l'hôte ou AMPR disposent de
# (C)onnect et de (T)elnet vers les hôtes locaux et ampr mais se voient
# interdire les autres adresses IP.
*       ax25    *       *       159
*       netrom  *       *       159
*       local   *       *       159
*       host    *       *       159
*       ampr    *       *       159
```

13.3 Exécution de *node* depuis *ax25d*

L'invocation du programme *node* par le démon *ax25d* nécessite l'ajout de règles appropriées au fichier `/etc/ax25/ax25d.conf`. Je souhaitais une configuration telle que les utilisateurs puissent se connecter soit à *node* soit à un service de leur choix. *ax25d* l'autorise par le biais d'une création astucieuse d'alias de ports. Par exemple, partant de la configuration d'*ax25d* donnée plus haut, on veut que tous les utilisateurs se connectant à `VK2KTJ-1` reçoivent le noeud. Pour cela, on ajoute la règle suivante au fichier `/etc/ax25/ax25d.conf` :

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

Linux répondra à toute demande de connexion sur le port AX.25 '`radio`' d'identifiant '`VK2KTJ-1`' en exécutant le programme *nde*.

13.4 Exécution de *node* depuis *inetd*

Offrir la possibilité d'ouvrir une session telnet sur votre machine et d'accéder au programme *node* est une tache plutôt facile. Commencez par choisir le port auquel les utilisateurs se connecteront. Dans mon exemple, j'ai pris arbitrairement le port 3694 bien que Tomi détaille dans sa documentation la marche à suivre pour remplacer le démon telnet usuel par le programme *node*.

Il faut modifier deux fichiers.

Ajouter au fichier `/etc/services` :

```
node    3694/tcp        #OH2BNS's node software
```

et au fichier `/etc/inetd.conf` :

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

Une fois *inetd* redémarré, tout utilisateur effectuant un telnet vers le port 3694 de votre machine se verra demander un login et, selon la configuration, un mot de passe avant d'être connecté à *node*.

14 Configuration de *axspawn*.

axspawn permet aux stations AX.25 qui se connectent d'ouvrir une session sur votre machine. Il peut être lancé par le programme *ax25d* décrit ci-dessus d'une façon similaire à *node*. Pour ouvrir une session utilisateur, vous ajouterez une variante de la ligne suivante au fichier `/etc/ax25/ax25d.conf` :

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

Si la ligne s'achève sur le caractère +, l'utilisateur devra appuyer sur la touche d'entrée avant de pouvoir s'identifier. Par défaut, il n'y a pas d'attente. Toutes les configurations d'hôtes qui suivent la ligne précédente déclencheront l'appel d'*axspawn* lorsqu'ils se connecteront. Quand *axspawn* s'exécute, il vérifie tout d'abord que l'argument de ligne de commande fourni est un identifiant licite, supprime le SSID puis parcourt le fichier `/etc/passwd` pour voir si l'utilisateur dispose d'un compte. Si c'est le cas et que le mot de passe associé est `""` (vide) ou `+`, la session utilisateur est ouverte. En présence d'un autre mot de passe, celui-ci est demandé. Si le compte n'existe pas, *axspawn* peut être configuré de façon à en créer un automatiquement.

14.1 Mise au point du fichier `/etc/ax25/axspawn.conf`

Le format du fichier est le suivant :

```
# /etc/ax25/axspawn.conf
#
# creation automatique de comptes utilisateur
create    yes
#
# compte d'invite en l'absence de creation automatique et si tout le reste
# echoue. Se desactive ave "no"
guest     no
#
# id ou nom du groupe pour le compte automatique
group     ax25
#
# id de depart
first_uid 2001
#
# id maximale
```

```
max_uid   3000
#
# emplacement des repertoires utilisateurs crees automatiquement
home      /home/ax25
#
# shell utilisateur
shell     /bin/bash
#
# lien entre les id utilisateur et le numero d'identification pour les
# connexions sortantes
associate yes
```

Détail des huit caractéristiques configurables de *axspawn* :

#

indique un commentaire.

create

si ce champ est positionné à `yes` alors *axspawn* tentera de créer un compte pour tout utilisateur qui n'apparaît pas dans le fichier `/etc/passwd`.

guest

fournit le nom du compte à employer pour les utilisateurs n'en ayant pas lorsque *create* est positionné à `no`. On y trouve souvent `ax25` ou `guest`.

group

indique le groupe pour les utilisateurs qui n'apparaissent pas dans le fichier `/etc/passwd`.

first_uid

valeur de départ des identités utilisateur lors de la création automatique

max_uid

identité utilisateur maximale disponible à la création automatique

home

répertoire dans lequel seront créés les comptes utilisateurs

shell

shell de login des nouveaux utilisateurs

associate

indique si les connexions sortantes de l'utilisateur ont lieu avec son identifiant d'appel personnel ou avec celui de votre station

15 Configuration de *pms*

pms fournit un système simple de messagerie personnelle. Il a été écrit à l'origine par Alan Cox. Dave Brown, N2RJT, <`dcb@vectorbd.com`> en a repris le développement. Les fonctionnalités sont restées simples : envoi de courrier électronique au propriétaire de la station et obtention d'informations limitée. Dave travaille actuellement à les enrichir.

Il faut tenir à jour quelques fichiers contenant des informations sur le système et ajouter les entrées adéquates au fichier `ax25d.conf` de telle sorte qu'il s'exécute pour les utilisateurs connectés.

15.1 Mise au point du fichier `/etc/ax25/pms.motd`

Le fichier `/etc/ax25/pms.motd` contient l'équivalent du message du jour affiché aux utilisateurs après qu'ils se sont connectés et ont reçu l'en-tête usuel de BBS. Il s'agit d'un simple fichier texte qui sera transmis tel quel.

15.2 Mise au point du fichier `/etc/ax25/pms.info`

`/etc/ax25/pms.info` est également un simple fichier texte dans lequel vous renseignerez des informations plus détaillées relatives à votre station ou à sa configuration. Ce fichier est transmis aux utilisateurs en réponse à la commande `Info` depuis l'invite `PMS`.

15.3 Associer les identifiants AX.25 aux comptes utilisateurs

Lors de l'envoi d'un courrier à destination d'un identifiant d'appel AX.25, *pms* s'attend à trouver une association avec une identité d'utilisateur usuelle sur la station. La section suivante décrit le processus.

15.4 Ajout de PMS au fichier `/etc/ax25/ax25d.conf`

L'ajout de *pms* au fichier `ax25d.conf` est très simple. Il vous faut néanmoins garder un élément en tête : Dave a ajouté la prise en compte d'arguments de ligne commande à PMS afin de gérer différentes conventions de fin de ligne. AX.25 et NetRom requièrent une fin de ligne et un saut de ligne tandis que le standard Unix comprend juste le caractère de fin de ligne. Par exemple, pour une entrée correspondant au lancement par défaut de PMS à l'ouverture d'une connexion sur un port AX.25, vous ajouteriez :

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

Cette ligne exécute *pms* en lui précisant qu'il s'agit d'une connexion AX.25 et que PMS a pour propriétaire `vk2ktj`. Consultez la page de *man* pour l'emploi d'autres méthodes de connexion.

15.5 Tester PMS

Exécutez depuis la ligne de commande :

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

En remplaçant votre identifiant d'appel par le mien, cela lancera pms en lui imposant l'emploi de la convention unix de fin de ligne et en donnant `vk2ktj` comme identité à l'utilisateur connecté.

Vous pouvez également demander à un autre noeud de se connecter afin de confirmer le fonctionnement de votre `ax25d.conf`.

16 Configuration des programmes *user_ call*

On trouve derrière ce nom les programmes *ax25_ call* et *netrom_ call*. Il s'agit de programmes très simples destinés à être lancés par *ax25d* pour automatiser les connexions depuis des hôtes distants. On peut bien sûr les employer dans des scripts ou via d'autres démons tels *node*.

Ils équivalent au programme *call* et n'effectuent aucun traitement sur les données, ce qui vous épargne le problème des conversions de fin de lignes.

Un exemple pour commencer. On suppose que vous disposez d'un petit réseau personnel, d'une station Linux tenant lieu de passerelle radio et d'une autre machine – on prendra un noeud BPQ – qui lui est connectée par un lien ethernet.

En principe, si vous voulez que les utilisateurs radio puissent joindre le noeud BPQ, il leur faudra le faire par l'intermédiaire de votre noeud Linux ou se connecter au démon node puis établir la connexion. *ax25_call* peut simplifier le processus s'il est invoqué par *ax25d*.

Prenons le cas d'un noeud BPQ d'identifiant `VK2KTJ-9`, la station Linux étant munie d'un port AX.25/ethernet nommé '`bpq`'. '`radio`' désignera le port radio de la machine passerelle.

Un enregistrement dans le fichier `/etc/ax25/ax25d.conf` du type :

```
[VK2KTJ-1 via radio]
default    * * * *   *   *  *
                root /usr/sbin/ax25_call ax25_call bpq %u vk2ktj-9
```

permet aux les connexions directes à '`VK2KTJ-1`' qui n'est autre que le démon Linux *ax25d*, celui ci les commutant automatiquement sur un lien AX.25 à '`VK2KTJ-9`' via l'interface '`bpq`'.

Vous pouvez essayer toutes sortes d'autres configurations. Les utilitaires '*netrom_call*' et '*rose_call*' opèrent de façon similaire. Un radioamateur en a fait usage pour faciliter l'accès à un BBS distant. En principe, on aurait dû entrer à la main une chaîne de connexion démesurément longue. Il a donc ajouté une entrée faisant apparaitre le BBS comme une entité appartenant au réseau local, *ax25d* servant en fait de proxy pour l'accès à la machine distante.

17 Configuration des commandes Rose Uplink et Downlink

Si vous avez l'habitude des réalisations Rose à base de ROM, vous ne serez pas dépaysé par la méthode d'appel AX.25 à travers un réseau Rose. Soit un noeud local d'utilisateurs Rose d'identifiant `VK2KTJ-5` et un appelant AX.25 souhaitant se connecter à `VK5XXX` au noeud Rose distant `5050882960`, il lancera la commande :

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

Au niveau du noeud distant, `VK5XXX` recevra une connexion avec l'identifiant des utilisateurs locaux AX.25 digipétée par l'intermédiaire de l'identifiant des noeuds Rose distants.

La couche protocolaire Rose de Linux ne gère pas cette fonctionnalité dans le noyau mais les deux applications *rsuplnk* et *rsdwnlnk* savent s'en charger.

17.1 Configuration d'une liaison Rose descendante

Afin que votre station Linux accepte un appel Rose et établisse une connexion AX.25 vers une destination à l'écoute de laquelle il n'est pas, vous devez ajouter un enregistrement à votre fichier `/etc/ax25/ax25d.conf`. En principe, cette ligne correspondra au comportement par défaut pour les connexions Rose entrantes. Par exemple, vous êtes à l'écoute des demandes d'accès Rose aux destinations telles `NODE-0` ou `HEARD-0` que vous gérez localement, mais toutes les autres connexions sont transmises à la commande *rsdwnlink* sous l'hypothèse qu'il s'agit d'utilisateurs AX.25.

Une configuration typique :

```
#
```

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

Avec cette configuration, tout appel qui effectue une connexion Rose sur votre noeud Linux vers une destination à l'écoute de laquelle vous ne vous tenez pas se verra converti en une connexion AX.25 sur le port `4800` avec `VK2KTJ-5` pour chemin.

17.2 Configuration d'un liaison Rose montante

Pour que votre station Linux accepte les connexions AX.25 d'une façon similaire à celle du noeud Rose, vous ajouterez à votre fichier `/etc/ax25/ax25d.conf` une ligne du type :

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

Notez la syntaxe particulière pour l'identifiant local. Le caractère '`*`' indique que l'application doit être invoquée si l'identifiant est reconnu dans le chemin de répétition d'une connexion.

Avec cette configuration, un appel AX.25 peut établir des appels Rose au moyen de la séquence présentée dans l'introduction. Toute personne demandant un relai via l'identifiant `VK2KTJ-5` sur le port AX.25 `4800` sera traité par la commande *rsuplnk*.

18 Association des identifiants AX.25 aux comptes utilisateurs

Dans de nombreuses situations, il est fortement souhaitable d'associer un identifiant à compte utilisateur. Par exemple lorsque plusieurs opérateurs radioamateurs partagent la même machine et souhaitent employer leur propre identifiant lorsqu'ils effectuent des appels ou lorsque des utilisateurs de PMS désirent dialoguer avec quelqu'un en particulier sur une station.

Les utilitaires AX.25 permettent de réaliser cette association. On l'a déjà évoqué dans la section relative à PMS mais je le répète ici afin de m'assurer que vous ne passerez pas à côté.

L'association s'effectue grâce à la commande *axparms*. Par exemple :

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

Cette commande associe l'identifiant AX.25 `vk2ktj` à l'utilisateur `terry`. Tout courrier destiné à `vk2ktj` sur *pms* sera transmis au compte Linux `terry`.

Songez à mettre ces correspondances dans vos fichiers *rc* de démarrage afin qu'elles soient disponibles à chaque réinitialisation.

Notez que vous ne devez surtout pas associer un identifiant au compte `root` vu que cela pourrait poser des problèmes de configuration à d'autres programmes.

19 Entrées du système de fichier /proc/

Le pseudo système de fichiers `/proc` contient divers fichiers spécifiques aux programmes AX.25 et NetRom. Ces fichiers sont normalement employés par les utilitaires AX.25 mais leur formatage est tel qu'ils peuvent vous intéresser. Le format est suffisamment simple pour ne pas nécessiter beaucoup d'explications.

/proc/net/arp

: liste des associations entre adresses IP et adresses de niveau MAC, qu'il s'agisse d'ethernet, d'AX.25 ou d'un autre protocole MAC.

/proc/net/ax25

: sockets AX.25 ouverts. Elles peuvent être en attente de connexion ou actives.

/proc/net/ax25_bpqether

: identifiants AX.25 de type Ethernet BPQ.

/proc/net/ax25_calls

: équivalences entre identités d'utilisateurs Linux et identifiants d'appel telles que définies par la commande *axparms -assoc*.

/proc/net/ax25_route

: informations sur les chemins AX.25

/proc/net/nr

: sockets NetRom ouvertes. Elles peuvent être en attente de connexion ou actives.

/proc/net/nr_neigh

: liste de voisins NetRom

/proc/net/nr_nodes

: informations sur les voisins NetRom

/proc/net/rose

: sockets Rose ouvertes. Elles peuvent être en attente de connexion ou actives.

/proc/net/rose_nodes

: correspondances entre destinations et voisins Rose

/proc/net/rose_neigh

: liste de voisins Rose

/proc/net/rose_routes

: connexions Rose en cours

20 Programmation réseau AX.25, NetRom, Rose

L'avantage le plus important lié à l'utilisation des protocoles par paquets radioamateurs du noyau réside en la facilité de développement des programmes et applications qui les emploient.

Bien que la programmation réseau sous Unix déborde du cadre de ce document, je vais décrire les principaux éléments d'utilisation des protocoles AX.25, NetRom et Rose au sein de vos programmes.

20.1 Familles d'adresses

La programmation AX.25, NetRom et Rose est assez semblable à la programmation TCP/IP sous Linux. LEs principales différences se font au niveau des familles d'adresses et des structures d'adresse à mettre en place.

Les noms de familles d'adresses pour AX.25, NetRom et Rose sont respectivement `AF_AX.25`, `AF_NETROM` et `AF_ROSE`.

20.2 Fichiers d'en-tête

Incluez toujours les fichiers '`ax25.h`', '`netrom.h`' ou '`rose.h`' si vous vous servez de ces protocoles. Les débuts de fichiers-types ressemblent à quelque chose du style :

Pour AX.25 :

```
#include <ax25.h>
int s, addrlen = sizeof(struct full_sockaddr_ax25);
struct full_sockaddr_ax25 sockaddr;
sockaddr.fsa_ax25.sax25_family = AF_AX.25
```

Pour 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;
```

Pour 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 Mise en forme des identifiants et exemples

La librairie `lib/ax25.a` du paquetage des utilitaires AX.25 contient des routines de conversion des identifiants. Vous pouvez bien sûr écrire les vôtres si vous le souhaitez.

Les programmes *user_ call* sont d'excellents exemples à partir desquels travailler. Leur source code est inclus dans les outils AX.25. Si vous passez un peu de temps à les examiner, vous remarquerez rapidement que quatre-vingt-dix pour cent du travail consiste à préparer l'ouverture des sockets. En fait la connexion est rapide mais la mise en place prend du temps.

Les exemples sont assez simples pour ne pas prêter à confusion. Si vous avez des questions, adressez-vous directement à la liste de diffusion `linux-hams` où quelqu'un vous aidera sûrement.

21 Quelques configurations-types

Ci-suivent des exemples de configurations parmi les plus typiques. Il ne s'agit que d'un guide dans la mesure où il y a autant de façons de configurer un réseau qu'il y a de réseaux disponibles mais il peut vous servir de point de départ.

21.1 Un petit réseau Ethernet local avec un routeur Linux vers un réseau radio local

Nombre d'entre vous disposent de petits réseaux locaux chez eux et désirent connecter les stations de ce réseau à un réseau radio local. J'ai ce type de configuration chez moi. J'ai réussi à obtenir un bloc d'adresses contiguës que je gère par une route unique sur mon Ethernet local. Votre coordinateur IP local vous aidera si vous souhaitez procéder ainsi. Les adresses du réseau Ethernet local forment un sous-ensemble des adresses radio. Voici ma configuration personnelle avec le routeur Linux :

```
                                         .      .   .    .    . .
   -+-                              .
    | Reseau        /---------\     .    Reseau
    | 44.136.8.96/29|         |    .     44.136.8/24        \ | /
    |               | Routeur |   .                          \|/
    |               |         |  .                            |
    |          eth0 |    &    |  .  /-----\    /----------\   |
    +---------------+         +-----| TNC |----| Radio    |---/
    |   44.136.8.97 | serveur |  .  \-----/    \----------/
    |               |         | sl0
    |               |  Linux  | 44.136.8.5
    |               |         |    .
    |               |         |     .
    |               \_________/       .
   -+-                                   .      .   .    .    . .
```

```
#!/bin/sh
# /etc/rc.net
# Configuration d'un port AX.25 de type KISS et d'une interface Ethernet

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
```

- L'option IP_FORWARDING doit être activée dans le noyau.
- Les fichiers de configuration AX.25 correspondent essentiellement à ceux donnés dans les sections précédentes. Reportez-y vous si nécessaire.
- J'ai décidé d'attribuer au port radio une adresse qui n'appartient pas au bloc attribué à mon réseau local. Rien n'y obligeait et j'aurais dû facilement y affecter l'adresse `44.136.8.97`.
- `44.136.8.68` correspond à ma passerelle d'encapsulation IP dans IP et est donc ma route par défaut.
- Chaque station sur l'Ethernet est munie de la route suivante :

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

  Les paramètres *mss* et *window* me permettent d'obtenir les meilleures performances possibles aussi bien pour les connexions Ethernet locales que pour les accès radio.
- smail, http, ftp et d'autres démons s'exécutent également sur le routeur qui est ainsi la seule station à fournir aux autres des services.
- Le routeur est un 386DX20 d'entrée de gamme avec 20 Mo de disque et une configuration Linux minimaliste.

21.2 Passerelle d'encapsulation IP dans IP

L'emploi de Linux comme passerelle d'encapsulation IP est maintenant courant à travers le monde. Le nouveau gestionnaire de tunnel IP accepte les routes multiples encapsulées et rend obsolète l'ancien démon *ipip*.

Un schéma classique :

```
                                        .      .   .    .    . .
---                                .
 | Reseau        /----------\     .    Reseau
 | 154.27.3/24   |          |    .     44.136.16/24       \ | /
 |               |  Linux   |   .                          \|/
 |               |          |  .                            |
 |          eth0 |          |  .  /-----\    /----------\   |
 +---------------+Passerelle+-----| TNC |----| Radio    |---/
 |   154.27.3.20 |          |  .  \-----/    \----------/
 |               |  IPIP    | sl0
 |               |          | 44.136.16.1
 |               |          |    .
 |               |          |     .
 |               \__________/       .
---                                     .      .   .    .    . .
```

Voici les fichiers de configuration intéressants :

```
# /etc/rc.net
# This file is a simple configuration that provides one KISS AX.25
# radio port, one Ethernet device, and utilises the kernel tunnel driver
# to perform the IPIP encapsulation/decapsulation
#
echo "/etc/rc.net"
echo "  Configuring:"
#
echo -n "    loopback:"
/sbin/ifconfig lo 127.0.0.1
/sbin/route add 127.0.0.1
echo " done."
#
echo -n "    ethernet:"
/sbin/ifconfig eth0 154.27.3.20 netmask 255.255.255.0 \
                broadcast 154.27.3.255 up
/sbin/route add 154.27.3.20 eth0
/sbin/route add -net 154.27.3.0 netmask 255.255.255.0 eth0
echo " done."
#
echo -n "    AX.25: "
kissattach -i 44.136.16.1 -m 512 /dev/ttyS1 4800
/sbin/ifconfig sl0 netmask 255.255.255.0 broadcast 44.136.16.255
/sbin/route add -host 44.136.16.1 sl0
/sbin/route add -net 44.136.16.0 netmask 255.255.255.0 window 1024 sl0
#
echo -n "    tunnel:"
/sbin/ifconfig tunl0 44.136.16.1 mtu 512 up
#
echo done.
#
echo -n "Routing ... "
source /etc/ipip.routes
echo done.
#
# end.
```

et :

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

/etc/ax25/axports

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

Quelques points à noter :

- Le nouveau gestionnaire de tunnel utilise le champ *gw* de la table de routage à la place du paramètre *pointopoint* pour fixer l'adresse de la passerelle IPIP distante. Il supporte ainsi plusieurs routes par interface.
- Vous **pouvez** attribuer la même adresse à deux interfaces réseau. Dans l'exemple courant, `sl0` et `tunl0` ont tous deux été munis de l'adresse IP du port Radio. La passerelle distante récupère ainsi la bonne adresse de votre passerelle dans les datagrammes encapsulés qu'elle reçoit.
- Les commandes de routage relatives aux routes encapsulées peuvent être générées automatiquement via une version modifiée du script *munge* incluse ci-dessous. Les instructions de routage sont écrites dans un fichier séparé et appelées par la commande `source /etc/ipip.routes` de *bash* (en supposant que vous employez les mêmes conventions). Le fichier source doit être au format de commande route NOS.
- Remarquez l'emploi de l'argument *window* dans la commande *route*. Ce paramètre améliore les performances de la liaison radio.

Le nouveau script tunnel-munge :

```
#!/bin/sh
#
# From: Ron Atkinson <n8fow@hamgate.cc.wayne.edu>
#
#  This script is basically the 'munge' script written by Bdale N3EUA
#  for the IPIP daemon and is modified by Ron Atkinson N8FOW. It's
#  purpose is to convert a KA9Q NOS format gateways route file
#  (usually called 'encap.txt') into a Linux routing table format
#  for the IP tunnel driver.
#
#        Usage: Gateway file on stdin, Linux route format file on stdout.
#               eg.  tunnel-munge < encap.txt > ampr-routes
#
# NOTE: Before you use this script be sure to check or change the
#       following items:
#
```

```
#     1) Change the 'Local routes' and 'Misc user routes' sections
#        to routes that apply to your own area (remove mine please!)
#     2) On the fgrep line be sure to change the IP address to YOUR
#        gateway Internet address. Failure to do so will cause serious
#        routing loops.
#     3) The default interface name is 'tunl0'. Make sure this is
#        correct for your system.

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

fgrep encap | grep "^route" | grep -v " XXX.XXX.XXX.XXX" | \
awk '{
        split($3, s, "/")
        split(s[1], n,".")
        if      (n[1] == "")    n[1]="0"
        if      (n[2] == "")    n[2]="0"
        if      (n[3] == "")    n[3]="0"
        if      (n[4] == "")    n[4]="0"
        if      (s[2] == "1")   mask="128.0.0.0"
        else if (s[2] == "2")   mask="192.0.0.0"
        else if (s[2] == "3")   mask="224.0.0.0"
        else if (s[2] == "4")   mask="240.0.0.0"
        else if (s[2] == "5")   mask="248.0.0.0"
        else if (s[2] == "6")   mask="252.0.0.0"
        else if (s[2] == "7")   mask="254.0.0.0"
        else if (s[2] == "8")   mask="255.0.0.0"
        else if (s[2] == "9")   mask="255.128.0.0"
        else if (s[2] == "10")  mask="255.192.0.0"
        else if (s[2] == "11")  mask="255.224.0.0"
        else if (s[2] == "12")  mask="255.240.0.0"
        else if (s[2] == "13")  mask="255.248.0.0"
        else if (s[2] == "14")  mask="255.252.0.0"
        else if (s[2] == "15")  mask="255.254.0.0"
        else if (s[2] == "16")  mask="255.255.0.0"
        else if (s[2] == "17")  mask="255.255.128.0"
        else if (s[2] == "18")  mask="255.255.192.0"
        else if (s[2] == "19")  mask="255.255.224.0"
        else if (s[2] == "20")  mask="255.255.240.0"
        else if (s[2] == "21")  mask="255.255.248.0"
        else if (s[2] == "22")  mask="255.255.252.0"
        else if (s[2] == "23")  mask="255.255.254.0"
        else if (s[2] == "24")  mask="255.255.255.0"
        else if (s[2] == "25")  mask="255.255.255.128"
        else if (s[2] == "26")  mask="255.255.255.192"
        else if (s[2] == "27")  mask="255.255.255.224"
        else if (s[2] == "28")  mask="255.255.255.240"
```

```
        else if (s[2] == "29")  mask="255.255.255.248"
        else if (s[2] == "30")  mask="255.255.255.252"
        else if (s[2] == "31")  mask="255.255.255.254"
        else                    mask="255.255.255.255"

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

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

21.3 Configuration d'une passerelle d'encapsulation AXIP

Nombre de passerelles Radio Amateur avec l'Internet encapsulent AX.25, NetRom et Rose dans IP. Le cas des trames AX.25 relève du RFC 1226 écrit par Brian Kantor. Mike Westerhof a réalisé un démon d'encapsulation AX.25 sous Unix en 1991. Le paquetage des utilitaires ax25-utils en contient une version légèrement améliorée.

Un programme d'encapsulation AXIP reçoit des trames AX.25 d'un côté, examine la destination AX.25 afin d'en déduire l'adresse IP à laquelle les envoyer et les encapsule dans un datagramme TCP/IP avant de les émettre. Il accepte également les datagrammes TCP/IP qui contiennent des trames AX.25, extrait ces dernières et les traite comme s'il s'agissait de trames AX.25 reçues depuis un port AX.25. La distinction des trames IP contenant de l'AX.25 se fait par l'intermédiaire d'un identifiant de protocole égal à 4 (la valeur 94 est possible quoique désuète). Le RFC 1226 décrit tout ça en détail.

L'outil *ax25ipd* inclus dans le paquetage ax25-utils se présente comme un programme gérant une interface KISS, au travers de laquelle passeront des trames AX.25, et une interface d'adaptation TCP/IP. Il se configure par l'intermédiaire du fichier `/etc/ax25/ax25ipd.conf`.

21.3.1 Options de configuration d'AXIP

ax25ipd opère dans deux modes : "digipeater" et "tnc". En mode "tnc", le démon agit comme un TNC kiss. Vous lui fournissez des trames d'encapsulation KISS et il les transmet comme dans la configuration normale. En mode "digipeater", le démon agit comme un noeud de transmission AX.25. Les différences entre ces deux modes sont subtiles.

Vous configurez dans le fichier à cet effet les "routes" ou correspondances entre les identifiants AX.25 et les adresses IP des machines auxquelles vous désirez également transmettre des paquets AX.25. Chaque route dispose d'options qui seront expliquées un peu plus tard.

Voici les autres options à configurer :

le tty du démon *ax25ipd* ainsi que sa vitesse (en général l'extrémité d'un tuyau)

l'identifiant souhaité pour le mode "digipeat"

l'intervalle beacon

le choix entre une encapsulation AX.25 dans des datagrammes IP ou bien dans des datagrammes UDP/IP. L'essentiel des passerelles AXIP emploie une encapsulation IP mais certaines sont situées derrière des filtres qui ne laisseront passer que les datagrammes UDP/IP. Le choix doit coïncider avec ce qui est attendu à l'autre extrémité.

21.3.2 Un fichier de configuration /etc/ax25/ax25ipd.conf typique

```
#
# fichier de configuration ax25ipd pour la station floyd.vk5xxx.ampr.org
#
# Transport axip. 'ip' garantit la compatibilite avec la plupart des
# autres passerelles.
#
socket ip
#
# Mode d'operation de ax25ipd (digi ou tnc)
#
mode tnc
#
# Si digi est selectionne, vous devez definir un identifiant. Si vous avez
# choisi tnc, l'identifiant est optionnel mais cela pourrait changer dans le
# futur (2 identifiants pour une kiss double port).
#
#mycall vk5xxx-4
#mycall2 vk5xxx-5
#
# En mode digi, on peut definir un alias (2 etc.).
#
#myalias svwdns
#myalias2 svwdn2
#
# ident toutes les 540 secondes ...
#
#beacon after 540
#btext ax25ip -- tncmode rob/vk5xxx -- Experimental AXIP gateway
#
# Port serie (ou tuyau connecte a kissattach dans mon cas)
#
device /dev/ttyq0
#
# Vitesse du peripherique
#
speed 9600
#
# niveau de log 0 - pas de sortie
# niveau de log 1 - informations de configuration
# niveau de log 2 - evenements majeurs et erreurs
# niveau de log 3 - evenements majeurs, erreurs et suivi des trames AX.25
# niveau de log 4 - tout
# niveau de log 0 pour le moment
#
loglevel 2
#
# En mode digi, on peut avoir un veritable tnc. param permet de passer les
```

```
# parametres tnc.
#
#param 1 20
#
# Adresses de broadcast. Chaque adresse figurant dans la liste sera relayee
# vers une des routes munies de l'indicateur idoine.
#
broadcast QST-0 NODES-0
#
# Definition des routes AX.25. Autant que necessaires
# Format :
# route <id destination> <ip destination> [indicateur]
#
# Indicateurs valides :
#         b  - broadcast
#         d  - route par defaut
#
route vk2sut-0 44.136.8.68 b
route vk5xxx 44.136.188.221 b
route vk2abc 44.1.1.1
#
#
```

21.3.3 Exécuter *ax25ipd*

Commencez par mettre en palce le fichier `/etc/ax25/axports`

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

Exécutez *kissattach* pour créer le port :

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

Lancez *ax25ipd* :

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

Testez la liaison AXIP :

```
call axip vk5xxx
```

21.3.4 Remarques concernant certains indicateurs des routes

"`route`" met en place les destinations d'envoi de vos trames AX.25 encapsulées. Lorsque le démon *ax25ipd* reçoit un paquet sur son interface, il compare l'identifiant de destination avec chacun de ceux présents dans sa table de routage. S'il trouve une correspondance, le paquet est alors encapsulé dans un datagramme IP et transmis à l'hôte spécifié.

Deux indicateurs peuvent être ajoutés à n'importe quelle route du fichier `ax25ipd.conf` :

b

tout trafic à destination d'une adresse repérée par le mot-clef "`broadcast`" doit transiter par cette route.

d

tout paquet ne correspondant à aucune autre route doit suivre ce chemin.

L'indicateur de broadcast est très utile puisqu'il permet l'envoi d'informations à destination de toutes les stations vers des stations AXIP : les routes axip fonctionnent normalement en point-à-point et sont incapables de gérer des paquets de diffusion générale.

21.4 Lier NOS à Linux au moyen d'un pipe

De nombreuses personnes aiment se servir de NOS sous Linux en raison de la richesse fonctionnelle et de la facilité d'emploi auxquelles il les a habituées. La plupart d'entre eux souhaitent que leur version de NOS puisse dialoguer avec le noyau Linux de façon à offrir certaines des possibilités de Linux aux radio-utilisateurs de NOS.

Brandon S. Allbery, alias KF8NH, a fourni les informations qui suivent relatives à l'interconnexion de NOS avec le noyau Linux par l'intermédiaire de tuyaux (pipe).

Linux et NOS gérant tous deux le protocole SLIP, il est possible de les relier au moyen d'une liaison slip. Vous pourriez le faire grâce à deux ports série et à un câble null-modem mais ce serait aussi coûteux qu'inefficace. Comme d'autres systèmes de type Unix, Linux dispose de tuyaux dits 'pipes' (prononcer paillepeu). Il s'agit de pseudo-périphériques qui émulent le comportement de tty usuels du point de vue des logiciels en redirigeant le flux vers d'autres tuyaux. Pour les utiliser, un programme doit d'abord ouvrir l'extrémité **maître** d'un tuyau après quoi un second programme peut ouvrir la terminaison **esclave**. Lorsque les deux bouts sont ouverts, les programmes peuvent communiquer l'un avec l'autre en écrivant des caractères dans les tuyaux comme s'il s'agissait de terminaux usuels.

Pour employer les tuyaux entre NOS et Linux, vous devez d'abord choisir un tuyau. Le répertoire `/dev` en regorge : les extrémités maîtres se nomment `ptyq[1-f]` et celles esclaves `ttyq[1-f]`. Gardez à l'esprit qu'elles fonctionnent par paires et que si vous utilisez `/dev/ptyqf` à un bout, vous devrez employer `/dev/ttyqf` à l'autre.

Une fois le tuyau choisi, vous allouez la terminaison maître à Linux et l'esclave à NOS (le noyau démarre le premier et l'extrémité maître doit être la première ouverte). N'oubliez pas que le noyau Linux doit être muni d'une adresse IP différente de celle de NOS. A mettre en place si ce n'est pas déjà le cas.

Le tuyau se configure comme un périphérique série. Pour une liaison slip, les commandes à exécuter seront donc du type :

```
# /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
```

Dans cet exemple, le noyau Linux dispose de l'adresse `44.70.4.88` et NOS de l'adresse `44.70.248.67`. La commande *route* de la dernière ligne indique simplement au noyau Linux qu'il doit router tous les datagrammes à destination d'amprnet via le lien slip créé par la commande *slattach*. Vous pouvez par exemple copier ces commandes dans le fichier `/etc/rc.d/rc.inet2` (selon votre installation) après toutes les autres commandes de configuration réseau afin que la liaison slip apparaisse automatiquement à la réinitialisation du système. Remarque : on ne gagne rien à utiliser *cslip* au lieu de *slip*. Au contraire, les performances diminuent de par la nature purement virtuelle du lien (on passe plus de temps à compresser les en-têtes qu'à transmettre toutes les données).

Essayez les commandes suivantes pour configurer la terminaison du côté NOS :

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

Ces commandes créent un port slip nommé 'linux' sur l'extrémité esclave du tuyau et ajoutent une route qui y pointe. Une fois NOS démarré, vous devriez pouvoir exécuter des ping et des telnet de NOS vers Linux et vice-versa. Si ce n'est pas le cas, vérifiez encore une fois que vous ne vous êtes trompé nulle part, surtout au niveau des adresses et des tuyaux.

22 Où trouver de l'information sur... ?

Ce document suppose une certaine expérience de la transmission paquets par radio, et, comme ce n'est pas forcément le cas, j'ai regroupé un ensemble de références à d'autres informations utiles.

22.1 Transmission paquets par radio

Vous trouverez des informations générales sur la transmission paquets par radio sur les sites suivants :

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 Documentation sur les protocoles

AX.25, NetRom - Jonathon Naylor a regroupé de nombreux documents sur le sujet qui sont disponibles via : *ax25-doc-1.0.tar.gz* `ftp://ftp.pspt.fi/pub/ham/linux/ax25/ax25-doc-1.0.tar.gz`

22.3 Documentation sur le matériel

Informations sur la carte **PI2** :

Ottawa Packet Radio Group `http://hydra.carleton.ca/` .

Informations sur le matériel **Baycom** :

Baycom Web Page `http://www.baycom.de/` .

23 Groupes de discussion radioamateurs et Linux

Il existe plusieurs endroits où parler de Linux ou de radio amateurisme. Par exemple dans les groupes de discussion `comp.os.linux.*`, sur la liste de diffusion `HAMS` de `vger.rutgers.edu`. Mentionnons également la liste `tcp-group` sur `ucsd.edu` (origine des discussions TCP/IP radio amateur) et le canal `#linpeople` sur le réseau irc `linuxnet`.

Pour vous abonner à la liste de diffusion Linux **linux-hams**, envoyez un courrier à :

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

avec dans le corps du message la ligne suivante :

```
subscribe linux-hams
```

La ligne de sujet sera ignorée.

La liste de diffusion **linux-hams** est archivée aux adresses :

zone.pspt.fi `http://hes.iki.fi/archive/linux-hams/`

et :

zone.oh7rba.ampr.org `http://zone.oh7rba.ampr.org/archive/linux-hams/` . Les débutants sont priés de commencer par utiliser les archives. Celles-ci contiennent des réponses à l'essentiel des questions courantes.

Pour souscrire à la liste `tcp-group`, envoyez un courrier à l'adresse :

```
listserver@ucsd.edu
```

avec dans le corps du message la ligne :

```
subscribe tcp-group
```

Remarque : n'oubliez pas que `tcp-group` a pour thème les discussions autour de l'emploi des protocoles évolués parmi lesquels figure TCP/IP. *Les questions spécifiques à Linux n'y ont normalement pas leur place.*

24 Remerciements

Les personnes dont les noms suivent ont contribué à l'élaboration de ce document (l'ordre n'a pas d'importance): Jonathon Naylor, Thomas Sailer, Joerg Reuter, Ron Atkinson, Alan Cox, Craig Small, John Tanner, Brandon Allbery, Hans Alblas, Klaus Kudielka, Carl Makin.

25 Copyright.