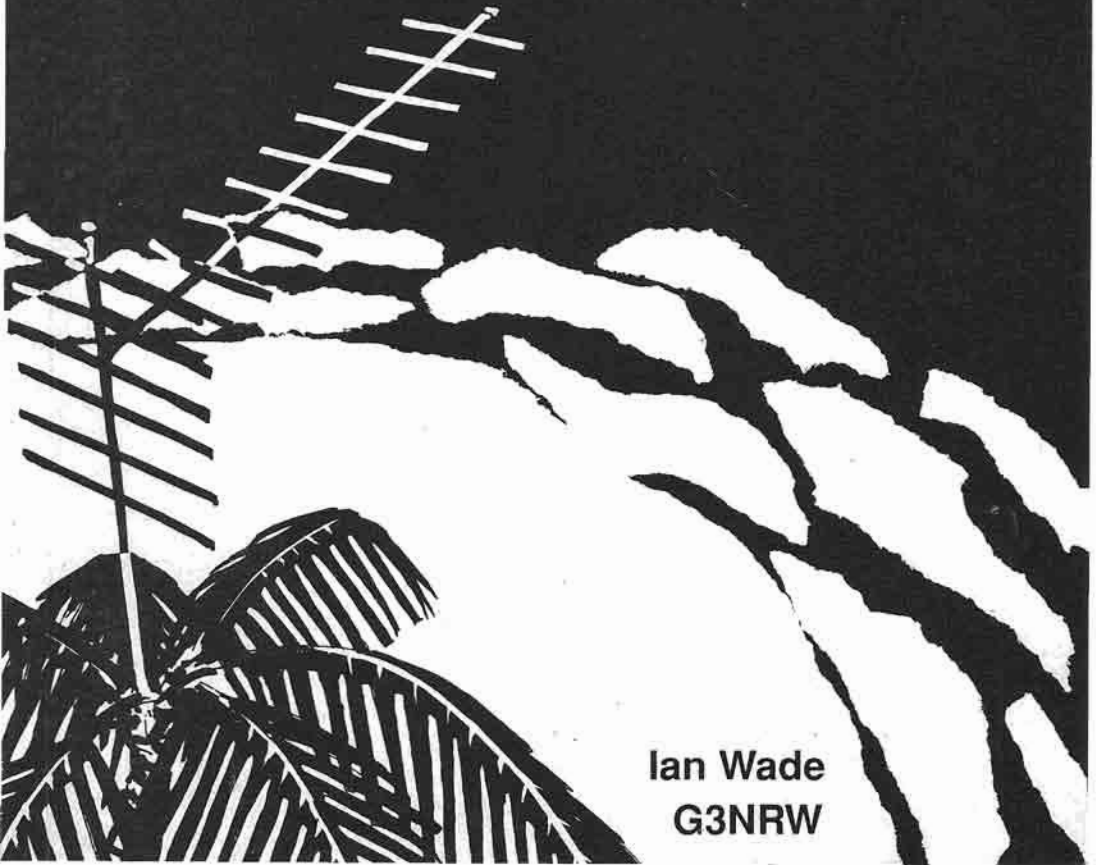
NOSintro

TCP/IP over Packet Radio

An introduction to the KA9Q Network Operating System



Ian Wade
G3NRW

NOSintro

TCP/IP over Packet Radio

An Introduction to the KA9Q Network Operating System

Ian Wade
G3NRW

NOSintro

TCP/IP over Packet Radio

An Introduction to the KA9Q Operating System

ISBN 1-897649-00-2

Published by Dowermain Ltd, Maxet House, Liverpool Road, Luton, Bedfordshire LU1 1RF, United Kingdom.

Printed by Maslands, Edward Neate House,
Unit 12, Howden Industrial Estate, Tiverton, Devon EX16 5HW, UK.

Text set in TrueType with Microsoft *Word for Windows,* version 2.0.
Graphics produced with Micrografx *Windows Draw!*, version 3.0.

Cover design by Jim Housego.

First edition: November 1992.

How to contact the author:

AMPRnet [TCP/IP]:	g3nrw @ g3nrw.ampr.org [44.131.5.2]
AX.25 Packet Radio:	G3NRW @ GB7BIL.#27.GBR.EU
Electronic Mail:	g3nrw @ dircon.co.uk
Post Office Mail:	Mr Ian Wade 7 Daubeney Close Harlington Dunstable Bedfordshire LU5 6NF, United Kingdom

CONTENTS

Chapter	1	Intro to NOSintro	1
	2	NOSview	7
	3	The Ground Rules	11
	4	NOS in a Nutshell	19
	5	Let's Meet the Locals	33
	6	The TNC Revisited	37
	7	A Peek at Protocols	49
	8	Names, Domains and Addresses	55
	9	Client/Server	63
	10	Hands On — Hardware Checkout	71
	11	Hands On — Software Installation	75
	12	NOS File Compendium	83
	13	Hands On — Session Manager	95
	14	The NOS Command Set Summary	109
	15	Hands On — *autoexec.nos*	115
	16	The *ftpusers* File	123
	17	Hands On — FTP	129
	18	NOS BBS — The Big Picture	141
	19	Setting up the NOS BBS	155
	20	The NOS BBS Command Set	161
	21	Hands On — NOS BBS File Server	173
	22	Hands On — Remote Sysop	181
	23	Forwarding SMTP Mail	185
	24	Pop Mail Collection	211
	25	PBBS Mail Forwarding	217

26	AX.25 Routing	231
27	Address Resolution Protocol	235
28	IP Routing	239
29	NET/ROM Routing	251
30	Going Live: Preparing the Files	263
31	Hands On — AX.25	267
32	Hands On — NET/ROM	275
33	Hands On — Ping and Hop	279
34	Hands On — Domain Name System	287
35	Trailing Flag	293
Appendix 1	Where to get the Software	295
2	NOS Command Set Reference	297
3	NOS Control Files	311
4	Character Codes	331
5	AMPRnet IP Address Coordinators	335
6	References	341
Index		343

1: INTRO TO NOSintro

Welcome to NOSintro, the beginner's guide to running TCP/IP over Packet Radio.

In this book you'll find a wealth of practical information, hints and tips for setting up and using the KA9Q Network Operating System (NOS). The emphasis throughout is on hands-on practicalities. You'll see exactly how to install NOS on a PC, how to set up the control files to suit your local environment, how to check out basic operations off-air before going live, and how to use NOS commands for transferring files, logging in to remote systems, sending mail, and so on.

Theoretical coverage is kept to a minimum — there are plenty of other publications describing the minutiae of TCP/IP and related packet protocols if you want to dig deeper. In this book there is just enough theoretical background to provide a framework for the hands-on sessions, so you get a good understanding of what's happening without being submerged in a morass of superfluous detail.

NOS and related packages such as NET run on all of the well-known families of microcomputers. These include the Apple Macintosh, Amiga, Archimedes, Atari, DEC VAX, IBM PC and Sun, running under MS-DOS, OS/2, VMS and various flavours of UNIX. This book is specifically about the PC version of NOS, but the other versions work in virtually the same way, so almost everything you read here is applicable to those versions as well.

NOS is a complex package, and requires you to set up a number of control files before you can use it. This isn't a difficult job, but there is quite a lot of work involved. To help get you on the road, this book contains full listings of typical NOS control files, which you can modify to suit your own environment.

Better still, you can obtain a copy of the G3NRW **NOSview** on-line documentation package for NOS. **NOSview** contains not only full reference documentation for NOS, but also a complete working set of

NOS software. This includes NOS itself and all of the control files listed in this book. You should get hold of **NOSview** if you can and install it on your PC, as the worked examples in this book relate directly to the files that come with the package. Full details of how to get **NOSview** are in Chapter 2.

I've said that this is a book for beginners to TCP/IP. The level is pitched at people who already know how to drive a PC at the MS-DOS command line, and how to make "ordinary" AX.25 packet radio connections with a conventional terminal node controller (tnc). Experience in sending and receiving messages via an AX.25 packet bulletin board system (PBBS) is also assumed.

In a book of this kind, it's impossible to explain everything about NOS. NOS is a big, complex package, with many more features than most commercial packages costing hundreds of dollars, and so it's only possible to scrape the surface here. My main hope is that there is more than enough information to get you started, with plenty of clues as we go along about what to explore next. In fact, the first two drafts of this book were much longer than originally intended, and savage wielding of the scalpel was eventually necessary to bring it down to a reasonable size. Given time, I plan to use some of the excised material in a follow-up book which will cover the advanced capabilities of NOS in much more detail.

NOS originally grew up in the world of amateur radio, but in more recent times it has found its way into "professional" environments as well. If you are a networking professional reading this book, please don't be misled by the word "amateur". Most of the techniques, the software and the networking infrastructures described here are the work of internationally respected professionals and academics, who also happen to be licensed radio amateurs.

The great attraction of the amateur environment is that people are free to experiment at will, without the constraints of fixed project goals and timescales, or bosses looking over their shoulders. Indeed, many of the techniques which are commonplace in the professional field today were originally developed by amateurs.

There's certainly nothing "amateurish" about NOS. You can install NOS on your PC in the office and connect into your LAN (or WAN or SLIP link), and you can use **ping**, **ftp**, **telnet**, **mail**, **news**, **ppp** and all the other well-known Internet services in exactly the same way that you probably do now. The big advantage of NOS is that it provides much

greater functionality than you'll find in most commercial packages, and it's free.

Reading a work of this nature is not a trivial undertaking. The best way to start is to spend an evening speed-reading the whole book from cover to cover, just to get the feel of it. Don't worry if there are parts you don't understand — just skip them and move on. Then read the book again, a little slower this time, perhaps taking a week of evenings to do so. If there are sections you still don't understand, skip them and read on to the end of the chapter. Then go back to the beginning of the chapter and read it again. Don't be afraid to dip and dive into different parts of the book to fill in the blanks — eventually the whole picture will become clear.

By then you should have a fairly good idea of what TCP/IP and NOS are about. The next step is a must: you must install NOS (ideally with **NOSview**) on your PC, so that you can try out the commands at first hand. Then read through the book yet again, this time concentrating on the hands-on sessions. Only at this point, when you type in NOS commands and see the results of your actions, will you really begin to understand what's happening.

All of this takes place with the radio switched off. When you eventually feel confident that you understand most of the capabilities of NOS, you are ready to modify the NOS control files to suit your own environment. You'll be replacing the dummy radio callsigns, network addresses and other parameters listed in this book with real callsigns, etc, and then you can switch the radio on and try out NOS on-air.

Chances are that if you follow these steps — it may take three or four weeks of spare time before you are ready for live tests — you'll be rewarded with almost everything working perfectly first time. Now you can login to other stations, transfer files, send mail, forward mail onto the PBBS network, run a NET/ROM node, etc, etc, and very quickly you'll be hooked! New avenues of exploration will open up, new software will come along to experiment with (TCP/IP is *the* growth area in networking software development these days), and I guarantee that there will always be something new to learn and try.

What if you can't make things work? The best people to help are obviously your neighbours who are already using TCP/IP, or you can put out a general bulletin on the PBBS network asking for advice.

If you are still having difficulties, I will be pleased to try to answer your queries. Full contact details follow immediately after the title page of this book. In general I would prefer to receive messages by packet radio or email, but if you write a letter, please enclose an SASE (and IRCs if appropriate) for your reply.

Note that my experience of NOS is with various MS-DOS implementations based on original versions from KA9Q. I haven't run NOS on any other platform. Therefore if you have specific detailed questions on the other platforms, please address them elsewhere; I don't want to mislead you with second guesses!

Acknowledgements

NOSintro is based on the work of many people. In the list below I hope I've included all of those who have written NOS software and documentation in the past, and who have played a significant part in the development of the amateur TCP/IP packet radio network throughout the world.

The number one acknowledgement goes, of course, to Phil Karn, KA9Q, the father of NOS. Phil has demonstrated to the world that it's possible to build a powerful, fully functional multi-tasking communications system, conforming to international networking standards, on the back of a primitive, memory-constrained, single-tasking operating system. It shouldn't work, but it does.

What's more, Phil has made his software freely available to the world, and several other people have now used it as a starting point for further development. Without Phil's contribution, it's unlikely that the amateur packet network would be anything like as advanced as it is today.

Now the roll call of other major contributors (in last name order):

John Ackermann, AG9V
Hayden Bate, G8AMD
Dave Brooke, G6GZH
Mike Chace, G6DHU
Tom Clark, W3IWI
Mike Dent, G6PHF
D R Evans, G4AMJ/NQ0I
Gary Ford, N6GF
Dan Frank, W9NK
Bdale Garbee, N3EUA
Fred Goldstein, K1IO
Gerard van der Grinten, PA0GRI
Allen Gwinn, NK5CKP
Charles Hedricks
Kelvin Hill, G1EMM
Gareth Howell, G6KVK
Pavel Jalocha, SP9VRC
Brian Kantor, WB6CYT
Anders Klemets, SM0RGV
Wally Linstruth, WA6JPR
Peter Meiring, G0BSX
Russell Nelson
Johan Reinalda, WG7J/PA3DIS
Bill Simpson
Mike Stockett, WA7DYX
Paul Taylor, G1PLT
Dave Trulli, NN2Z
Stanley Wilson, AK0B

If you are missing from the list and feel you should be there, please don't be offended. Treat it as an inadvertant omission on my part. If you care to drop me a line I'll be glad to add your name to the credits in the next edition of **NOSintro**.

Good luck with TCP/IP. You'll have fun!

73

Ian Wade, G3NRW

November 1992

2: NOSview

NOSview is an on-line public domain documentation package for the KA9Q Network Operating System (NOS). First released in September 1991, **NOSview** is a complete reference work describing in detail all of the commands to be found in the major NOS releases. This chapter outlines its main features, and how to get a copy.

Introducing NOSview

Over the years, many documents have appeared on the networks describing various features of NOS, but much of that material is incomplete. Some of it is inaccurate, and, because it was written and edited by many hands, sometimes very misleading and inconsistent.

In **NOSview** I have attempted to pull together all the available documentation and massage it into a consistent whole. The final product is almost 300 pages long, around 20 percent being new material. All of the NOS commands are described in detail, and there is at least one example included with each command. There are also many examples of display outputs, showing the results of executing the commands.

Consistency

Because NOS contains software modules originating from several different sources, the associated documentation inevitably contains inconsistencies.

For example, the words *label* and *interface* apparently describe different objects, whereas in actuality they are the same thing. On the other hand, the word *address* can have different meanings, depending on the command.

A lot of effort has gone into **NOSview** to eliminate these inconsistencies. Command parameter names are now consistent throughout. Callsigns in the examples follow a set pattern: calls for NOS stations are in the *NS9xxx* series, vanilla AX.25 stations are *AX9xxx*, NET/ROM stations are *NR9xxx*, and so on.

Also, to distinguish between IP hostnames and AX.25 callsigns, hostnames are shown in lower case (*ns9bob*), whereas AX.25 callsigns are in upper case (*NS9BOB-5*).

These seemingly simple rules make a tremendous difference to the readability of the documentation. There is now no doubt about whether a parameter should be an IP hostname or an AX.25 callsign, or whether you need an IRQ number or an interrupt vector address, and so on.

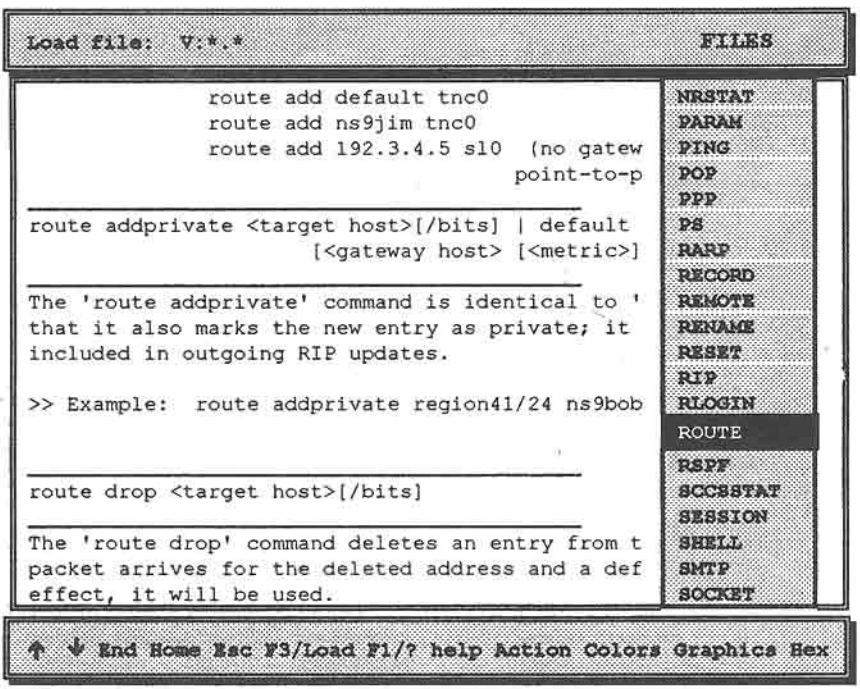


Fig 2-1: On-line NOS documentation with NOSview. There is a separate help file for each NOS command, selected from the pop-up FILES menu.

NOSview On-line

But this is only half the story. The real power of **NOSview** comes into its own when used with **VIEW**, a public domain file viewer which is included with **NOSview**. **VIEW** lets you hot-key to the **NOSview** documentation without breaking out of NOS, providing instant on-line help whenever you need to know what to do next.

Figure 2-1 opposite shows an example of the **VIEW** screen. To take full advantage of **VIEW**, **NOSview** is supplied as a set of over 90 individual help files, one file for each NOS command. This provides immediate access to the command of interest, saving time and effort when searching for detailed information.

A further benefit of supplying **NOSview** as individual files, rather than as one monolithic document, is that you can place the files in your NOS public directory. Then when someone logs into your system, they can download selected **NOSview** information in manageable pieces, rather than saturate the network for hours on end trying to download one enormous file.

NOSgas: The NOS Get-Away Special

Yet another feature of **NOSview** is that it contains a complete working set of NOS software, dubbed **NOSgas** — the NOS Get-Away Special. **NOSgas** incorporates a complete set of supporting files (such as *autoexec.nos*, *ftpusers* and so on) which you can use on your system. The templates are accompanied by full descriptions of their formats, plus warnings about the "gotchas" which can cause lots of frustration if you are unaware of them.

All you have to do is edit these templates to match your system (by modifying callsigns, etc), and you have a ready-made environment to try out NOS.

How to get NOSview

The current release of **NOSview** is version 244; i.e. released in 1992, week *44*. By now, **NOSview** should be available on the major telephone bulletin boards worldwide, and also on Internet host *ucsd.edu*

in directory *hamradio/packet/tcpip/docs*. Look for the files *NOSVIEW.ZIP* and *NOSVW244.ZIP*.

Alternatively, you can get a free copy by sending me a clean DOS-formatted diskette (any size *except* 360K) and return mailer to the address on the reverse of the title page of this book. Note that I can only supply **NOSview** for DOS machines (i.e. running under DR-DOS, MS-DOS or PC-DOS).

Please enclose return postage with your mailer as follows:

United Kingdom:	UK postage stamps
Europe:	3 IRCs
The Americas:	7 IRCs
Rest of World:	9 IRCs

Any unused IRCs will of course be returned.

Other Versions of NOS

Appendix 1 contains a list of people you can contact to obtain other versions of NOS for non-DOS platforms.

3: THE GROUND RULES

This book contains many abbreviations and acronyms, and a lot of networking and computer jargon. Many of these terms mean different things to different people, and even the experts use the same words in quite different ways. So to pull everything together, this chapter presents a unified description of these words and abbreviations, explaining how they are used in the book. Having defined the terminology and the ground rules here, the rest of the book should then be much easier to read.

Abbreviations and Acronyms

PC: Personal Computer, in the widest sense, not just an IBM PC running DOS. Although this book specifically describes the IBM PC environment for NOS, most of what you see is equally applicable to the other popular machines such as Apple Macintosh, Amiga, Atari, and so on.

DOS: Disk Operating System. This means Digital Research's DR-DOS, Microsoft's MS-DOS or IBM's PC-DOS, or any of the many vendor-specific work-alikes.

NOS: Network Operating System. This is the basic TCP/IP software package and the subject of this book. Earlier PC versions were known as **NET**, and some versions for other platforms are still known as NET. The examples in this book are based on PA0GRI's NOS version 2.0m released in Summer 1992, which is in turn based on KA9Q's version of 29 December 1991. Other currently available DOS versions of NOS include WNOS and JNOS.

NET/ROM: This is the networking software for switching nodes from Software 2000, or work-alike packages such as TheNet.

Networking Protocols

The world of data communications is overflowing with abbreviations, acronyms and protocols. Here is a checklist of the protocols used in NOS:

AMPRnet: Amateur TCP/IP Packet Radio Network.

ARP: Address Resolution Protocol. Handles the association between IP hostnames and AX.25 callsigns or Ethernet adapter addresses.

AX.25: Amateur X.25 Link Layer Protocol. Handles level 2 frame transfer between stations.

AXIP: AX.25 over IP protocol. Used for encapsulating AX.25 packet frames for transmission through an IP "wormhole".

BOOTP: Boot Protocol. Used for bootstrapping NOS.

FINGER: Finger Protocol. Allows users to find out about other users.

FTP: File Transfer Protocol. The principal protocol for transferring ASCII and binary files between stations.

ICMP: Internet Control Message Protocol. Handles IP transmission errors.

IP: Internet Protocol. The workhorse network protocol in the TCP/IP combination.

KISS: "Keep It Simple, Stupid!" Protocol. Handles data transfer between the host computer and the tnc.

NET/ROM: Handles Transport Layer data transfer.

NNTP: Network News Transfer Protocol. Handles distribution of news files.

NRS: NET/ROM control protocol for managing an external (non-NOS) NET/ROM node.

PING: Packet Internet Groper protocol. Used for checking the availability of other stations.

POP, POP2, POP3: Post Office Protocols. Handle reverse forwarding of SMTP mail.

PPP: Point-to-Point Protocol. Handles serial link data transfers.

RIP: Routing Information Protocol. Handles IP routing table broadcasts.

RLOGIN: Remote login. Allows login to remote computers.

RSPF: Radio Shortest Path First protocol. Another protocol for handling IP routing table broadcasts.

SLIP: Serial Link Internet Protocol. Another point-to-point serial link protocol.

SLFP: Serial Link Frame Protocol. Handles serial link compression.

SMTP: Simple Mail Transfer Protocol. Handles the forwarding and reception of mail.

TCP: Transmission Control Protocol. Handles reliable virtual circuits between stations, flow control and error recovery (e.g. duplicate or missing packets).

TELNET: Remote login. In NOS systems, handles login to the NOS BBS.

TIP: Terminal Interface Protocol. Handles direct communication with a serial port, character-by-character, with no additional protocol overhead. Can be used for initialising a tnc or modem.

TTYLINK: Chat Protocol. Handles interactive character-by-character conversations.

UDP: User Datagram Protocol. Handles one-shot data transfers (which may get lost en-route).

UUENCODE: Encodes binary files into ASCII prior to transmission.

UUDECODE: Decodes uuencoded files from ASCII back to binary.

Conventions

NOS is based on software which has been around for a long time in the UNIX world. This means that many of the UNIX conventions apply.

NOS commands are case-sensitive. That is, they consist entirely of lower-case letters; for example, NOS understands the command **session** but doesn't understand **SESSION** or **Session** or **SeSsIoN**, or any other variation containing capital letters.

Unlike UNIX or DOS, NOS understands abbreviated commands. You can abbreviate most commands down to one or two letters, provided the abbreviation is still unique. For example, NOS understands **sess** or **ses** or even **se** to mean **session**, but **s** by itself is ambiguous, as there are several other commands beginning with the letter **s**.

In this book, command names are given in full for clarity.

NOS file paths use forward slashes (/), not backslashes (\) like DOS. Thus you will see NOS filenames written like */spool/mail/sysop.txt*, not *\spool\mail\sysop.txt*. Also, NOS filenames are in shown in lower-case.

All NOS directories are rooted on DOS drive letter *N:* — thus the NOS file */spool/mail/sysop.txt* corresponds to the DOS file *N:\SPOOL\MAIL\SYSOP.TXT*. In this book, we define the NOS root with the command:

```
SUBST N: C:\NOS
```

and so the NOS file */spool/mail/sysop.txt* is really the DOS file *C:\NOS\SPOOL\MAIL\SYSOP.TXT*.

For the few DOS-specific files described in this book, the usual DOS conventions apply. That is, they are written in upper-case and with backslashes; e.g. *C:\DOS\ANSI.SYS*.

Station Identification

To run NOS you will need to choose an IP hostname. This is the name of your system by which other TCP/IP stations will know you, and will normally be your callsign. In this book, IP hostnames are in lower-case; e.g. *ns9bob*.

To distinguish between IP hostnames and AX.25 callsigns, the latter are in upper-case; e.g. *NS9BOB-5*.

Almost all callsigns in this book are fictitious. The following prefixes apply to make it easier to distinguish between different types of station:

- *ns9...* NOS (TCP/IP) station
- *NR9...* NET/ROM node
- *AX9...* AX.25 end-user station
- *BB7...* AX.25 PBBS station

NET/ROM aliases in this book are of the form #<suffix>. For example, ns9bob has the alias #BOB. (In reality, the choice of alias is largely a matter of personal preference, but usually it begins with # or the letters IP or TCP, to distinguish it from ordinary NET/ROM aliases).

Keyboard Characters

The following abbreviations apply:

- **CR** carriage-return (enter)
- \r carriage-return (enter)
- **LF** line-feed (newline)
- **^** the CTRL key (e.g. ^Z means control-Z)
- **CTRL** the CTRL key
- **SHIFT** the SHIFT key
- **ALT** the ALT key
- **ESC** the ESCAPE key
- **Fn** Function Key **n**

Mail and Bulletin Boards

The generic word *mail* encompasses personal *messages* and public *bulletins*.

PBBS: The PBBS is the traditional packet bulletin board system, featuring the familiar AX.25 mailer (using commands like **SP** to send mail, **R** to read it, and so on).

NOS BBS: The NOS BBS is the bulletin board system built into NOS. This handles AX.25 mail in the same way as a PBBS, and also handles SMTP mail for AMPRnet/Internet.

External mailers (such as PCElm, ELM and BM) are alternative programs to handle SMTP mail. You can run these programs completely separately from NOS, or you call them from within NOS with the **mail** command.

PMS: A PMS is the Personal Messaging System built in to conventional tncs.

The Terminal Node Controller

The tnc operates in three basic modes:

Native Mode is the normal mode for which the tnc was originally designed. That is, it lets you access the packet network directly from the keyboard, by giving commands in response to the familiar *cmd:* prompt. The tnc controls everything to do with sending and receiving AX.25 packets, and does not even need a host computer (all that is required is a dumb terminal).

Host Mode requires the use of a host computer with the tnc. The host computer takes over virtually all of the functionality of the tnc, allowing a much greater degree of control, but is still restricted to sending and receiving AX.25 packets.

KISS (Keep it Simple, Stupid!) Mode is a variation of host mode. The host computer runs almost all of the network software, and communicates with the tnc using the KISS protocol. In KISS mode, the tnc can handle all the protocols supported by AMPRnet, Internet, NET/ROM and AX.25.

Origin/Target and Source/Destination

When sending information via intermediate stations, it is important to understand the distinction between terms like *origin*, *source*, *destination* and *target*. In this book, these terms are used as follows (see Fig 3-1):

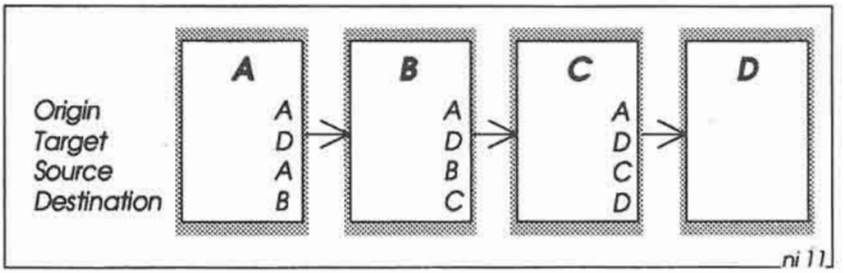


Fig 3-1: *Origin* and *Target* refer to the extreme end-points of communication. *Source* and *Destination* refer immediate neighbours.

Origin: is the station originating the information. If you are sending a message to somebody else, your station is the origin.

Target: is the final intended recipient of your information.

Source: is the station transmitting the information at this point.

Destination: is the station receiving the information at this point.

Thus, referring to Fig 3-1, station A is the origin of the information, and station D is the final target. For the path between A and B, A is the source and B is the destination, and for the path between B and C, B is the source and C is the destination, and so on.

We'll see that the situation can get quite complicated when considering a multi-layer path between NOS stations, where the end-to-end path may take in IP gateways, NET/ROM nodes and AX.25 digipeaters. In this situation it's very important to keep a clear head when referring to source and destination, as these terms may refer to different stations at the different network layers.

Routers and Gateways

TCP/IP has been around for several years, and a whole vocabulary has grown up around it. More recently, the International Standards Organisation (ISO) has formulated the Open Systems Interconnection (OSI) model — the so-called *7-layer model* — to describe network communications, and this too has its own vocabulary and jargon.

It turns out that there is some commonality between the TCP/IP and OSI models, but there is also a lot of overlap and conflict, with the

same terms having quite different meanings in the two models. Predictably, this can cause a lot of confusion. This is not the place to compare the two models; instead we will say here that the TCP/IP terminology will be used throughout most of this book, with just occasional references to the OSI model for comparison.

The main candidate for confusion is the word *gateway*. In the TCP/IP world it is referred to as an *IP gateway*, which corresponds roughly to an *OSI Router* (and is nothing to do with an *OSI gateway*).

4: NOS IN A NUTSHELL

This chapter provides a brief overview of the KA9Q Network Operating System (NOS).

NOS is a multi-tasking operating system that provides an extremely flexible and powerful set of communications services for use on packet radio networks, telephone lines and local area networks. NOS supports most of the commonly used Internet protocols such as TCP, IP, TELNET, FTP, SMTP and so on, plus the packet radio protocols AX.25, NET/ROM and PBBS mail.

With NOS you can communicate with virtually any kind of computer (Fig 4-1). An Amstrad can talk to an Apple, an Amiga can talk to an IBM mainframe, a laptop PC can talk to a Cray, and so on. What's more, you can send electronic mail via worldwide networks, and you can even log into remote systems, just as if you were directly connected to them.

NOS supports the *AMPRnet* (Amateur Packet Radio network), which rides on the back of the Internet protocols. These protocols are operating system independent. This means, for example, that you can run NOS on a PC running DOS or UNIX/XENIX, or a DEC VAX running VMS, or a Sparc workstation running SunOS. You can send binary or ASCII files between them, handle mail, and set up gateways to link different types of network.

Probably the most important aspect of NOS is that all of these protocols and services conform to *internationally agreed standards*, and are available in one form or another on virtually every micro, mini and mainframe system in use today. This means that you are not locked into non-standard software (such as YAPP or 7PLUS) that nobody outside the amateur world understands, and you can communicate with almost any type of computer in the world in exactly the same way. NOS is truly an Open System.

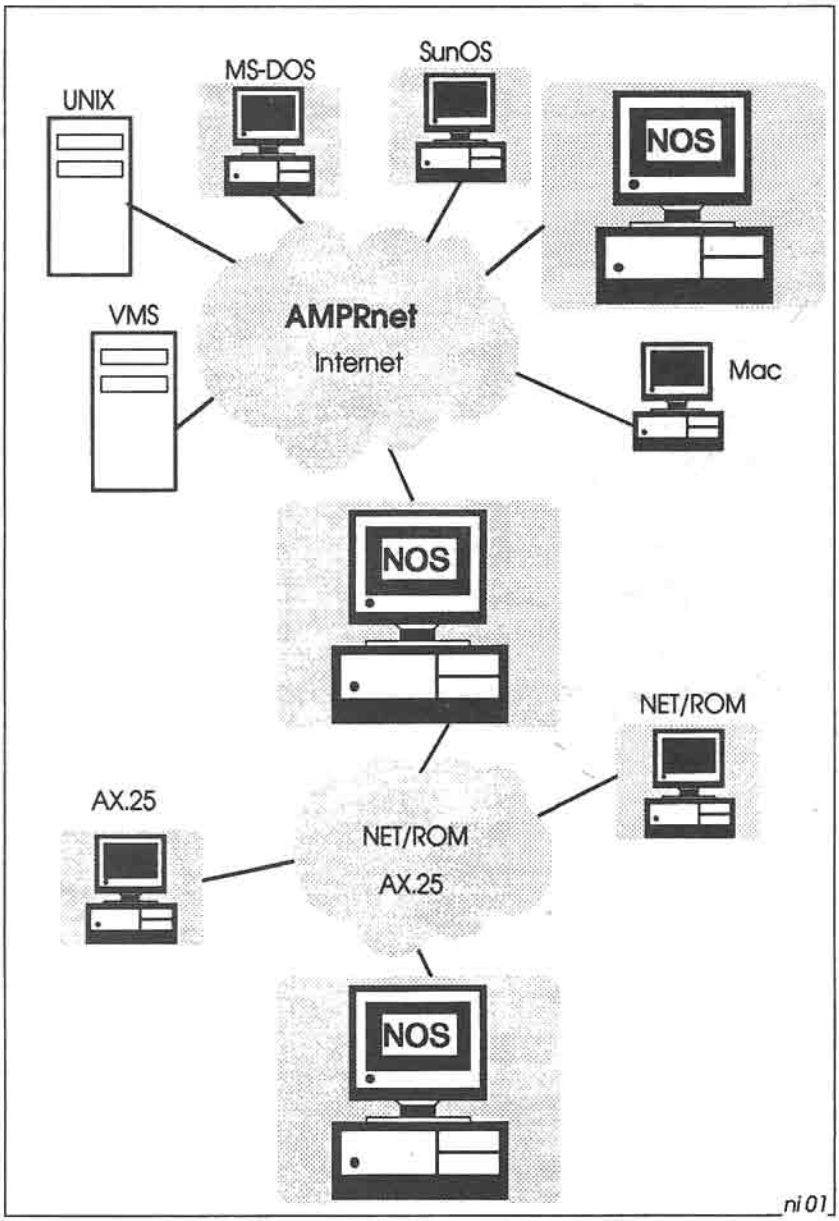


Fig 4-1: NOS provides connectivity with AMPRnet, Internet, NET/ROM and AX.25

There's more. As well as supporting TCP/IP and AX.25, NOS also understands NET/ROM. You can even set up your own NET/ROM node if you want to.

Why support NET/ROM? Well, in the ideal world, all NOS systems would talk TCP/IP directly to each other, and would handle node-to-node routing at the IP level. Unfortunately we have yet to reach this ideal state, so in most cases we have to rely on existing networks to carry our TCP/IP traffic instead. The most widespread packet radio network which already exists is NET/ROM, so that is why NOS supports it.

Thus when you monitor TCP/IP traffic, you may see AX.25 frames which contain NET/ROM packets which contain IP packets which contain TCP packets — see Fig 4-2. Sounds complicated, but once you've read this book you'll see that it's really quite straightforward to set up, provided you keep a clear head and understand the functions of the different network layers.

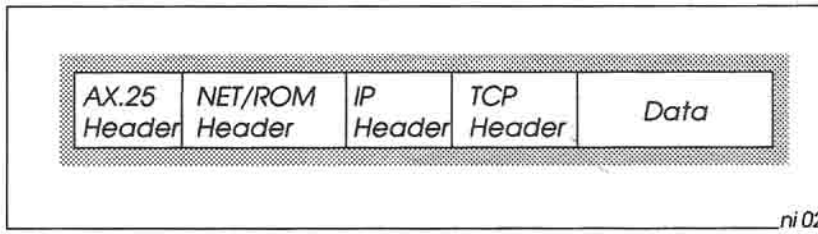


Fig 4-2: A Multi-protocol NOS Packet.

And there's even more. NOS also supports PBBS forwarding and reverse forwarding, allowing us to communicate with the established PBBS mail network. NOS stores the PBBS mail files in the same directories as TCP/IP mail files, and you can read and send mail in either format.

Thus we have the best of both worlds. We can choose to send our mail either via the AMPR network or via the AX.25 PBBS network, and we can read mail from both of those networks as well.

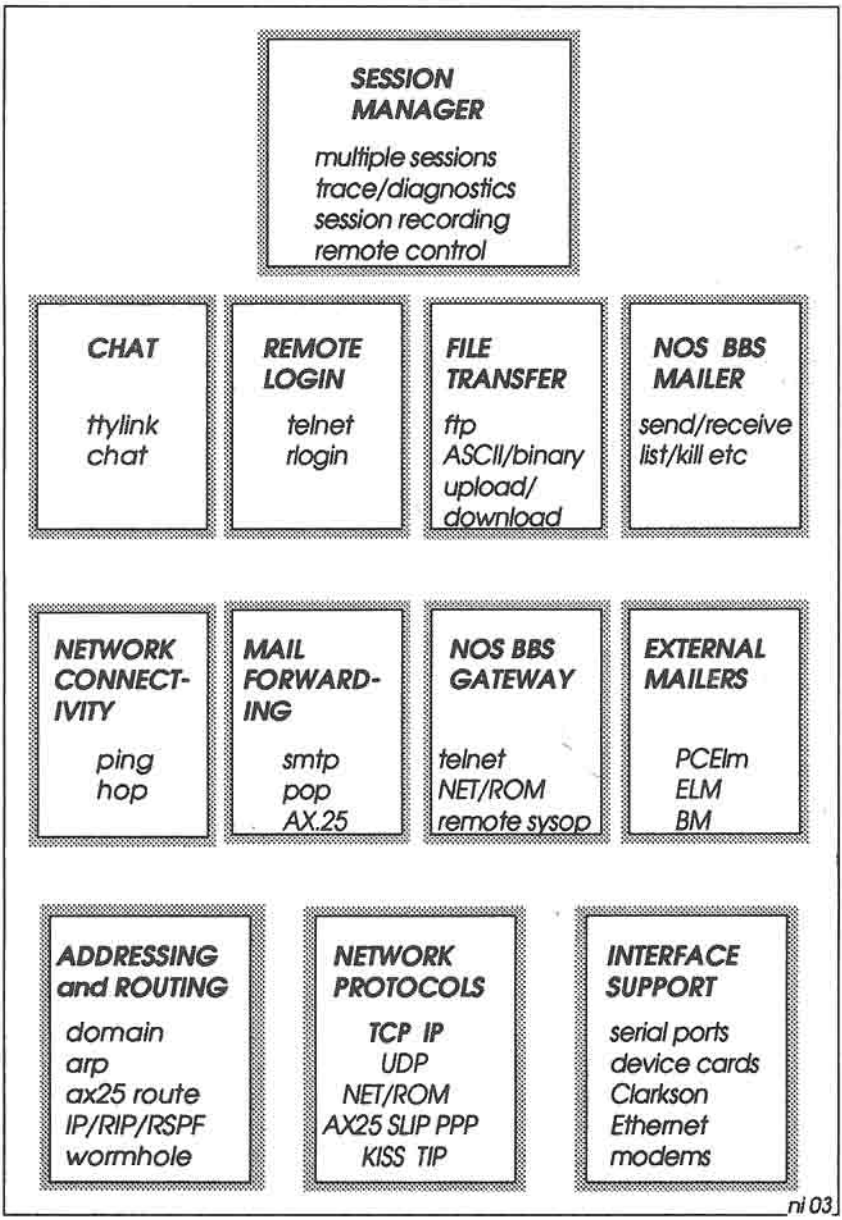


Fig 4-3: NOS — The Big Picture

The Basic Requirements

To run NOS in a DOS environment, you need the following:

- a PC
- a tnc capable of KISS operation (most are today)
- a copy of the NOS software
- NOS documentation (e.g. the **NOSview** documentation package)
- an Internet (IP) Address.

The PC can be almost any model in the 80x86 family, with at least 1MB of memory. Obviously the machine should be the fastest you can afford, at least 8 MHz. With a full set of run-time software and on-line documentation, you will need about 2.5MB of hard disk space (although you can run a bare-bones system on a laptop with just dual 720K diskette drives at a pinch).

NOS software and full reference documentation are available in the **NOSview** package, already described in the Chapter 2.

Internet addressing is explained below.

The Internet Protocols

Figure 4-3 opposite shows the main building blocks of NOS. The two networking protocols at the heart of NOS are the Transmission Control Protocol (TCP) and the Internet Protocol (IP), at the bottom of the diagram. These protocols were developed under the aegis of the Defense Advanced Research Projects Agency (DARPA) in the United States, and have been in general use in data networks throughout the world for many years.

However, the raw TCP and IP protocols are not of very much interest by themselves, at least not while you're still learning how to set up NOS. Much more important are the network services that use TCP/IP, which you will use to transfer files, send mail and so on.

The five main classes of network services which you will use are (again see Fig 4-3):

- Chat
- Remote Login
- File Transfer
- Mailers
- Network Connectivity

Chat

The chat service lets you do just that, using the NOS command **ttylink** (or **chat** in some versions of NOS). Thus if you want to chat to NS9KEN, you give the command **ttylink ns9ken**, and once you are connected you can converse in exactly the same way as in vanilla AX.25. NOS saves keystrokes in a buffer as you type, and then transmits the buffer when you hit **CR**.

Remote Login

There are two different remote login services provided in NOS. The one you are most likely to use is TELNET. When you give a command such as **telnet ns9ken**, you will normally be connected to his NOS BBS, where you can read and send mail, and use various network gateways if you have permission.

The alternative login service is **rlogin**. This command lets you perform a login to a remote computer which supports the RLOGIN protocol.

File Transfer

The NOS command for file transfer is **ftp**. To transfer files between your system and NS9KEN's system, you give the command **ftp ns9ken**, and when you are connected you can give the **get** command to fetch a file from NS9KEN (e.g. **get yourfile.txt**), or use **put** to send a file to NS9KEN (e.g. **put myfile.txt**).

You can transfer ASCII or binary files, simply by giving the **ascii** or **binary** command before starting the transfer. You don't need to worry about lost packets or duplicate packets; FTP takes care of error

detection and correction, so when the transfer is done you can be confident that it was successful.

Mailers

The basic function of a mailer program is to let you compose messages and bulletins ready for forwarding, and to read incoming mail. There are no less than four mailers which you are likely to come across in NOS systems:

- BM
- ELM
- PCElm
- NOS BBS

The first three of these mailers are not actually part of NOS, but are separate programs which you can call from NOS when you want to access your mailbox. Alternatively you can use them completely independently of NOS, starting them from the DOS command line.

The fourth mailer, the NOS BBS, is built in to NOS, and has several extra features in addition to handling mail.

Why so many mailers? It's really a matter of history. In early versions of NOS there was no built-in mailer, and BM ("Bdale's Messy Mailer" from N3EUA) was provided instead. This had very basic functionality and was cumbersome to use, but it served its purpose at the time.

Next in line came ELM, which provides a much nicer full-screen menu environment. With ELM you can compose mail using your favourite text editor, include files in your messages, set up mailing lists and so on. Many people use this mailer today.

PCElm is a more recent mailer which looks and works very much like ELM, but is in fact unrelated. As well as providing all the facilities of ELM, PCElm also has a built-in text editor and lets you set up screen colours, define message file name extensions and delimiters, filter out unwanted message headers and so on.

The built-in NOS BBS contains a simple mailer which works in a very similar way to the familiar AX.25 PBBS. You give commands like **L** to list mail, **SP** or **SB** to send it, **R** to read it and so forth. However, the NOS BBS also provides a set of gateway commands which let

users break out into the NET/ROM network or telnet into another NOS BBS, or even take over control of your station as a remote sysop — but you'll be glad to hear they can't do any of these things unless you give them permission!

Use of the NOS BBS is not restricted to TCP/IP users. An ordinary AX.25 user can connect to your NOS BBS, read and send mail just like a TCP/IP user, and can use the gateway commands as well if they have permission. In other words, this gives an ordinary AX.25 user the capability of accessing the NET/ROM network and AMPRnet if they want.

So which of these four mailers do you use? If you are logging into someone else's system, you have no choice: the built-in NOS BBS on that system is the only mailer you can access. On your own system you can use the NOS BBS if you want, but you'll probably prefer to use PCElm (or perhaps ELM) as it has a much nicer user interface.

Mail Forwarding

The main function of the mailers just described is to let you read and compose mail. To send and receive this mail, NOS provides three mail forwarding services:

- Simple Mail Transfer Protocol (SMTP)
- Post Office Protocol (POP)
- AX.25 PBBS Forwarding

SMTP handles the sending and receiving of mail via AMPRnet, and is the default method of handling mail. Using SMTP you can transfer mail between any computers which understand it; i.e. virtually any kind of machine in the world.

POP is the reverse forwarding protocol that works with SMTP. With POP you can nominate another machine as your Post Office, and when you run POP, your own machine will automatically login to the Post Office and collect any mail waiting for you.

AX.25 PBBS forwarding and reverse forwarding is fully compatible with the PBBS network, so if you don't have access to a local NOS system which can forward your mail using SMTP, you can still communicate with the outside world via the PBBS network.

Network Connectivity Services

NOS provides a number of network support services which let you check the availability of other stations on the network. These services include **ping** and **hop**.

The **ping** command is known officially in the trade as the "Packet Internet Groper" (... amazing but true!), and is useful when you're not sure if a local station is responding to your traffic. Whenever you want to check if a local station is active, you "ping" it; e.g. **ping ns9ken**. If NS9KEN is running NOS, it will respond to your ping, and you will see on the screen a number representing the round-trip time for your ping packets. If you get no response, or if the round-trip time is unexpectedly long, you know that something is wrong.

The **hop** commands let you check the availability of routes to a particular station. For example, to find out which gateways your packets pass through to reach NS9LIZ, you would give the command **hop check ns9liz**. This is very useful to verify that a route exists to the target station — and can sometimes show up some bizarre routings that you never knew existed!

Station IP Addresses

Every NOS station has an IP address, a unique 32-bit number which is usually expressed as four decimal numbers separated by dots (the so-called "dotted-decimal" notation). For example, NS9BOB's IP address in this book is 44.199.41.1.

The first byte is always 44, which represents the AMPRnet.

The second byte (199) usually represents a country (or a state in the United States).

The third and fourth bytes are an address within that country. Typically the third byte will represent a region or area, and the last byte will be a station number in that region.

Incidentally, you may see some documentation which shows IP addresses enclosed in square brackets; e.g. [44.199.41.1]. This convention is a relic of early NOS systems, and is not used today.

Each country or state where there is AMPRnet activity has a local IP address coordinator who allocates addresses on request. A list of

coordinators is shown in Appendix 5. You should contact your local coordinator listed in the appendix to get an address. If your country does not yet have a coordinator, you should contact the international coordinator in the United States instead (but be prepared — he will probably nominate you as the country coordinator!).

From time to time the coordinators issue a full list of IP addresses in their area, as a set of bulletins on the PBBS network. When you set up your NOS station, you will use this list to create the file *domain.txt*, which NOS uses whenever you make a network connection. (Strictly speaking, you don't really need to have a *domain.txt* file — you could use IP addresses instead of symbolic hostnames; e.g. you could give the command **ping 44.199.41.2** instead of **ping ns9ken**, but obviously it is more meaningful to use names rather than addresses).

Keeping *domain.txt* up to date is clearly a problem. One way round this is to nominate a local station as a *Domain Name System (DNS) Server*, which keeps a master copy of the file and makes it available to other users. (This is somewhat similar to a PBBS White Pages server, which keeps a record of AX.25 stations and their local mailboxes). If you then set up NOS to use the DNS server and attempt to make a network connection, NOS will first look in your own *domain.txt* file for the hostname you have given. If it can't find the hostname there it then automatically make a request to the DNS server machine for the IP address of the station you are trying to contact.

Address Resolution Protocol

When setting up a network connection, NOS needs to know not only the IP address but also the link address of the station you wish to talk to. If you are using a radio link, the link address is the other station's callsign (e.g. NS9KEN-5). If you are on Ethernet, the link address is the 48-bit hardware address of the Ethernet adapter card in the other station's PC (e.g. 00:00:C0:AC:01:26).

To set up the table of link addresses, NOS provides the **arp** (Address Resolution Protocol) set of commands. Thus, for example, to communicate with NS9KEN, you could give a command such as **arp add ns9ken ax25 NS9KEN-5**, thus forming an association between the IP hostname (ns9ken) and the link address (NS9KEN-5).

Routing

Routing controls how packets get to their destination. NOS supports no less than three completely independent levels of packet routing:

- AX.25 routing
- NET/ROM routing
- IP routing

You'll already be familiar with AX.25 routing, particularly when it's referred to by its more usual name: *digipeating*. NOS has a set of **ax25 route** commands which let you set up digipeater paths to nominated destinations.

For example, to route packets via digipeater AX9DGC to reach NS9PAM-5, the NOS command to set up the AX.25 routing table entry will be **ax25 route add NS9PAM-5 AX9DGC**.

Similarly, there is a set of **netrom route** commands, with which you can set up NET/ROM routes and aliases which ordinary NET/ROM nodes understand.

IP routing is basically an extension of the NET/ROM routing idea, specifically for forwarding IP packets onwards to their final destination. NOS has a set of **route** commands for setting up and maintaining the IP routing table.

Each of these three levels of routing is quite independent of the other two.

Routing Table Updates

In the amateur packet network, nothing lasts for ever — or even for a lot less time than ever! Routes between nodes are continually changing as stations come and go, as frequencies change and so on. This means that for there to be any realistic chance of communicating with other users on the network, your station has to be kept up-to-date with the current routing situation.

NOS achieves this in two ways. Firstly, it listens to user traffic on the frequency, and when it hears stations it dynamically updates the appropriate routing tables. These updates remain in memory for a finite period (usually measured in minutes or tens of minutes), and if a

station is not heard again the routing information for that station eventually disappears.

The second way that NOS keeps up-to-date is by routing broadcasts. NOS regularly sends broadcasts of the NET/ROM routing table in just the same way as a native NET/ROM node, and also sends IP routing table broadcasts at regular intervals.

For IP routing broadcasts, NOS supports two protocols: RIP (Routing Internet Protocol) and RSPF (Radio Shortest Path First) protocol. RIP is the protocol to use if your station is part of a well-established and stable network such as Ethernet, whereas RSPF works better in a dynamic radio environment.

Wormhole Routing

Another method of packet routing supported by NOS is the wormhole, which provides AX.25 connectivity over a TCP/IP link. This is useful where you are linking two AX.25 stations via an Ethernet or telephone connection. In effect, the NOS wormhole acts just like a (rather complicated) digipeater — see Fig 4-1 opposite.

Interface Support

NOS is noted for its very wide range of supported interfaces (although not every version of NOS will support all of them). These include:

- The serial ports (COM1 - COM4), for tncs or modems
- Modem control
- Ethernet adapters
- Clarkson drivers
- Baycom AX.25 driver
- DRSI PCPA 8530 card
- HAPN 8273 adapter
- High speed DRSI/HAPN driver
- Eagle 8530 card
- NET/ROM control
- Single- and multi-port KISS TNCs
- PACcom PC100

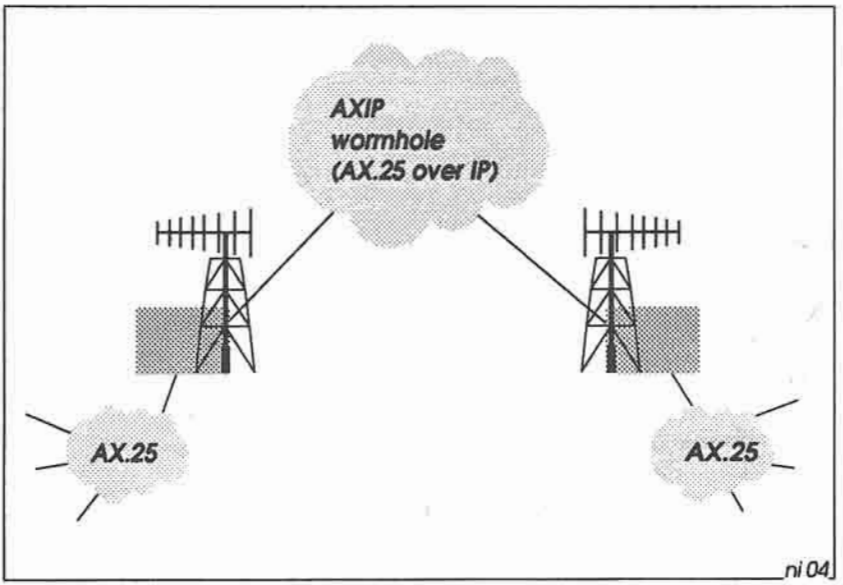


Fig 4-4: The AXIP *wormhole* lets AX.25 users communicate over AMPRnet.

In other words, NOS will talk to virtually any tnc or modem, or any of the well-known network adapters. The Clarkson drivers are freely available as public domain software, and support all of the generally available Ethernet and Token Ring LAN adapters.

NOS talks a number of low-level protocols via these interfaces:

- KISS for tnc control
- SLIP and PPP for serial point-to-point telephone links
- NRS for NET/ROM control
- Ethernet and ARCnet for Ethernet adapters

The NOS Session Manager

Because NOS is a multi-tasking system, you can run many sessions in parallel. Hence it's possible, for example, to telnet to NS9KEN, do file transfers with NS9LIZ, access your own mailbox, ping NS9BOB and have a chat with AX.25 station AX9AAA, all at the same time. In

principle you can run many more sessions as well, but you'll really need a Jekyll and Hyde personality to handle it all!

The NOS Session Manager maintains a virtual screen and keyboard for each session, and you can hot-key from session to session at will. You can find out the status of any session on demand, and you can trace all the traffic flowing through your station, right down to the hexadecimal byte level if you want to. You can also record any session on disk for later use. Furthermore, you can allow other stations to drive your Session Manager remotely if, for example, your station is on a remote hilltop site.

That's NOS

By now it will be clear that NOS is a very complex package, with many advanced features which make it usable in a wide variety of environments. To the beginner, some of the features in NOS may appear to be daunting, but fortunately it isn't necessary to understand everything before you can use it.

Just like when using a tnc for the first time, you can get away with using default setup parameters; performance may not be optimum, but it will at least work. Then as you gain experience you can dig deeper into the software and start to experiment with different configurations.

Driving NOS is a bit like driving a car. Think of the network protocols (TCP, IP, SMTP and so on) as the engine, and think of the network services (like TELNET and FTP) as the brakes and steering. To drive NOS (the car), you have to know about TELNET and FTP (the brakes and steering), but learning about TCP and IP (the engine) can wait until later.

With NOS, all you really need to know at first is how to set up the tnc, how to configure the address and routing tables, and how to use the basic network services to transfer files and handle mail. Fine tuning of the network protocol parameters can wait until much later.

5: LET'S MEET THE LOCALS

For the purposes of illustration throughout this book, there are many examples of NOS commands which include callsigns, IP addresses and so on. To achieve continuity and consistency from chapter to chapter, it's useful to present a network showing who's who. So let's meet the inhabitants of the hypothetical country of Nosland — see Fig 5-1.

The Nosland network consists of a mixture of ordinary AX.25 stations (the small boxes in Fig 5-1), NET/ROM nodes (middle-size boxes) and NOS TCP/IP stations (the shaded boxes). Stations have callsigns in the following series:

AX.25 stations	AX9xxx
AX.25 PBBS stations	BB7xxx
NET/ROM nodes	NR9xxx
NOS stations	NS9xxx

The IP network

The IP network (AMPRnet) is spread over three geographical regions, arbitrarily called Regions 41, 45, and 47. The IP addresses of stations in these regions are of the form 44.199.rr.xx, which breaks down as follows:

- 44 is the AMPRnet network code.
- 199 is the country code for Nosland.
- rr is the Region code (i.e. 41, 45 or 47).
- xx is the station address within the region.

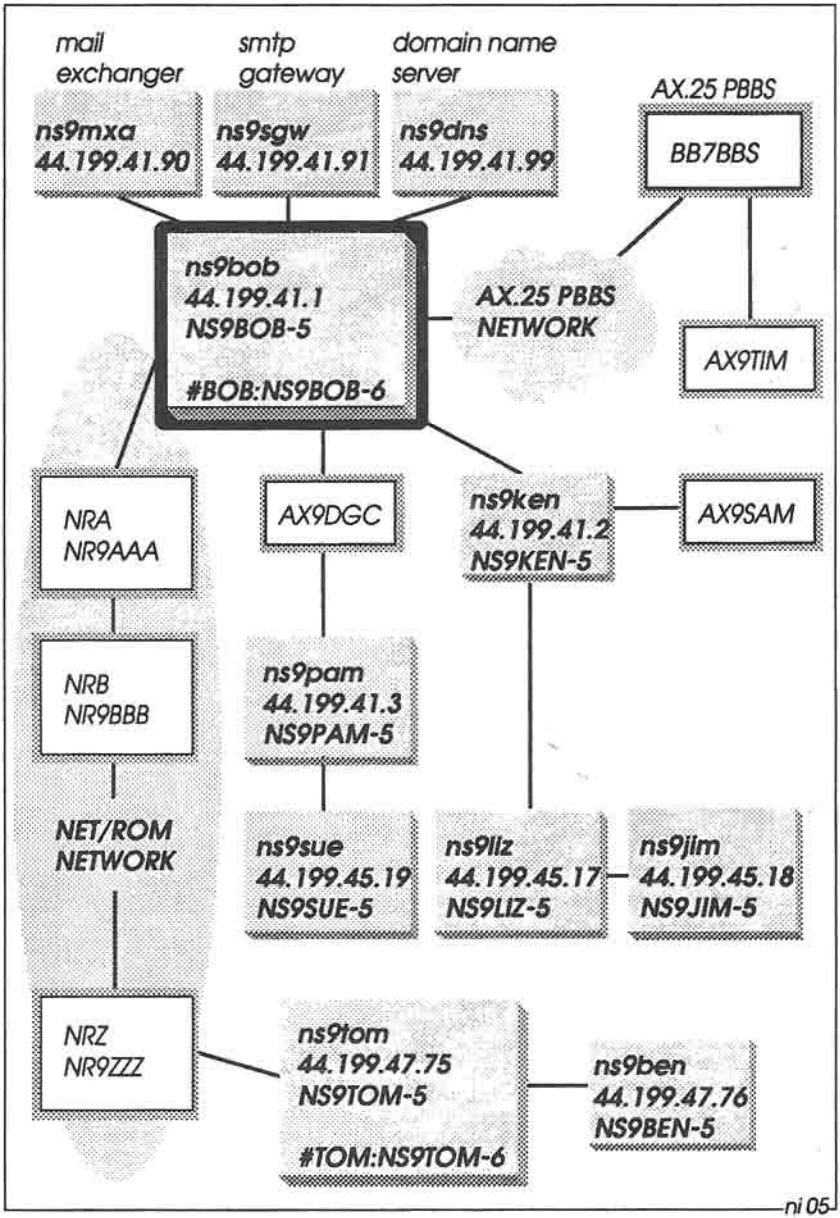


Fig 5-1: The NOSLAND Network.

Star of the show (on whom most examples in this book are based) is Bob, shown near the top of Fig 5-1. His AX.25 callsign (in upper-case letters) is NS9BOB-5, and he has the IP hostname (in lower-case letters) ns9bob.ampr.org, shortened to ns9bob. His IP address is 44.199.41.1.

Bob's immediate IP neighbours in Region 41 are NS9PAM (via digipeater AX9DGC) and NS9KEN. Through Pam he can talk to Sue in Region 45, and through Ken he can talk to Liz and Jim, also in Region 45.

Bob also talks to NS9MXA, which acts as a mail exchanger gateway. Bob uses this gateway to forward mail addressed to certain specific stations. He also forwards mail to other stations via the general SMTP gateway NS9SGW.

Further, he uses the Domain Name System server NS9DNS, to get IP addresses for stations which are not included in his own IP name-and-address file.

The NET/ROM Network

The ordinary NET/ROM network on the left of Fig 5-1 links Regions 41 and 47, and consists of three nodes, NR9AAA, NR9BBB and NR9ZZZ. These nodes have NET/ROM aliases NRA, NRB and NRZ respectively.

In addition, Bob and Tom run NET/ROM nodes within their NOS stations, using the alias:callsign pairs #BOB:NS9BOB-6 and #TOM:NS9TOM-6 respectively.

The AX.25 Network

There are several stations which run ordinary AX.25. These include digipeaters AX9DGA, AX9DGB and AX9DGC, plus end stations AX9TIM and AX9SAM (AX9DGA and AX9DGB are not shown on this diagram).

Station BB7BBS is a regular AX.25 mailbox, capable of receiving and forwarding mail over the PBBS network. Bob acts as a mail gateway, forwarding mail to and from BB7BBS.

Connectivity

The stations which can talk direct to each other are joined by lines in Fig 5-1. Certain stations function as bridges, routers and gateways, to provide connectivity for other stations which are out of direct range of each other. The Nosland network has been carefully designed to show how to configure each of these stations, to handle just about every forwarding scenario you'll encounter in practice.

6: THE TNC REVISITED

Most amateur packet radio systems use a terminal node controller (tnc) to interface the computer to the radio (Fig 6-1). The tnc is basically a Packet Assembler/Disassembler (PAD), with a built-in modem to convert outgoing digital bit streams into audio tones suitable for modulating the radio transmitter, and to convert incoming audio from the radio receiver into digital data for the PC.

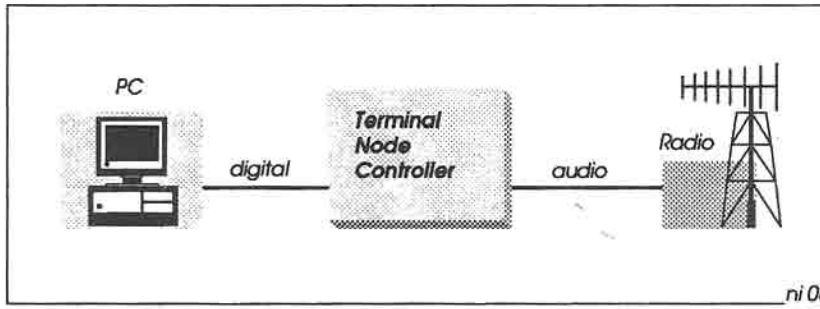


Fig 6-1: The Terminal Node Controller contains a packet assembler/disassembler (PAD) and modem.

[As an alternative, some people are now using the Baycom modem for packet radio. In this case, the PAD functionality is implemented in a special software driver which you load into memory before starting NOS].

This chapter examines briefly what is inside a tnc, and the various modes it can operate in.

TNC Modes

The tnc can operate in three different modes:

- Native mode
- Host mode
- KISS mode

Native Mode

Native mode is the mode for which the tnc was originally designed. As its name implies, the tnc controls a terminal node on the packet network, and when operating in native mode you don't even need a computer; a dumb terminal is enough.

When the tnc starts up it displays the familiar *cmd:* prompt on the terminal, and you can then give around 80 commands to start and stop connections, monitor network traffic, send and receive mail, and so on.

All of these operations are controlled by firmware within the tnc (Fig 6-2). The firmware has five main components:

- Command interpreter
- TNC control
- Packet Assembly/Disassembly
- Radio Control
- Personal Messaging System

Command Interpreter: The command interpreter directs user commands to the other firmware components.

TNC Control: This component understands dozens of commands to set up the tnc; e.g. uart control, terminal link flow control, clock initialisation, callsign setup, etc, etc.

Packet Assembly/Disassembly: This is the major component of the tnc firmware, handling the connection and disconnection of AX.25 virtual circuits, AX.25 timers, digipeater routing, beacons, network monitoring, packet sequencing/flow control and HDLC frame assembly/disassembly.

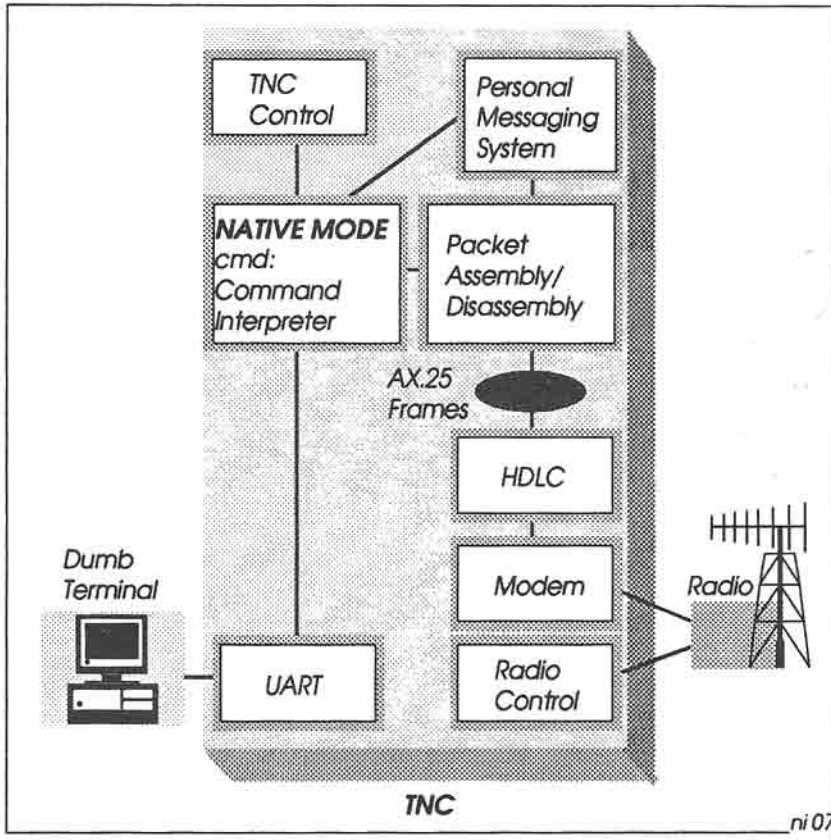


Fig 6-2: The tnc in native mode. All packet handling takes place inside the tnc.

Radio Control: This component supports the radio-dependent aspects of the tnc, such as TXDELAY and other timers, persistence count and so on.

Personal Messaging System: The PMS is a simple messaging system that stores personal messages which people have sent to you.

These tnc functions are only briefly listed here, simply to allow us to compare native mode operation with host mode and KISS mode. If you want to find out more about native mode, the book *Your Gateway to*

Packet Radio by Stan Horzepa is highly recommended (details in Appendix 6).

Shortcomings of Native Mode

When the first tncs were designed in the early 1980s, the goal was to give users the opportunity to get into packet radio with the minimum of equipment. Together with just a dumb terminal and a radio, you had everything you needed to make AX.25 network connections, chat interactively with your neighbours, and send and receive short personal messages.

But that was all. If you wanted to do more adventurous things like file transfers, or set up a store-and-forward bulletin board, or set up a network switch, you had to replace the dumb terminal with a PC.

To do these things properly, the PC has to be in control of the packet station, not the tnc. But with the firmware which existed in the early tncs, it was the *tnc* that was in charge of proceedings. The tnc decided when to send a message to the PC, and what format the message was in. Incoming status messages got mixed up with user data, and the file transfer was a hit-and-miss affair.

The basic difficulty was that the tnc's user interface had been designed for people, not computers. Human users were not greatly troubled if status messages arrived at random times in different formats, and were mixed up with data, but programming a PC to cope with all these possibilities was a nightmare.

To overcome these shortcomings, host mode was introduced.

Host Mode

In host mode (Fig 6-3), the tnc is like a well-behaved child — it only speaks when spoken to! The PC is now in charge of proceedings, and commands and responses across the serial link are in simple, consistent formats which are easy to program. The PC only asks the tnc for information when it is ready to receive it.

This makes it much easier to display session status, switch between multiple data streams, and so on. Programming network services such as file transfer and bulletin boards is now straightforward, and more

flexible — it's much easier to change software in the PC than to change firmware in the tnc.

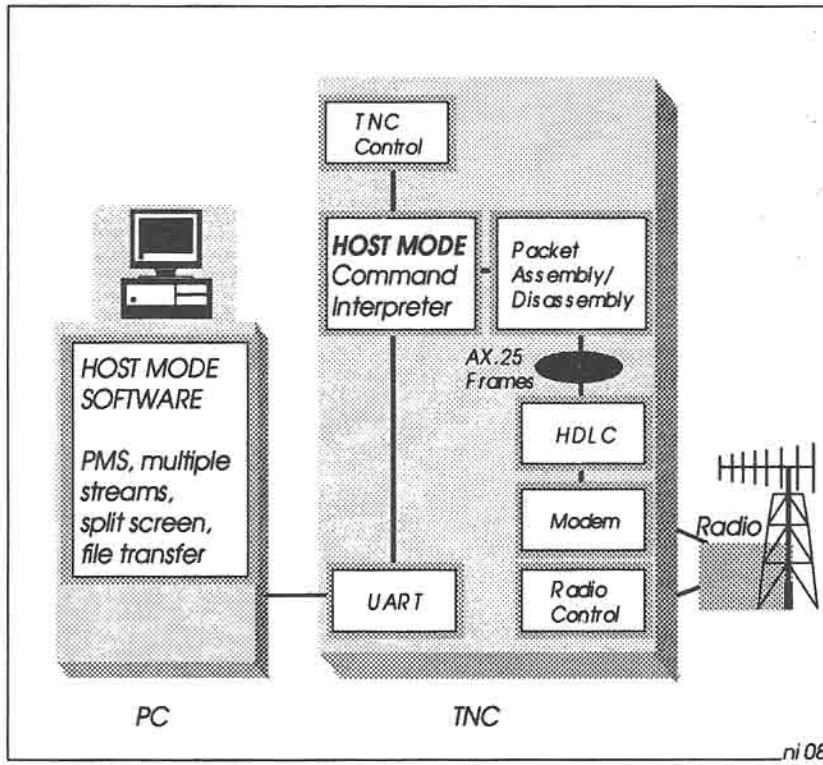


Fig 6-3: When the tnc operates in host mode, the PC is in control. The PC handles the higher level functions such as the Personal Messaging System, multiple streams and split-screen operation. The tnc still handles low-level packet assembly/ disassembly, however, and is still restricted to AX.25.

However, in host mode, most of the low-level packet handling still takes place within the tnc. This is fine for AX.25 virtual circuits, but not suitable for other protocols such as TCP/IP. What's really needed

is the capability of the PC to control the content of frames at the lowest level. This is what you get when the tnc operates in KISS mode.

KISS (Keep it Simple, Stupid!) Mode

When the tnc operates in KISS mode, almost all of the station's functionality takes place within the PC (Fig 6-4). The PC provides the high-level network services for file transfer, bulletin boards and so on, together with lower level protocol software which has access to every HDLC frame that enters and leaves the tnc.

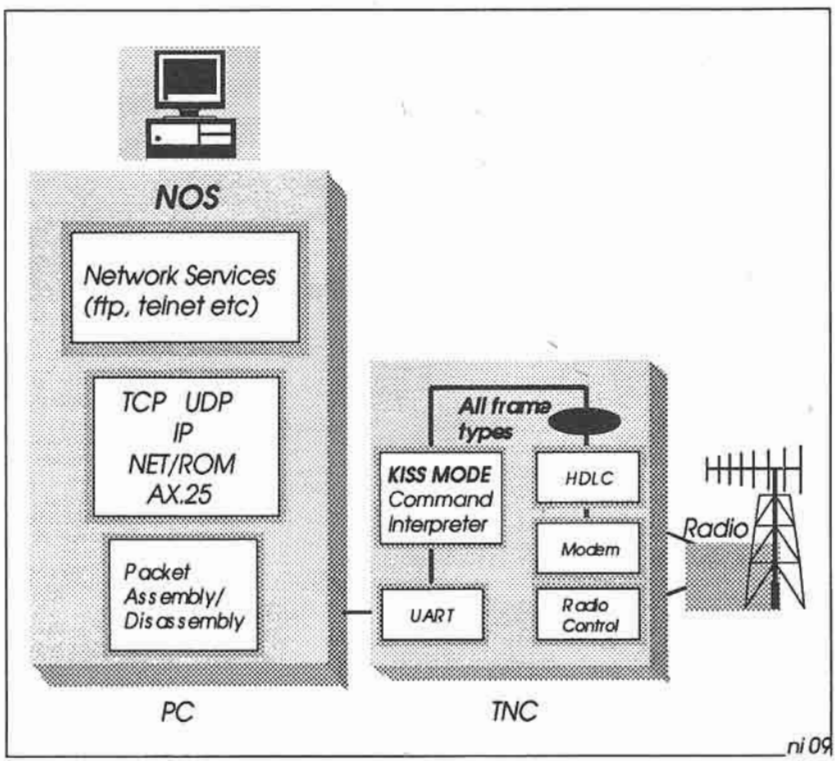


Fig 6-4: In KISS mode, the tnc handles all frame types. NOS in the PC contains a complete set of AX.25 software (replacing the tnc AX.25 firmware), together with support for all the AMPRnet protocols and NET/ROM.

This means that it's now possible to control exactly what goes into each individual HDLC frame, making it straightforward to multiplex several different protocols over the same radio link. The downside, of course, is that these protocols have to be implemented within the PC — this makes it necessary for NOS to contain a complete set of AX.25 software which completely replaces the AX.25 firmware in the tnc.

The KISS Protocol

To communicate between the PC and the tnc, the KISS protocol is used. This is a very simple asynchronous packet protocol, whose main purpose is to provide an envelope for HDLC frames (Fig 6-5). Each frame starts and finishes with a Frame End (FEND) character. There is no checksum or CRC.

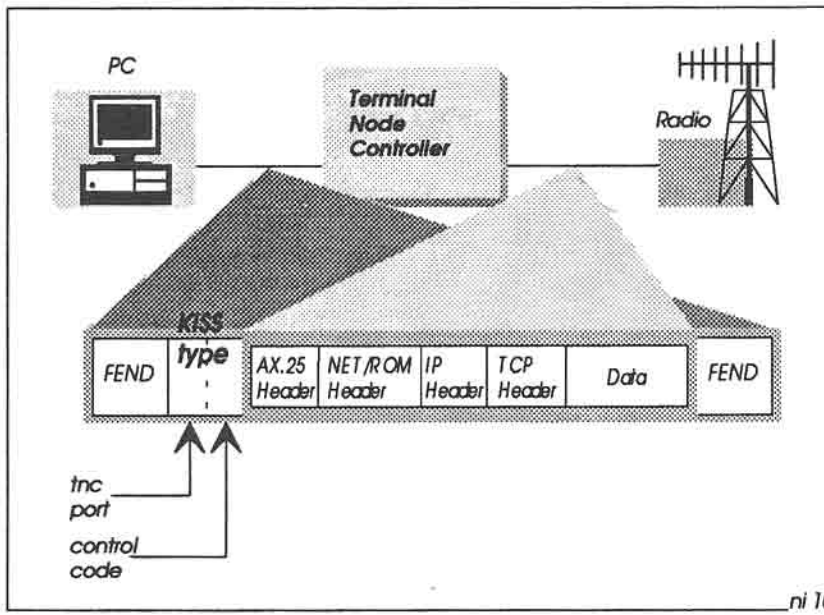


Fig 6-5: KISS Frame format. When a frame reaches the tnc, the *FEND* and *KISS type* bytes are removed, leaving the original packet for transmission.

Immediately following the leading FEND character is a KISS *type* byte. The low-order 4 bits of this byte contain a control code.

If the code is 0, this is a data frame, and the high-order 4 bits specify the tnc port number (0-15) for which the frame is applicable.

If the control code is non-zero, the frame contains a tnc setup command.

The KISS link is set to 8-bit data, one stop bit and no parity. If a frame happens to contain a data byte which looks like a Frame End character, the byte is replaced with a 2-byte Frame Escape/Transposed Frame End (FESC/TFEND) sequence. If a frame contains a FESC, this is replaced with a 2-byte Frame Escape/Transposed Frame Escape (FESC/TFESC) pair.

0	Data Frame
1	TX Delay (x 10mS)
2	Persistence (0-255)
3	Slottime delay (x 10mS)
4	TX Tail (x 10mS)
5	0=half duplex, 1=full duplex
6	Hardware dependent
7	TX mute
8	0=DTR low, 1=DTR high
9	0=RTS low, 1=RTS high
10	Baudrate
11	End delay
12	Group
13	Idle
14	Min
15	Max key
16	Wait
17	Parity: 0=none, 1=even, 2=odd
129	Down
130	Up
254	Prepare to switch tnc from KISS to native mode
255	Switch tnc from KISS to native mode

Table 6-1: KISS Control Codes (expressed in decimal). Most of these codes are tnc-specific. Only codes 0–3, 5 and 255 are understood by all tncs.

When a KISS frame arrives at the tnc, the FEND characters and KISS type byte are stripped off, and any escaped characters are replaced with their original values. Then, if the frame is a data frame, it is passed to the HDLC controller chip for transmission.

If the frame is a tnc control frame, the tnc executes the command specified in the KISS type byte. Some of the commands require additional parameters, which are included in the rest of the frame. The actual command codes and their functions are tnc-dependent.

A more-or-less complete list of known codes is shown in Table 6-1 opposite. All tncs understand codes 0-3, 5 and 255, whereas the remaining codes are mostly for experimental use.

For a more detailed description of KISS mode, see the paper by Mike Chepponis and Phil Karn (details in Appendix 6).

Switching the TNC to KISS mode

When you power up a tnc, it will normally start in native mode. To make the tnc ready for NOS, you first need to initialise it with your AX.25 callsign, and you also need to set up the CW ID interval for Morse code station identification, if local licence regulations require it.

For example:

```
cmd: MYCALL NS9BOB-5
cmd: MID 84
```

Then, to switch to KISS mode, the command:

```
cmd: KISS ON
```

is probably all that's necessary — see Fig 6-6 on the next page.

Once the tnc is in KISS mode, it won't understand any more native mode commands. From now on, all communication with the tnc uses the KISS protocol.

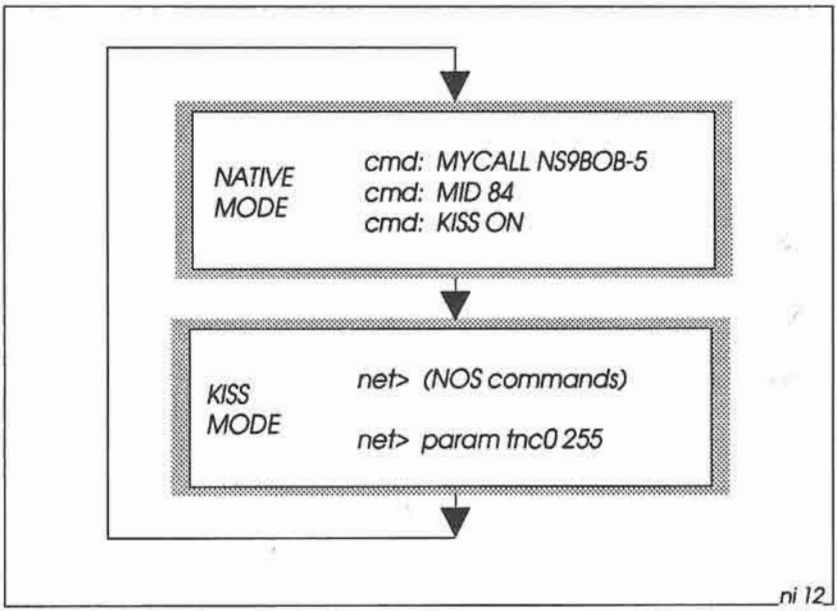


Fig 6-6: Switching the tnc between native mode and KISS mode.

However, some older tncs may require a sequence of commands to switch from native to KISS mode; e.g.

```
cmd: AWLEN 8
cmd: CONMODE TRANS
cmd: HID OFF
cmd: HPOLL OFF
cmd: KISS ON
cmd: PARITY 0
cmd: PPERSIST ON
cmd: RAWHDLC ON
cmd: START $00
cmd: STOP $00
cmd: TRACE OFF
cmd: XON $00
cmd: XOFF $00
cmd: XFLOW OFF
cmd: HOST ON
```

Switching Back to Native Mode

If you want to switch your tnc out of KISS mode back to native mode, you need to send a KISS command frame with command code 255. This is achieved in NOS with the **param** command:

```
net>  param tnc0 255
```

where tnc0 is the name of the interface to the tnc. Some tncs may also require the **param tnc0 254** command as well:

```
net>  param tnc0 254
net>  param tnc0 255
```

7: A PEEK AT PROTOCOLS

In this chapter we take a fresh look at some of the protocols which you are probably already using, such as AX.25 and NET/ROM, and then explain how TCP/IP and the AMPRnet ride on top of them.

Protocol Stacks

Present-day networking design usually follows the OSI model, the so-called 7-layer protocol stack (the left half of Fig 7-1). This isn't the place to go into detail on the functions of each layer in the stack; suffice to say here that it's convenient to break down the stack into two parts. The lower part contains the Physical, Data Link and Network layers, and the upper part contains the remaining four layers.

The essential difference between the two sets of layers is that the lower layers are basically network-dependent, whereas the upper layers are virtually independent of the underlying network.

However, the TCP/IP world which NOS supports was well established long before the OSI model became accepted. The TCP/IP stack is shown in the right half of Fig 7-1, and from this we can see the approximate correspondence between the two models.

The reason for showing both of these stacks here is that some of the protocols which NOS supports (such as KISS, AX.25 and NET/ROM) best fit the OSI model, whereas the Internet protocols best fit the TCP/IP model. Predictably, this mixture of old and new protocols has caused many headaches for the software developers who squeezed them all into one package. To the purist the result is a mess, but to the pragmatist it works, and that's what counts!

So let's merge the protocols which NOS supports into one diagram (Fig 7-2). It looks pretty complicated now, but if we take it piece by piece it won't be too painful. Let's start at the bottom.

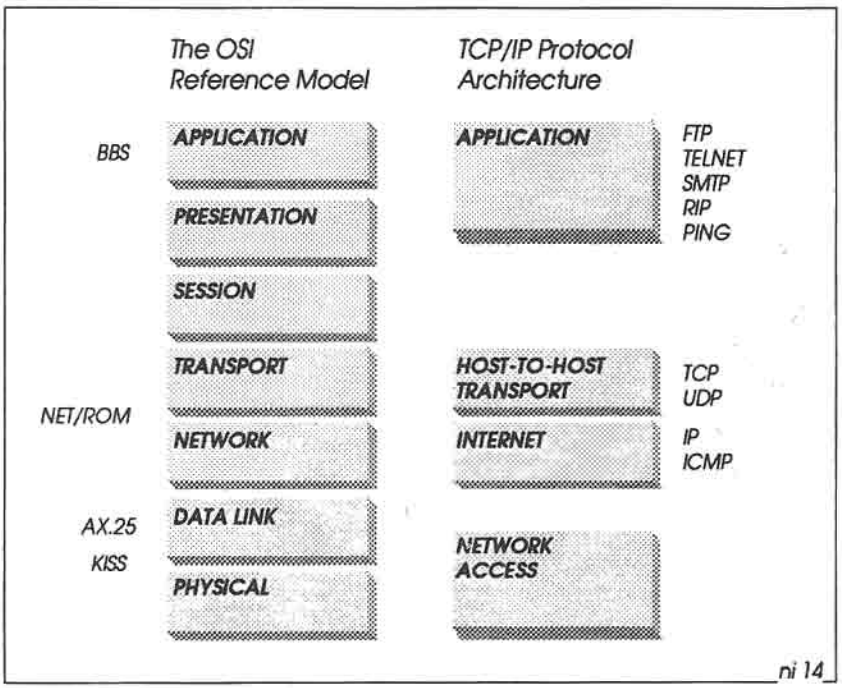


Fig 7-1: The OSI and TCP/IP Protocol Stacks

The Physical Layer

The physical layer is concerned with the physical connections to the network. NOS provides support for three main types of connection:

- modems
- packet radio terminal node controllers
- local area network adapters

The connection between the PC and a modem can be a simple 3-wire cable (TD, RD and ground), but is much more likely to include all the modem control lines, such as RTS, CTS and so on.

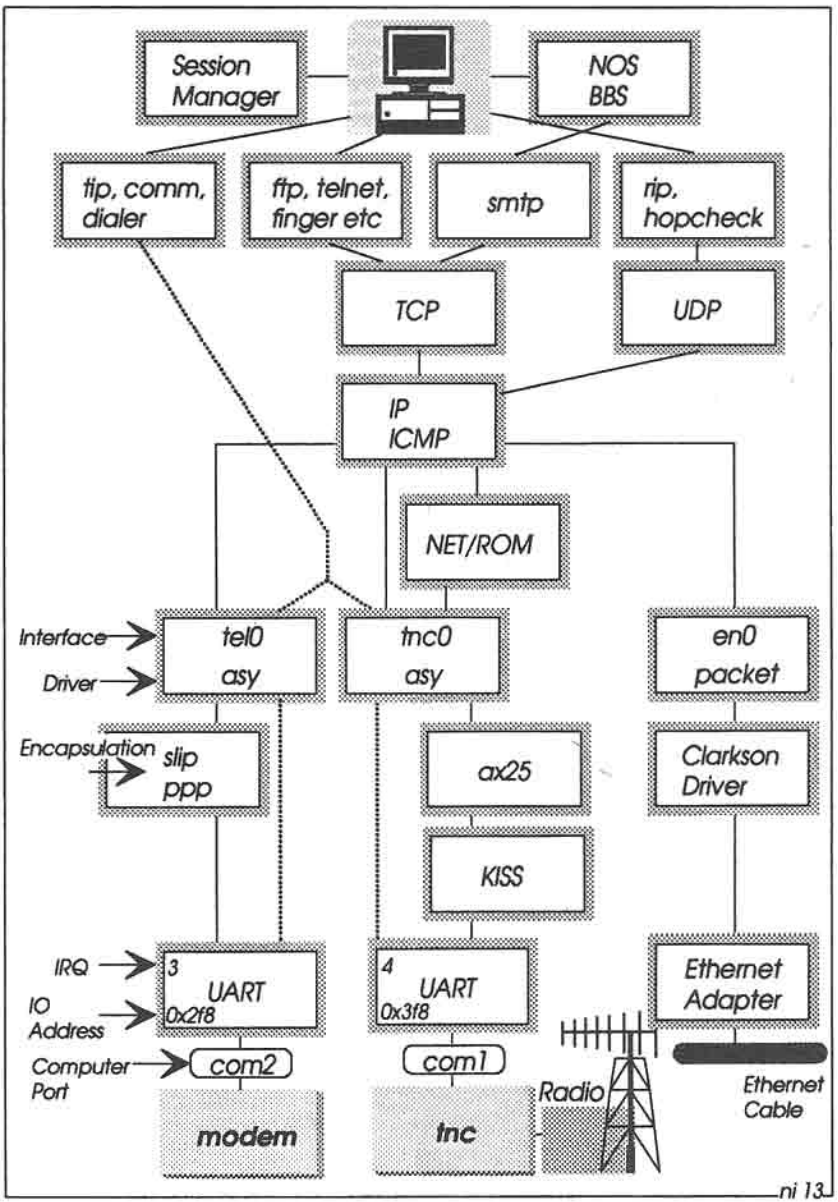


Fig 7-2: NOS Protocols

The connection between the PC and a tnc can also be a 3-wire cable, as the KISS protocol which NOS uses when running TCP/IP does not support flow control. However, some versions of NOS do support hardware flow control, so it would make sense to have a cable with all the modem control lines present anyway.

The LAN adapter can be any of a wide range of commonly available Ethernet or token ring adapters. We will see shortly that NOS allows you to install one or more of the public domain Clarkson drivers which support these adapters.

NOS Drivers and Interfaces

For each of the I/O controllers and the uarts in your system you will need to define a DOS I/O address and an IRQ vector number. For example, for the COM1 port, the uart uses I/O address 0x3f8 and IRQ 4. These numbers appear as parameters in NOS **attach** commands which run when you start NOS. For example:

```
attach asy 0x3f8 4 ax25 tnc0 2048 256 4800
```

We'll look in detail later at what all the parameters mean, but two of them are relevant here. The parameter **asy** is the name of the NOS asynchronous driver, and the parameter tnc0 is the NOS interface name for the driver. NOS driver names like **asy** are fixed, but you can choose any meaningful names for the interfaces. (In this book we use the name tnc0 for the tnc interface, but other documentation which you may have seen uses the names ax0 or pk0 instead).

Interface names are used in many commands. For example, if you want to trace packets passing through the tnc, you can give the command **trace tnc0 211**, or if you want to chat with AX9SAM using AX.25 you can give the command **connect tnc0 AX9SAM**. Similarly, if you use the interface name tel0 for the modem port, you can run a dialer script with a command like **dialer tel0 /scripts/dialbob.scr**.

Data Link Layer

The Data Link layer is concerned with protocols which encapsulate packets into frames in readiness for transmitting them (and also, of

course, for decapsulating received packets). When sending and receiving data over a telephone line, the most common protocol is SLIP, but the PPP point-to-point protocol is nowadays gaining in popularity.

For the packet radio network, you will almost certainly be using the AX.25 protocol at this level, with the AX.25 packets being enveloped in KISS frames for the tnc. The AX.25 driver is fully compatible with the AX.25 Level 2 Version 2 specification, so you can use it not only for NOS but also for ordinary AX.25 connections if you want to.

An added feature of NOS is the ability to talk directly to a serial port, without encapsulating the data in any way. This is useful if you want to send commands to the tnc in native *cmd:* mode, or to set up or interrogate a modem.

NOS provides three commands to do this:

- **tip** allows interactive access to a serial port via the keyboard and screen;
- **comm** lets you send previously prepared strings from a file to a serial port;
- **dialer** lets you control a serial port with scripts containing modem control commands, time delays and tests for expected responses.

For LAN adapters you can use external drivers compatible with the FTP Inc packet interface. As the drivers are not part of NOS, you load them into memory before starting NOS. You can get suitable drivers from the FT/TCP package available from FTP Inc, or more probably you will use one of the public domain Clarkson drivers. Appendix 1 gives the details.

The "Workhorse" Protocols

Working our way up Fig 7-2, we now come to the "workhorse" protocols which do the hard work of transferring information between systems. These include TCP, IP, UDP, ICMP, NET/ROM and others. As already mentioned in Chapter 4, think of these protocols as the "engine" of the NOS system. We are not particularly interested at the moment in exactly what they do or how they work; all we need to know now is where they fit into the overall picture.

The Network Services

The services shown at the top of Fig 7-2 (**tip, ftp, telnet** etc) form the interface between NOS and the user. You can give commands like **ftp ns9bob** from the keyboard and see responses on the screen, or you can include the commands in script files if you want to run them repeatedly (in a similar way to **.BAT* files in DOS). You can also give many of the commands from your built-in NOS BBS.

The Session Manager

The Session Manager is the part of NOS which pulls everything together. You give commands to the Session Manager, which then either executes them immediately, or starts new sessions to handle them. With the Session Manager you can monitor what is happening within NOS at any time, start and stop network services, abort data transfers, run command scripts, trace network packets, change interface configurations and so on.

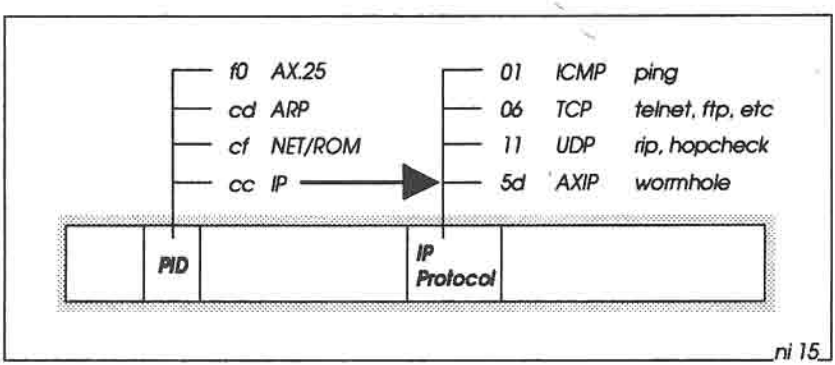


Fig 7-3: The Protocol ID (PID) code in the AX.25 frame specifies the type of packet. When the PID = cc (hex), it is an IP packet, and the IP Protocol code then specifies the higher level protocol. (All codes shown here in hexadecimal).

8: NAMES, DOMAINS AND ADDRESSES

Before you can communicate using TCP/IP, you need an Internet hostname and address. Your hostname will usually be your callsign, followed by the domain name for packet radio: *.ampr.org* (e.g. *ns9bob.ampr.org*). A domain is a logical area in the Internet network; *ampr* means "amateur packet radio" and *org* means "organisation."

To get an Internet address, you need to contact your local address coordinator (see Appendix 5 for a list of coordinators). The IP address will be 32 bits long, and, like your radio callsign, it will be unique — nobody else in the world will have the same address. The address is almost always written in dotted-decimal notation; e.g. 44.199.41.1.

The *domain.txt* File

Having received your IP address, you now edit it into the NOS name-and-address file, *domain.txt*. Appendix 3 contains an example of a typical *domain.txt* file (see pages 318-319). Fig 8-1 on the next page shows a short extract.

Each entry (known as a "resource record") needs one line, and the fields are separated by any combination of tabs or spaces. You can include comments, prefixing them with the # character.

Special Addresses

The first few resource records in *domain.txt* specify some special addresses for network broadcasts and packet routing. Later chapters describe how to use these.

The loopback address is a dummy IP address: 127.0.0.1. When you send anything to this address, you are really sending it to yourself! For

example, you can transfer a file to yourself with the command **ftp loopback**, or log into your own NOS BBS with **telnet loopback**. You won't use loopback very often once you are familiar with NOS, but it's extremely useful for testing off-air — this way you can learn a lot about NOS and make as many mistakes as you like, without treading on your neighbours' toes. Incidentally, some documentation uses the word *localhost* instead of loopback; this means exactly the same thing.

```
# ------------------------------------
# SPECIAL ADDRESSES
# ------------------------------------
ampr.ampr.org.          IN A      44.0.0.0
nosland.ampr.org.       IN A      44.199.0.0

region41.ampr.org.      IN A      44.199.41.0
region45.ampr.org.      IN A      44.199.45.0

loopback.ampr.org.      IN A      127.0.0.1
# ------------------------------------
# RADIO REGION 41
# ------------------------------------
ns9bob.ampr.org.        IN A      44.199.41.1

ns9ken.ampr.org.        IN A      44.199.41.2
ken.ampr.org.           IN CNAME  ns9ken.ampr.org.

ns9pam.ampr.org.        IN A      44.199.41.3

ns9zzz.ampr.org.        IN MX 0   ns9ken.ampr.org.

                        IN NS     ns9dns.ampr.org.
ns9dns.ampr.org.        IN A      44.199.41.99
```

Fig 8-1: The file *domain.txt* relates symbolic hostnames to Internet addresses.

Ordinary Internet Address Records

You need an entry in *domain.txt* for your own station. For example:

```
ns9bob.ampr.org.        IN A      44.199.41.1
```

Note that the callsign/domain string ends in a dot (after the letters *org*).

The letters IN A signify that this is an Internet Address entry; we'll see later that there are several other types of entry as well.

You'll also need an entry in *domain.txt* for every other station you want to communicate with using TCP/IP. There may be only a few entries if you just want to talk to the locals, or there may be hundreds of entries if you want to venture further afield. Your IP address coordinator should be able to provide you with an up-to-date file containing the details you need.

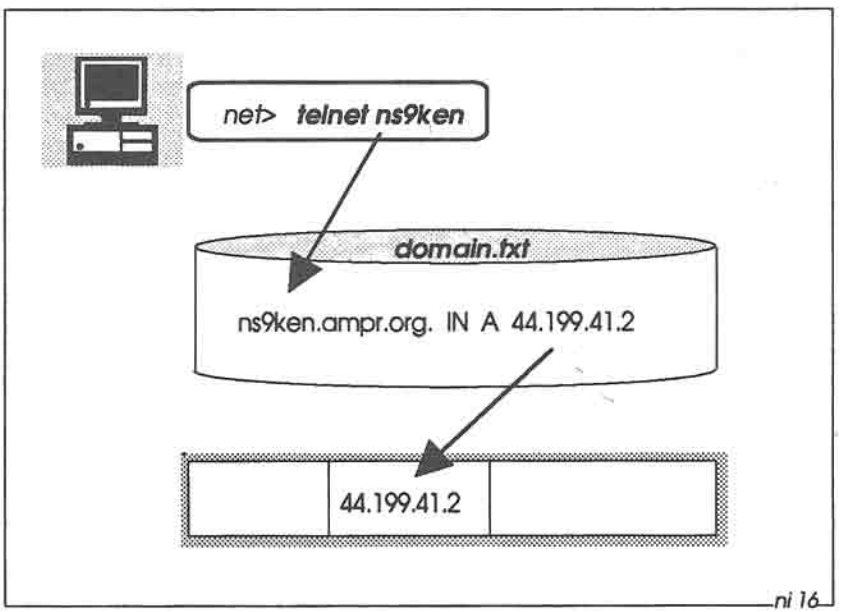


Fig 8-2: NOS uses *domain.txt* when translating symbolic hostnames to numeric IP addresses for insertion in packets.

The file *domain.txt* is not just restricted to containing entries for AMPRnet stations. You can also include entries for other domains if you have access to other networks.

For example, your NOS system may be connected to your company's Ethernet LAN, so you will have entries like this:

```
alpha.acme.com.        IN A     192.93.94.95
beta.acme.com.         IN A     192.93.94.96
```

Here the domain name is acme.com, where com is short for "company."

Default Domain Suffix

By default, you have to include the full name of a station when giving a command; e.g. to transfer a file to Pam, the full command is **ftp ns9pam.ampr.org**. However, using full domain names like this gets tedious after a (very short) while, and so to make life easier you can define a new default, by putting this command in the NOS startup file *autoexec.nos*:

```
domain suffix ampr.org
```

Thereafter, all you need to say is **ftp ns9pam** — NOS then automatically adds the suffix *.ampr.org* to ns9pam to get the full hostname.

However, you can still use the full name if you wish, so if you want to send a file to host alpha on the LAN, you can say **ftp alpha.acme.com**. In this case, because you have given the full name, NOS ignores the default domain suffix.

Canonical Name (Nickname) Records

For stations that you talk to regularly, it may be convenient to add a nickname entry to *domain.txt*. This is termed a CNAME (Canonical NAME) resource record, which refers back to an existing IN A record:

```
ken.ampr.org.          IN CNAME ns9ken.ampr.org.
```

Then you can talk to NS9KEN with a command like **ftp ken**.

Mail Exchanger Records

Another type of entry in *domain.txt* is the MX (Mail EXchanger) record. For example:

```
ns9zzz.ampr.org.      IN MX 0   ns9ken.ampr.org.
```

This states that mail addressed to ns9zzz is to be sent to ns9ken, who will act as a "Mail Exchanger" and forward it on to ns9zzz. The digit 0 after IN MX is called the *preference value* of the exchanger. You can have several mail exchangers for a given destination, each with a different preference value, and NOS will attempt to forward to the exchanger with lowest value. If that fails, it will try again with the next lowest value, and so on. This is covered in more detail in the chapter on SMTP mail forwarding.

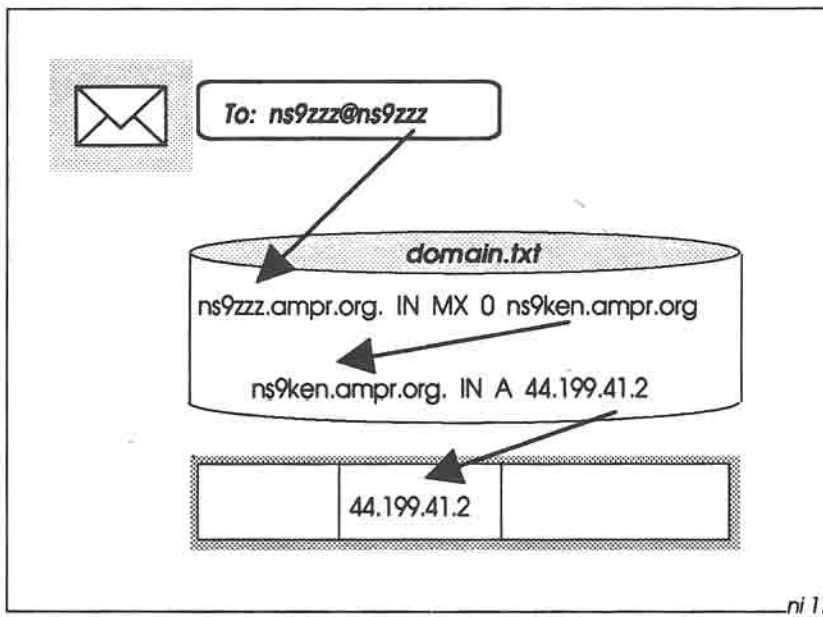


Fig 8-3: The Mail Exchange (MX) record specifies a mailhost which knows how to forward mail for a particular station.

Name Server Records

There are of thousands of people throughout the world with AMPRnet IP addresses, and hundreds of thousands of people with IP addresses for other networks. It's obviously unrealistic to put all of these addresses in *domain.txt* (and even if you did, it would be an impossible task to keep them up-to-date manually), and so it makes sense to put these addresses on special machines called Domain Name System (DNS) servers; see Fig 8-4 opposite.

Name servers are usually large, fast machines which communicate with each other to keep master copies of resource records up-to-date, and which allow ordinary users to interrogate them for particular records of interest.

Some versions on NOS incorporate a DNS server. If you have a server in your area, you can refer to it by adding an NS (Name Server) record to your *domain.txt*. For example:

```
                    IN NS     ns9dns.ampr.org.
```

The NS record specifies the name of a Name Server machine. That machine will need an ordinary IN A entry as well:

```
ns9dns.ampr.org.      IN A      44.199.41.99
```

Now, when you attempt to communicate with any station, NOS will first look in *domain.txt* for the IP address of that station. If it isn't there, NOS will then automatically send a request to the name server in an attempt to get the address from there.

Optimising *domain.txt*

If you don't have access to a name server, your *domain.txt* file may be very long. As NOS needs to read this file whenever you want to contact a station, it makes sense to optimise its layout to minimise access time — if the file has hundreds of records, it may take several seconds to find a record if the machine or disk are slow.

The most obvious way of reducing access time is to put all the entries for local stations at the front of the file, as you're more likely to talk to them than to other stations further afield.

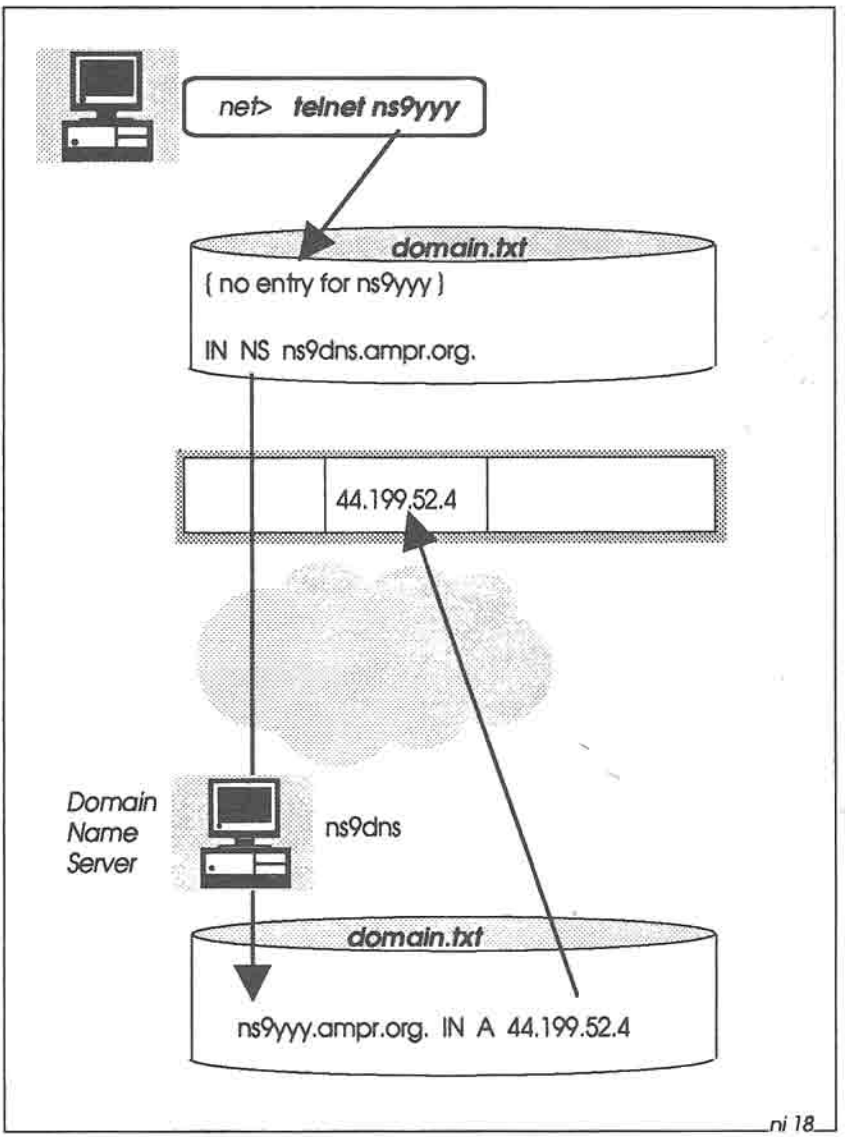


Fig 8-4: If there is no entry for a wanted hostname in *domain.txt*, NOS can make a request across the network to a Domain Name Server to get the required IP address.

In addition, you should remove all extraneous comments and whitespace (tabs and spaces) from the file. This will markedly improve system response if you have a long descriptive comments for each resource record.

Some versions of NOS also support another technique for reducing the length of *domain.txt*. You can include an origin statement, specifying the default domain name for entries in the file, and then you don't need to include the domain name in individual resource records.

For example:

```
$origin ampr.org.

ns9bob   IN A      44.199.41.1

ns9ken   IN A      44.199.41.2
ken      IN CNAME  ns9ken

ns9pam   IN A      44.199.41.3

ns9zzz   IN MX 0   ns9ken

         IN NS     ns9dns
ns9dns   IN A      44.199.41.99

alpha.acme.com.        IN A      192.93.94.95
```

This makes the file a lot shorter (and much easier to prepare!).

9: CLIENT / SERVER

Fundamental to the operation of NOS is the concept of "client/server" organisation. This chapter covers the basics of client/server, explaining why it is so powerful and flexible in networked environments.

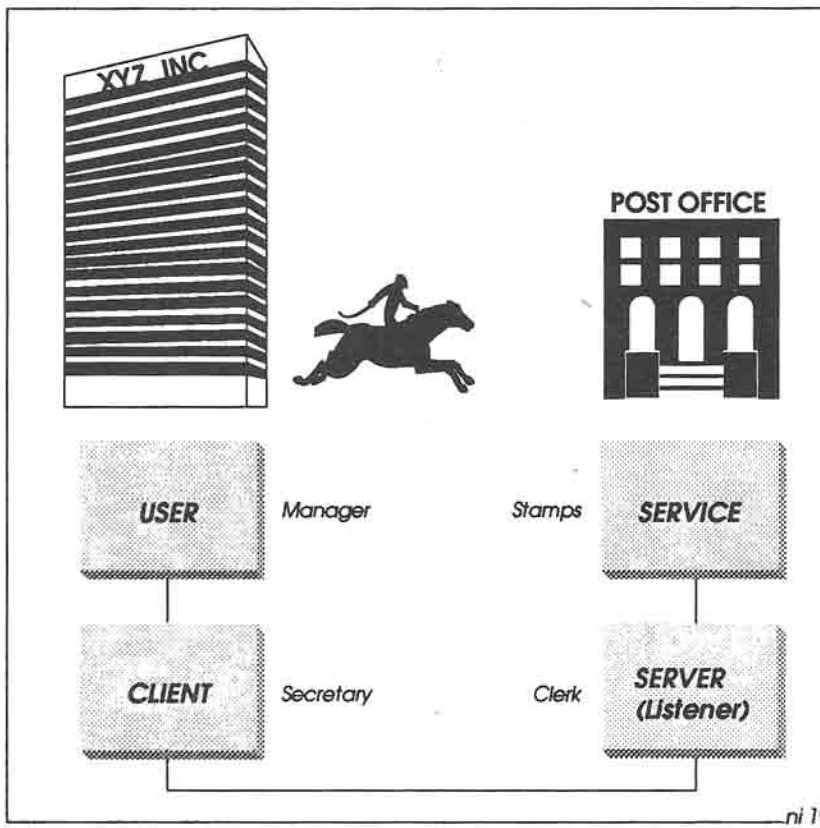


Fig 9-1: Client/Server Organisation.

It's convenient to use a post office analogy to show how client/server works (Fig 9-1). Let's say that a manager in the offices of XYZ Inc requires some postage stamps. He asks his secretary to go to the post office to buy them. To do this, the secretary leaves the building, goes down the street, enters the post office, and stands in line waiting to be served at a counter.

Eventually, the secretary reaches the front of the line and asks the counter clerk for the stamps. The clerk hands over the stamps. Then the secretary leaves the post office and returns to XYZ Inc, arriving back in the manager's office to hand over the stamps.

A simple enough transaction. Now let's translate this into networking jargon. The manager is a *user*, who issues an instruction to a *client* (the secretary) to request a *server* (the counter clerk) to provide a *service* (issue postage stamps). The client (secretary) then submits the result of the service (the stamps) back to the user (the manager). That's basically all there is to "client/server".

[As an aside, you'll see the words *listener* or *daemon* scattered throughout NOS documentation. These are alternatives for the word *server*].

The *telnet* Client

In the world of TCP/IP, there are several types of client (just as in the office there are several types of secretary), used for requesting different services. One of the most important clients is *telnet* (Fig 9-2). When you give a command such as **telnet ns9ken**, you are asking your telnet client to make a connection with the telnet server at NS9KEN.

When the connection is made, the telnet server then sets up a logical path between the client and the NOS BBS service. The BBS sends back a login prompt to the client, which displays it on your screen. You then log in, and you're away. The connection remains in place until you give the **B** command to the BBS, which then disconnects itself from the client.

The numbers 1024 and 23 in Fig 9-2 are called *port numbers*. Packets passing between client and server contain these port numbers, so that each end knows which service and which user session they relate to. The number 1024 is an arbitrary port number allocated by the client, and 23 is the port number for the telnet service.

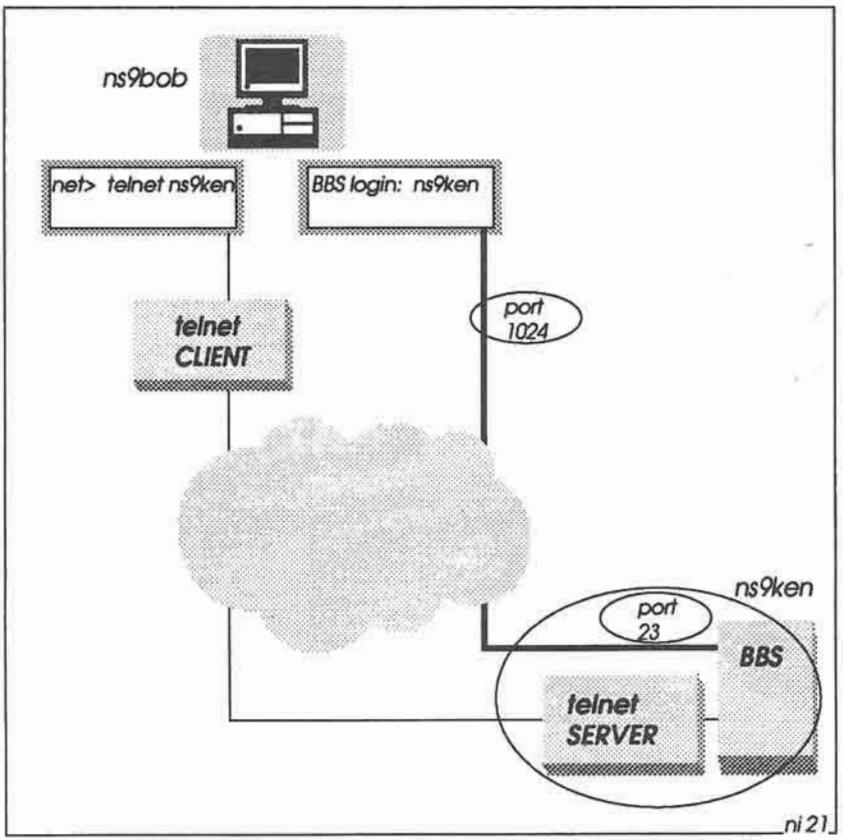


Fig 9-2: When NS9BOB gives the command *telnet ns9ken*, his telnet client connects to the telnet server at NS9KEN. The server then starts the BBS, which sends a login request back to NS9BOB.

Multiple Sessions

You are not restricted to just one telnet session. In principle, you could start many sessions, accessing the BBSs of several stations at the same time. Fig 9-3 shows two such session, with NS9KEN and NS9LIZ.

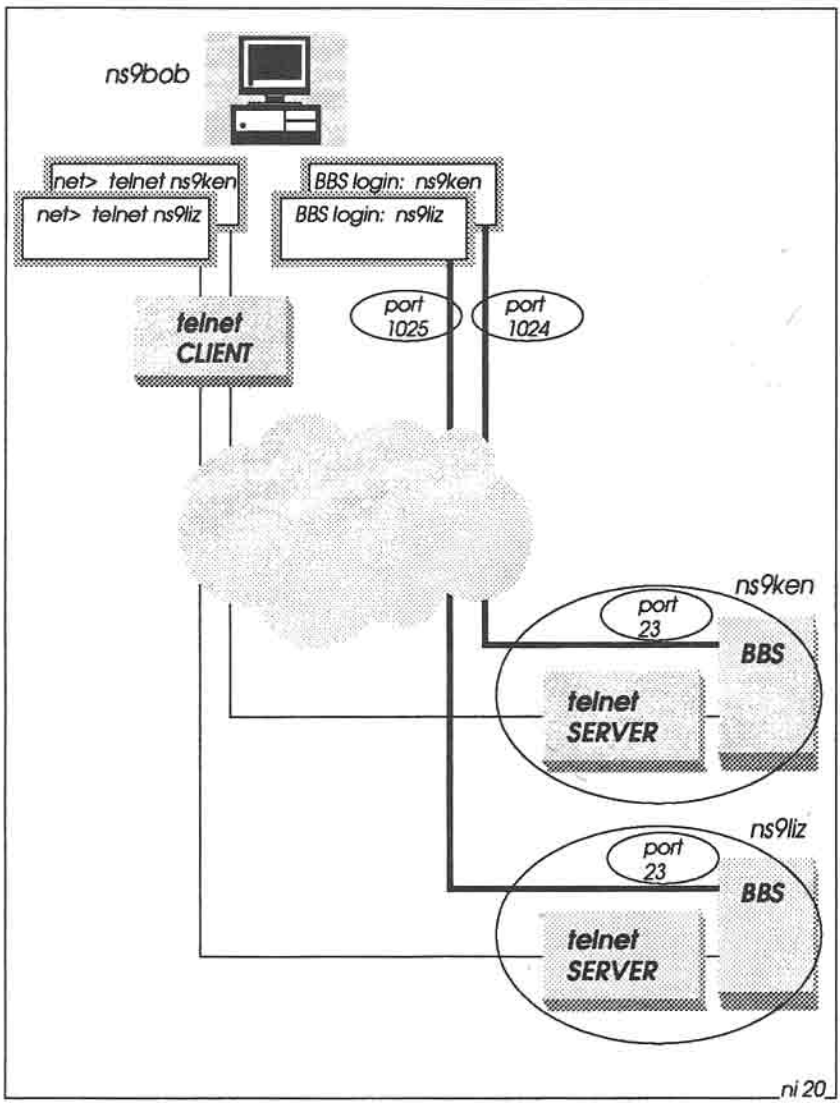


Fig 9-3: Multiple TELNET sessions.

A further concept associated with client/server is the *socket*. A socket is basically an end point for communication, and is expressed as a combination of hostname:port number. For example, in Fig 9-3, socket

ns9bob:1024 is connected to socket ns9ken:23, and socket ns9bob:1025 is connected to socket ns9liz:23. These socket numbers appear in several NOS status messages, making it possible uniquely to identify individual sessions.

Behind the *telnet* Server

It's important to realise that when your telnet client connects with a telnet server on another machine, you may not always be greeted by a NOS BBS. See Fig 9-4. If the server is running on a UNIX machine, the telnet server will connect your client to a UNIX login prompt instead, which will then take you into a UNIX shell. If the telnet server is running on a DEC VAX machine, the server will connect you to a VMS login sequence instead.

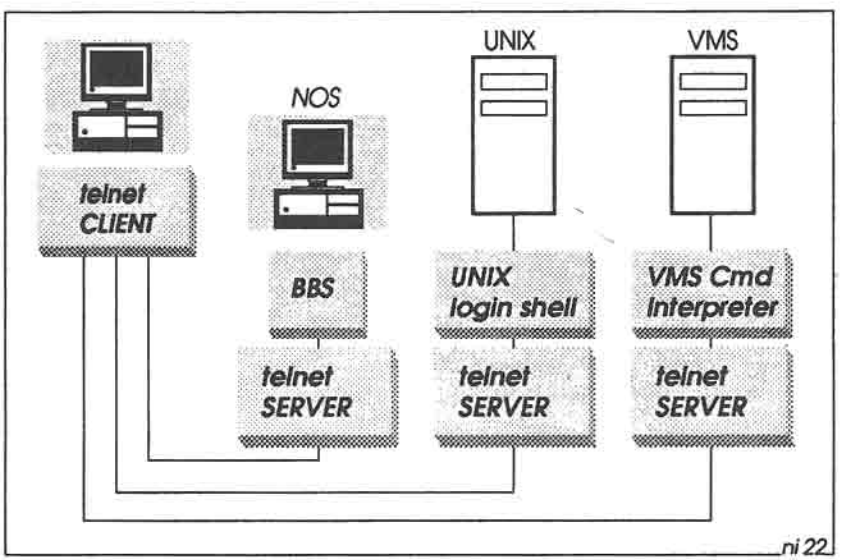


Fig 9-4: The NOS *telnet* client can communicate with *telnet* servers on all kinds of systems.

In other words, what lies behind the TELNET server depends very much on the machine it is running on and the network services it supports. In NOS you have the potential to login to *any* machine of any type which is running a telnet server — almost every well-known

machine supports telnet these days — giving you the opportunity to connect to all manner of network services.

Different Services

Let's take the post office analogy a little further. When the secretary talks to the the clerk, it doesn't always have to be a request for stamps. It could be a request to mail a parcel, or to ask for a sheet of airmail stickers, or whatever.

That is, the server can handle several kinds of related request from the client. In the telnet world, the user can ask for a different service by adding a service number to the telnet command; for example, **telnet ns9ken 21**. The number 21 is called a "well-known" port number. The default well-known port number for the telnet server is 23 (that is, **telnet ns9ken 23** means the same as **telnet ns9ken**).

There are around 370 assigned well-known port numbers. Here are some of them:

7	echo
9	discard
20	ftp-data
21	ftp-control
23	telnet
25	smtp
67	bootp
69	tftp
79	finger
87	ttylink (chat)
109	pop2
110	pop3
119	nntp
513	rlogin

Having to add a port number to a **telnet** command is a bit of a chore, and so to make things easier, NOS provides more meaningful client names for the more frequently used services. For example, you can say **ftp ns9ken** instead of **telnet ns9ken 21**, or **finger ns9ken** instead of **telnet ns9ken 79**, and so on.

Note that some versions of NOS don't have a **ttylink** (or **chat**) client, so in this case you'll need to include the well-known port number 87 in **telnet** commands to chat to those versions; e.g. **telnet ns9ken 87** — see Fig 9-5.

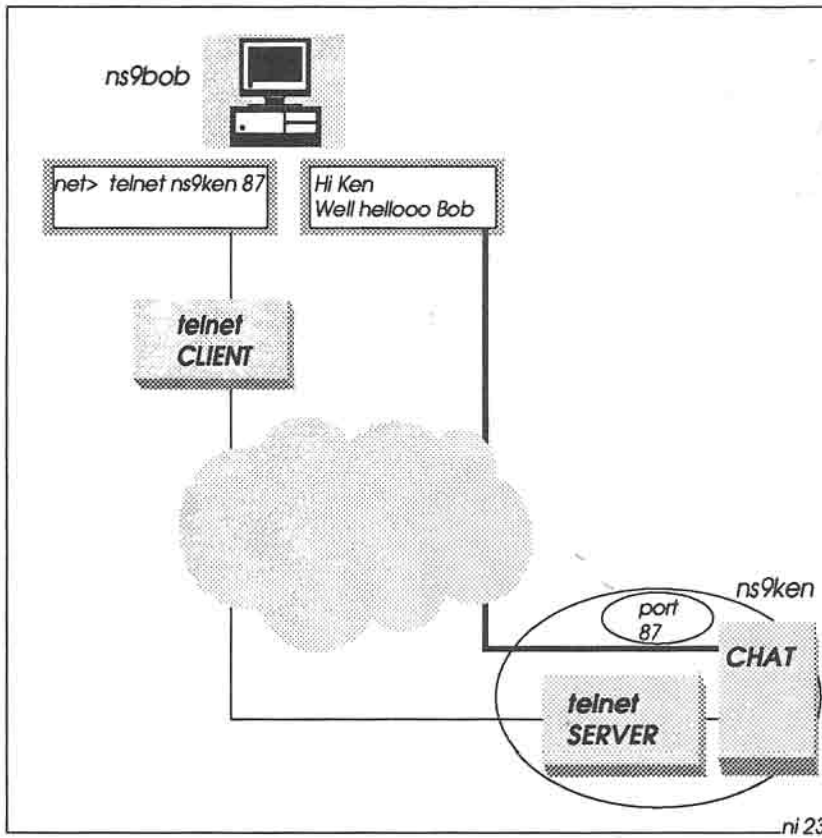


Fig 9-5: It is possible to access several different types of server by adding a port number to the *telnet* command. For example, to chat to NS9KEN, the command will be *telnet ns9ken 87*.

Talking to Yourself

A further feature of client/server architecture is that you are not restricted to having *only* clients on some machines and *only* servers on others. Any machine can have both clients *and* servers. This gives you the opportunity to talk to yourself in exactly the same way as you would talk to somebody else over the network — very useful for testing and familiarising yourself with NOS.

Thus if you are running NS9BOB, you could log into your own NOS BBS with the command **telnet ns9bob** (Fig 9-6). In this case, the telnet client in your machine connects directly to the telnet server in your machine. Now you can drive your own BBS in exactly the same way as if you had logged into somebody else's NOS BBS. In fact, logging into your own BBS is such a common requirement that NOS has a special command for this: **bbs**.

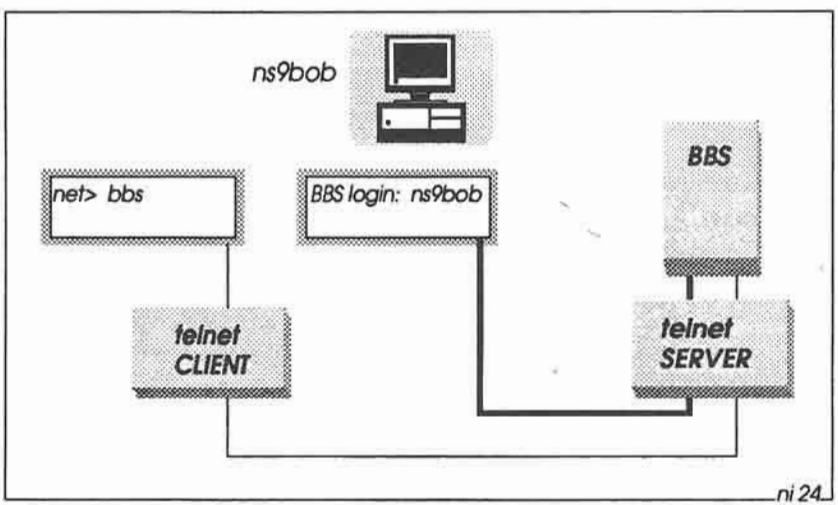


Fig 9-6: NS9BOB can log into his own NOS BBS with any of three different commands: *bbs*, *telnet loopback*, or *telnet ns9bob*.

10: HANDS ON — HARDWARE CHECKOUT

Now that we've seen something of the background to TCP/IP and NOS, it's time to get down to business. This chapter and the next one explain how to set up the hardware and install the software.

To run most versions of NOS in a packet radio environment, you need the following hardware (see Fig 10-1):

- A PC
- A tnc (terminal node controller)
- An RS-232 cable connecting the PC to the tnc
- A radio transceiver

The PC

The PC may be any IBM-compatible 80x86 machine, running DOS or similar. NOS needs as much memory as you can give it, and it's unlikely that it will run with less than 640K of RAM. A hard disk is obviously preferable, although NOS will run (slowly) on a dual-floppy laptop as well.

The TNC

Most tncs have an internal battery which saves certain setup parameters in battery-backed RAM. This can be a nuisance at times, especially if you get the setup wrong and accidentally get the tnc into a locked up state. The only remedy is to take the cover off of the box, disconnnect the battery, re-connect it again a few seconds later, and then replace the cover.

It's highly likely that you'll get the tnc settings wrong when first setting up NOS, and so it's recommended that you fit a single-pole on/off switch in the battery circuit. Then if you lock up the tnc, it's a simple

matter to turn the switch off and on again to make the tnc usable again, without the hassle of removing and replacing the cover. (In fact, you can probably operate the tnc without a battery at all, as you will configure NOS to initialise the tnc to KISS mode each time you start NOS, and reset it back to native AX.25 mode when you are done).

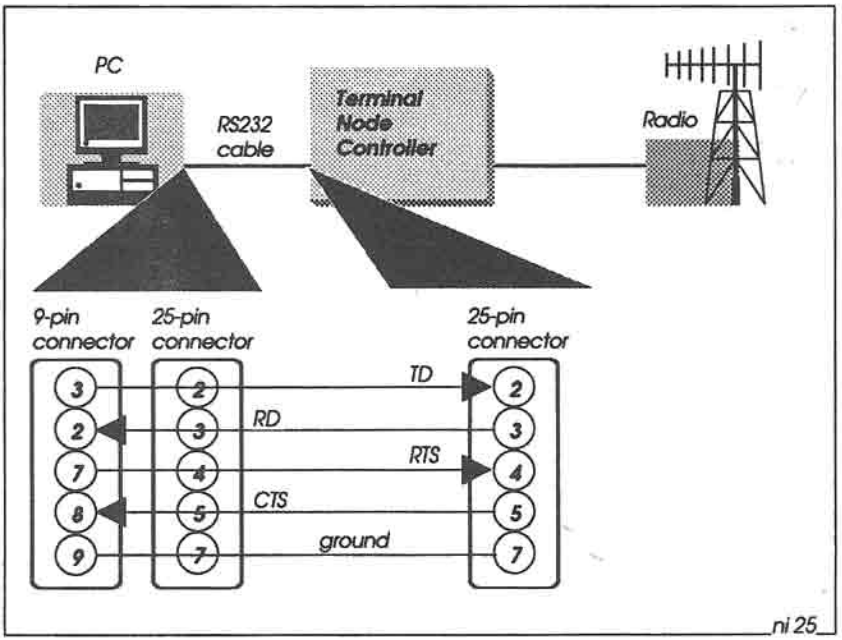


Fig 10-1: The RS-232 cable connects the PC to the TNC.

The RS-232 Cable

The RS-232 cable connecting the PC to the tnc should be a 5-wire cable, wired for hardware (RTS/CTS) flow control; again see Fig 10-1. Some tncs with older firmware do not support RTS/CTS in KISS mode, in which case a simple 3-wire cable (TD/RD/GROUND) will suffice.

In any case it's probably preferable to use the 5-wire cable to avoid possible lockup problems when transferring non-printing ASCII

characters in the tnc's native mode — XOFF (**CTRL-S**) may stop data transfer if software flow control is used.

When hardware flow control is working properly, the following conditions apply:

```
RTS HIGH:  PC tells the tnc it can start/resume sending data
RTS LOW:   PC tells the tnc it must stop sending data
CTS HIGH:  tnc tells the PC it can start/resume sending data
CTS LOW:   the tnc tells the PC it must stop sending data
```

Native Mode Checkout

Before using NOS, it's advisable to check out the tnc in native AX.25 mode with hardware flow control. Set the speed of the PC-to-tnc link to a reasonably high value (e.g. 4800 bps), with an 8-bit data path, and disable software flow control:

```
cmd:  AWLEN  8
cmd:  START  $00
cmd:  STOP   $00
cmd:  TBAUD  4800
cmd:  XON    $00
cmd:  XOFF   $00
cmd:  XFLOW  OFF
```

Connect to your local AX.25 PBBS as usual and read a few long messages. Also send yourself a long message and read it back. Make sure that nothing is missing from the text on the screen. If some characters are missing, or if the keyboard or screen lock up, it's probable that the hardware flow control is not working.

Make it work properly before proceeding. If flow control does not work in AX.25 mode, it's unlikely that NOS will work either. If everything works with software but not hardware flow control, you may need to check the state of the RTS and CTS lines (tnc pins 4 and 5 respectively) with a breakout box or multimeter. Most of the time these lines should be high, dropping only occasionally to the low state when transferring long messages.

Assuming all is well, you are now ready to install the NOS software.

11: HANDS ON — SOFTWARE INSTALLATION

Several steps are required when installing the software:

- Optimising DOS
- Modifying *CONFIG.SYS* and *AUTOEXEC.BAT*
- Loading the NOS software
- Configuring the NOS control files

Optimising DOS

As mentioned earlier, NOS is memory hungry and will use as much memory as you can give it. This means that you should minimise the DOS overhead as far as possible. There are several ways to do this.

First, as NOS does not require graphics memory, the biggest potential saving is to tell DOS to release the video RAM that it uses for the graphics pages (for example, with the DR-DOS command **MEMMAX +V**, or with the Quarterdeck VIDRAM program). This step alone will make an extra 96K of RAM available in conventional memory.

You should load as much of DOS as possible into upper memory or high memory.

You should also load your favourite TSR (Terminate and Stay Resident) programs into upper/high memory if possible.

With these precautions in place, you'll probably find that you now have over 700K of conventional RAM available for NOS.

Modifying *CONFIG.SYS* and *AUTOEXEC.BAT*

A small number of additions and modifications to the DOS files *CONFIG.SYS* and *AUTOEXEC.BAT* are required to run NOS.

CONFIG.SYS should have at least the following commands:

```
BREAK=ON
BUFFERS=20
FILES=20
LASTDRIVE=z
SHELL=c:\command.com /P /E:1024
DEVICE=c:\dos\ansi.sys
```

The LASTDRIVE variable is required because NOS will be configured to use SUBSTituted drive letters up to the letter *V:*.

The SHELL variable has the /E parameter, to set the environment space to 1024 bytes. This is to ensure there is sufficient room for NOS environment variables.

The ANSI.SYS device driver lets you use ANSI escape sequences in text strings (for example, to display the command prompt in reverse video).

AUTOEXEC.BAT will contain all the usual commands, plus the following:

```
CALL c:\nos\nosenv.bat
```

(assuming that you place NOS in directory *C:\NOS*).

The batch file *NOSENV.BAT*, described below, sets up the NOS environment variables.

Loading the NOS Software

The simplest way to install NOS is to acquire a copy of **NOSview**, which contains all the files you need, plus an installation script which puts them in the proper place. Details of how to obtain **NOSview** were included in Chapter 2, and full installation instructions are in the !!README.1ST file in **NOSview**.

If you don't have a copy of **NOSview**, you'll have to install NOS and prepare all the control files by hand. Full listings of the control files are included in Appendix 3.

Whichever way you install NOS, you'll need decide on the NOS root directory; i.e. the top level DOS directory which NOS will use. In this book we use *C:\NOS* as the NOS root, but in principle you can place it anywhere.

However, be aware that when NOS is operational, other people will be logging into your system and may browse around your files, from the NOS root level downwards. For this reason, *C:* is not recommended as the NOS root!

It's convenient to use a SUBSTituted drive for the NOS root — see Fig 11-1. This is set up in *NOSENV.BAT*, with a command of the form **SUBST N: C:\NOS**. Thereafter, you can make all references to NOS files relative to drive *N:*.

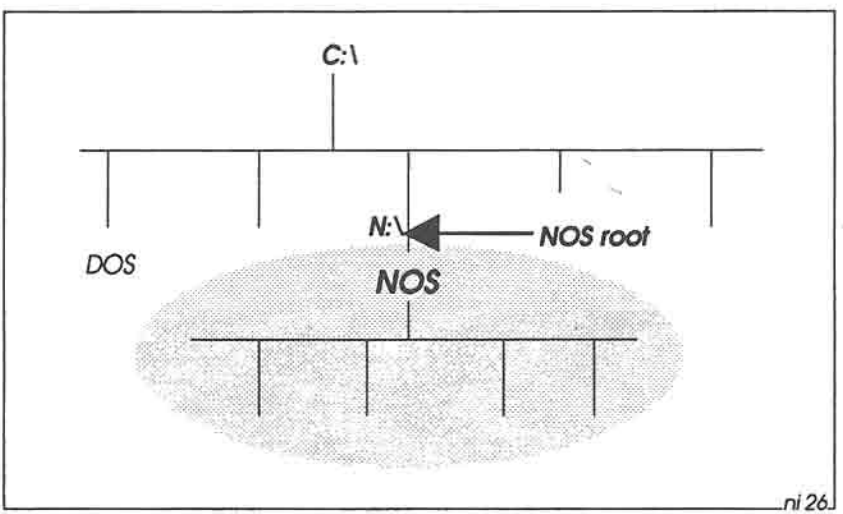


Fig 11-1: The NOS root directory is defined as drive *N:* in this book, using the DOS command *SUBST N: C:\NOS*.

The NOS Directory Tree

NOS requires the following directory tree. If you are installing NOS manually, you'll need to create the following directories (with the DOS **MKDIR** command), replacing **N:** with your chosen NOS root directory (e.g. *C:\NOS*).

```
N:\

N:\DUMP\
N:\DUMP\RECORD\
N:\DUMP\TRACE\

N:\FINGER\

N:\PUBLIC\
N:\PUBLIC\MASTERS\
N:\PUBLIC\NOSDOCS\
N:\PUBLIC\NOSVIEW\

N:\SCRIPTS\

N:\SPOOL\
N:\SPOOL\HELP\
N:\SPOOL\MAIL\
N:\SPOOL\MQUEUE\
N:\SPOOL\NEWS\
N:\SPOOL\RQUEUE\
N:\SPOOL\SIGNATUR\

N:\TMP\
```

Once the directory tree is in place, the files listed in the box opposite need to be installed.

Configuring the NOS Control Files

Several of the files listed opposite require tailoring before you use NOS on-air. This mostly involves changing callsigns, IP addresses, routing tables, and so on, to suit your local environment. Details of exactly how to make the necessary changes are included in subsequent chapters. However, the files supplied with **NOSview** are usable as they stand — provided, of course, that you leave your radio switched off.

```
N:\ALIAS
N:\AUTOEXEC.NOS
N:\AX25.COM
N:\CLEANQ.BAT
N:\DOMAIN.TXT
N:\FTPUSERS
N:\NET.RC
N:\NOSxxxxx.EXE                (the NOS executable)
N:\NOSENV.BAT
N:\PCELM.EXE
N:\PCELM.MSG
N:\PCELM.RC
N:\POPUSERS
N:\REMOTE.BAT
N:\SIGNATUR
N:\STARTNOS.BAT
N:\UUDECODE.EXE
N:\UUENCODE.EXE
N:\VIEW.COM
N:\VIEW.HLP
N:\WD8003.COM

N:\FINGER\SYSOP

N:\PUBLIC\MASTERS\*.*          (NOSview control file masters)
N:\PUBLIC\NOSDOCS\*.*          (NOSview documentation files)
N:\PUBLIC\NOSVIEW\*.*          (NOSview HELP files)

N:\SCRIPTS\FKEYS.LST
N:\SCRIPTS\FKEYS.SCR
N:\SCRIPTS\KISSON.DIA
N:\SCRIPTS\TNCRESET.DIA
N:\SCRIPTS\TNCRESET.SCR

N:\SPOOL\AREAS
N:\SPOOL\FORWARD.BBS
N:\SPOOL\REWRITE

N:\SPOOL\HELP\*.*              (the NOS BBS help files)

N:\SPOOL\SIGNATUR\SYSOP.SIG
```

Fig 11-2: NOS Files. Their functions are described in the next chapter.

In fact, you can learn a great deal about NOS before using it for real, so it's recommended that initially you leave the files alone. Try working through the hands-on sessions in this book as far as you can with the radio disconnected, then edit the control files to reflect your own environment before going live.

Checking the VIEW Fileviewer

Having installed the software, you should then re-boot the system. When *AUTOEXEC.BAT* runs, it will now call *NOSENV.BAT*, which sets up the NOS environment. You should see the message:

```
Loading VIEW - Clockwork's Resident File Viewer.
To activate press:-       RShift and SPC
```

This confirms that the **VIEW** TSR file viewer is loaded. **VIEW** lets you examine any file in the system, ASCII or binary, in text or hexadecimal format. Test its operation by pressing the right shift key and the space bar simultaneously. The Clockwork View introduction screen should appear. Use the **PgDn** and **PgUp** keys to examine the help menus.

Then test the file viewer, by pressing **F3**. The prompt:

```
Load file: *.*
```

should then appear at the top of the screen. Now press **CR**. This should pop up a list of files in the current directory, similar to the example in Fig 2-1. Move the arrow keys up or down to select a particular file, then press **CR** again. The selected file should now appear.

Use the **G** and/or **H** key to change the display format if necessary. Finally press **ESC** to exit from **VIEW**.

SUBSTituted DOS Drives

Type the DOS **SUBST** command, and confirm that all of the SUBSTituted drives defined in *NOSENV.BAT* are available:

```
M: => C:\NOS\SPOOL\MAIL
N: => C:\NOS
Q: => C:\NOS\SPOOL\MQUEUE
R: => C:\NOS\DUMP\RECORD
T: => C:\NOS\DUMP\TRACE
V: => C:\NOS\PUBLIC\NOSVIEW
```

Change directory to the NOS root directory (*N:*\), and check that all the files and directories listed above are accessible. Also, change to directory *V:*\ and check that the **NOSview** files are present.

Now type the DOS **SET** command, and check the NOS environment variable settings defined in *NOSENV.BAT* are correct. Also check that the *N:*\ directory has been appended to the original PATH list:

```
PATH=C:\dos; ... N:\
HOME=C:\nos
MAILER=N:\pcelm.exe
TMP=N:\tmp
TZ=UTC
USER=ns9bob
```

The meanings of these variables are explained in later chapters.

With software installation complete, it's now time to look closer at the files.

12: NOS FILE COMPENDIUM

In this chapter we take a closer look at the directory structures and data files in the NOS environment. This includes the files listed in the previous chapter, plus other files which NOS generates at runtime.

Examples of all the text files are included in Appendix 3.

The Directory Tree

The complete directory/file tree for NOS is shown in Fig 12-1 on the next page. NOS can reside in any directory on any drive, but as explained earlier, it's best to allocate a new drive letter for NOS (e.g. *N:*), using the DOS **SUBST** command. This has the advantage that the path declarations in the NOS control files are kept simple, and, more important, it prevents people browsing through your private files when they log into your system. You don't want other people climbing too high up your tree!

The NOS directories under drive *N:* are as follows:

/: The root directory contains most of the files for controlling NOS.

/dump: This directory is for log and trace files.

/dump/record: Holds session recordings and files downloaded from a mailbox.

/dump/trace: Holds trace files.

/finger: The NOS **finger** command allows you to "put the finger on somebody"; i.e. find out more about them. The *finger* directory contains one or more text files with information about yourself (the so-called "brag" files).

```
/alias
/autoexec.nos
N:\AX25.COM
N:\CLEANQ.BAT
/domain.txt
/ftpusers
/net.rc
/netrom.sav
N:\NOSxxxxx.EXE
N:\NOSENV.BAT
N:\PCELM.EXE
N:\PCELM.MSG
N:\PCELM.RC
/popusers
N:\REMOTE.BAT
/seqf
/signatur
N:\STARTNOS.BAT
N:\UUDECODE.EXE
N:\UUENCODE.EXE
N:\VIEW.COM
N:\VIEW.HLP
N:\WD8003.COM

/dump/session.log

/dump/record/*.*

/dump/trace/*.*

/finger/sysop

/public/*.*
```

```
/public/masters/*.*
/public/nosdocs/*.*
/public/nosview/*.*

/scripts/fkeys.lst
/scripts/fkeys.scr
/scripts/kisson.dia
/scripts/tncreset.dia
/scripts/tncreset.scr

/spool/areas
/spool/forward.bbs
/spool/history
/spool/mail.log
/spool/rewrite

/spool/help/*.hlp

/spool/mail/*.txt

/spool/mqueue/*.txt
/spool/mqueue/*.wrk
/spool/mqueue/*.lck
/spool/mqueue/sequence.seq

/spool/news/*.*

/spool/rqueue/*.txt
/spool/rqueue/*.wrk

/spool/signatur/*.sig

/tmp/*.*
```

Fig 12-1: The complete NOS file tree. DOS filenames are in capital letters, NOS filenames in lower-case. Filenames in light text are created by NOS at run-time.

/public: This is a general-purpose directory which anyone can access when logged into your system. The directory contains files for general consumption, and certain nominated users can also write into it.

/public/masters: This directory contains master copies of the NOS control files. The files in this directory are read-only, to prevent accidental damage.

/public/nosdocs: The *nosdocs* directory contains a set of miscellaneous NOS documentation files. The files in this directory are also read-only.

/public/nosview: This directory contains the complete set of **NOSview** reference documentation files, so that other people can read them to find out more about NOS. The files in this directory are read-only. DOS drive letter *V:* also points to this directory, so that you can instantly access **NOSview**.

/scripts: This directory contains scripts for use with the NOS **source** command, together with dialer scripts for setting up a modem or tnc.

/spool: The spool directory contains a number of files to do with mail handling.

/spool/help: The help subdirectory contains a set of help text files used by the built-in NOS mailer.

/spool/mail: This is the default directory for incoming mail, for outgoing POP mail, and for outgoing mail to be forwarded into the PBBS network.

/spool/mqueue: This is the directory containing all outgoing mail awaiting SMTP forwarding.

/spool/news: This directory contains files received from the news network with the NNTP protocol.

/spool/rqueue: The *rqueue* directory is an alternative directory for incoming mail (i.e. instead of */spool/mail*).

/spool/signatur: This directory holds one or more text files containing your "signature" to be appended to messages which you send with the NOS BBS.

/tmp: This directory is for temporary work files.

DOS Files

This section describes the DOS files required by NOS.

C:\AUTOEXEC.BAT: (see page 312 for listing) This file serves its usual purpose during DOS startup. The only special requirement is that it includes a **CALL** to *NOSENV.BAT*, to set up the correct working environment for NOS. If you include any TSRs in *AUTOEXEC.BAT*, it is recommended that you load them in upper memory, to free up as much conventional RAM as possible for NOS. Further, you may also care to use a utility such as Quarterdeck's VIDRAM to free up the PC's graphics buffer area, thus making it available to NOS.

C:\CONFIG.SYS: (page 317) The usual DOS configuration file. The only special requirements are the inclusion of the LASTDRIVE environment variable (allowing us to define a number of SUBSTituted drives), and the use of the /E option with the SHELL variable to make the DOS environment space large enough.

C:\DOS\ANSI.SYS: The usual device driver for screen and keyboard handling. Alternatives such as *NANSI.SYS*, *ZANSI.SYS* or *VGAANSI.SYS* are equally suitable here.

N:\AX25.COM: The special TSR driver for the Baycom modem.

N:\CLEANQ.BAT: (page 317) This batch file removes any unsent messages and lock files from the outgoing mail queue, and any unsent messages awaiting PBBS forwarding. This is a useful utility for cleaning up after experimenting with mail handling off-air.

N:\NOSxxxxx.EXE: This is the NOS runtime executable. The letters xxxxx represent a date code or version number.

N:\NOSENV.BAT: (page 325) Called by *AUTOEXEC.BAT* at DOS startup, this file initialises the NOS environment. There are six **SUBST** commands to define new drive letters:

```
M:    defines the incoming mail queue
      (useful for taking a quick look at files in the queue)
N:    defines the NOS root directory
Q:    defines the NOS outgoing mail queue
      (useful for taking a quick look at what is in the queue)
R:    defines the NOS record directory
      (useful for taking a quick look at downloaded files)
T:    defines the NOS trace directory
      (useful for scrolling through trace files)
V:    defines the NOSview directory
      (useful for examining NOSview files)
```

NOSENV.BAT also contains a number of environment variable definitions:

```
HOME: this is the home directory for the external mailers (BM,
      PCElm and ELM).

MAILER: this is the name of the external mailer which you
      call when you give the NOS mail command.

TMP:  this the the name of the temporary directory for the
      external mailers.

TZ:   this is the time-zone, used by mailers.

USER: this is the name of the user, used by the ftp command.
```

NOSENV.BAT then calls the **VIEW** TSR. **VIEW** is quite large (around 13 KB), so you should load it into upper memory if possible.

Finally, *NOSENV.BAT* installs the device driver(s) for devices which NOS does not support directly. This will be necessary for devices like Ethernet adapters, or for the Baycom AX.25 driver (*AX25.COM*).

N:\PCELM.EXE: PCELM is a popular external mailer, invoked with the NOS **mail** command. The DOS environment variable MAILER specifies the name of this file. (Alternative external mailers commonly supplied with NOS are ELM and BM).

N:\PCELM.MSG: This file contains the text of PCElm help and status messages. You can edit this file to provide help in different languages if you wish.

N:\PCELM.RC: This is the startup file for PCELM (*rc* means "run commands"). The corresponding startup files for ELM and BM are, predictably, *ELM.RC* and *BM.RC*.

N:\REMOTE.BAT: *(page 326)* This batch file runs NOS in a continuous loop, and is intended for use at remote site locations. If NOS stops running for some reason, DOS goes back to the beginning of the loop and restarts it.

N:\STARTNOS.BAT: *(page 328)* This is a simple DOS batch file that starts NOS running.

N:\UUENCODE.EXE: This command performs binary-to-ASCII file conversion. Encoding of 8-bit binary files into 7-bit ASCII is

necessary prior to uploading/downloading of binary files to/from the NOS BBS (This is only required for AX.25 users of the BBS. FTP can handle 8-bit binary file transfer without encoding).

N:\UUDECODE.EXE: This command converts uuencoded files from ASCII back to binary.

N:\VIEW.COM: This is the **VIEW** fileviewer from Clockwork Software.

N:\VIEW.HLP: The help file for **VIEW**.

N:\WD8003E.COM: This is an example of a Clarkson Ethernet driver.

NOS Files

This section describes the complete set of NOS files. All the file pathnames given below are relative to the NOS root drive, *N:*.

/alias: (page 311) The *alias* file contains a list of alternative names for mailing, allowing you to use easily-remembered names when sending mail. Also, the file can contain distribution lists for sending messages to groups of people.

/autoexec.nos: (pages 312-317) This is the big daddy of the NOS control files. The name *autoexec.nos* is the default name for this file, but you can choose a different name if you wish. You can create a number of different startup files for different situations or configurations, and switch quickly from one scenario to another without having to edit the startup file every time.

In this chapter we only look at the main functions of this file, and then in later chapters we will analyse each of the functions in detail. The main sections of *autoexec.nos* are as follows:

```
Miscellaneous Setup: These commands are for general NOS
housekeeping.

Domain Defaults:  These defaults make it easier to use domain
addressing.  They let you use simple commands like ftp ns9ken
instead of the full ftp ns9ken.ampr.org, and in status reports
you will see host names displayed as ns9ken rather than the
less meaningful 44.199.41.2.
```

Station Identification: Here you define your IP and AX.25 addresses.

TNC Setup: The commands in this section initialise the tnc at the physical level. The *attach* command defines the interface name (*tnc0*), so that later commands can use this name when addressing the interface. The dialer script *kisson.dia* switches the tnc to KISS mode, and the *param* commands set up various parameters inside the tnc.

The Baycom AX.25 Packet Driver: This command makes the link between NOS and the Baycom AX.25 driver TSR.

Setting up AX.25: Here you define the usual parameters for the AX.25 level.

Other Interfaces: At this point in the startup file are commands for defining interfaces for the Ethernet adapter and for an AX.25-over-IP (*axip*) "wormhole" link.

Interface Configuration: The *ifconfig* command lets you define broadcast and network mask parameters for the tnc interface.

TCP/IP defaults: Here are the initial values for critical TCP and IP parameters.

Starting Network Services: This is where you start all of the network servers (listeners). Once these are running, clients on other machines can then establish communication with them.

NET/ROM Configuration: Where TCP/IP traffic is carried over a NET/ROM link it is necessary to set up a large number of NET/ROM parameters. These commands start the NET/ROM server, define your own NET/ROM alias (*#BOB* in the example), and initialise a number of timers and counters.

NET/ROM Filtering: In some instances it may be preferable to accept NET/ROM broadcasts from only a limited number of nominated stations, rather than to accept broadcasts from any station which might be in range. The *netrom nodefilter* commands let you nominate the stations of interest (or alternatively filter out unwanted stations).

IP Routing Table for NET/ROM: This table is required for NOS stations which are contacted via NET/ROM.

The ARP Table for NET/ROM: The ARP table specifies the relationship between IP hostnames and NET/ROM callsigns.

NET/ROM Routing: Entries in this section let you set up the NET/ROM routing table.

AX.25 Routing: The entries in the AX.25 routing table allow you to specify any digipeaters in paths to AX.25 destinations.

AX.25 Modes: Here you can specify whether virtual circuits or datagrams are to be used.

More ARP Table entries: This part of the file contains more ARP table entries specifying the relationship between IP hostnames and AX.25 callsigns (or Ethernet adapter addresses).

IP Routing Table: The IP routing table contains entries for groups of stations and/or individual stations, specifying the local interfaces to be used and the hostnames of remote IP gateways which will handle the traffic.

Routing Interface Protocol: The RIP commands control IP routing table broadcasts, and allow you to filter particular broadcasts in a similar way to NET/ROM node filters.

RSPF: The RSPF commands are an alternative to RIP.

Hop Check: These commands specify timer constants for use with the *hop* commands when tracing the availability of particular routes.

Remote Control: These commands kick other hosts within radio range into life. The resulting traffic thus forces an automatic update of the routing tables.

Setting up the NOS BBS: The *smtp* commands initialise SMTP mail forwarding. The *smtp kick* command forces the SMTP client to scan the outgoing mail queue and to forward any IP mail. The *mbox kick* command wakes up the mailbox client and forces it to forward any AX.25 PBBS mail.

POP Commands: These commands configure the POP mail forwarding service.

FTP Defaults: These parameters specify how FTP is to transfer files by default.

Function Key Definition: The script *fkeys.scr* contains definitions for all of the function keys and cursor keys. These definitions let you execute frequently-used NOS comands with just one or two keystrokes, making it much easier to drive.

/domain.txt: *(pages 318-319)* The *domain.txt* file contains a list of station names, such as ns9ken.ampr.org, together with their corresponding IP addresses. (In some versions of NOS, this file is called *hosts.net* and has a different format).

/ftpusers: (pages 322-323) The *ftpusers* file contains a list of people who are allowed special privileges on your system, when using the **ftp** command or when logged into your mailbox. You can therefore give permission to specific individuals to create, write or delete files in nominated directories, and let them access various network ports via your mailbox. And you can even let them operate your station remotely if you want!

/net.rc: (page 324) The *net.rc* file contains username/password pairs for one or more remote hosts. When you **ftp** to those hosts, you will be automatically logged in and can start FTP transfers immediately.

/netrom.sav: This file is created when you give the **netrom save** command. It contains the dynamic NET/ROM routing table entries which have been received in broadcasts. You can later give the **netrom load** command, which will read this file and restore the NET/ROM routing table to its original state at the time when the file was created. This means that you don't have to wait for new broadcasts after restarting NOS.

/popusers: (page 326) This file contains a list of username/password pairs used with the POP mail forwarding protocol.

/seqf: This file is maintained automatically by the ELM mailer, and contains the current message number for an outgoing message.

/signatur: (page 327) This file contains the text to be appended to the end of any messages you send with the PCElm mailer. Note that there is a separate signature subdirectory which is used by the NOS built-in mailer; see below. [Having two different methods of adding a signature to a message, depending on whether you're using PCelm or the NOS BBS, is just plain silly and should be unnecessary. However, we have to live with it at present].

/dump/session.log: The session log is maintained by NOS, and contains NOS startup and shutdown times, together with a detailed record of almost everything that NOS does. This is invaluable when trying to find out what actually happened and what didn't!

/dump/record/.*:* These files are session recordings (created by the NOS **record** command) or files downloaded from a BBS.

/dump/trace/.*:* The files in this directory contain full trace information of packet traffic sent and received by this station. These trace files are essential when setting up NOS and debugging.

/finger/sysop: (page 329) This is an example of a "brag" file. When a remote user gives the command **finger sysop@ns9bob**, this file is sent to the user, and tells him all about you.

/public/masters/.*:* These are master copies (read-only) of the NOS control files.

/public/nosdocs/.*:* These are miscellaneous NOS documentation files.

/public/nosview/.*:* These are the on-line **NOSview** documentation files.

/scripts/fkeys.lst: (page 319) This is a text file which is displayed when you hit the **F1** help key. It lists all the function key definitions.

/scripts/fkeys.scr: (pages 320-321) This is the file which programs the function keys and cursor keys, and corresponds to the *fkeys.lst* file.

/scripts/kisson.dia: (page 324) This script sets the speed of the PC-to-tnc serial link to 4800 bps, and then sends a number of asterisks to the tnc. Hopefully the tnc will recognise them and autobaud into native command mode. The script then sets up the tnc with your callsign and Morse ID (MID) interval; the value 84 corresponds to 840 seconds (14 minutes). Finally, the script sets the tnc into KISS mode.

This script was written for a PK-88 tnc; you may need to change it if you have a different tnc.

/scripts/tncreset.dia and *tncreset.scr: (pages 329-330)* These scripts reset the tnc to native mode.

/spool/areas: This text file is used by the NOS BBS, and is output in response to the BBS A command. It contains a list of public mailbox names.

/spool/forward.bbs: (page 321) The *forward.bbs* file specifies how and when messages are to be forwarded onto the AX.25 PBBS network.

/spool/history: The *history* file is generated by the built-in NOS mailer, and contains a list of received bulletin IDs. If a bulletin arrives in the system with an ID which already exists in the history file, NOS rejects it.

/spool/mail.log: The mail log contains details of every attempt to send and receive mail. This is where to look if mail transfer is not working.

/spool/rewrite: *(page 327)* The *rewrite* file contains the rules for re-addressing mail (e.g. for forwarding onto the PBBS network).

/spool/help/*.hlp: These are short text files containing instructions on how to use the NOS BBS commands. There is one file per command.

/spool/mail/*.txt: The *.txt* files contain incoming messages, which you can then read with a mailer. There is one file per individual user and one per public mailbox area; for example, all of NS9BOB's messages are in file *ns9bob.txt*, and all bulletins in the TCPIP area are in the file *tcpip.txt*.

In addition, there may be other *.txt* files in this directory, for outgoing mail to be forwarded into the AX.25 PBBS network or for POP forwarding.

/spool/mqueue/*.txt: The *.txt* files in this directory contain the text of outgoing messages.

/spool/mqueue/*.wrk: The *.wrk* files contain message To: and From: information, together with the name of the host to which the message is to be sent.

/spool/mqueue/*.lck: The *.lck* files are lock files which SMTP creates when attempting to send a message.

/spool/mqueue/sequence.seq: This file is maintained automatically by the NOS BBS and PCElm mailers, and contains the current message number for an outgoing message.

/spool/news/*.*: The files and subdirectories below */spool/news* are used by the network news protocol NNTP.

/spool/rqueue/*.txt and ***/spool/rqueue/*.wrk:*** This directory is an alternative to the default incoming mail directory (*/spool/mail*). The *.txt* files have the same format as those in the default mail directory. The *.wrk* files have a similar but not identical format to those in the default mail directory.

/spool/signatur/*.sig: The signature files in this directory are short text files which the built-in NOS mailer automatically appends to the end of messages which you create. Which file is used depends

on how you log into the NOS BBS. If you log in as ns9bob, then the mailer uses the signature file *ns9bob.sig*, but if you log in as sysop, then *sysop.sig* is used instead.

Note that these signature files are used by the NOS BBS. If you use the PCElm mailer, the file */signatur* will be used instead.

/tmp/*.*: The files in this directory are temporary working files, usually created by external mailers.

13: HANDS ON — SESSION MANAGER

Having installed the hardware and software, and become familar with the functions of most of the files, you are now almost ready to start NOS. This chapter covers the final steps required to tailor NOS to your environment, then explains in detail exactly what happens when NOS starts up. Once NOS is running, you can then get to know the Session Manager.

The first time you run NOS, make sure that the radio is turned off. Only when you are confident that all the control files are set up properly with real callsigns and IP addresses should you consider going live.

Choosing a Serial Port

Before you can communicate with the tnc, you have to specify which asynchronous serial port you are going to use (i.e. COM1, COM2, COM3 or COM4), by removing the # symbol from the front of *one* of the following **attach asy** commands in autoexec.nos:

```
# attach asy 0x3f8 4 ax25 tnc0 2048 256 4800   # COM1
# attach asy 0x2f8 3 ax25 tnc0 2048 256 4800   # COM2
# attach asy 0x3e8 4 ax25 tnc0 2048 256 4800   # COM3
# attach asy 0x2e8 3 ax25 tnc0 2048 256 4800   # COM4
```

The first two numeric parameters (e.g. 0x3f8 and 4) are the I/O address and IRQ number for the port respectively.

The name tnc0 is a symbolic interface name for the port. You can choose anything you like here (but tnc0 is used throughout this book for consistency). You will use this name in future commands to refer to this port.

The next two numeric parameters (2048 and 256) are to do with buffer and packet sizes, and need not be changed.

The last parameter on the line (4800) is the speed of the serial link to the tnc. You may care to increase this to 9600 or even higher if your machine is fast enough; in general, the faster the better, to minimise bottlenecks on the link.

So, for example, if your tnc is attached to COM1 and the link runs at 9600 bps, the line will read:

```
attach asy 0x3f8 4 ax25 tnc0 2048 256 9600   # COM1
```

Starting NOS

There are several different ways you can start NOS, and to make things easy it's convenient to put the startup commands in a *.BAT* file — you'll be starting and stopping NOS many times, so a *.BAT* file saves a lot of typing. You can also make several *.BAT* files for different startup situations.

Appendix 3 contains a typical NOS startup file, *STARTNOS.BAT* (see page 328). It performs several functions.

First, the PROMPT symbol is redefined to read:

```
$e[5mEXIT TO RETURN TO NOS $e[0m$e[44m$p$g$e[m
```

(The $e sequences in the PROMPT string are ANSI escape sequences to make the text string EXIT TO RETURN TO NOS flash, and to display the current directory name in green. You can remove these sequences if you have an aversion to flashing lights or green text!).

Why change the prompt from its usual C:\> form? Well, when NOS is running, you can escape to a DOS shell if you want to do DOS things. The new prompt reminds you that you are not in a default DOS shell, and that you need to give the **EXIT** command to return to NOS.

Next, for DR-DOS users, the startup file executes the **MEMMAX +V** command. This makes 96 KB of additional RAM available. (Alternatively, if you are using Quarterdeck's QEMM package, you

can give the **VIDRAM ON** command to gain access to this extra memory).

The **IF EXIST** statement tests for the existence of **.LCK* files in the *Q:* directory (recall that *Q:* is a SUBSTituted drive for the outgoing mail directory */spool/mqueue*). These lock files appear when NOS is attempting to forward mail to another station, and normally disappear when forwarding is complete. However, if you stop NOS before it has a chance to remove the lock files, the forwarding system is left in an indeterminate state. Thus during startup, *STARTNOS.BAT* forcibly removes any lock files.

The next command:

```
N:\NOS_20M.EXE /autoexec.nos
```

starts NOS itself. The NOS executable is *NOS_20M.EXE*, and *autoexec.nos* is the NOS startup file.

IMPORTANT: Note that the slash preceding the startup file name is a *forward* slash, not a backslash. The slash indicates that the file is at the NOS root level (i.e. drive *N:*).

The general syntax for NOS startup is:

```
nos.exe [-b] [-s n] [-d /directory] [-v] [startup_file]
```

The -b option: This option specifies the use of the PC's BIOS for console output. Normally NOS writes direct to the video display buffer for speed, but in certain circumstances (e.g. when running a windows package) this may cause bleedthrough. Selecting the -b option should remove this problem; e.g.

```
N:\NOS_20M.EXE -b /autoexec.nos
```

The -s option: This option defines the number of sockets for use by NOS. Sockets are data arrays used for network connections; you typically need four sockets for most types of connection. The default number of sockets is 40. Increase this number if you anticipate running many sessions in parallel. For example:

```
N:\NOS_20M.EXE -s 60 /autoexec.nos
```

The -d option: This option lets you specify a different root directory for the NOS configuration and spool files. Its use is not recommended, as some of the commands in NOS (e.g. **finger**) do not understand it. It's much cleaner and more secure to root NOS on a SUBSTituted drive, and to make all file references relative to that root.

The -v option: This option gives a verbose output at NOS startup time, displaying each startup command as it executes. This is useful for detecting errors in the startup file, or if NOS hangs during initialisation. For example:

```
N:\NOS_20M.EXE -v /autoexec.nos
```

The default name for startup_file is *autoexec.nos* (so it isn't really necessary to include it in the above examples). You can create any number of NOS startup files if you wish, to cater for different operating scenarios.

The rest of *STARTNOS.BAT* restores the original DOS environment after you finally exit from NOS. That is, the 96 KB block of video RAM is released and the prompt is restored to its original state.

Lift Off!

You are now finally ready to start NOS. Switch on the tnc, and make sure that it's working properly in native *cmd:* mode. Then type the **STARTNOS** command at the DOS prompt. NOS will run *autoexec.nos*, and eventually something like the display in Fig 13-1 opposite will appear on the screen.

net> is the Session Manager prompt. If it doesn't appear immediately, just hit **CR** a couple of times — it should then appear.

Switching the TNC from native mode to KISS mode

The last few lines (starting at Dialing on tnc0) appear when NOS calls the dialer script near the start of *autoexec.nos*. The script itself, *kisson.dia*, is in directory */scripts*; see Appendix 3 for a listing (page 324).

```
KA9Q NOS version 911229 (PA0GRI v2.0m)
This version produced by PA0GRI
Copyright 1991 by Phil Karn (KA9Q) and contributors.
TxDelay: 20
Persist: 63
SlotTime: 10
TxTail: 10
FullDup: 0
DTR: 1
RTS: 1
N:/DUMP/RECORD
Dialing on tnc0
MYCALL NS9BOB-5
MID 84
XMITOK ON
KISS ON
Dial tnc0 complete

net>
```

Fig 13-1: The NOS startup screen.

The purpose of the script is to switch the tnc from native *cmd:* mode into KISS mode — see Fig 13-2. After setting the link speed to 4800 bps, the script then sends a number of asterisks at 200 millisecond intervals, giving the tnc an opportunity to autobaud to the correct speed. The script now sends a **\r (CR)**, waits up to one second for the *cmd:* prompt, and then sets up the AX.25 callsign **(MYCALL NS9BOB-5)**. This is followed by **MID 84** (to set the Morse ID interval to 840 seconds — 14 minutes) and **XMITOK ON** (to enable the tnc PTT line). Finally, the **KISS ON** command sets the tnc into KISS mode.

This script is applicable to an AEA PK-88 tnc with the RAM backup battery removed. That is, each time NOS starts up it is assumed that the tnc is in its reset state — if it's already in KISS mode, then the tnc will ignore the script commands. If you have a different type of tnc, or want to use a different sequence of commands to set it into KISS mode, then you will need to modify the script. (On the other hand, if the backup battery is in place and you leave the tnc permanently in KISS mode, you won't need the script at all).

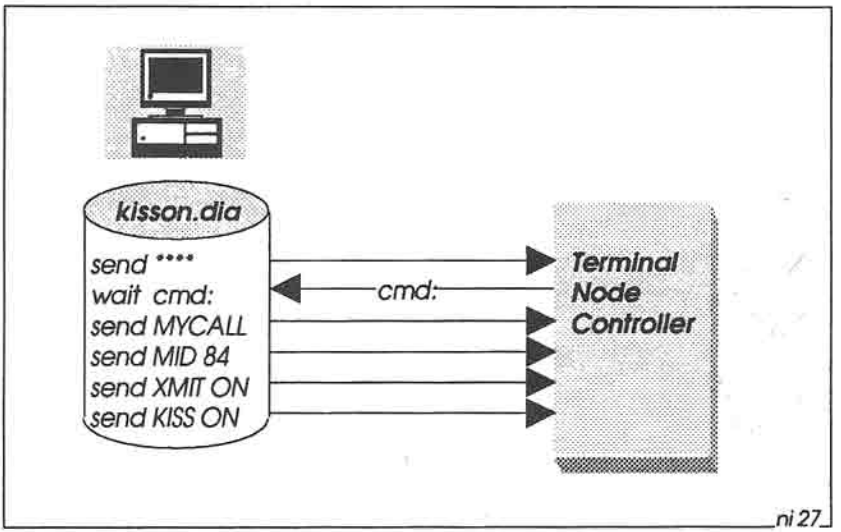


Fig 13-2: NOS startup. The dialer runs *kisson.dia*, which autobauds the tnc (by sending a stream of asterisks). The tnc responds with the native mode *cmd:* prompt. Then the dialer sends commands to set up the callsign and CWID interval, and to enable the PTT line. Finally the dialer sends the *KISS ON* command to switch the tnc to KISS mode.

First Steps

To find out which commands this version of NOS supports, give the **?** or **help** command — see Fig 13-3.

Note that all of these commands are in lower case; NOS understands **cd**, but will not understand **CD** or **Cd** or **cD**.

Note also that you can abbreviate commands, provided that they are still uniquely recognisable. For example, you can type **star** instead of **start** (but not **sta**, as **status** also begins **sta**).

```
net> ?
!          abort      arp        autoroute    asystat
attach     attended   ax25       bbs          connect
cd         close      cls        comm         disconnect
dir        delete     detach     dialer       domain
dump       echo       eol        escape       exit
finger     fkey       ftp        ftype        help
hop        hostname   icmp       ifconfig     info
ip         isat       kick       lock         log
lzw        mail       mbox       memory       mkdir
mode       more       motd       multitask    netrom
nntp       param      ping       popmail      ps
pwd        record     remote     rename       reset
rip        rmdir      route      session      sccstat
shell      skick      smtp       socket       source
start      stop       status     tcp          tail
telnet     test       ttylink    third-party  tip
trace      udp        upload     watch        watchdog
```

Fig 13-3: The NOS command set.

To find out some more information about this particular release, give the **info** command:

```
net> info
NOS configuration information.   (PA0GRI version 920424)
Containing: AXIP, POP2client, POP2server, POP3client,
 POP3server, TCP servers,  RIP, HOP, MAILBOX, LZW, and the
 Russell Nelson modsets
Generic async interface (via 8250/16450/16550)
SLIP async interface (via 8250/16450/16550)
KISS async interface (via 8250/16450/16550)
FTP Software's PACKET driver interface
Generic SCC (8530) driver
NET/ROM network interface
Van Jacobson compression on SLIP/PPP
IP access control
```

The **status** command gives us further useful information:

```
net>  status
KA9Q Internet Protocol Package, v911229 (PA0GRI v2.0m)
This version produced by PA0GRI

NOS load information: Code Segment=1f75 - Data Segment=71e3

The system time is Sat Aug 15 13:55:47 1992
NOS was started on Sat Aug 15 13:54:52 1992

Elapsed time => 0 days:00 hours:00 minutes:55 seconds.

The station is currently Attended.
The 'Message Of The Day' is
If I'm not here, please leave a message in the mailbox.

                Table of Open Files.
Name          length  offset hnd acc PSP device    type/owner
SESSION .LOG 107238      0    1  rw 1EAF drive C: [nos_20m.exe]
```

Escape to DOS

To escape to a DOS shell, give the **!** or **shell** command. You should now see the special DOS prompt already set up in *STARTNOS.BAT*, and you should be able execute DOS commands. Be aware, however, that there will only be a small amount of memory available to the shell, as NOS is still running and occupying most of the available RAM, so this will limit what you can do in DOS.

Furthermore, you may find when you attempt to escape to DOS that the PC locks up completely. This almost certainly means that there is no room for the DOS shell, and you will have to experiment with *CONFIG.SYS* and *AUTOEXEC.BAT* to move as much as possible out of conventional memory.

When you have finished with DOS, give the **EXIT** command to return to NOS.

Let's Start a Session: The *more* Command

The **help, info** and **status** commands are examples of direct NOS commands; they respond immediately, and do not start a new session.

As an example of starting a session, give the command **more /autoexec.nos** (note the forward slash — we're in NOS now). The screen clears, and then the first page of *autoexec.nos* appears, with the word —More— at the bottom, indicating that there is "more" to come.

Now hit **CR** a few times. You'll see that you are scrolling down the file, one line at a time. To get the next screenful, hit the spacebar instead.

Then, *before you reach the end of the file*, hit the **ESC** key (or **F10** if you are running a version of NOS which doesn't support function key mapping). This will take you back to the Session Manager. Now give the **session** command:

```
net> session
 #  S#  Type      Rcv-Q Snd-Q State        Remote socket
*1  -1  More          0     0 Limbo!       /autoexec.nos
```

The first column contains the session number, and the asterisk indicates that this is the currently active session. The Type and Remote socket columns verify that this is a **more** session. The remaining columns are meaningless in this context.

If you now hit **CR**, you'll find that you are back in the **more** session. That is, whenever you are in Session Manager and hit **CR** by itself, NOS takes you back to the currently active session (indicated with an asterisk) if there is one.

Now hit **ESC** again to return to the Session Manager, and try the command **more /nosenv.bat**. Again the screen will clear, and the *NOSENV.BAT* file appears. Once again, hit **ESC**, and give the **session** command:

```
net> session
 #  S#  Type      Rcv-Q Snd-Q State        Remote socket
 1  -1  More          0     0 Limbo!       /autoexec.nos
*2  -1  More          0     0 Limbo!       /nosenv.bat
```

This time, the asterisk is alongside session 2, so if you were now to hit **CR** by itself you would return to the *NOSENV.BAT* file. But how do you get back to the first session? Simple: just give the command **session 1**.

In other words, you can switch between sessions by giving the **session** command, followed by the number of the particular session you are interested in. (It turns out that this is a round-about way of doing things — we'll soon see that there's a much better solution if your version of NOS supports function key mapping).

Terminating a Session

Sessions will either terminate naturally (e.g. you have **mored** the complete file), or you can terminate them prematurely with the **reset** command; for example, **reset 2** will terminate session 2. When you give the **reset** command, you will find that NOS takes you back to the session one last time, asking you to confirm termination by hitting **CR**.

Keyboard Mapping

Almost all versions of NOS recognise function key **F10** to mean "escape to the Session Manager". You can also define an additional key to do the same thing, by putting the **escape** command in *autoexec.nos*. For example, if you really wanted to use the ":" character as an escape character (not recommended), put this command in *autoexec.nos*:

```
escape  :
```

In practice, it's much more useful to use the **ESC** key:

```
escape  ESC
```

Note that the letters **ESC** represent the Escape key. That is, you need to enter this key as a *single literal character* (for example, if you use DOS EDIT to edit *autoexec.nos*, you prefix the **ESC** key with **CTRL-P**), and it will not be visible in a printed listing of the file.

If you use a variant of PA0GRI's NOS, you have much more flexibility in re-defining individual keys, using the **fkey** command. It's best to put the definitions in a NOS script file, and call this script from *autoexec.nos*; see the command **source /scripts/fkeys.scr** towards the end of *autoexec.nos* — **source** means "run a script containing NOS

commands." Appendix 3 contains a listing of *fkeys.scr* (pages 320-321).

The listing includes the numbering scheme for the keys which you can map: the function keys **F1** to **F10** (in normal, shift, control and alt modes), plus the **PgUp**, **PgDn**, **Home**, **End**, **Ins**, **Del** and arrow keys. For example, key 60 corresponds to **F2**, key 85 to **SHIFT-F2**, key 95 to **CTRL-F2** and so on.

The definition for each key is defined in the fkeys script as a text string between inverted commas (" "). To include a control character, prefix the character with a caret (**^**) symbol; e.g. ^M means **CTRL-M (CR)**.

Each command begins with **^[**, which means **ESC**. That is, you first escape from the current session back to the Session Manager, before executing the rest of the string.

Some of the commands end in **^M**, which means they are executed immediately. Others do not end in **^M**, giving you the chance to add extra information, or the opportunity to think before you hit **CR** — you certainly don't want to reset the tnc with **CTRL-F1** by accident, for example!

Note that **F1** (fkey 59) is defined as "^[tail /scripts/fkeys.lst^M". Thus when you hit **F1**, NOS will display the tail end (the last few lines) of *fkeys.lst*, which just happens to contain a help list for the mapped keys; see Appendix 3 for a listing of *fkeys.lst* (page 319).

Note also that the up-arrow (fkey 72) is defined as **CTRL-B**. This means "repeat the last command."

Now try repeating the **more** commands described earlier, using the **ESC** key to return to the Session Manager, and **F2** to list the sessions, and **CTRL-F5** to reset a session. Use **F1** whenever you need help. Then start several other **more** sessions, and use **ALT-F1** to switch to session 1, **ALT-F2** to switch to session 2, **ALT-F3** to switch to session 3, and so on. A lot easier than typing **session 1**, **session 2**, etc.

The keyboard mappings keys defined in this book save a lot of time and typing, and make NOS much easier to drive. Obviously you are free to change them to anything you like, but it's recommended that you stay with the mappings defined here, at least until you've finished reading this book!

Some more Filesystem Commands

Let's try a few more miscellanous commands to get our bearings:

cd: By itself, **cd** tells you the current DOS directory. At NOS startup this is */dump/record*, set at the very end of *autoexec.nos*. The command **pwd** ("print working directory") does the same thing.

To change to a different directory, it's almost like DOS; e.g. **cd /spool/help**. Note however that **cd..** (without a space after **cd**) doesn't work; you need to give the **cd ..** command (with a space before the two dots) to go up a directory level.

dir: You can use the **dir** command by itself to get a directory listing of the current directory, or with a directory path; e.g.

```
net> dir /spool
help/          8:05  8/13/92   mail.log    77 10:16  8/13/92
mail/          8:05  8/13/92   mqueue/        8:05  8/13/92
news/          8:05  8/13/92   rqueue/        8:05  8/13/92
signatur/      8:05  8/13/92
7 files. 4,669,440 bytes free. Disk size 41,246,720 bytes.
```

Resetting the TNC

You may wish to switch the tnc back to native mode, either because you have finished with NOS, or because the tnc seems to have locked up and is not responding as expected — Fig 13.4. The basic NOS command to do this is **param tnc0 255** (some tncs may require **param tnc0 254** as well).

The key combination **CTRL-F1** takes this a step further. The keyboard mapping file (*fkeys.scr*) shows that **CTRL-F1** runs the script *tncreset.scr*. This gives the **param** command to switch the tnc back to native mode, then calls the dialer script *tncreset.dia* (see pages 329-330 for listings of these files).

The dialer script resets the tnc, then, after autobauding with the asterisks, gives the **XMITOK OFF** command to disable the transmitter.

Try **CTRL-F1** to verify that you can reset the tnc. Then, to switch it back into KISS mode again, you can use **SHIFT-F1**, which calls the *kisson.dia* script described earlier.

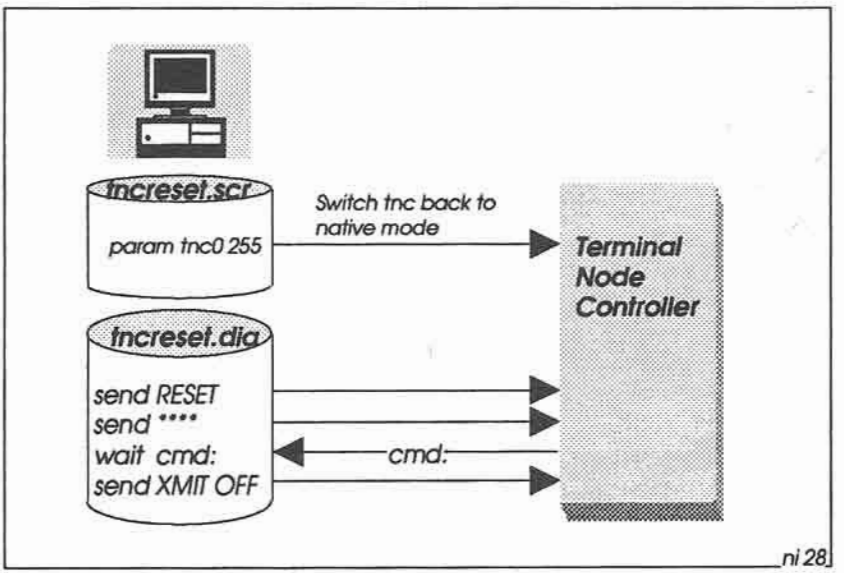


Fig 13-4: The NOS command *param tnc0 255* switches the tnc from KISS mode back to native mode. The *tncreset.dia* dialer script then resets the tnc and disables the PTT line.

Talking Direct to the TNC in Native Mode

If you want to communicate directly with the tnc in native mode, you can use the **tip** (Terminal Interface Program) command — see Fig 13-5.

You must first reset the tnc to native mode (for example, with **CTRL-F1**), then give the command:

```
net> tip tnc0
```

This clears the screen and starts a new session. Hit **CR** a couple of times, and you should see the *cmd:* prompt appear. From now on you can talk to the tnc just like you used to!

Note than when you are running a tip session, you cannot run TCP/IP at the same time — **tip** lets you talk in native mode to the tnc, whereas TCP/IP requires KISS mode access to the tnc.

To exit the tip session, you simply escape to the Session Manager, then give the **reset** command. Then you can give the **SHIFT-F1** command again, to put the tnc back into KISS mode.

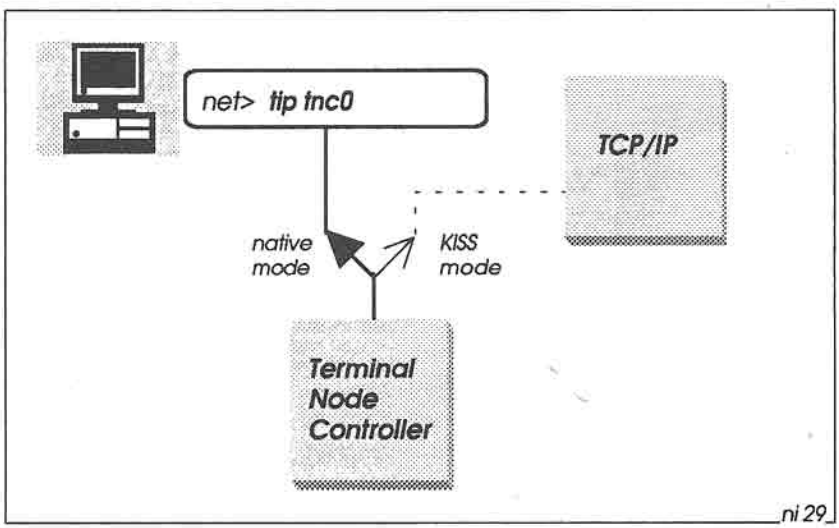


Fig 13-5: The *tip* command lets you communicate direct with the tnc in native mode. It is not possible to run TCP/IP at the same time.

The Next Step

If you've successfully navigated this chapter, you are now ready to get to know the NOS command set.

14: THE NOS COMMAND SET SUMMARY

NOS has over 80 separate commands, and getting your mind around them is not a 5-minute job! Also, most of the commands have several options, complicating matters even further. This chapter puts the commands into logical groups, and provides a brief description of each one. In later chapters we will look at most of them again in much more detail. (See Appendix 2 for full command syntax information).

Session Manager Commands

? (or help)	lists the main command names
F10 (or **ESC**)	returns you to the Session Manager `net>` prompt
attended	defines whether station is attended
cls	clears the screen
dump	displays memory contents
escape	defines the Session Manager escape character
exit	exits from NOS back to DOS
fkey	defines function keys and cursor control keys
info	displays NOS release information
isat	("is AT") defines method of system clock handling
lock	locks the keyboard
log	logs NOS sessions
memory	displays memory utilisation
multitask	allows NOS multitasking
ps	displays process status
! (or shell)	escapes to a DOS subshell
record	records a session in a file
source	runs a NOS command file
status	displays Session Manager status
test	performs a system test
trace	traces packet traffic
watch	calculates execution times
watchdog	controls NOS restart after failure

Directory Access Commands

cd	changes DOS directory
delete	deletes a file
dir	lists the files in a directory
mkdir	makes a DOS directory
pwd	displays current DOS directory
rmdir	removes a DOS directory

File Access Commands

more	displays a file a page at a time
rename	renames a file
tail	displays the last few lines of a file
upload	uploads a file

Session Control Commands

abort	aborts an FTP transfer
close	closes a session normally
connect	initiates an AX.25 connection
disconnect	terminates an AX.25 connection
kick	wakes up a NOS client
reset	forces a connection to close
session	displays session status
skick	kicks a socket
start	starts a NOS server
stop	stops a NOS server

NOS BBS Commands

bbs	calls your own NOS BBS
lzw	controls data compression
mail	calls external mailer
mbox	displays mailbox status
motd	defines "message of the day"
pop, popmail	handles reverse forwarding of SMTP mail
smtp	controls the SMTP client
telnet	calls a remote NOS BBS
third-party	controls third-party message handling

FTP Commands

abort	aborts an FTP transfer
echo	controls command echoing
eol	controls end-of-line handling
ftp	initiates a file transfer session
ftype	sets default file transfer type (ASCII or binary)

Status Commands

arp	displays ARP address table
asystat	displays asynchronous port status
drsistat	displays DRSI interface status
dump	displays memory contents
eaglestat	displays Eagle interface status
etherstat	displays Ethernet status
hapnstat	displays HAPN interface status
icmp	displays ICMP status
ifconfig	displays an interface configuration and status
ip status	displays IP status
mbox	displays mailbox status
memory	displays memory status
nntp	displays network news status
nrstat	displays NET/ROM status
param	displays tnc parameters
ps	displays process status
sccstat	displays SCCS interface status
session	displays session status
socket	displays socket status
tcp status	displays TCP status
udp	displays UDP status

Routing Commands

autoroute	remembers IP routing
ax25 route	manipulates the AX.25 routing table (digipeaters)
hop	traces the route to a remote station
netrom route	manipulates the NET/ROM routing table
rip	controls the Routing Interface Protocol
route	manipulates the IP routing table
rspf	controls the Radio Shortest Path First protocol

Network Client and Server Commands

bbs	starts a local NOS BBS session
chat	starts a chat session
finger	starts a finger session
ftp	starts a file transfer session
kick	kicks a client
mode	sets up transfer mode (datagram or virtual circuit)
nntp	handles network news
ping	pings a remote station
pop	handles SMTP reverse forwarding
rarp	handles reverse ARP
remote	handles remote control of a station
reset	aborts a session
rlogin	starts an rlogin session
smtp	handles the SMTP client
start	starts a network server (listener)
stop	stops a network server
telnet	starts a remote NOS BBS session
telnet 25	starts an SMTP session
telnet 87	starts a chat session
tip	communicates directly with tnc/modem
ttylink	starts a chat session

Interface Commands

attach	attaches an interface
detach	detaches an interface
ifconfig	configures an interface

Trace Commands

domain trace	traces domain cache handling
hop trace	traces packets to nominated destination
icmp trace	traces ICMP packets
nntp trace	traces network news packets
rip trace	traces routing broadcasts
smtp trace	traces SMTP mail handling
tcp trace	traces TCP packets
trace	traces network interface packets

TNC/Modem Commands

comm	sends commands to a tnc/modem
dialer	runs a tnc/modem initialisation script
param	sets tnc parameters
ppp	controls point-to-point protocol
tip	communicates directly with a tnc/modem

Network Name and Address Commands

arp	manipulates Address Resolution Protocol table
bootp, bootpd	handles network booting
domain	handles domain caching
hostname	sets local host name

15: HANDS ON — autoexec.nos

We now take a closer look at some of the NOS startup commands to be found in *autoexec.nos*. As usual, this chapter assumes you are using the default *autoexec.nos* listed in Appendix 3 (pages 312–317). This means that you are NS9BOB, so when you give a command like **finger sysop@ns9bob**, you are really talking to yourself. Try the commands as we go along.

Command Ground Rules

As explained earlier, all NOS commands are lower case, and you can abbreviate them as long as they are still unique. However, you must be careful when adding parameters to the commands: you shouldn't abbreviate parameters, as NOS is not always thorough about checking them for validity.

If you are not sure about what parameters are needed for a particular command, just type the command followed by a question mark. NOS will reply with a list of required parameters. For example:

```
net> mbox ?
valid subcommands: attend expert fwdinfo haddress jumpstart
kick maxmsg motd nrid password qth secure smtptoo status
timer tiptimeout trace utc zipcode
```

Alternatively, you can consult the NOS Command Set Reference in Appendix 2, or use **VIEW** to look at the same file on-line (when in **VIEW**, use **F3** to display file *V:\!CMDSET*).

Note that many of the commands are of a "display or set" nature. That is, if you give the command by itself, you can display the current values of its parameters, but if you follow the command name with parameters you can change the parameter values to something else.

Let's take a look at the beginning of *autoexec.nos*.

Miscellaneous Setup

```
# Miscellaneous setup ****************************************
attended        on
escape          ESC             # <ESC> character
isat            yes              # 286/386 clock
multitask       on
log             /dump/session.log
watchdog        off

motd  "If I'm not here, please leave a message in mailbox."
```

attended: The attended flag indicates whether the station is attended and you are ready to accept incoming chat requests. If attended is set to off, NOS sends a message back to the caller saying that the station is unattended.

isat: The isat flag states whether the PC is an AT (i.e. 286/386/486) or not. When set to on, NOS measures time intervals in milliseconds, but if set to off it can only measure time in 55 millisecond clock ticks.

multitask: When multitask is set to on, NOS continues to run when you escape to a DOS shell. If set to off, NOS activity ceases until you exit from DOS and return to NOS.

log: The session log file contains a log of every time you start and stop NOS, and a summary of every session you run. This can be useful in tracing system problems. Not surprisingly, the file can grow to be very large, so you should delete it from time to time to save disk space.

watchdog: When set to on, NOS runs a watchdog process that monitors internal activity. If the watchdog times out (after 5 minutes), NOS will automatically exit. This is useful for unattended site operation, where *STARTNOS.BAT* can be part of an endless loop, as in the file *REMOTE.BAT*:

```
:loop
N:\STARTNOS
GOTO loop
```

motd: Specifies the "message of the day", used to welcome callers.

Domain Commands

There are many domain commands, most of which you'll rarely use. They specify how NOS is to resolve and display IP addresses:

```
# Set up domain defaults ************************************
domain cache size 30
domain suffix    ampr.org.
domain translate on              # display host names
domain verbose   off             # do not display suffix
domain addserver ns9dns
```

domain suffix: NOS automatically adds the suffix to any hostname you type which does not end in a dot, to generate the full domain name. For example, when you give the command **ftp ns9ken**, NOS will look in *domain.txt* for the name ns9ken.ampr.org.

domain translate: When set to on, NOS converts numeric IP addresses to host names, making it much easier to understand status reports. For example, the **route** command gives an output like this when domain translate is set to on:

```
Dest          Len Interface Gateway     Metric P Timer Use
ns9tom        32  netrom                1        man   0
region45      24  tnc0      ns9ken      1        man   0
```

This is much more meaningful than:

```
Dest          Len Interface Gateway     Metric P Timer Use
44.199.47.75  32  netrom                1        man   0
44.199.45.0   24  tnc0      44.199.41.2 1        man   0
```

when domain translate is set to off. (Unfortunately some versions of NOS are broken; even when domain translate is set to on, the conversion does not take place).

domain verbose: When set to on, NOS displays the full domain names of hosts, including the suffix; e.g. ns9ken.ampr.org. This can be useful if *domain.txt* contains hosts from several different

domains, but in an AMPRnet-only environment you don't need to see the suffix, so you can set domain verbose to off.

Station Identification

Here is where you tell NOS who you are:

```
# Station Identification ************************************
ip address        ns9bob
hostname          ns9bob
ax25 mycall       NS9BOB-5        # This MUST precede 'attach'
```

ip address: This is the IP address of your station. This corresponds to the word *host* in descriptions of command syntax. For example, the syntax for the **ftp** command is "**ftp** *host*", which means that the **ftp** command needs an IP address (or a name in *domain.txt* which corresponds to that address).

(N.B. A lot of existing reference documentation uses the word *hostid* to mean IP address. This is quite wrong. A hostid is an IP address expressed in hexadecimal; e.g. the IP address 44.199.47.75 is equivalent to a hostid of 2cc72f4b. You can say **ftp tom** or **ftp 44.199.47.75**, but **ftp 2cc72f4b** won't get you very far. Except for mentioning it here, the word *hostid* is not used anywhere else in this book).

hostname: The hostname is the name by which you greet users of your FTP server and your NOS BBS. In principle you can use any name here, but this can cause complications when people try to reply to mail, so it's best to use the name corresponding to your IP address in *domain.txt*.

ax25 mycall: This tells NOS your AX.25 callsign. Note that it does not set up your tnc with the callsign as well; you must do this separately (e.g. in the script *kisson.dia*) before starting NOS.

Setting up the TNC

We've already seen how to attach the serial interface to the tnc, with the **attach asy** command. You can also set up various parameters in the tnc with the **param** command:

```
# Set up the TNC ********************************************
  attach asy      0x3f8 4 ax25 tnc0 2048 256 4800        # COM1
# attach asy      0x2f8 3 ax25 tnc0 2048 256 4800        # COM2
# attach asy      0x3e8 4 ax25 tnc0 2048 256 4800        # COM3
# attach asy      0x2e8 3 ax25 tnc0 2048 256 4800        # COM4

# trace tnc0      211

# Initialise the tnc to KISS mode ***************************
dialer tnc0 /scripts/kisson.dia

param tnc0        1 20              # TX delay     (x 10mS)
param tnc0        2 63              # Persistence  (0-255)
param tnc0        3 10              # Slot Time    (x 10mS)
param tnc0        4 10              # TX tail      (x 10mS)
param tnc0        5 0               # 0=HDX
param tnc0        dtr 1
param tnc0        rts 1
```

Here is a list of **param** commands which may be applicable to your tnc (all values of *n* are decimal in the range 0-255):

```
param tnc0 1    n       TX Delay (x 10mS)
param tnc0 2    n       Persistence
param tnc0 3    n       Slottime delay (x 10mS)
param tnc0 4    n       TX Tail (x 10mS)
param tnc0 5    n       0=half duplex, 1=full duplex
param tnc0 6            Hardware dependent
param tnc0 7            TX mute
param tnc0 8    n       0=DTR low, 1=DTR high
param tnc0 9    n       0=RTS low, 1=RTS high
param tnc0 10           Baudrate
param tnc0 11   n       End delay
param tnc0 12   n       Group
param tnc0 13   n       Idle
param tnc0 14   n       Min
param tnc0 15   n       Max key
param tnc0 16   n       Wait
param tnc0 17   n       Parity: 0=none, 1=even, 2=odd
param tnc0 129  n       Down
param tnc0 130  n       Up
param tnc0 254          Prepare to switch tnc to native mode
param tnc0 255          Switch tnc from KISS to native mode
```

Note that many of these parameters are tnc- and NOS-version dependent; only parameters 1-3, 5 and 255 are common to all implementations.

Interface Configuration

The interface configuration command **(ifconfig)** sets up various network parameters for each of the NOS interfaces. The parameter you may need to change is description:

```
ifconfig tnc0 description "144.625 MHz port"
```

This description provides useful information to people who want to use this interface as a gateway to other networks.

To obtain the current status of all NOS interfaces, use the **ifconfig** command by itself:

```
net> ifconfig
tnc0      IP addr ns9bob MTU 256 Link encap AX25
          Link addr NS9BOB-5
          flags 0 trace 0x0 netmask 0xff000000 broadcast
                                                44.255.255.255
          sent: ip 0 tot 6 idle 0:00:00:07
          recv: ip 0 tot 0 idle 0:00:00:07
          descr: 144.625 MHz port
loopback  IP addr loopback MTU 65535 Link encap None
          flags 0 trace 0x0 netmask 0xffffffff broadcast
                                                255.255.255.255
          sent: ip 0 tot 0 idle 0:00:00:07
          recv: ip 0 tot 0 idle 0:00:00:07
encap     IP addr 0.0.0.0 MTU 65535 Link encap None
          flags 0 trace 0x0 netmask 0xffffffff broadcast
                                                255.255.255.255
          sent: ip 0 tot 0 idle 0:00:00:07
          recv: ip 0 tot 0 idle 0:00:00:07
```

This tells us several things about each interface:

- The IP address
- The MTU (maximum transmission unit); i.e. largest packet size.
- The encapsulation: i.e. whether the packets are AX.25 or NET/ROM packets etc.

- The link address: either an AX.25 callsign or Ethernet adapter address
- Various flags and masks
- Numbers of packets sent and received through the interface.

In addition to the tnc0 interface defined in *autoexec.nos*, there are two other interfaces which appear automatically: **loopback** and **encap**. The loopback interface is a software loopback within NOS; that is, it is not attached to a physical interface port. This can be useful for testing. The encap interface is used for encapsulating AMPRnet traffic inside packets which the Internet understands.

The *finger* Command

To show how you can use NOS to communicate with other stations, let's look at the **finger** command. This allows you to "put a finger on" (i.e. find out about) users on another system. The syntax of the command is:

```
finger @host
finger user@host
```

To find out who is known on particular system, use the first form of this command. Try:

```
net>  finger @ns9bob
```

Alternatively, you can try **finger @loopback**, which amounts to the same thing. N.B. There is no space after the @ symbol.

NOS replies with the message:

```
@ns9bob — Resolving ns9bob... trying ns9bob:finger
Known users on this system:
sysop
```

The first line confirms that NOS is resolving (looking for) ns9bob in *domain.txt*, and is then trying to connect to the finger server at ns9bob.

The finger server then responds with a list of known users on the system; in this case, just one (sysop).

Where did the name **sysop** come from? Take a look in directory */finger* (with the **dir /finger** command). Here you'll find a text file named *sysop*, containing everything you ever wanted to know about sysop.

Now that you know the file *sysop* exists in the finger directory, you can ask NOS to send you a copy of the file:

```
net>  finger sysop@ns9bob
```

(N.B. No spaces either side of the @ symbol).

This time, NOS responds with the contents of */finger/sysop*:

```
sysop@ns9bob - Resolving ns9bob... trying ns9bob:finger

                Hello and welcome to ns9bob

        User:           bob (NS9BOB)
        Real Name:      Robert R Roberts
        Class:          Extra
        Address:        12345 Anystreet
                        Anytown, Anystate, AnyZIP
        Telephone:      (111) BOB-3456
        System Config:  PK-88
                        Yaesu FT-27RB
                        144.625 MHz

        Occupation:     Professor of Anglo-Saxon
           Hobbies:     Sheepshearing
```

So all you have to do now is prepare a separate text file for each user on your system that you want to tell the world about. Please remember KISS — keep it *short*, stupid! One screenful is quite enough. Give each file a name up to 8 letters long (no extension), and place it in directory */finger*.

Incidentally, if you give the **ifconfig** command again, you will now see that the send and receive counts on the loopback interface have increased. In other words, **ifconfig** is useful for checking that traffic is actually passing through an interface.

16: THE ftpusers FILE

The file */ftpusers* is probably the most critical control file in NOS, as it determines what individual users are allowed to do in your system. If you get this file wrong, you could be in for a few surprises!

The file contains a list of login names, passwords, top-level root directories and access rights. Although called *ftpusers*, the file is actually used for general access control, for TELNET and PPP as well as for FTP.

The format of each entry in *ftpusers* is as follows (all on one line):

```
login_name  password  root_dir[;root_dir]  permissions
                                                [ip_address]
```

For example, here is an extract from the *ftpusers* file listed in Appendix 3 (pages 322–323):

```
# Miscellaneous accounts requiring no password:
anonymous * /public 3
anon * /public 3
bbs * /public 3
guest * /public 3

# Special accounts:
superuser supasswd /public 67
ns9bob bobby /public 7
roberto robertspw / 127          # Login name > 6 characters

# Friendly visitors
ns9ken kenneth /public 7
ns9liz lizzie /public 7

# Unwanted visitors:
NS9NRD * /public/tmp 128         # Sorry, no access
```

There must be *exactly one space* between each field, and at least the first four fields must be present. The fifth field (ip_address) is only used for PPP.

(N.B. If you are already familiar with the UNIX file */etc/ftpusers*, be aware that this serves the opposite purpose of the NOS *ftpusers* file. The UNIX file contains a list of those users who are *not* allowed to use FTP, whereas the NOS file specifies those who *are* allowed).

Login Name

The login name may be any length, but only the first 8 characters are significant. In most cases the login name will be a callsign, and will appear in the From: field of any messages which the user subsequently sends. However, there's nothing to stop you having a login name like nebuchadnezzar, but your messages will then be From: nebuchadnezzar@ns9bob, which may not be particularly helpful to anyone who wants to reply to you!

In addition to ordinary login names, you should also include the names anonymous, anon, bbs and guest. These four accounts are intended for anyone who doesn't otherwise have an entry in *ftpusers*. You will allow people to login to any of these accounts with any password, but they will only have restricted read-only access to "safe" public directories.

Password

The password may be any string of characters, with no spaces or tabs. Note that the password is "in clear" (i.e. not encrypted), so it's not a good idea to allow anyone to access the root directory (*N:*\) where *ftpusers* resides. Otherwise, if somebody discovers the password of a user who does have access to *N:*\ (by monitoring the on-air packets from that user), it is then possible to read *ftpusers* and discover all the passwords.

For the anonymous, anon, bbs and guest accounts you should put an asterisk is the password field. This means that these users may use any password when logging in. By convention, it's usual for such users to give their callsign as the password.

Password Checking

It's important to realise that NOS only refers to *ftpusers* when users connect to the system with the **ftp**, **telnet**, **bbs** and **ppp** commands. If someone connects to the NOS BBS using ordinary AX.25 or NET/ROM, NOS does *not* ask for a password. In this case the login name becomes the user's AX.25 callsign.

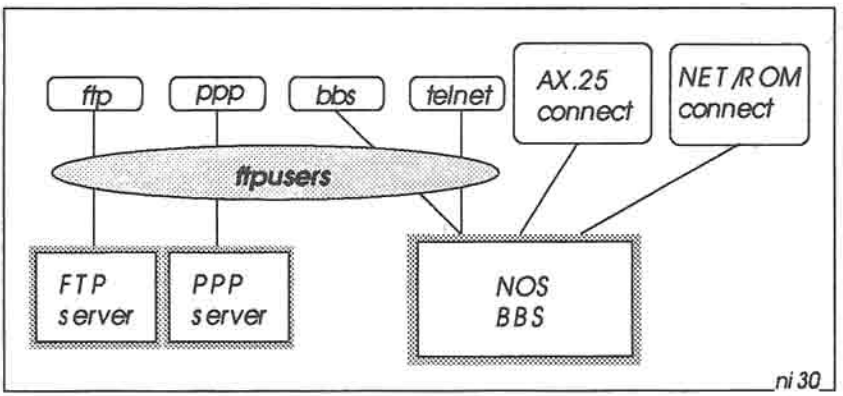


Fig 16-1: The *ftpusers* file authenticates access to the NOS BBS (when entered with the *bbs* or *telnet* command) and also to the FTP and PPP servers. If a user enters the BBS with an ordinary AX.25 or NET/ROM connect, there is *no* access check against the *ftpusers* file.

So how can you prevent ordinary AX.25 or NET/ROM users gaining password-free access? The brute-force way is to turn off the AX.25 and NET/ROM servers (with the **stop ax25** and **stop netrom** commands). An alternative, more subtle approach is to choose login names with 7 or more characters for accounts with sensitive access permissions. As AX.25 callsigns have a maximum length of 6 characters, these will never match any 7- or 8-character login name.

Root Directory

The root directory is the highest directory level which the user is permitted to access. This must be expressed as an absolute full

pathname from the NOS root (*N*:\), but without drive letter. (The reason for recommending that you use the **SUBST N:** command to define the NOS root should now be crystal clear: users are not able to go any higher up the file tree than *N*:\, so they can never access any files outside of NOS).

You are not restricted to one root directory — you can specify a list of them, using the semicolon as directory name separator. For example:

```
/public;/private
```

Note that there are no spaces in the directory list, and that all root directories are relative to the *NOS* root. (Newer versions of NOS are now becoming available which allow you to specify root directories relative to any *DOS* root, such as *D*:\ or *H*:\ or whatever, allowing you to specify LAN network drives or CD-ROM drives, for example).

When you login to the NOS BBS or use the **ftp** command, NOS automatically takes you to the first root directory in the list. Then you can give a **cd** command to change to another root directory. (This applies only to the **ftp** command; you cannot change to a different root in the BBS).

Access Permissions

The access permissions are expressed as a numerical code derived from Table 16-1 opposite.

To determine the permissions code for a user, add up all the values in the table which apply to the user. For example, for ns9bob's account, the permissions code is 7; i.e. 1+2+4, meaning that Bob can read, create, write and delete files in the */public* directory, or in any directory below */public*. Anybody logging in as anonymous, anon, bbs or guest has permissions code 3, meaning that they can read or create files in or below */public*, but they cannot update or delete any existing files.

The meanings of the other permissions are discussed later.

Note that anyone could masquerade as Bob, and login using AX.25 or NET/ROM with the AX.25 callsign NS9BOB. In this case, because the callsign matches Bob's password entry, *the user immediately gains Bob's access permissions, without being asked for a password.*

That is why Bob has two password entries: ns9bob and roberto. Account ns9bob is potentially vulnerable to intrusion from an AX.25 station, and is therefore access is restricted to */public*.

Account roberto can never be accessed by an AX.25 station (the login name roberto is 7 characters long), and so general access is allowed to all directories and all gateways. Provided that Bob only uses this account locally — that is, not over the air where other people could see his password when he logs in — he is perfectly safe from intrusion.

```
ftp and telnet

   1      read files
   2      create new files
   4      write/delete existing files

telnet only

   8      allow AX.25 Gateway access
  16      allow Telnet Gateway access
  32      allow NET/ROM Access
  64      allow Remote control
 128      Disallow all access

ppp only

 256      PPP connection
 512      peer ID/password lookup

miscellaneous

1024   disallow send commands (except to sysop)
2048   disallow read commands
4096   disallow third-party mail
8192   this station is a known BBS
```

Table 16-1: Access Permissions in the *ftpusers* file. (N.B. Some versions of NOS do not support all of these permissions).

Selective Read-Only Permission

You may wish to give users general read/write/delete permissions for a particular directory such as */public*, but how can you prevent them

writing to sensitive files below */public*? For example, in the **NOSview** distribution, the *masters*, *nosview* and *nosdocs* directories are located below */public*, and you certainly don't want people tampering with them!

This is where DOS comes to the rescue — you can use the DOS **ATTRIB** command to add the read-only permission to all the files you want to protect:

```
N:\>  ATTRIB +R N:\PUBLIC\MASTERS\*.*
N:\>  ATTRIB +R N:\PUBLIC\NOSVIEW\*.*
N:\>  ATTRIB +R N:\PUBLIC\NOSDOCS\*.*
```

17: HANDS ON — FTP

With FTP (File Transfer Protocol) you can connect to another NOS station, browse around the filesystem there, and transfer ASCII or binary files to or from that system — provided you have permission to do so, as determined by that station's *ftpusers* file.

Before using FTP, you should use **cd** or **pwd** to select a safe NOS directory on your system. For example, you don't usually want to download files into your NOS root directory. It's best to select the default directory */dump/record*:

```
net>  cd /dump/record
```

This directory corresponds to the SUBSTituted *R:* drive. Thus you can examine any file in this directory by selecting **F3** *R:**.* when in **VIEW**.

Initialising FTP

Prior to using FTP, you can give the **ftype** command to set the mode of file transfer:

```
net>  ftype binary
```

This sets the default transfer mode to 8-bit binary (also known as *image* mode), and is normally good for the transfer of *all* files, ASCII and binary.

However, if you suspect that the path will only handle 7-bit file transfers, you can change the mode, using the command:

```
net>  ftype ascii
```

Starting an FTP Session

The hands-on session described in this chapter is summarised in Fig 17-1 opposite.

The syntax of the FTP command is:

```
ftp host
```

To try out FTP, you can start a session with yourself, with the command:

```
net>  ftp ns9bob
```

(Assuming you are NS9BOB).

NOS responds by starting a new session window, which displays:

```
Resolving ns9bob... Trying ns9bob:ftp...
FTP session 1 connected to ns9bob
220 ns9bob FTP version 911229 (PA0GRI v2.0m) ready on Sun Aug
                                                23 05:45:41 1992
Enter user name (ns9bob):
```

As you are logging in as ns9bob, you can can just hit **CR**; if you are logging in with any other account name in *ftpusers*, give that name.

Later, when you go on-air and log into somebody else's system, you may not know if you have an account on that system. In that case, log in as anonymous.

The FTP server now requests your password:

```
331 Enter PASS command
Password:
```

Now enter the password for ns9bob (bobby) and hit **CR**. The password does not echo. (If you log in to another system as anonymous, give your callsign as the password).

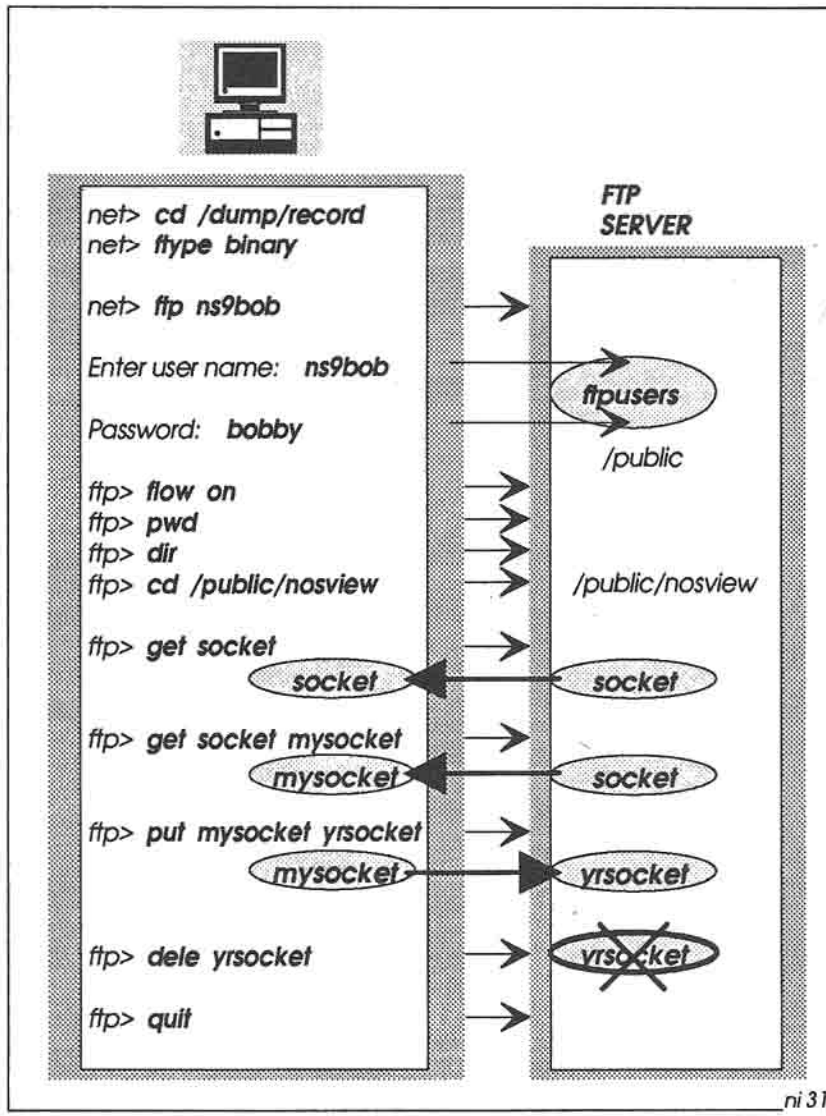


Fig 17-1: A typical FTP session. After connecting with the server and entering your login name and password, FTP takes you to the directory listed in your *ftpusers* entry (in this case, */public*). You can then transfer files between client and server.

If you get the password wrong, FTP responds with Permission denied and an ftp> prompt. To try again, give the **pass** command, together with the password:

```
ftp>  pass bobby
230 Logged in
```

If you subsequently decide to log in as another user, give the **user** command:

```
ftp>  user roberto
```

FTP Subcommand Summary

Having logged in, you can now give several subcommands at the ftp> prompt. Here is their syntax:

ascii
batch [on|off]
binary
cd *remote_dir*
dele *remote_file*
dir [*remote_dir* | *remote_file* [*local_file*]]
flow [on|off]
get *remote_file* [*local_file*]
hash
list [*remote_dir* | *remote_file* [*local_file*]]
ls [*remote_dir* | *remote_file* [*local_file*]]
mget *remote_file* [*remote_file* ...]
mkdir *remote_dir*
mput *local_file* [*local_file* ...]
nlst [*remote_dir* | *remote_file* [*local_file*]]
pass *password*
put *local_file* [*remote_file*]
pwd
quit
rmdir *remote_dir*
type [**a** | **i** | **l** *bytesize*]
user *username*
verbose *n*

n=0: errors only
1: + summary
2: + progress
3: + hash

Let's try some of these. First, it's a good idea to turn flow control on. Some FTP commands can produce several screenfuls of output, which will zip by too quickly unless you say:

```
ftp>  flow on
```

Now check the remote directory that you have logged into:

```
ftp>  pwd
257 "/public" is current directory
```

This will be the directory listed in the *ftpusers* entry for this user.

Now get a full directory listing:

```
ftp>  dir
200 Port command okay
150 Opening data connection for LIST /public
masters/        8:05  8/13/92   nosdocs/         8:05  8/13/92
nosview/        8:05  8/13/92
3 files.  4,079,616 bytes free.  Disk size 41,246,720 bytes.
LIST : 169 bytes in 0 sec (1158/sec)
226 File sent OK
```

If you just want a list of filenames, without sizes or date/timestamps, just give the **ls** (list) command instead.

Now change directory to *nosview*, and get a full listing of files there:

```
ftp>  cd /public/nosview
257 "/public/nosview" is current directory
ftp>  dir
200 Port command okay
150 Opening data connection for LIST /public/nosview

{then follows a screenful of filenames}

-More-
```

Then repeatedly hit the spacebar to get the rest of the listing.

Downloading Files

To get a copy of one of these files (say, *socket*) into your local download directory, use the **get** command:

```
ftp>  get socket
200 Type i OK
200 Port command okay
150 Opening data connection for RETR /public/nosview/socket
RETR socket: 3267 bytes in 0 sec (9497/sec)
226 File sent OK
```

You can now examine your local directory (*/dump/record*) to confirm that the file is indeed there. To do this, escape to the Session Manager, and give the **dir** command. NOS should then report that the file *socket* is there. You can then read the file, with the command **more socket**.

(You will probably prefer to use **VIEW** instead of the NOS **dir** and **more** commands. In that case, just hot-key to **VIEW,** then **F3** *R:\SOCKET* to read the file).

Now return to the FTP session, with **ALT-F1**. In the **get** request above you simply said **get socket**, which means that the local copy will have the same name as the original file. To give the downloaded copy a different name, just include the new name in the **get** command:

```
ftp>  get socket mysocket
```

Monitoring the Transfer

When you are downloading files to yourself in this manner, the transfer rates are obviously high, because the transfers are taking place completely within your system; speeds of thousands of bytes per second are typical. Unfortunately, when you use FTP in earnest over the air, the rate will drop to hundreds or even tens of bytes per second, depending on the speed of the radio link and on other traffic on the channel. This means that file transfers can take a very long time.

To monitor the progress of a transfer, you can give the hash command:

```
ftp> hash
```

Now, whenever you transfer information, FTP will display a hash character (#) for every 1024 bytes transferred. Try getting a large file:

```
ftp> get tcp
200 Type i OK
200 Port command okay
150 Opening data connection for RETR /public/nosview/tcp
##############
RETR tcp: 13591 bytes in 1 sec (12503/sec)
226 File sent OK
```

You will see 13 # marks, corresponding to a file length of about 13 KB. This is an extremely useful indication that things are still happening.

If you want to download several files, you can use the multiple get (**mget**) command. It's usual to use a wildcard with this command. For example, to get all filename starting with the letters *re*:

```
ftp> mget re*
```

In this case, the downloaded copies have the same names as the originals.

Uploading Files

To upload files from a local directory to a remote directory, you use the **put** command. For example, go back to the *public* directory, then upload the file *mysocket*:

```
ftp>  cd /public
257 "/public" is current directory
ftp>  put mysocket yrsocket
200 Type i OK
200 Port command okay
150 Opening data connection for STOR /public/yrsocket
###
226 File received OK
STOR yrsocket: 3267 bytes in 0 sec (11626/sec)
```

Now give the **dir** command to confirm that the upload was successful:

```
ftp>  dir
200 Port command okay
150 Opening data connection for LIST /public
masters/        8:05  8/13/92   nosdocs/         8:05  8/13/92
nosview/        8:05  8/13/92   yrsocket  3,267  7:19  8/25/92
4 files.  4,079,616 bytes free.  Disk size 41,246,720 bytes.
LIST : 201 bytes in 0 sec (1158/sec)
226 File sent OK
```

Deleting Files

You can now try deleting this file:

```
ftp>  dele yrsocket
```

A further **dir** command should confirm that the file has gone.

Note that the **dele** command was successful because NS9BOB's entry in *ftpusers* has permission code 7, which means he is allowed to read, write, create *and delete* files in directory */public* downwards. If the permission code were only 3 (as it should be for the *anonymous* account, for example), it would not have been possible to delete the file.

Finally, when you have done all you want to in the FTP session, you terminate with the **quit** command:

```
ftp>  quit
221 Goodbye!
```

FTP is nothing but friendly!

Automated Login

When you use FTP to access other stations, you can save time in the login process by setting up the file */net.rc*, which contains all the information needed to login automatically:

```
# NET.RC
# Format:
# ------------------------------------------------------------
# <hostname> <username> <password>
# ***********  EXACTLY ONE SPACE BETWEEN FIELDS  ********
# ------------------------------------------------------------
loopback ns9bob bobby
ns9ken ns9bob mypasswd
ns9liz anonymous ns9bob
```

When you now give the command **ftp loopback**, the FTP client will automatically supply the login name ns9bob to the FTP server, and will then automatically provide the password bobby when asked.

Similarly, when you say **ftp ns9ken**, NOS will provide the login name ns9bob and password mypasswd — see Fig 17-2. For the login to be successful, the login name and password must match an entry in Ken's *ftpusers* file.

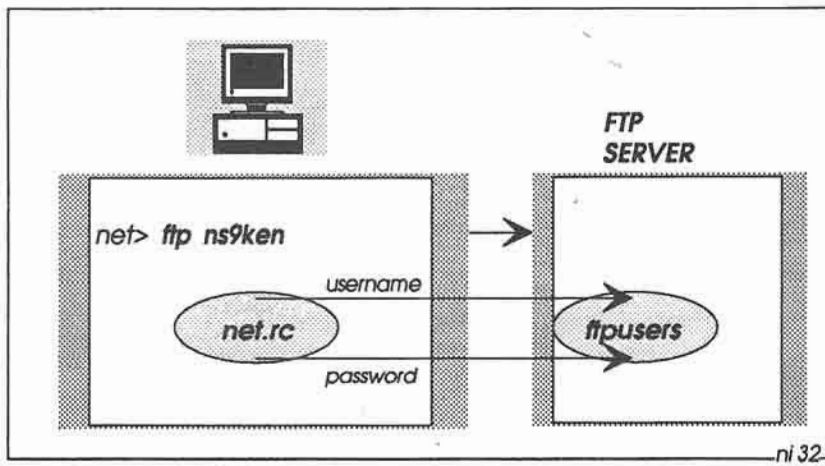


Fig 17-2: The *net.rc* file contains username and password entries which are supplied automatically to the server when you give an *ftp* command.

When you say **ftp ns9liz**, NOS will log you into Liz's FTP server as *anonymous* and provide your callsign ns9bob as the password. In this case you will gain access to the directory specified in the anonymous entry in Liz's *ftpusers* file.

Binary File Transfers

The transfer of 8-bit binary files works well if the path to the other station supports 8-bit data all the way. You simply set **ftype** to binary as explained earlier, and then **get** or **put** the files you need.

However, if any part of the path only supports 7-bit data (and very often you don't know), 8-bit binary transfer is out. In this case you'll need to convert the 8-bit file to a 7-bit ASCII format, transfer the 7-bit file, and then have the sysop at the other end convert it back to its original 8-bit format.

The **NOSview** distribution provides two utilities to do this: **UUENCODE** and **UUDECODE**. See Fig 17-3. The letters UU betray the origin of these commands — they were originally developed for Unix-to-Unix file transfers. The UU commands are DOS programs, so you need to shell out from NOS to use them.

To encode an 8-bit binary file (say *AX25.COM*) into its 7-bit uuencoded equivalent (say *AX25.UU*), shell out to DOS and then give the **UUENCODE** command:

```
net>  shell
EXIT TO RETURN TO NOS  N:\>  UUENCODE < AX25.COM > AX25.UU
EXIT TO RETURN TO NOS  N:\>  EXIT
```

On returning to the NOS FTP session, set the transfer mode to ASCII and send the uuencoded file to the other station:

```
ftp>  type ascii
ftp>  put ax25.uu
```

At the other end, file conversion back to binary is achieved with the DOS command:

```
N:\> UUDECODE < AX25.UU > AX25.COM
```

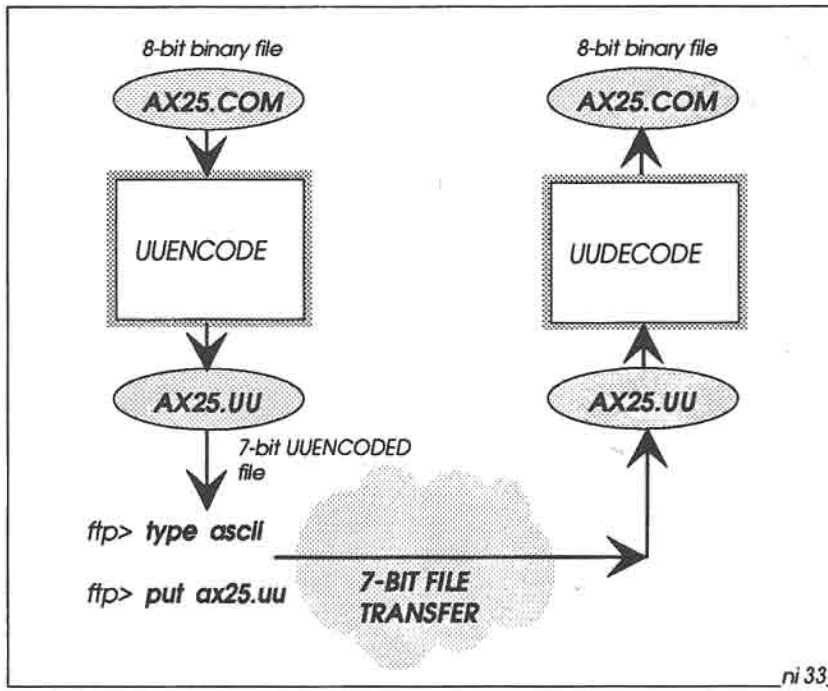


Fig 17-3: If the network can only support 7-bit file transfers, it is necessary to encode an 8-bit binary file into 7-bit ASCII format before sending it. At the receiving end, the file is decoded back to its original 8-bit format. You have to escape to DOS to encode and decode the file.

Aborting an FTP Command

Sometimes things don't go to plan, and you may want to abort an FTP session. You can use the NOS **reset** command as a last resort, but it's usually much cleaner to terminate the session with the NOS **abort** command.

To do this, escape to the Session Manager, then give the **abort** command, with a session number if necessary. For example, to abort an FTP transfer in NOS session 3:

```
net>  abort 3
```

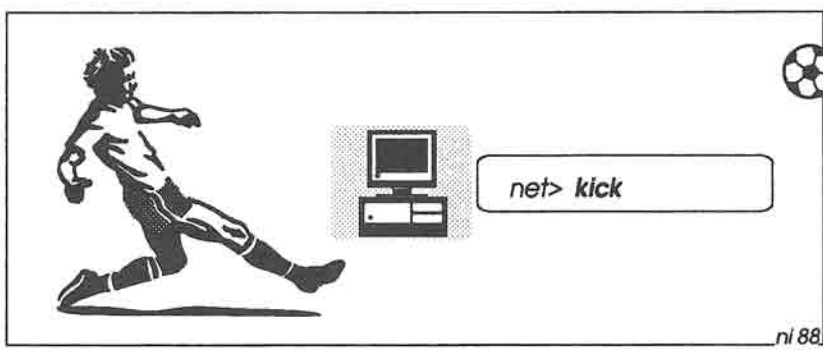
The **abort** command only works with FTP transfers. Be aware that when you abort a file transfer, you may finish up with a partial copy of the file on the destination machine. You will then have to delete the file manually.

Kick the Session into Life!

Sometimes you'll find that sessions appear to stop running for no apparent reason. To bring them back to life, you can try the **kick** command. For example, to kick session 3:

```
net>  kick 3
```

You can use this command for all types of session, not just **ftp** sessions.



18: NOS BBS — THE BIG PICTURE

In the AX.25 PBBS world, it's common for users to run their own personal messaging system (PMS), with automatic forwarding in both directions between the PMS and a local PBBS. People can log into the PMS, but they can only leave private messages there for the PMS owner.

In the NOS world, you also have a personal messaging system. Here it is called the NOS BBS, and is very much more powerful and flexible than an AX.25 PMS.

You can login to your own NOS BBS with the **bbs** command, and into a remote BBS with the **telnet** command. Ordinary AX.25 users can also login to the NOS BBS, simply by connecting to the station in the usual way.

The NOS BBS comprises six main parts:

- The Mailbox Server
- The Built-in NOS Mailer
- File Server
- Gateway Server
- Finger Server
- Remote Sysop Server

See Fig 18-1.

The *mailbox server* contains a command interpreter which understands simple one- or two-letter commands to invoke the BBS services.

The *NOS mailer* lets you create, send and receive mail. You can transport mail to other NOS stations using the SMTP protocol, and collect your mail using the POP protocol. You can also forward and reverse-forward mail to and from the conventional AX.25 PBBS network.

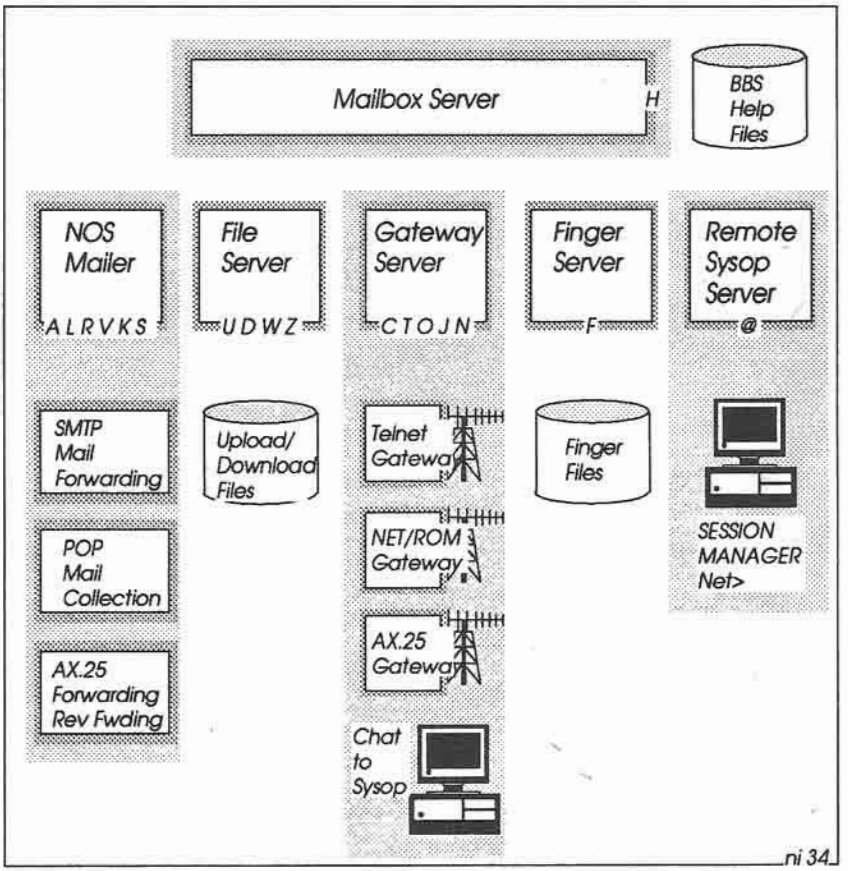


Fig 18-1: The NOS BBS. The individual capital letters in the boxes (e.g. A L R V K S) are BBS commands.

The *file server* lets you upload and download files, and, if you have permission, delete them also. File transfers are in 7-bit format. If you want to transfer 8-bit binary files, it's necessary to encode them into 7-bit format first. The file server is really intended for AX.25 users who connect to the BBS — NOS users would normally use FTP instead.

The *gateway server* lets you make outgoing AX.25, NET/ROM and TELNET connections to other stations. The server is likewise intended for AX.25 users, so that after connecting to the BBS they can then initiate a new connection to another station, possibly on another network. The server also provides a chat facility, letting you talk direct to the sysop.

The *finger server* is also intended for AX.25 users, letting them read the finger files on the system; NOS users would normally use the **finger** command instead.

The *remote sysop server* lets you communicate directly with the NOS Session Manager on the system, giving you the power to do almost anything. This is particularly useful if you need to change settings on a difficult-to-get-to hilltop system, for example, without physically having to go there. For security reasons you need to have the sysop permission in the *ftpusers* file on the remote system, and you may also have to give an additional password, before you can go ahead and do your worst.

Mail Handling

Fig 18-2 shows the principal building blocks for handling mail within NOS. Starting at the top of the diagram, the main steps in composing, sending and receiving mail are as follows:

- BBS Login
- Composing Mail
- Forwarding Mail with the SMTP Client
- Receiving Mail with the SMTP Server
- Reading the Mail
- POP Mail Collection
- PBBS Forwarding

BBS Login

There are several ways of logging into the BBS, depending on whether the BBS is local or remote, and whether the connection to the BBS is made via **telnet** or ordinary AX.25.

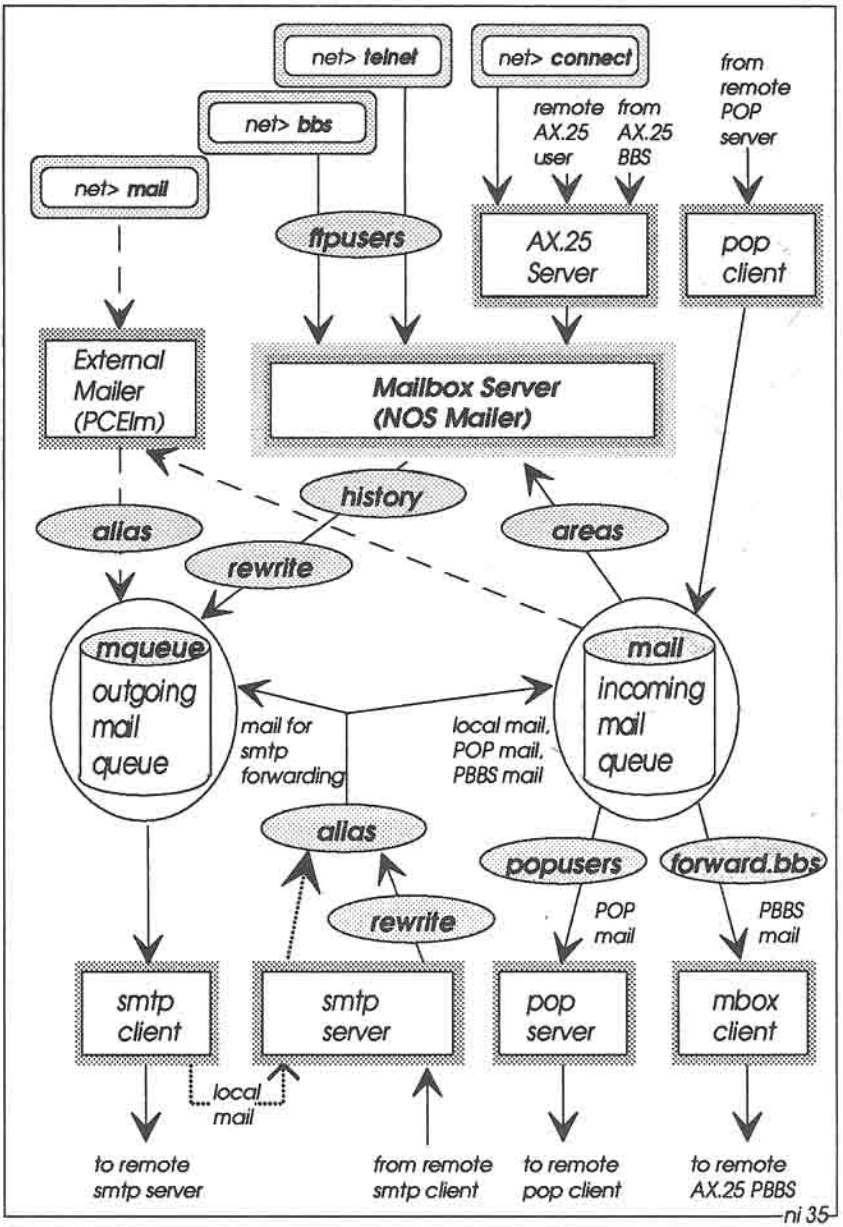


Fig 18-2: The NOS BBS and PCElm external mailer.

To login to your own built-in BBS (see Fig 18-3 below), give command:

```
net> bbs
```

Alternatively, if you prefer to use an external mailer, the command to start the mailer is:

```
net> mail
```

Note that before starting NOS and giving the **mail** command, you must set up the DOS environment variable MAILER, to specify which mailer you want to use. For example, in *NOSENV.BAT*:

```
SET MAILER=N:\PCELM.EXE
```

You can even run an external mailer independently of NOS if you wish. For example, at the DOS prompt:

```
N:\> PCELM
```

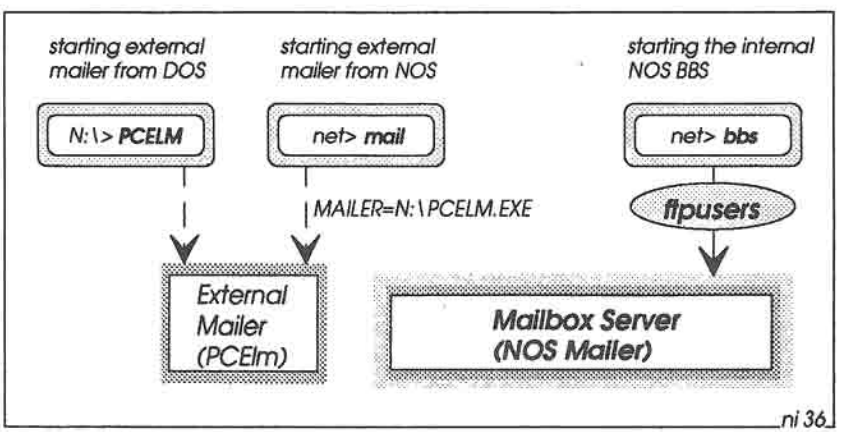


Fig 18-3: Logging in to your own NOS BBS (or external mailer).

To log into a remote BBS (Fig 18-4 below), you will normally use the **telnet** command; e.g.

```
net>  telnet ns9ken
```

When you use **telnet** (or **bbs**), NOS will ask you for a user name and password. These must match an entry in *ftpusers*. If you can't provide a suitable name and password, you could try connecting to the BBS with the AX.25 **connect** command instead. For example:

```
net>  connect tnc0 NS9KEN-5
```

This sets up an ordinary AX.25 connection, and NOS will not ask you for a login name or password.

Ordinary AX.25 users can also connect to the NOS BBS, and AX.25 PBBS mailboxes can connect when they have mail to forward to the BBS.

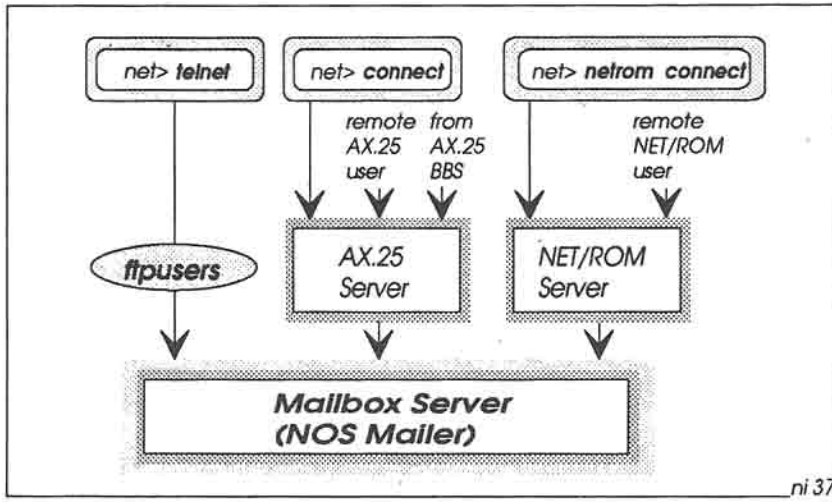


Fig 18-4: Logging in to a remote NOS BBS (using *telnet* or an ordinary AX.25 or NET/ROM connection).

Composing Mail

Once logged in to the BBS, you can then give commands to the mailer (see Fig 18-5). To compose a message for sending, you use a command like:

```
NS9BOB-5} sp ns9liz@ns9liz
```

When you have finished preparing the message, you terminate it with **CTRL-Z**, in a similar way to conventional PBBS mailbox systems. The NOS mailer then checks to see if it is a bulletin which it has already received (the file */spool/history* contains details of all received bulletins).

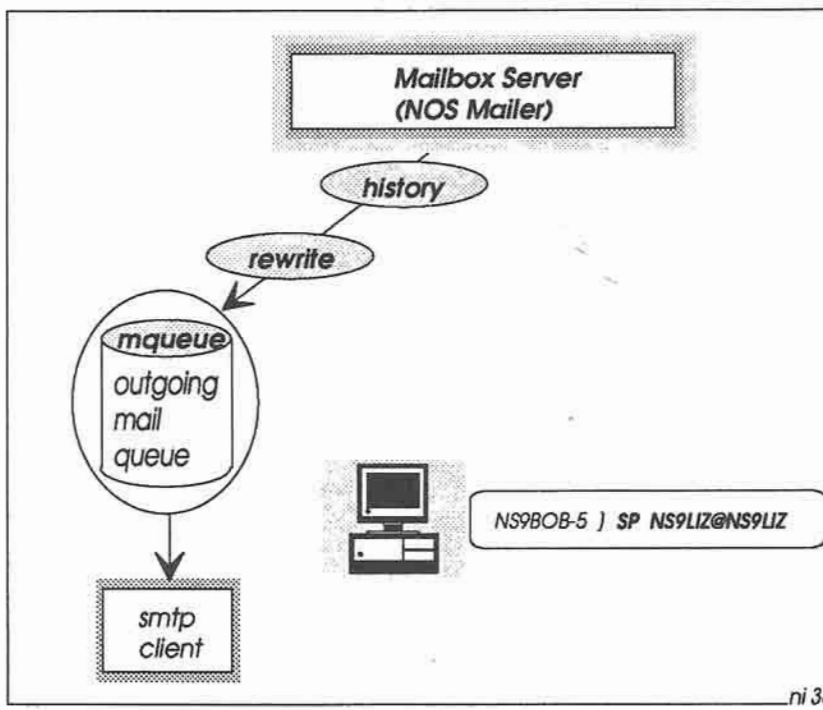


Fig 18-5: The mailer puts the message into the outgoing mail queue (*/spool/mqueue*).

Also, if necessary, the mailer converts the destination address into a more suitable format (by reference to the file */spool/rewrite*) — the *history* and *rewrite* files are described later. The mailer then places the message in the outgoing mail queue, in directory */spool/mqueue*, and wakes up the local SMTP client.

Forwarding Mail with the SMTP Client

The SMTP client takes the message from the outgoing mail queue, and attempts to forward it to its destination (Fig 18-6).

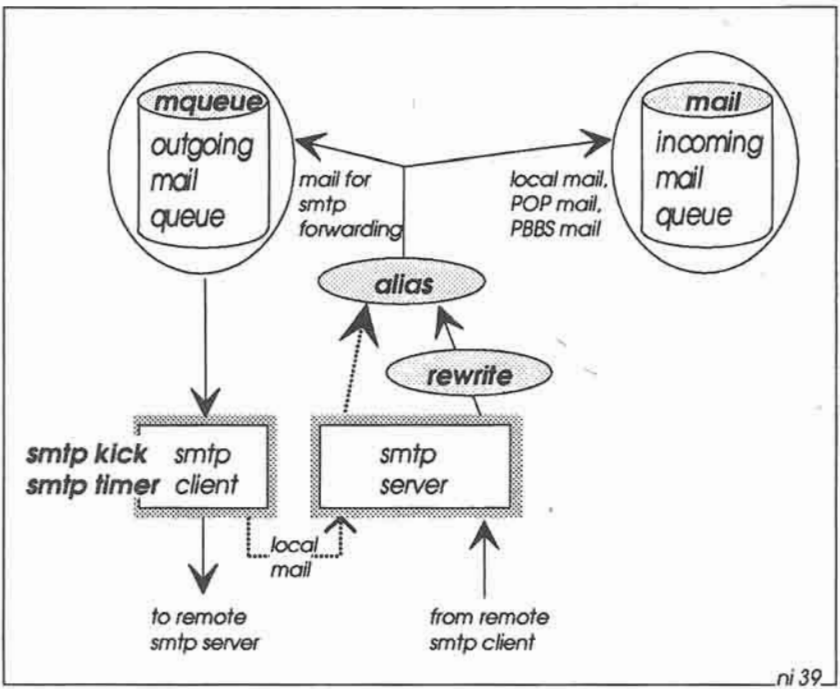


Fig 18-6: To forward the mail, the SMTP client connects to the addressed SMTP server. This may be a local server (for local mail), or a remote server across the network. The *smtp kick* and *smtp timer* commands bring the SMTP client to life.

If the destination is on a remote system, the SMTP client will make a connection with the SMTP server on that system. If the message is addressed to a user on the local system, the SMTP client will make a direct connection with the SMTP server on the same system.

Normally the SMTP client attempts to forward messages as soon as the mailer places them on the outgoing mail queue. However, if the attempts are unsuccessful, the client will try again at regular intervals — you can set up the retry interval with the **smtp timer** command.

Sometimes you may find that the SMTP client appears to have gone to sleep for too long (it doesn't seem to be forwarding any messages). In this case you can give it a nudge with the **smtp kick** command.

Receiving Mail with the SMTP Server

The SMTP server listens continuously for incoming connection requests from local and remote SMTP clients (again, see Fig 18-6). When a connection is made, the client passes the message to the server. The server then checks to see if the addressee has an entry in the file */alias*. If there is an alias, it is expanded; depending on the alias entry, this may result in several copies of the original message.

The server then places the original message (and any copies) into the appropriate queue. Messages to be sent onwards to another station go back into the outgoing mail queue, for forwarding by the SMTP client.

Messages intended for local users go into the incoming mail queue (by default, in directory */spool/mail*), and a message pops up on the screen saying that mail has arrived.

Each message goes into a text file in the incoming mail queue. There is a separate file for each addressee; for example, all mail addressed to ns9bob goes into file */spool/mail/ns9bob.txt*, mail addressed to tcpip goes into */spool/mail/tcpip.txt*, and so on.

Reading the Mail

To read incoming mail, you first have to select the correct mailbox *area*. The default mailbox area name is the same name that you used to login to the BBS. If you logged in as ns9bob, the default area is

ns9bob, which means that you can read all messages in */spool/mail/ns9bob.txt* (using the BBS **R** command).

If you logged in via an AX.25 connection (and therefore did not provide a login name or password), the default area is your callsign.

The file */spool/areas* defines public areas to which you are allowed access (in addition to your default area), and you can switch to any of these areas with the BBS **A** command.

For example, the BBS command **A TCPIP** (or **a tcpip**) switches you to the tcpip area, and then you can read all the messages and bulletins in */spool/mail/tcpip.txt* (see Fig 18-7).

You can only switch to areas listed in the *areas* file, plus your own private login area. If you want to see the private mail of another user, you have to login first as that user.

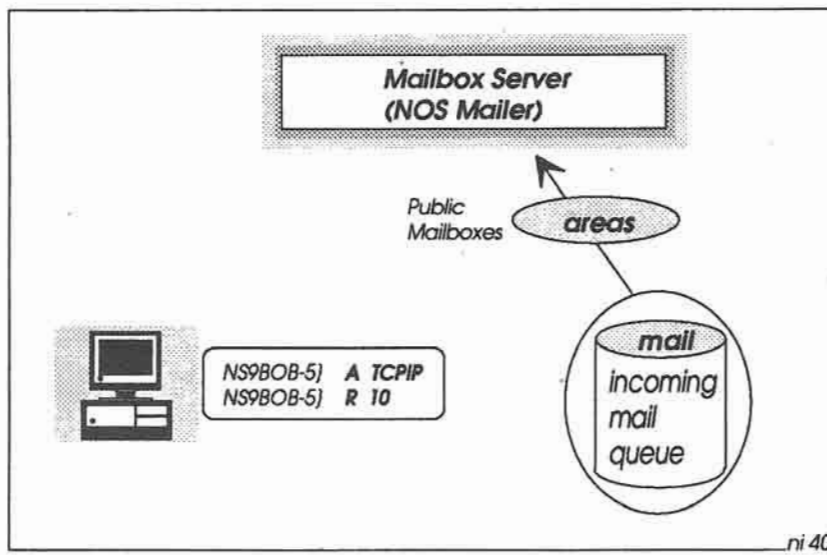


Fig 18-7: Reading the mail. The file */spool/areas* specifies public bulletin mailboxes.

POP Mail Collection

Normally, mail delivery to a remote host takes place automatically, provided that the remote host is actually switched on and ready to receive it. In many instances, however, this will not be the case — many people shut down their systems when they are not actually using them. This can lead to network overload, where a machine makes repeated attempts to forward mail which can never succeed.

A solution to this problem is to nominate a machine as a "poste restante" — a system which will hold mail indefinitely for people, and which will only deliver the mail when those people wake up and ask for it. See Fig 18-8.

The poste restante machine arranges for such mail to go into the incoming mail queue in *.txt* files. The POP server on this machine listens continuously for requests from POP clients on neighbouring machines. When it receives a POP request, the server checks the user name and password against the file */popusers*, and if they match, the server transfers the corresponding *.txt* file across the network to the client.

The POP client on the receiving machine then deposits the file in the incoming mail queue, and displays a message on the screen saying that the mail has arrived.

PBBS Forwarding

Mail for forwarding onto the PBBS network is handled specially in NOS. Irrespective of whether you create messages locally for PBBS forwarding, or whether they come into the system from another machine, they eventually find themselves in one or more *.txt* files in the incoming mail queue. See Fig 18-9.

On their journey to the incoming mail queue, the destination addresses for the messages will probably be changed, by reference to the *rewrite* file. PBBS addressing rules are quite different from SMTP, varying considerably throughout the world, and the *rewrite* file serves as an address swap file — it contains all the necessary rules for converting addresses into formats which the PBBS network will understand.

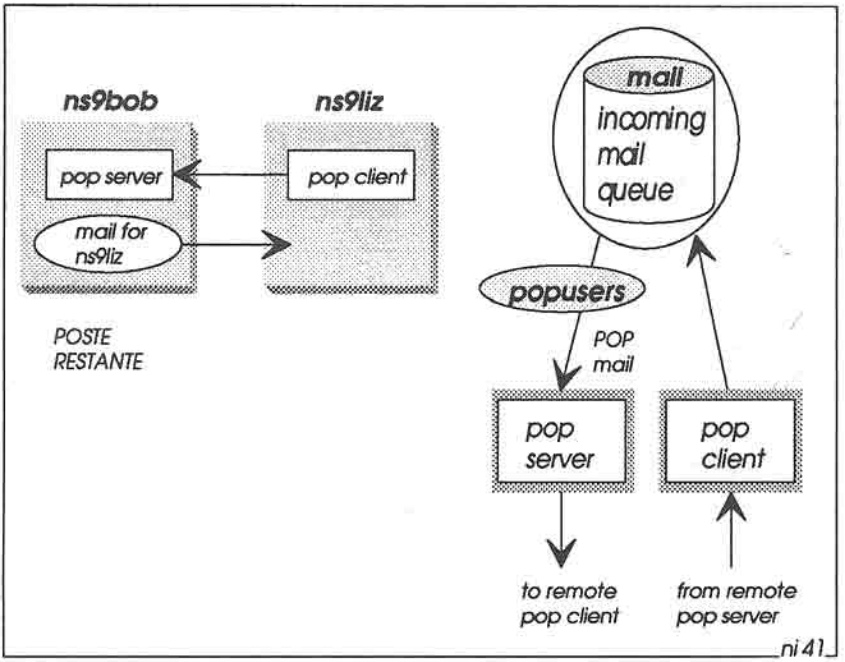


Fig 18-8: POP Mail Collection. Mail for ns9liz resides in Bob's incoming mail queue awaiting collection. When Liz is ready to collect the mail, her POP client connects to Bob's POP server, which then forwards the mail. The file *popusers* in Bob's system authenticates Liz's POP request.

At regular intervals, the mailbox client wakes up and examines the file */spool/forward.bbs*, to see if the time is right to forward PBBS mail. You can define different timeslots in *forward.bbs* for different PBBSs if you wish. If there is any outstanding PBBS mail in the incoming mail queue when a timeslot becomes valid, the mailbox client connects to the designated PBBS and then forwards the mail to it.

After forwarding any mail to the PBBS, the roles are then reversed and the PBBS forwards any outstanding mail that it has for the NOS system.

You can define the mailbox client scan interval, using the NOS **mbox timer** command. If you want to force the client to forward mail immediately to a PBBS, you can use the NOS **mbox kick** command.

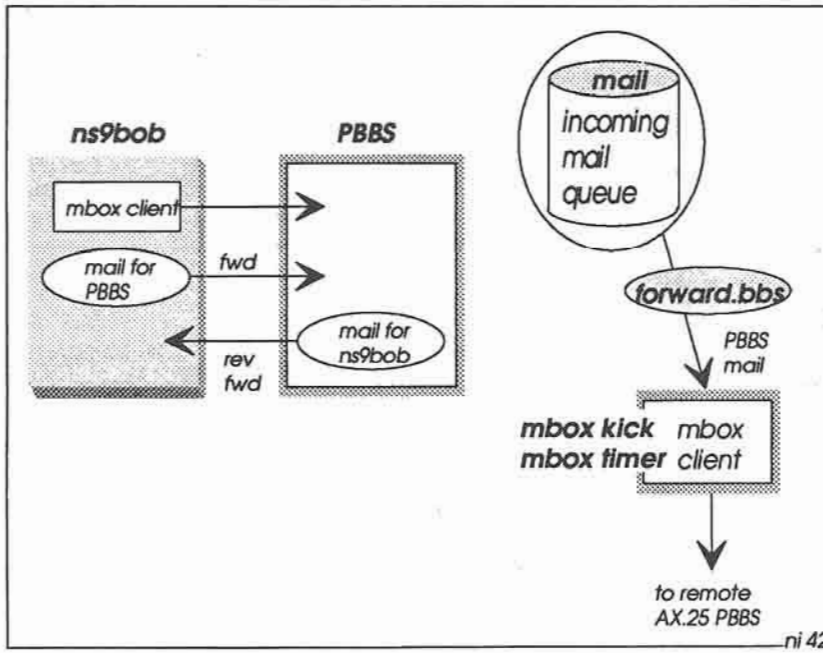


Fig 18-9: PBBS forwarding. Mail for forwarding onto the PBBS network resides on the incoming mail queue. The file */spool/forward.bbs* specifies when the mailbox client is to forward the mail, and to which PBBSs it is to go. The *mbox kick* and *mbox timer* commands bring the mailbox client to life. When forwarding is complete, any outstanding mail on the PBBS is reverse-forwarded to the NOS system.

19: SETTING UP THE NOS BBS

Before using the NOS BBS, you have to set up a number of control files. In this chapter we look at two of the files which you need for simple mail handling via the AMPRnet (using SMTP):

- */spool/areas*
- */autoexec.nos*

Examples of these files are included in Appendix 3.

The *areas* file

The file */spool/areas* (page 311) lets you set up public message areas in your BBS. The first character of each area name must be the first character on the line. Any other lines containing descriptive information must start with a space.

```
all     ........  General chit-chat.
tcpip   ........  General TCP/IP and NOS messages.
sysop   ........  Messages for Bob (NS9BOB).
```

Users logging in to the BBS can list this file by giving the **A** command by itself. Or they can select a particular area with a command like **A TCPIP**, and then access public bulletins in that area.

The *autoexec.nos* file

The NOS BBS requires several parameters to be defined in *autoexec.nos* (pages 312–317). The first of these is **ifconfig description**:

```
ifconfig tnc0 description "144.625 MHz port"
```

This description appears on the screen in response to the BBS **P** (Ports) command. Note that you need to enclose the description text with inverted commas (" ").

Starting the Servers

Autoexec.nos contains several commands to start the servers required for mail handling:

```
start ax25          # allow AX.25 users to log in to my BBS
start finger        # allow people to finger me
start pop2          # allow my BBS to reverse-forward mail
start pop3          #
start remote        # allow people to control my station
start smtp          # turn on SMTP mail forwarding
start ttylink       # allow people to chat to me
```

These **start** commands turn on the network servers which the BBS uses. If you don't want people to use some of these services, you can comment them out with the hash character; e.g.

```
# start       ax25
# start       finger
# start       pop2
# start       pop3
# start       remote
# start       ttylink
```

Third-Party Traffic

If you are permitted to handle third-party traffic, include this in *autoexec.nos*:

```
third-party   on
```

Configuring the SMTP Client and Server

To configure the SMTP client and server, include the following commands in *autoexec.nos*:

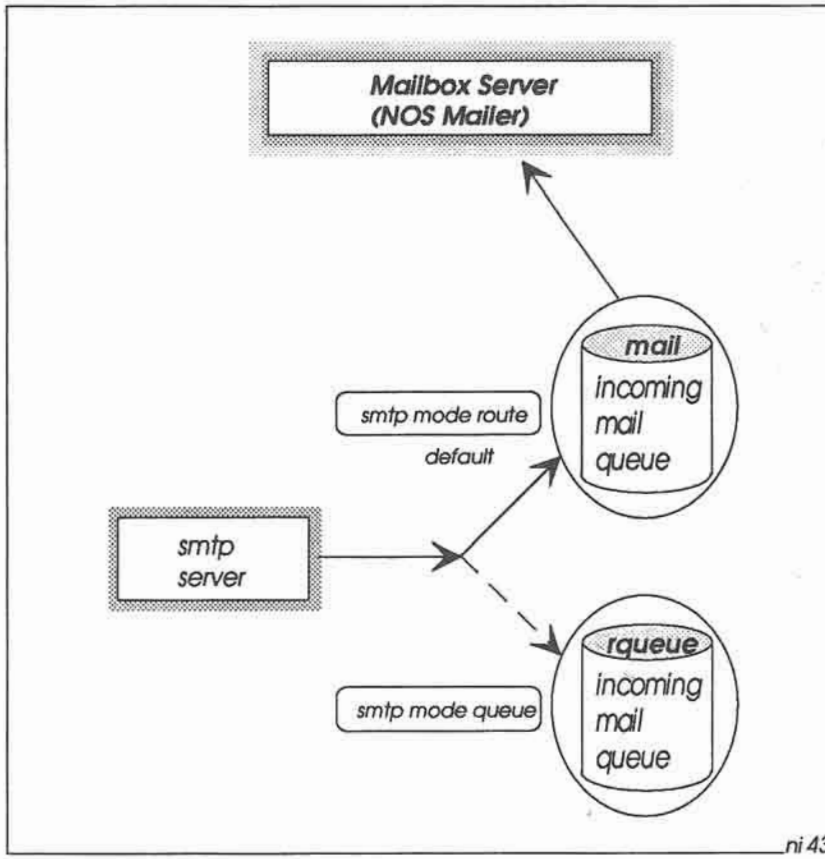


Fig 19-1: The *smtp mode* command specifies which queue will hold incoming mail. When the mode is set to *route* (the default), the mail goes into */spool/mail*, from where the NOS mailer can read it. If the mode is set to *queue*, the mail goes into */spool/rqueue*. Special software (not supplied with NOS) is then required to handle the mail in this queue.

```
smtp timer    600
smtp gateway  ns9sgw
smtp usemx    on
smtp mode     route
smtp kick
```

The **smtp timer** value is the time interval, in seconds, between automatic attempts at forwarding mail. Thus in this example the BBS will attempt to forward any outstanding mail every 600 seconds (10 minutes).

The **smtp gateway** (ns9sgw) is the name of the station to which the BBS will forward mail for destinations which do not appear in *domain.txt*.

The **smtp usemx** command specifies that SMTP is to use MX entries in *domain.txt* for forwarding mail. Chapter 23 describes this in more detail.

The **smtp mode route** command states that SMTP is to place incoming mail in the *M:* directory (*/spool/mail*), ready for the mailer to read it. This is the usual setting — see Fig 19-1.

As an alternative, you can change the command to:

```
smtp mode queue
```

In this case all incoming mail goes into the directory */spool/rqueue* instead. This directory is intended for use by different mail handling programs (not supplied with NOS), to allow you to experiment with alternative mailing strategies. You cannot read mail in this directory with the BBS mailer.

Configuring the Mailbox Client

Finally, you need to set up a number of parameters for the mailbox client (see the box opposite).

When **mbox attend** is set to on, users can start chat sessions with you using the **O** (Operator) command. If set to off, the BBS tells the user that the box is unattended.

```
mbox attend   on
mbox maxmsg   100
mbox motd     "Please use SP NS9BOB to leave a message"
mbox expert   on
mbox nrid     on
mbox password "Maximum 30-character password."
mbox secure   off

mbox qth      "[London]"
mbox utc      0
mbox zipcode  "1234567"
mbox fwdinfo  "BOBBBS"
mbox haddress "BB7BBS.#41.GBR.EU"
mbox smtptoo  off
mbox kick
```

The **mbox maxmsg** parameter lets you specifiy the maximum number of messages per mailbox area. Be aware that you should keep this number (and the number of areas) reasonably low, as the BBS reserves a corresponding amount of memory for the maximum number of messages.

The **mbox motd** is a short "message-of-the-day" that appears as a welcome message when a user logs into the BBS. Note that the text of the message must be enclosed in inverted commas (" ").

When **mbox expert** is on, the BBS will display a short prompt by default when a user logs in. When set to off, the prompt contains a list of all the command letters. The user can change the expert state with the BBS **X** command.

When **mbox nrid** (NET/ROM ID) is on, your callsign appears in the BBS prompt instead of the simple **>** prompt:

```
NS9BOB-5}
```

If you are running a NET/ROM node, your NET/ROM alias appears in the prompt as well:

```
#BOB:NS9BOB-6}
```

The **mbox password** is contained within inverted commas, and can be up to 30 characters long. The BBS uses this password to authenticate

a user who wishes to control the station remotely; how to do this is described in detail in Chapter 22.

The **mbox secure** command controls access to the BBS gateway commands. When set to on, users are not allowed to use the gateways.

The remaining mbox commands (**qth, utc, zipcode, fwdinfo, haddress, smtptoo** and **kick**) are to do with AX.25 PBBS forwarding, and are described in detail in Chapter 25. If you are not using PBBS forwarding, you can omit these commands from *autoexec.nos*.

20: THE NOS BBS COMMAND SET

This chapter summarises the NOS BBS commands available to you when you log in.

Logging In

To log into your own NOS BBS, you simply give the **bbs** command. To gain access to somebody else's BBS, you use the **telnet** command; e.g. **telnet ns9ken** will connect you to the NOS BBS on NS9KEN's machine.

Try logging into your own BBS:

```
net>  bbs
```

You will then see something like this:

```
Trying ns9bob:telnet...
Telnet session 1 connected to Local BBS
```

Having connected with the BBS, you now log in:

```
KA9Q NOS (ns9bob)
login:  ns9bob
Password:  bobby        (this does not echo)
```

The login name and password you use should be in the *ftpusers* file, described earlier in the Chapter 16.

[If you are logging in to a remote BBS, you probably won't know if you have an entry in their *ftpusers* file, so log in as **anonymous** or **bbs** instead, and give your own callsign in response to the **password**

prompt; your privileges will be restricted, but you should at least be able to get in to the box].

Remember, the requirement for a login name and password only applies if you connect to the BBS using **bbs** or **telnet**. Ordinary AX.25 or NET/ROM users connecting to the BBS will not be asked for them.

Assuming your login was successful, you now get a welcome "message-of-the-day" and the command prompt:

```
[NOS-H$]
Welcome ns9bob,
to the ns9bob TCP/IP Mailbox (911229 (PA0GRI v2.0m))
Please use  sp ns9bob  to leave a message for NS9BOB

NS9BOB-5} Current msg# 0 :
 ?,A,B,C,D,E,F,H,I,J,K,L,M,N,O,P,R,S,T,U,V,W,X,Z >
```

Help

The first command you can then try is the **?** command, to find out more about what BBS commands are available:

```
 ?,A,B,C,D,E,F,H,I,J,K,L,M,N,O,P,R,S,T,U,V,W,X,Z >
?
(?)help    (A)rea    (B)ye       (C)onnect (D)ownload (E)scape
(F)inger   (H)elp    (I)nfo      (J)heard  (K)ill     (L)ist
(M)busers  (N)odes   (O)perator  (P)orts   (R)ead     (S)end
(T)elnet   (U)pload  (V)erbose   (W)hat    (X)pert    (Z)ap
```

The actual commands you see here will depend on the version of NOS. Several early versions have a slightly different command set; in those versions, **C** means **Chat** (equivalent to **Operator**) and **G** means **Gateway** (equivalent to **Connect**)

You can use either lower-case or upper-case letters for NOS BBS commands.

The NOS BBS Command Set

The NOS BBS commands break down into 6 groups:

- NOS BBS User Interface: **B, E, H, I, M and X**
- NOS BBS Mailer: **A, K, L, R, S and V**
- NOS BBS File Server: **D, U, W and Z**
- NOS BBS Gateway Server: **C, J, N, O, P and T**
- NOS BBS Finger Server: **F**
- NOS BBS Remote Sysop Server: **@**

Let's look at them in detail.

NOS BBS User Interface Commands

B (Bye): This logs you off the BBS.

E (Escape): The **Escape** command lets you define an escape character which you can use to break out of a NOS Gateway session. The default escape character is **CTRL-X**. Thus, for example, when you want to finish an operator chat session (started with the **O** command — see below), you simply hit **CTRL-X** to return to the NOS BBS prompt.

H (Help): The **H** command by itself will give you a list of topics on which the BBS has further information:

```
HELP.HLP

USAGE
 H[elp] [<command-name>]

DESCRIPTION
The help command will display help for a given command.  The
help command by itself, displays this particular message.  To
get help for a specific command, enter "help" followed by a
space and then the name of the command you want described.

                          {etc etc}
```

This listing is actually the file */spool/help/help.hlp*, which is located along with all the other NOS BBS help files in directory */spool/help*. The help files are plain ASCII text, and you can modify them if you wish (for example, translate them into another language). They should not be longer than about 20 lines; otherwise when a user calls them up they will scroll off the top of the screen.

As explained in the example above, you can then request further help with a command such as **help area**. You can shorten this to **help a**, or even **h a**, as the **help** command simply displays the help file whose first letter matches the letter you give in the **help** command.

I (Info): The **Info** command displays the Info help file (*spool/help/info.hlp*). This is where you can describe your system, and give brief instructions on how to use the gateways and any other services you may offer to users logging in.

M (Mbusers): The **M** command displays the names of all users logged into the mailbox.

X (Expert): The **Expert** command toggles the prompt between a full prompt:

```
NS9BOB-5} Current msg# 0 :
 ?,A,B,C,D,E,F,H,I,J,K,L,M,N,O,P,R,S,T,U,V,W,X,Z >
```

and an abbreviated prompt:

```
NS9BOB-5}
```

The NOS BBS Mailer Commands

A (Area): The **Area** command lets you select a particular mail area within the BBS. All the mail addressed to ns9bob is in the mail area *ns9bob*, all the mail addressed to tcpip is in mail area *tcpip*, everything for sysop is in area *sysop*, and so on. This is somewhat different from the ordinary AX.25 PBBS mailer, which has only one mail area which holds all the mail, irrespective of whom it is addressed to.

When you log in, the default mail area is the same as your login name; see Fig 20-1. That is, if you log in as ns9bob, then your default area is *ns9bob*, and here you will find all the personal messages addressed to ns9bob. (These messages are contained in the text file */spool/mail/ns9bob.txt*).

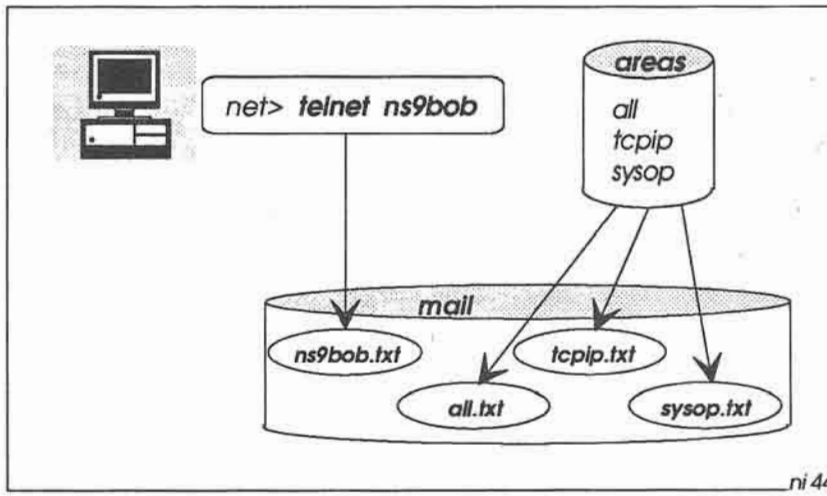


Fig 20-1: The file */spool/areas* specifies the public bulletin mailboxes.

Your default mail area is the only area where you are allowed to access personal messages. If you want to access messages addressed to ns9ken, then you will have to log in as ns9ken, whereupon your default mail area will now become *ns9ken*.

The NOS BBS automatically creates a new personal message area whenever a message arrives for someone who doesn't already have such an area. Thus when the very first message addressed to ns9liz arrives, for example, NOS creates the file */spool/mail/ns9liz.txt*.

K (Kill): The **Kill** command lets you mark mail for deletion from the current area, provided you have permission to do so (as determined by your account entry in *ftpusers*). For example, **K 10** marks message 10 for deletion. The message doesn't actually disappear until you log out, so if you have second thoughts before you log out you can still read the message.

L (List): The **List** command lets you list mail in the current area. The listing will contain a letter **Y** if you have already read a message, or an **N** if not.

R (Read): The **Read** command lets you read mail in the current area. For example, you say **R 10 13** to read message numbers 10 and 13 (or you can even say simply **10 13**; the letter **R** is not strictly necessary).

S (Send): This is the basic command for sending mail. As with AX.25 PBBSs, you append a second letter to the **S**, such as **P** or **B**; e.g. **SP** for personal messages, **SB** for bulletins, and so on.

There are two special **S** commands:

SR lets you reply to a message; e.g. **SR 10** to reply to message number 10, or **SR** by itself to reply to the message you have just read.

SF lets you forward (send a copy of) a message to someone else; e.g. **sf ns9liz@ns9liz** forwards the current message to Liz.

There are several methods of addressing mail, depending on whether or not the addressee is in *domain.txt*, and on whether NOS is to forward the mail to another NOS system (using SMTP), or to forward it via the AX.25 PBBS network. These methods are described in detail in later chapters.

V (Verbose): The **Verbose** command works the same way as the **Read** command, but displays not only the message itself but also the message header. This can be useful if there has been a message forwarding problem and you want to see how the message got to your BBS.

The NOS BBS File Server Commands

The NOS file server commands give access to the public files area, and are intended primarily for ordinary AX.25 users who have connected to the NOS BBS (**telnet** users would normally use **ftp** instead to access public files).

D (Download): The **Download** command lets you download *7-bit ASCII* files from the BBS to your system. For example: **d yourfoo.txt**. See Fig 20-2.

To download a binary file, you use the **DU (Download Uuencoded)** command instead; e.g. **du prog.exe**. In this case, the BBS will automatically use the built-in uuencode function to convert the binary file to ASCII, before downloading it to you. When downloading is finished, you have to convert the encoded file off-line back to its original binary form with the DOS **UUDECODE** command.

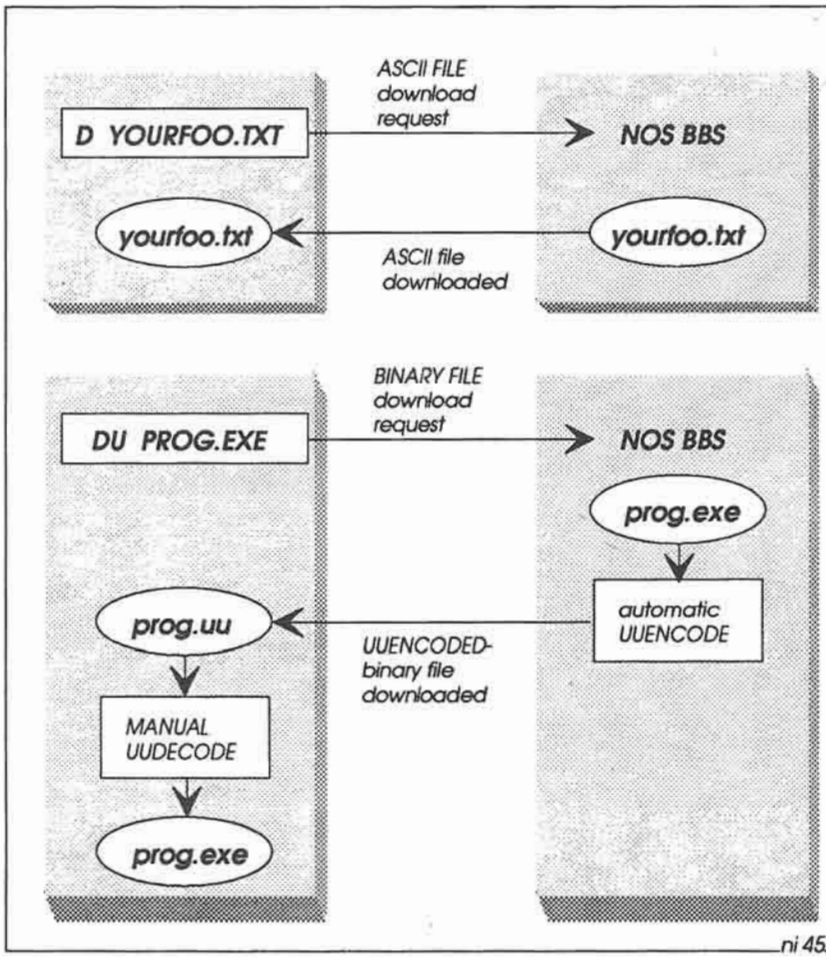


Fig 20-2: BBS download always transfers files in 7-bit ASCII format. If you request the download of an 8-bit file, the BBS uuencodes the file first into 7-bit format before transfer.

U (Upload): The **Upload** command lets you upload *7-bit ASCII* files from your system to the BBS. For example: **u myfoo.txt**. There is no UU command for uploading binary files. Instead, you must first convert the file off-line from binary to 7-bit uuencoded-ASCII (as already explained in Chapter 17) before uploading it.

W (What): The **What** command by itself lists the files in the root directory to which you have access, specified by your entry in *ftpusers*. (If your entry has more than one root directory in *ftpusers* — for example /public;/private — you can only list the files in and below the first directory). To list files in sub-directories below the top level, you simply specify the subdirectory path; e.g. **w public/nosview**.

Z (Zap): The **Zap** command lets you delete files, if you have permission to do so; e.g. **z yourfoo.txt**. Your entry in *ftpusers* specifies your file access permissions.

The NOS BBS Gateway Server Commands

The NOS BBS provides a number of gateway commands which let you access the AX.25 network and the NET/ROM network, and which let you log in to another NOS BBS using **telnet**. See Fig 20-3. To terminate any sessions which you start with these commands, simply type the BBS Escape character (by default, **CTRL-X**).

C (Connect): The **Connect** command allows you to connect to an AX.25 station (e.g. **C tnc0 AX9ABC-3**) or to a NET/ROM node (e.g. **C #TOM**). When connecting to an AX.25 station, you need to include the interface name (tnc0) to specify which port you want to use for the connection. To find out the names of available ports, use the **P** command. To find out the names of known NET/ROM nodes, use the **N** command.

J (Justheard): The **J** command gives you a list of recently heard stations. This may be useful if you are not sure of the exact callsigns or NET/ROM node aliases of accessible stations.

N (Nodes): The **Nodes** command displays a list of accessible NET/ROM nodes.

O (Operator): The **O** command starts a chat session with the operator. To terminate the session, give the BBS Escape command (default **CTRL-X**).

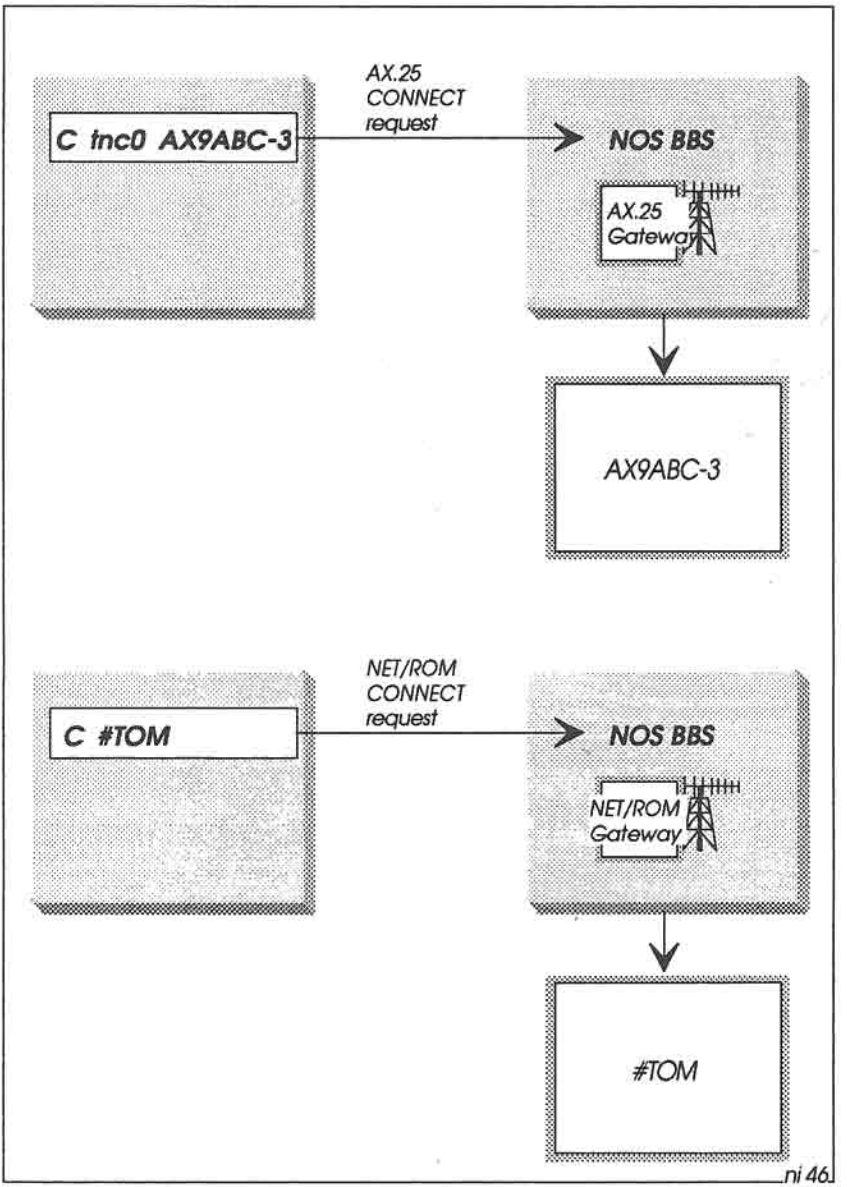


Fig 20-3: After connecting to the NOS BBS, users can connect to AX.25 and NET/ROM stations via the NOS BBS Gateways.

P (Ports): The **Ports** command gives a list of available interfaces through which you can make connections, together with brief descriptive comments; e.g:

```
tnc0    144.625 MHz Port
ec0     In-house Ethernet LAN
tel0    1200/2400 bps modem
```

This information comes from **ifconfig description** commands in *autoexec.nos*.

T (Telnet): The **Telnet** command lets you make a **telnet** connection with another NOS station; e.g. **t ns9liz**. See Fig 20-4. In this way an ordinary AX.25 user can gain access to the AMPRnet.

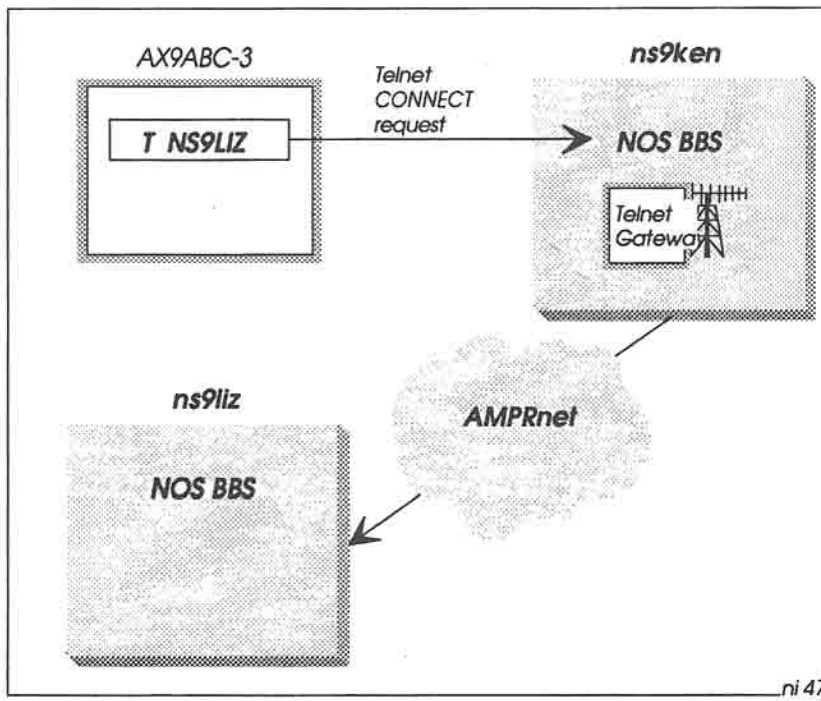


Fig 20-4: After logging in to a NOS BBS, an ordinary AX.25 station can access AMPRnet via the NOS Telnet Gateway.

The NOS BBS Finger Server

The **F** command lets you "finger" another station, that is, to find out more information about it. See Fig 20-5. The command **f @ns9ken** will return a list of known users having finger files on **ns9ken**, and the command **f sysop@ns9ken** will return the contents of file **/finger/sysop** at **ns9ken**. The finger files are plain ASCII, and, like the help files, they should be short and to the point.

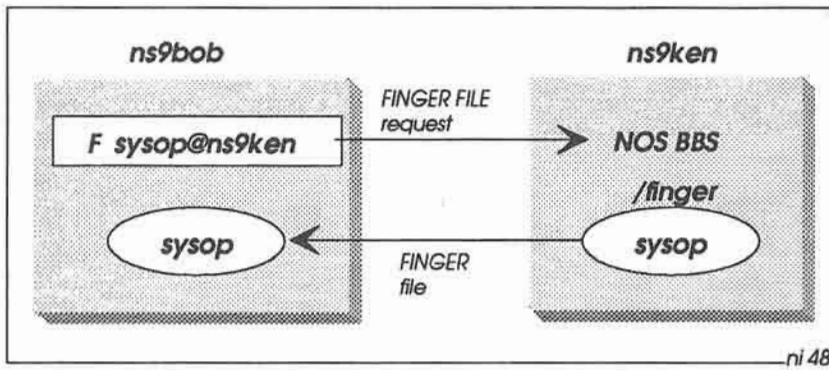


Fig 20-5: Finger files.

The NOS BBS Remote Sysop Command

In addition to the commands just described, there is also a Remote Sysop command, @, that lets you take over control of the station. This is described in detail in Chapter 22.

21: HANDS ON — NOS BBS FILE SERVER

The built-in NOS BBS file server is intended for people who connect to the system via an ordinary AX.25 or NET/ROM link; NOS users will normally use **ftp** instead to access files on the system.

The server has five commands:

- **W** What (list the available files)
- **D** Download an ASCII file
- **DU** Download a uuencoded file
- **U** Upload a file
- **Z** Zap (delete) a file

To see how these commands work, check that your current NOS directory is */dump/record*, then log into your own BBS:

```
net> cd /dump/record
net> bbs
```

Log in as roberto (with the password robertspw), then try the commands.

The *What* Command

The W command lists all files in a particular directory. The default directory is the directory specified in the *ftpusers* file for the user. If more than one directory is specified in *ftpusers* for the user, the default is the first directory in the list.

For example:

```
NS9BOB-5} W
alias           662 10:09 5/25/92 autoexec.000 7,587  5:26 8/16/92
autoexec.nos  7,755  8:59 8/31/92 cleanq.bat     285 11:20 8/13/92
domain.txt    1,925  8:37 8/28/92 dump/                8:04 8/13/92
finger/             8:07 8/13/92 ftpusers     2,699 10:32 8/16/92
net.rc          545  8:54 8/23/92 netrom.sav      16  8:43 7/26/92
nos_20j.exe 198,556 20:03 5/03/92 nos_20m.map 75,514  8:23 8/03/92
nos_20m.exe 203,201  8:23 8/03/92 nosenv.bat   1,041 15:14 8/20/92
pcelm.msg     9,898 21:53 2/19/91 pcelm.exe   68,992 23:54 1/04/92
pcelm.rc      8,344 11:48 5/25/92 popusers       429 18:13 9/18/91
public/             8:04 8/13/92 remote.bat     333 18:14 9/18/91
scripts/            8:04 8/13/92 signatur       533 14:34 5/14/92
spool/              8:04 8/13/92 startnos.bat   847 15:13 8/20/92
tmp/                8:04 8/13/92 view.hlp     6,910  0:00 9/01/89
view.com     10,221  4:24 5/21/91
```

Directories are indicated with a forward slash; e.g. finger/.

You can also use wildcards to narrow down the search:

```
NS9BOB-5} w *.exe
nos_20j.exe 198,556 20:03 5/03/92 nos_20m.exe 203,201 8:23 8/03/92
pcelm.exe    68,992 23:54 1/04/92
```

To find out the names of files in a subdirectory, add the subdirectory name to the W command, using forward slashes; e.g.

```
NS9BOB-5} w /public/nosview
```

The *Download* Command

The D command downloads a file to the user. The download is in 7-bit format, suitable for plain ASCII files. For example:

```
NS9BOB-5} d /public/nosview/arp
```

But there's a catch! You'll see that the downloaded file appears on the screen, but it is not saved on disk. To save the file on disk, you need to escape back to the Session Manager to start a **record** session; e.g.

```
net>  record /dump/record/myarp
```

[For convenience, the **SHIFT-F10** key combination is programmed to provide most of this command automatically, so all you have to type is **SHIFT-F10***myarp*].

Then return to the BBS session (by hitting **CR**), and give the download command again.

When the download is finished, you terminate the **record** session by escaping back to the Session Manager, then giving the command:

```
net>  record off
```

[For convenience, the **CTRL-F10** key combination does the same thing].

Now when you examine the */dump/record* directory, you should find the file *myarp* is present:

```
net>  more myarp
```

[Or you can hot-key to **VIEW** and **F3** *R:\MYARP*].

Downloading uuencoded Binary Files

To download a binary file from the NOS BBS, you use the **DU** command. This automatically reads the binary file and passes it through the uuencode program built in to the file server (see Fig 20-2). The output from uuencode consists of 7-bit printable ASCII characters, and it is this encoded form which is downloaded.

As before, you must first start a **record** session before giving the **DU** command; e.g.

```
net>  record /dump/record/pcelm.uu
```

Then return to the BBS and give the download command:

```
NS9BOB-5}  du pcelm.exe
```

When the transfer is finished, you should escape back to the Session Manager and finish recording:

```
net>  record off
```

Now you will find the uuencoded file in */dump/record/pcelm.uu*. Take a look at this file:

```
net>  more pcelm.uu
```

[or hot-key to **VIEW**, then **F3** *R:\PCELM.UU*]

You will see something like this:

```
du pcelm.exe
begin 755 /pcelm.exe
M35J  8< ?P!       __^0$(           /@     $ ^S!J<@
M                          !    6P   *     #*     ! $  )0 0P[N $,.
M- %##G8!0P[6 4,.T@%##LX!0P[* 4,.Q@%##L(!0PZ^ 4,.N@%##K8!0PZR
M 4,.K@%##JH!0PZF 4,.H@%##IX!0PY"!$,.;@1##EP%0P[.!4,.$@9##@8&
M0PY2"4,.3@E##DH)0PY&"4,.0@E##CX)0PXZ"4,.-@E##C()0PXN"4,.*@E#

    { many more lines }

M!04% @(" @(" @(" @<*%0H,"0(""P(4#@(" @("" ("$@("$ (0 @(" @(&
M!PH*"@P) @(- A$.$P("#P(( @(2 @(" @(" @  'P [ %H > "7 +4 U #S
M
M
'

end
size 68992
>
```

To restore this to its original binary form, you need to exit from NOS and then decode the file under DOS. However, before performing the decode, take a closer look at the begin line of the uuencoded file:

```
begin 755 /pcelm.exe
```

This contains the *absolute* pathname of the file that will be created when you decode the file. In other words, if you decode this file as it stands, it will appear as *PCELM.EXE* in the *NOS root directory, N:/*. In all probability you won't want this file in the root directory, so you should edit the file to remove the slash character:

```
begin 755 pcelm.exe
```

Then, to decode the file, escape to DOS and use the commands:

```
N:\>  CD \DUMP\RECORD
N:\DUMP\RECORD\>  UUDECODE < PCELM.UU
```

This takes a few seconds. When finished, you should find the file *PCELM.EXE* in the current directory, and it should be identical to the original binary file in *N:*.

Finally, you can delete the encoded file:

```
N:\DUMP\RECORD\>  DEL PCELM.UU
```

The Upload Command

To upload a 7-bit ASCII file to the BBS, you need to give the U command. Be careful to ensure you include the full pathname where you want to store the file on the BBS, otherwise it may finish up in the wrong place.

For example, to upload the local file *myarp* (in */dump/record*) and call it *myarp2* on the BBS:

```
NS9BOB-5} u /public/myarp2
Send file, Terminate in /EX or ^Z in first column (^A aborts):
```

Note that all you have done here is tell the BBS that you want to upload a file and call it *myarp2*. You haven't actually started the upload operation. To do this, you escape to the Session Manager and give the **upload** command:

```
net> upload myarp
```

The upload now starts. To check its progress, you can give the **upload** command from time to time. NOS will respond with the current upload status. When uploading is done, the status is reported as uploading off:

```
net> upload
uploading myarp
net> upload
uploading myarp
net> upload
uploading off
```

When uploading is finished, return to the BBS session to terminate the **U** command with **/EX**, and the BBS prompt should then re-appear:

```
/EX
NS9BOB-5}
```

[Sometimes you may need to enter **/EX** twice to make the prompt re-appear].

Uploading Binary Files

The **U** command just described will only upload 7-bit ASCII files. If you want to upload an 8-bit binary file, you first need to exit to DOS and encode the file first.

For example:

```
N:\DUMP\RECORD\>  UUENCODE < MYFILE.EXE > MYFILE.UU
```

This creates the file *myfile.uu,* which you can then upload to the BBS. When it arrives at the BBS, there is no automatic way of decoding it back to its original form; the BBS owner will have to do this with the DOS **UUDECODE** command as already described.

The Zap Command

The **Z** command simply deletes a file, if you have permission in *ftpusers* to do so. For example:

```
NS9BOB-5}  z /public/myarp2
```

22: HANDS ON — REMOTE SYSOP

The NOS BBS allows remote access to the Session Manager — very useful for setting up and checking the operation of a system which you can't physically reach easily.

Gaining Access to the Session Manager

To allow remote access to the Session Manager, you first need to set up a Remote Sysop account with the sysop permission in *ftpusers*. You also need to set up the mailbox password for remote sysop access.

The sysop permission in *ftpusers* is the code 64. This number should be added to the permissions codes for all users whom you will allow to become a remote sysop. For example:

```
superuser supasswd /public 67
```

Note that the login name (superuser) is more than six characters long, thus preventing access to ordinary AX.25 users. Note also that the permissions are 64+2+1; i.e. read, create and sysop, with */public* as the root directory — this reduces the possibility of serious damage to the system if someone discovers the password by snooping on the channel and then logs in as superuser.

The Mailbox Password

The mailbox password is set up in *autoexec.nos*. This password may be up to 30 characters long; e.g.

```
mbox password "Maximum 30-character password."
```

Now write out the password on a piece of paper and number each character, starting at 0:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
M	a	x	i	m	u	m		3	0	-	c	h	a	r	a	c	t	e	r		p	a	s	s	w	o	r	d	.

You will need to know these character numbers when logging in.

Becoming Superuser

To see how to use the mailbox password, login to your own BBS as superuser, with the password supasswd. See Fig 22-1. Then, at the BBS prompt, give the @ command. The BBS responds with a warning to be very careful, and then outputs a string of five numbers. You must now reply by inputting the corresponding characters of the mailbox password; 22 means character 22, 21 means character 21, and so on.

After inputting the five characters, hit **CR** twice. Assuming the input was correct, a new Session Manager prompt then appears — but with a capital "N", to distinguish it from the normal prompt.

For example, the complete dialog looks like this:

```
NS9BOB-5} @

Type 'exit' to return

Be VERY careful!!
22 21 18 2 12
apexh          {a is char 22, p is char 21, e is char 18, etc}
               {blank line to finish password input}
Net>           {Remote Session Manager prompt}
```

From now on you are talking direct to the remote Session Manager, so, as it says in the message, *be very careful*! You now have complete control over the station, and can do almost anything with it.

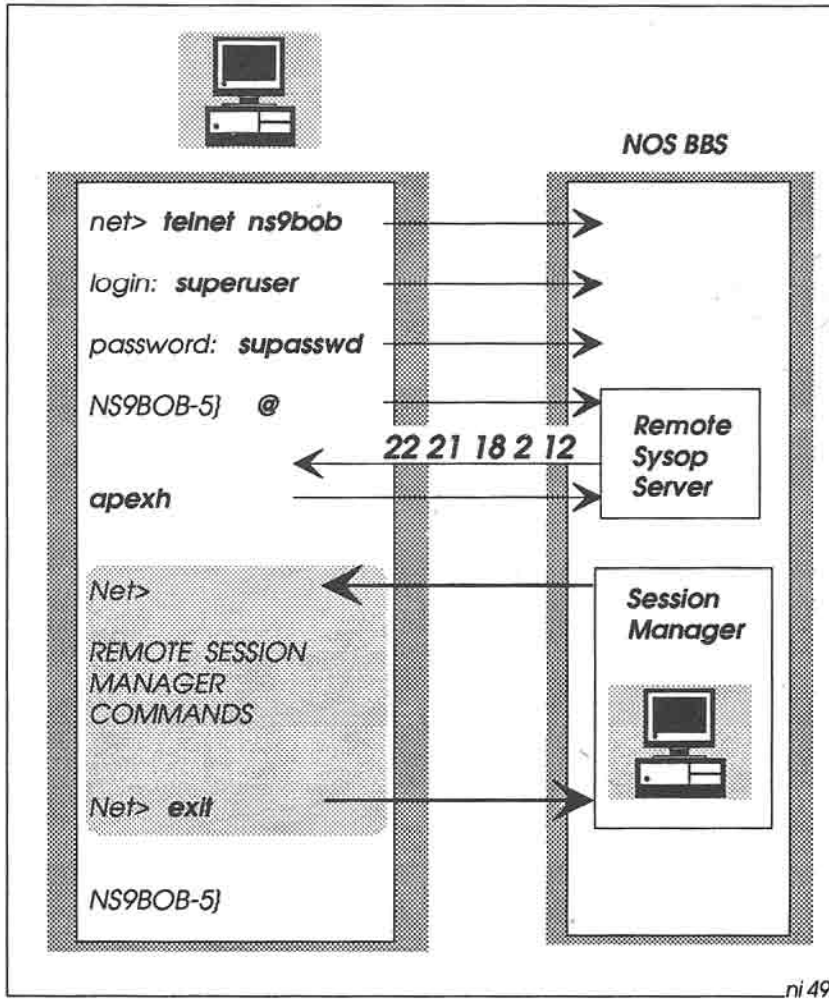


Fig 22-1: Logging in as a Remote Sysop. The Remote Sysop server sends back 5 numbers between 0 and 29, corresponding to character positions in the mailbox password. You have to respond with the correct password characters before the server grants access to the remote Session Manager.

When you are finished, leave the Session Manager with the **exit** command. This takes you back to the BBS prompt:

```
Net> exit
NS9BOB-5}
```

Dummy Responses

When inputting the mailbox password characters in response to the series of five numbers, you can type several dummy lines if you wish, to fool anyone who may be snooping. For example:

```
22 21 18 2 12
zf8wy          {dummy line}
apexh          {the real line}
qwert          {another dummy line}
               {blank line to finish password input}
Net>
```

Provided that one of these lines contains the correct mailbox password characters, you will gain access to the remote Session Manager.

Unsupervised Operation

If the system operates completely unsupervised, it is a good idea to turn on the NOS watchdog timer (in *autoexec.nos*):

```
watchdog on
```

The watchdog timer is normally reprimed at regular intervals, and doesn't time out. But if something goes wrong inside NOS and the timer expires — after a few minutes — then NOS automatically exits. If you originally started NOS with a batch file something like the following, NOS will then restart again automatically:

```
loop:
  CALL STARTNOS
  GOTO loop
```

23: FORWARDING SMTP MAIL

In the next three chapters we take a closer look at how to address and forward mail. There are several different methods, depending on whether you want your mail to go via the AMPRnet using SMTP, or via the AX.25 PBBS network, or a combination of both.

N.B. The techniques described in these three chapters show what is technically possible with NOS mail forwarding. This includes third-party message handling. If you are not licensed to handle third-party messages, not all of these techniques will be available to you.

Which Network?

The first question to consider is which network is to carry your mail. You may launch some messages into the AMPRnet, and others into the AX.25 PBBS network. This is like deciding whether to send a letter via the public Post Office network or via a private courier service.

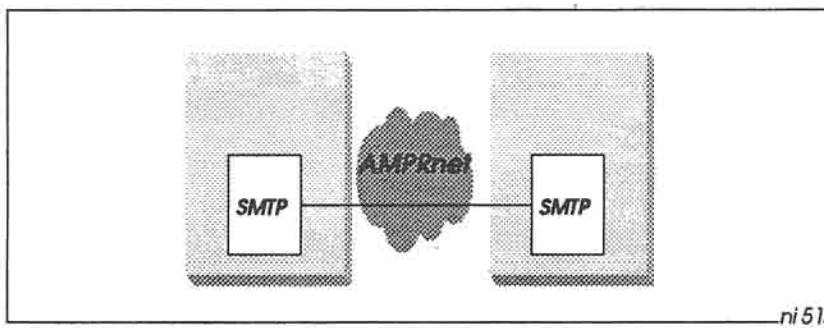


Fig 23-1: Launching mail onto the AMPRnet. All stations understand SMTP.

In the ideal AMPRnet environment, you would use SMTP to forward all messages to their destinations (Fig 23-1). In this case, all forwarding stations and recipient stations understand SMTP. Think of this as the public Post Office network.

On the other hand, you could use the AX.25 PBBS network to forward all messages (Fig 23-2). This would be necessary if the recipient does not understand SMTP, and/or there is no SMTP routing available to the destination. Think of this as a private courier network.

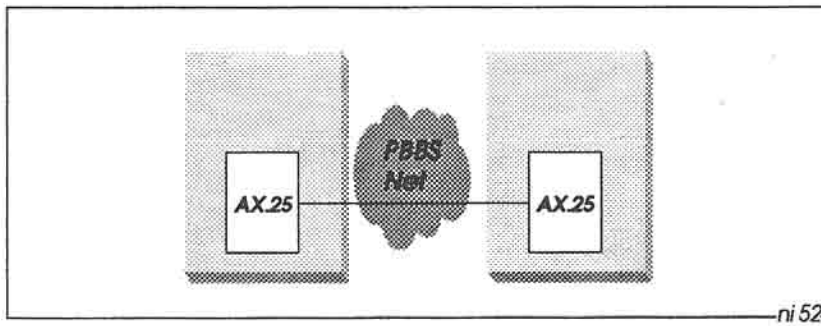


Fig 23-2: If it is not possible to connect to a local SMTP station, all mail has to be launched onto the AX.25 PBBS network.

SMTP/PBBS Mail Gateways

In reality, of course, things aren't as clear cut as this. Just as the Post Office makes use of private courier services, and private courier services make use of the Post Office, individual packet radio messages may in fact travel partly via SMTP routes and partly via AX.25 PBBS routes.

The network to support this (Fig 23-3) consists of SMTP/PBBS mail gateways which accept messages via SMTP and re-launch them on to the PBBS network (and vice versa). As we've already seen, NOS can handle both SMTP mail and AX.25 PBBS mail, and so can function as such a gateway.

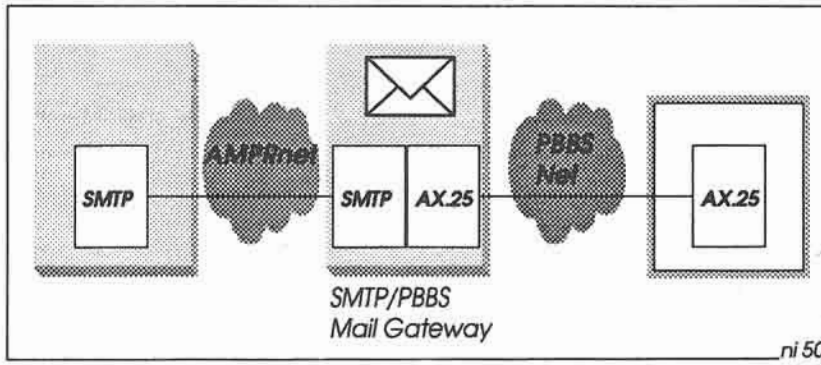


Fig 23-3: SMTP/PBBS mail gateways convert message formats between AMPRnet and the PBBS network.

Launching the Mail

NOS allows you to choose how to launch mail into the combined SMTP/PBBS network. To decide which method to use, the following general rules are useful:

1. Choose a "next-door neighbour" station through which you want to forward mail (ignore for now any intermediate digipeaters or NET/ROM nodes you may need to go through to get there). Your neighbour may be a NOS station which understands SMTP, or a PBBS station which only understands AX.25.
2. If your neighbour understands SMTP, you can use SMTP to launch all of your mail. This includes mail addressed to ordinary AX.25 stations which don't understand SMTP; somewhere along the route there will be an SMTP/PBBS mail gateway to convert messages from SMTP to PBBS format. (However, if you prefer, you can launch mail for AX.25 stations directly on to the PBBS network if you want to).
3. If your neighbour doesn't understand SMTP, you'll have to use AX.25 PBBS forwarding for *all* your mail, even for addressees running NOS.

General Rules for Addressing

In general, you can address mail in two ways. The examples that follow are for the built-in NOS BBS, but you will address mail in the same way if you are using an external mailer such as PCElm.

The syntax for addressing mail is as follows:

```
NS9BOB-5} sp user@target_mailhost
NS9BOB-5} sp user%target_mailhost@intermediate_mailhost
```

(N.B. There is no space either side of the % sign). For example:

```
NS9BOB-5}  sp liz@ns9liz
NS9BOB-5}  sp AX9TIM%BB7BBS@ns9bob
```

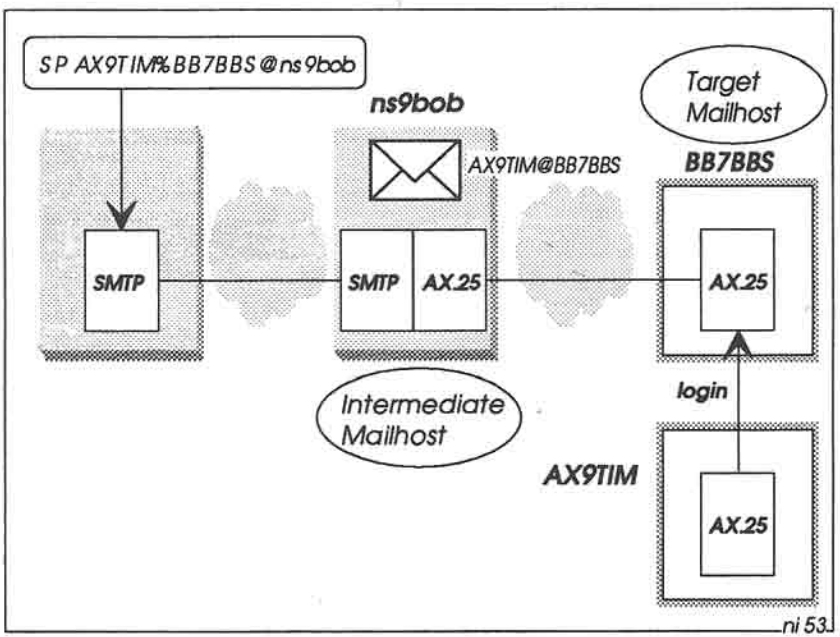


Fig 23-4: You can address mail via an intermediate mailhost if that helps message forwarding towards the target. The intermediate host translates the % character in the address to an @ character.

The *target_mailhost* is the eventual destination of your mail, and the *user* is someone (or something) that can access that mailhost to retrieve the mail (see Fig 23-4).

The *intermediate_mailhost* is a host somewhere along the route to the target_mailhost (again, see Fig 23-4). The intermediate_mailhost changes the first part of the address (user%target_mailhost) to read user@target_mailhost, and then sends it on towards the target.

This form of addressing, using the % symbol, is useful when you want to launch a message in a particular direction, rather than rely on automatic forwarding into unknown territory. It can save a lot of time if you know something about the network to help the message along!

Incidentally, you can use this form of addressing in a pure AX.25 PBBS network as well. This may be useful for sending messages to other countries, where some routes are known to be unreliable or where the local network has strange forwarding tables. See Fig 23-5.

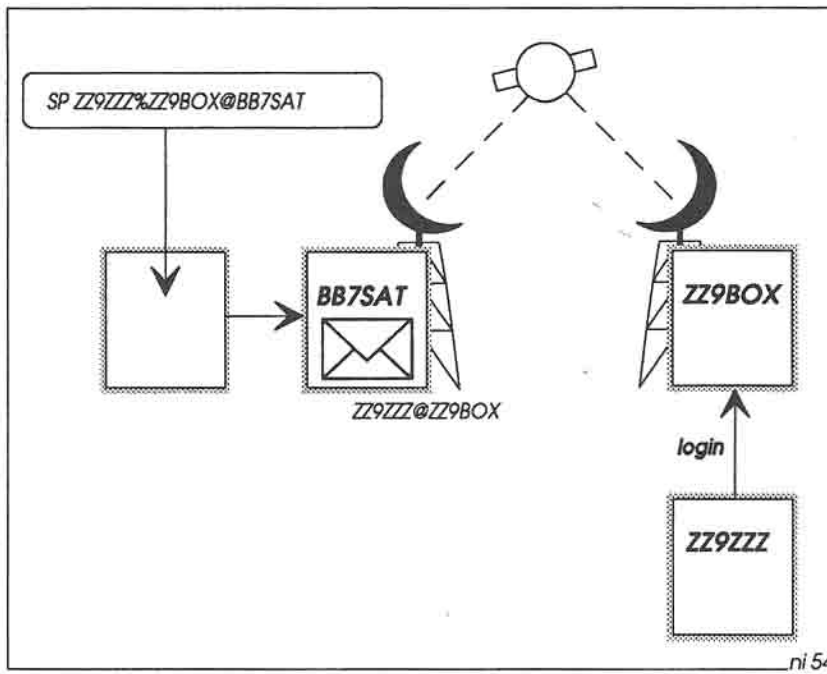


Fig 23-5: Using % in addresses on the AX.25 PBBS network.

For example, **SP ZZ9ZZZ%ZZ9BOX@BB7SAT** lets you direct a message to the local satellite gateway BB7SAT, which then changes the address to ZZ9ZZZ@ZZ9BOX. This is probably preferable to addressing the message simply as ZZ9ZZZ@ZZ9BOX, as you then have no control over how your local network handles it, and it may finish up going via an hf AMTOR link instead. On the other hand, that may be a better bet after all ...

We'll now look at some of the various methods of addressing mail in detail.

SMTP to a Known IP Address

This is the simplest method of addressing. Bob can send a message to Ken using the command:

```
NS9BOB-5} sp ns9ken@ns9ken
```

NOS discovers ns9ken's IP address by looking it up in *domain.txt*:

```
ns9ken.ampr.org. IN A 44.199.41.2
```

Assuming that IP routing is set up properly, NOS connects with 44.199.41.2 and then sends the message (or, more accurately, the SMTP client in Bob's system connects with the SMTP server in Ken's system, and then the client and server co-operate with each other to transfer the message).

Thus for all your immediate neighbours, and for any other stations to whom you wish to send mail and for which a known route exists, you should have an IN A entry in *domain.txt*.

SMTP Client and Server

Fig 23-6 shows in more detail what happens when you send a message. Starting at the top left-hand corner, you use a mailer (such as the NOS BBS or PCElm) to compose the message.

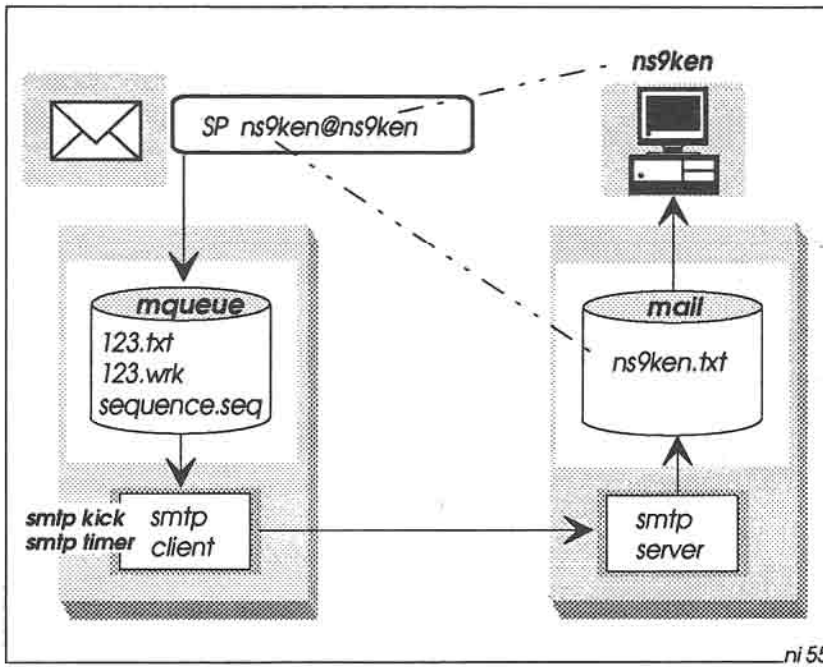


Fig 23-6: Sending SMTP mail. The mailer places the message onto the outgoing mail queue (*/spool/mqueue*), then the SMTP client connects to the addressee's SMTP server. The server places the message onto the incoming mail queue (*/spool/mail*), ready for reading.

The mailer places the message in the outgoing mail queue (*/spool/mqueue*), in two files: *n.txt* and *n.wrk*. The value *n* is the sequence number of the message. Thus for message number 123, the two files are called *123.txt* and *123.wrk*.

The sequence number itself is maintained in the file */spool/mqueue/sequence.seq*.

The *n.txt* file contains the text of the message, in exactly the form that you composed it in the mailer, together with To:/From: address and date/timestamp information which the mailer included automatically.

The *n.wrk* file contains three lines. For example:

```
ns9ken
ns9bob@ns9bob
ns9ken@ns9ken
```

The first line states the target_mailhost.

The second line states who the message is from.

The third line contains the addressee.

At regular intervals (defined with the NOS **smtp timer** command — normally every 10 minutes), the SMTP client wakes up and looks at the outgoing message queue, by examining the *n.wrk* files. For each message in the queue, the client then attempts to make a connection with the SMTP server at the host listed in the first line of the *n.wrk* file.

In the above example, the SMTP client on the local machine attempts to connect with the SMTP server at ns9ken.

(If you don't want to wait until the next SMTP timer interval expires, you can force the SMTP client to wake up immediately with the NOS command **smtp kick**).

Once contact is established, the client then transfers the message to ns9ken, where the SMTP server places it in the incoming mail directory (*/spool/mail*). Each user has a separate *.txt* file in this directory; thus a message addressed to user ns9ken@ns9ken appears in file *ns9ken.txt*.

Finally, the SMTP server at ns9ken outputs a message to the screen, saying that a "message for ns9ken has arrived". Ken can then log into his BBS to read it.

SMTP to an AX.25 Station at an SMTP Mailhost

When you want to send a message to an ordinary AX.25 station whose local mailhost is running SMTP (Fig 23-7), all you have to do is address the message to the AX.25 callsign @ the mailhost.

For example:

```
NS9BOB-5} sp AX9SAM@ns9ken
```

Using SMTP, the message will find its way to ns9ken, where it will be saved in Ken's NOS BBS, in file */spool/mail/ax9sam.txt*. AX9SAM can then connect to NS9KEN in the usual way, and will find the message waiting for him there.

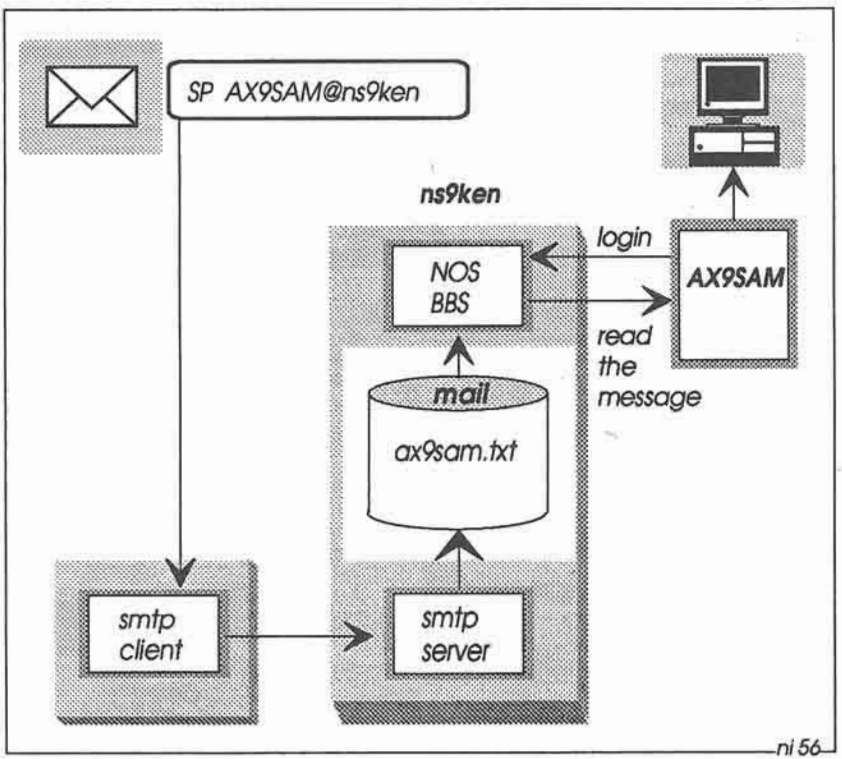


Fig 23-7: Sending mail to an AX.25 station for collection at a NOS station.

SMTP via a Mail Exchanger

As already noted in Chapter 8, it's obviously unrealistic for *domain.txt* to contain every known IP address in the world, and sooner or later

you'll want to send a message to a particular station whose callsign you know but whose IP address you don't. So what do you do?

If you're lucky, there may be a reasonably local station which has a comprehensive *domain.txt* file containing the address you want, and which knows how to forward your message on to its destination. In this case, you can use that station as a mail exchanger (Fig 23-8).

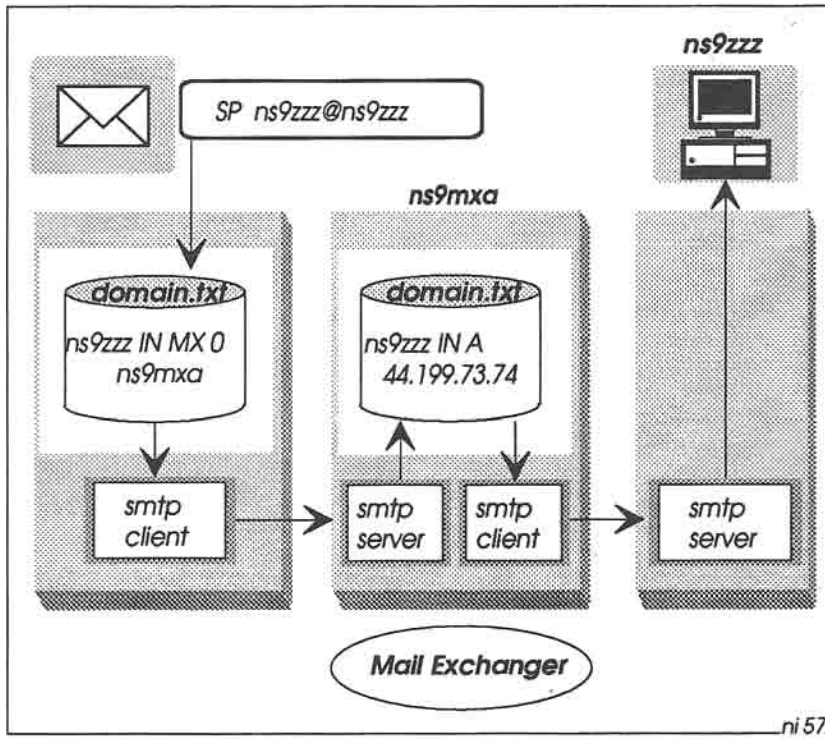


Fig 23-8: You can nominate a Mail Exchanger system with an *IN MX* record in *domain.txt*. Mail for ns9zzz passes to the exchanger, which then forwards it to its destination.

To do this, you first need to include the IP address of the gateway in your *domain.txt*:

```
ns9mxa.ampr.org. IN A  44.199.41.90
```

Then you add MX (Mail Exchanger) records to *domain.txt* for particular stations you wish to send messages to via the exchanger. For example, for messages to ns9zzz:

```
ns9zzz.ampr.org. IN MX 0   ns9mxa.ampr.org.
```

The digit 0 after IN MX is the *preference value* of this exchanger. A value of 0 is the highest preference.

There can be more than one MX record in *domain.txt* for a particular addressee, allowing you to specify alternative MX exchangers:

```
ns9zzz.ampr.org. IN MX 0   ns9mxa.ampr.org.
ns9zzz.ampr.org. IN MX 10  ns9mxb.ampr.org.
ns9zzz.ampr.org. IN MX 20  ns9mxc.ampr.org.
```

By default, NOS will attempt to forward messages for ns9zzz via the MX exchanger with the lowest preference value (i.e. ns9mxa, with preference 0). If that attempt fails, NOS will then try to forward via the exchanger with the next lowest preference (i.e. ns9mxb, with preference 10), and so on.

You can also use wild cards in MX records. This is very useful if you want to direct messages to a particular network (which may not be anything to do with AMPRnet).

For example, you can forward messages to anyone having a .com Internet address with the record in *domain.txt*:

```
*.com. IN MX 0 ns9int.ampr.org.
```

assuming that the MX exchanger ns9int knows how to forward on to the Internet. The wild card (*) matches any name ending in .com; e.g. messages for a.com, a.b.com and a.b.c.com all go via ns9int.

Having set up the MX records in *domain.txt*, you then tell NOS to use them, with the command (in *autoexec.nos*):

```
smtp usemx on
```

From now on, you can address mail via the MX exchangers with simple commands like:

```
NS9BOB-5} sp ns9zzz@ns9zzz
NS9BOB-5} sp bob@mwc.com
```

SMTP will discover that there are MX records which match ns9zzz and mwc.com, and will forward accordingly.

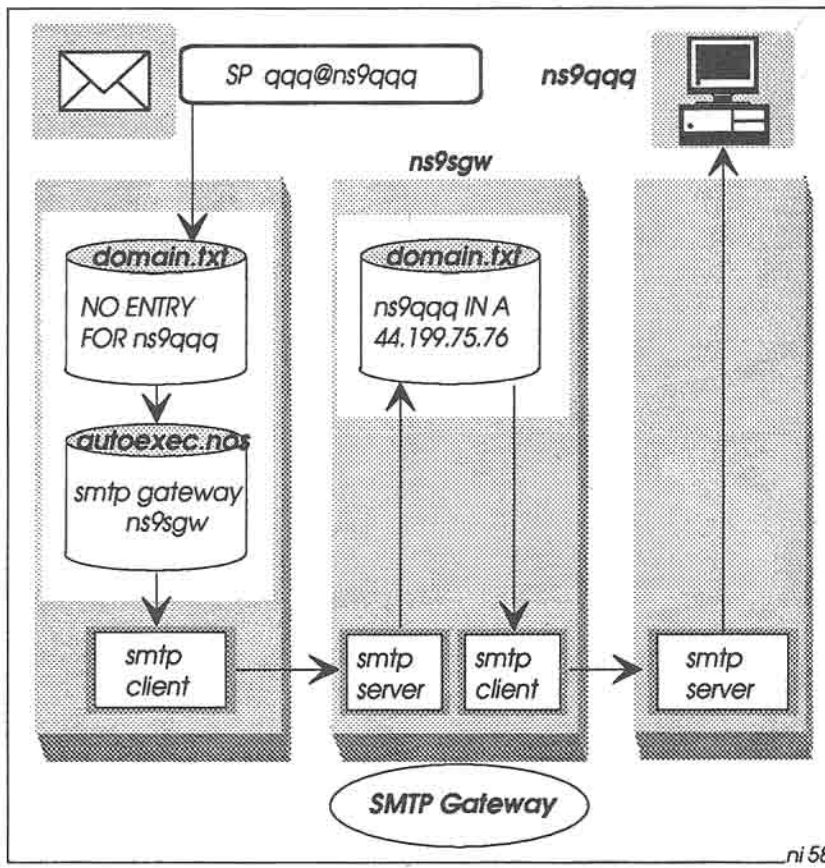


Fig 23-9: SMTP to an unknown IP address. If there is no entry for an addressee in *domain.txt*, the mail is sent to an SMTP gateway (*ns9sgw*), which knows how to forward it to its destination.

SMTP to an Unknown IP Address

The MX records just described are fine for forwarding mail to particular stations (or to particular networks) via specified exchangers.

More generally, however, you'll want to send messages to people whose IP addresses you don't know, and you have no idea how to forward these messages to them.

To handle this situation, you can nominate a default SMTP gateway to handle mail — see Fig 23-9 opposite. For example, in *autoexec.nos*:

```
smtp gateway ns9sgw
```

Thereafter, when you address mail to any station which does not have an IN A or IN MX entry in *domain.txt*, SMTP will forward the mail to the gateway. That gateway will then (hopefully!) know how to forward it onwards.

SMTP to an AX.25 Station at a PBBS

If you want to send a message to an ordinary AX.25 station whose local mailhost is an AX.25 PBBS, you can still launch the message using SMTP — even though neither the addressee nor his mailhost understands SMTP.

This works because there will be an SMTP/PBBS mail gateway somewhere along the route which will re-launch your SMTP message into the PBBS network (see Fig 23-10).

To address such messages, you simply use the same method as you would with an ordinary AX.25 PBBS. For example:

```
NS9BOB-5} sp AX9XXX@BB7BXA
```

NOS will look in *domain.txt* as usual for the IP address of the destination mailhost (BB7BXA), but won't find it there — remember that *domain.txt* contains IP addresses, not AX.25 callsigns. BB7BXA is an AX.25 PBBS, so it doesn't have an IP address.

Thus SMTP treats BB7BXA as an IP station whose address is not known, so all it can do is forward it to the default SMTP gateway

station. The gateway will then either forward the message on to another SMTP gateway, or will re-launch it onto the the PBBS network. The setting up of an SMTP/PBBS gateway is described in full in Chapter 25.

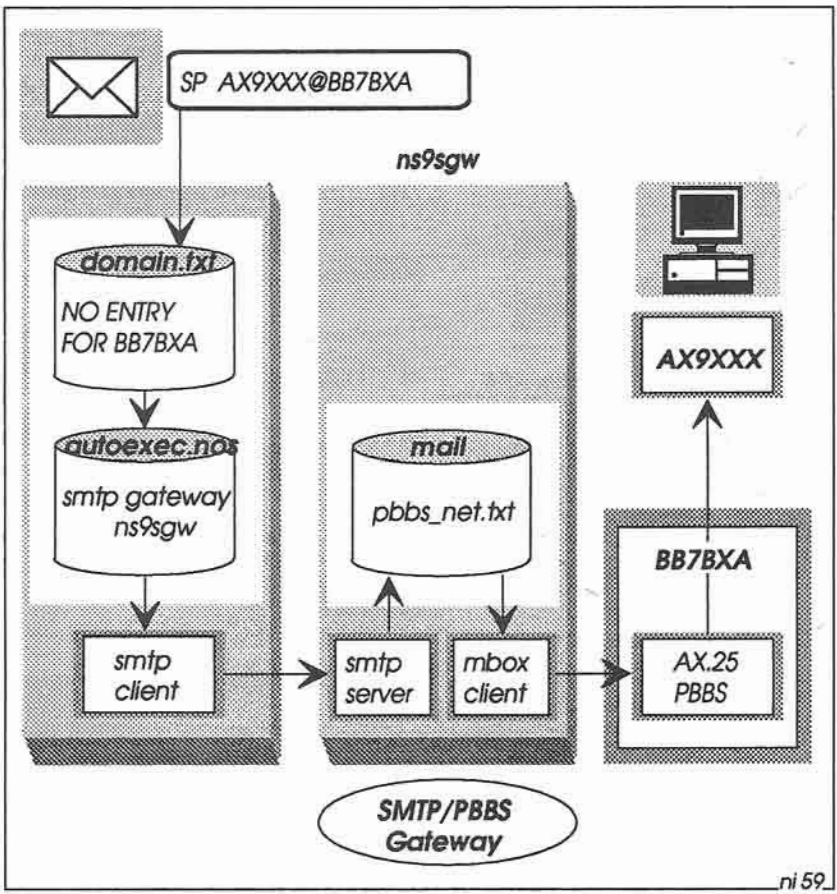


Fig 23-10: SMTP to an AX.25 station at a PBBS. The SMTP/PBBS gateway converts the message to PBBS format then forwards it to the target PBBS.

The REWRITE File

There will be many times when you want to simplify the addressing of your mail, or when you want to send a particular message to more than one recipient. The NOS files */spool/rewrite* and */alias* are especially useful here.

The *rewrite* file gives us great flexibility in addressing mail, being particularly useful for handling incomplete addresses, inaccurate addresses and hierarchical addresses.

Essentially, NOS uses *rewrite* to map the original destination address to a new destination address. This is a one-to-one mapping, whereby the original address changes to a new address in a different format.

This is in contrast to alias expansion, where you can specify several new destinations for one original message.

Each record in *rewrite* is a single line containing a template for the original destination and a mapping for the corresponding replacement address — see Fig 23-11.

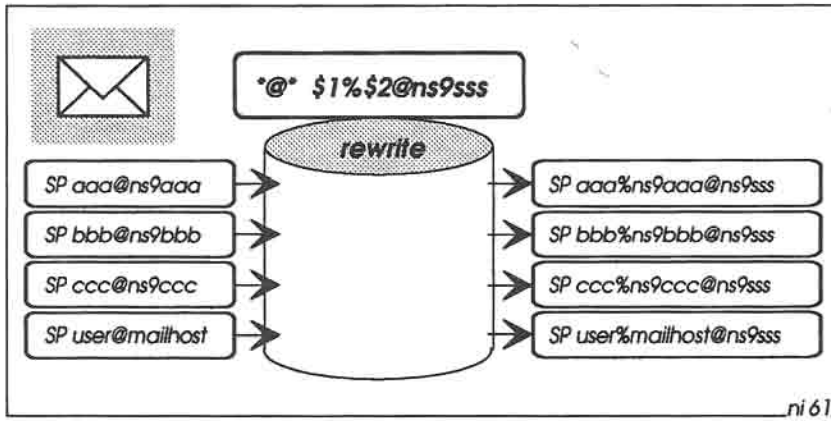


Fig 23-11: The file */spool/rewrite* translates addresses from one format to another to suit forwarding.

A simple example of a *rewrite* record:

```
*@*   $1%$2@ns9sss
```

The template for the original destination is *@*. The asterisk is a wildcard, and so the template matches any user name and mailhost; e.g. aaa@ns9aaa.

In the mapping for the replacement address ($1%$2@ns9sss), the variables $1 and $2 correspond to the first and second asterisks in the template. Thus for a message addressed to aaa@ns9aaa, $1 is aaa and $2 is ns9aaa, and so the new destination address will become aaa%ns9aaa@ns9sss.

In other words, any message originally addressed to user@mailhost will be re-addressed to user%mailhost@ns9sss.

Note that the order of the *rewrite* records may be important. NOS starts at the beginning of *rewrite* when attempting to match an address to a template, and when a match is found the scanning stops (except in the special case of a matching record containing a letter **r** at the end, when scanning re-starts from the beginning — this is described in more detail below).

It's also important to note that the new destination address goes into the *n.wrk* file for the message. The original address (user@mailhost) remains unchanged in the associated *n.txt* file. In other words, *rewrite* changes the forwarding information in *n.wrk* to make it easier to launch messages in the right direction, but does not change the message itself (in *n.txt*) in any way.

Using the Post Office analogy again, you may write the To: address on both the letter and the envelope, but someone may change the address on the envelope afterwards to make it easier to deliver — the To: address on the letter inside the envelope remains unchanged.

Using wildcards for partial addresses

As well as using the asterisk wildcard to represent complete items such as user names or callsigns in the template field of the *rewrite* file, you can also use the asterisk to match part of an item.

For example, if you want to send all traffic addressed to German stations with callsign prefixes dj or dk via the gateway ns9deu, you could set up a *rewrite* record something like:

```
*@dj*   $1%dj$2@ns9deu
*@dk*   $1%dk$2@ns9deu
```

Then a message addressed to fritz@dj9zzz is re-addressed to fritz%dj9zzz@ns9deu, and similarly a message to hans@dk9zzz is re-addressed to hans%dk9zzz@ns9deu.

Rewrite File Processing

When does NOS read the *rewrite* file? It depends on how the message gets into the system (Fig 23-12). If you are sending a message using the NOS BBS (or if someone has logged in to your NOS BBS via **telnet** or an AX.25 connect), then the *rewrite* mapping takes place when the message goes onto the outgoing mail queue. (N.B. This only happens when you use the NOS BBS. External mailers such as PCElm are unaware of the *rewrite* file).

On the other hand, if the message comes into NOS from a remote system via the SMTP server, *rewrite* mapping takes place then.

But if a message enters the SMTP server locally, *rewrite* mapping does not take place; the mapping was already done when the message was put onto the outgoing mail queue.

Re-Scanning the *rewrite* File

In certain circumstances it's useful to include several records in the *rewrite* file to handle incorrectly addressed mail. For example, the address fritz@dj9zzz.deu would not match any of the templates in the *rewrite* records listed so far. We've seen that the simple address fritz@dj9zzz does match one of the templates, but any other form of this address is unrecognisable.

To handle this situation, NOS provides a re-scan capability for the *rewrite* file. Where a *rewrite* record has the letter **r** at the end, this tells NOS that if the address matches the template in that record, perform

the mapping and then go back to the beginning of the *rewrite* file to scan through it again.

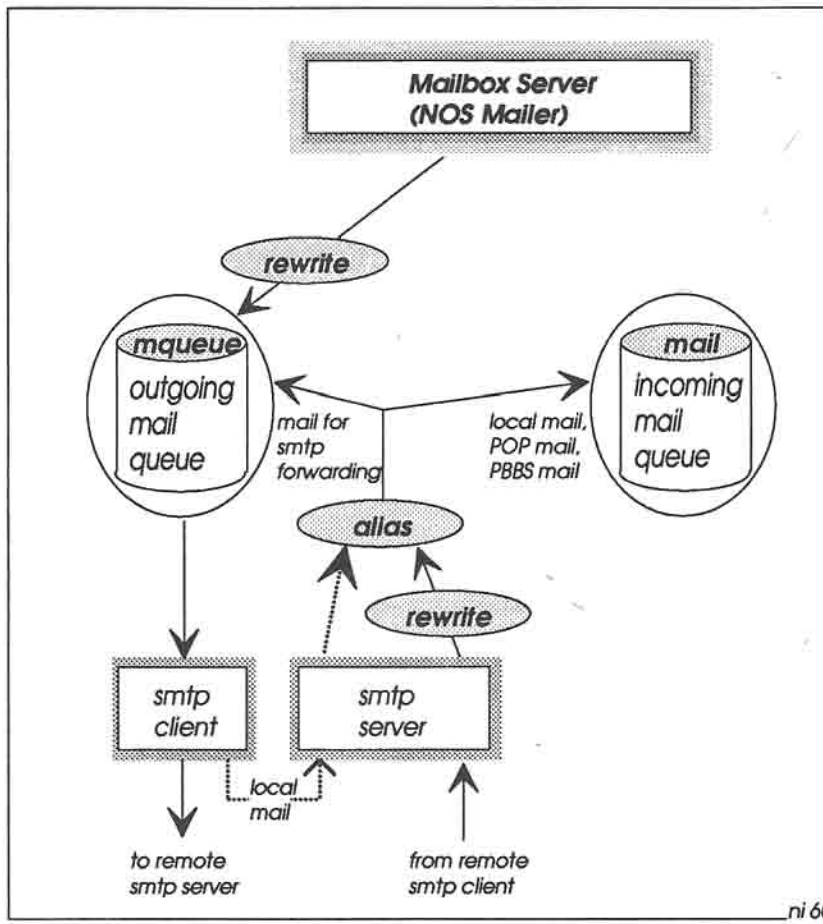


Fig 23-12: The *rewrite* file is read either when a message arrives from a remote system via the SMTP server, or when a locally-addressed message is written to the outgoing mail queue by the NOS mailer.

So, to handle incorrectly addressed messages to DJ stations, the *rewrite* file could contain:

```
*@dj*  $1%dj$2@ns9deu
*@dj*.deu  $1@dj$2 r
```

The address fritz@dj9zzz.deu does not match the template in the first line, but it does match in the second line, and so the address changes to fritz@dj9zzz. Because the second line contains the letter **r** at the end, the scan now re-starts at the beginning of the file, and this time the new address does match the template. The address changes again, to its final form: fritz%dj9zzz@ns9deu.

Handling Incoming Bulletins

Another use for *rewrite* is to handle incoming bulletins (Fig 23-13).

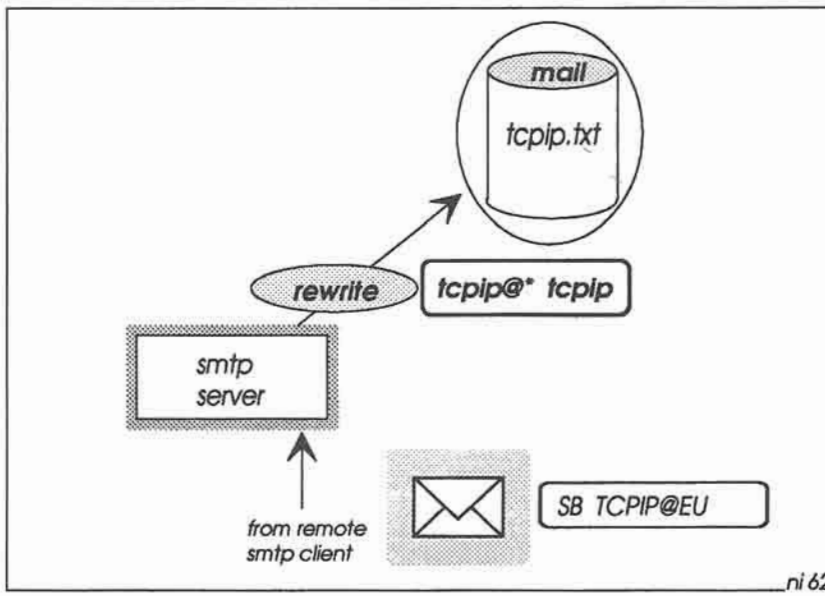


Fig 23-13: The *rewrite* file re-addresses incoming bulletins for delivery into the mail queue.

For example, you may wish to save incoming bulletins addressed to tcpip@gbr or tcpip@eu or tcpip@www.

In this case, the *rewrite* mapping is simple:

```
tcpip@*  tcpip
```

Thus any incoming bulletins addressed to tcpip@anything will go into your local tcpip area.

The Alias File

The *alias* file contains a look-up list which the SMTP server and external mailers use to address mail. For example, if the file contains the alias:

```
ken ns9ken@ns9ken
```

you can send a message to Ken using the BBS command **sp ken**, and the BBS will automatically re-address the message to ns9ken@ns9ken.

(Of course, you could have put this entry in the *rewrite* file instead, where it would have the same effect).

More typically, you can set up a mailing list of several recipients:

```
thegirls ns9pam@ns9pam ns9sue@ns9sue ns9liz@ns9liz outtray
```

and then simply send messages to the list members with the command **sp thegirls**. The last entry on the line (outtray) is a local mailbox file for saving a copy of each outgoing message.

There are three important points to keep in mind when setting up the *alias* file:

1. The alias name (the first entry on each line) must be a local name. That is, it must not contain an @ character.
2. There must be exactly one space between each field in the file.
3. If the list of recipients is more than one line long, you can continue the list on subsequent lines. These continuation lines must start with a space or tab.

For example:

```
theworld ns9ccc@ns9ddd ns9eee@ns9fff ns9ggg@ns9hhh
 ns9iii@ns9jjj ns9kkk@ns9lll outtray
```

There is a space before ns9iii at the beginning of the second line.

Alias File Processing

It's interesting to see how NOS uses the *alias* file (see Fig 23-14).

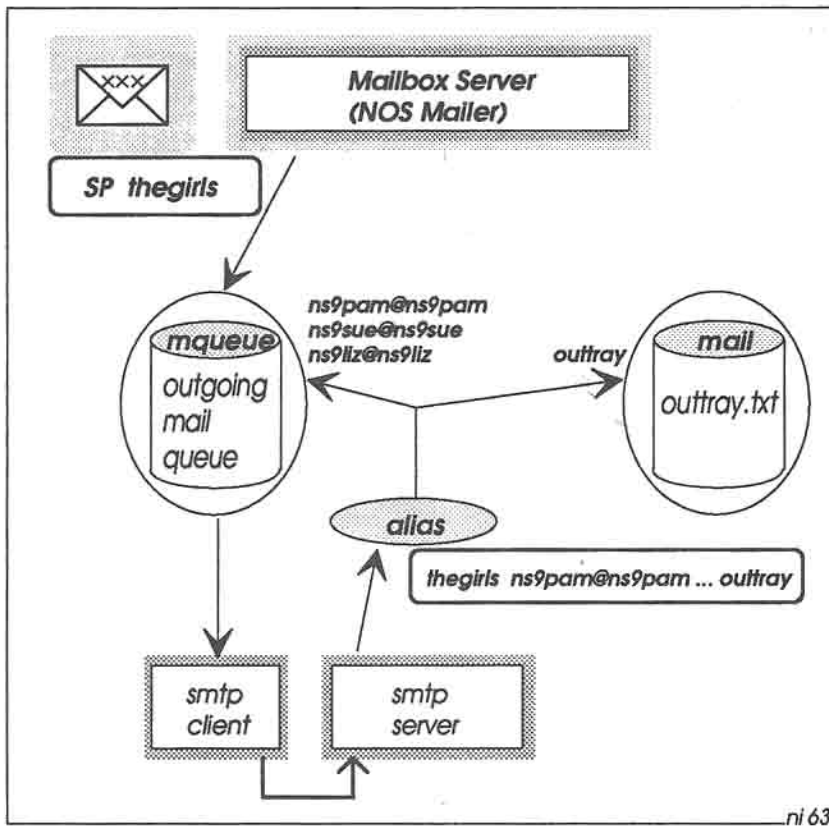


Fig 23-14: The SMTP server uses the *alias* file to create multiple copies of messages.

Starting at the top left-hand corner, you can address a message to the girls with the BBS command **sp thegirls**. The BBS places the message as usual in the outgoing mail queue.

When the SMTP client processes the message, it finds that it is addressed to a local user (thegirls), and so the SMTP client makes an internal connection with the local SMTP server.

It is the SMTP server which reads the *alias* file. For each non-local recipient in the alias list (ns9pam@ns9pam, ns9sue@ns9sue and ns9liz@ns9liz), the server now places a separate copy of the original message in the outgoing mail queue. Thus when the SMTP client next wakes up it will then attempt to deliver these messages to the remote hosts.

And for each local recipient in the alias list (i.e. outtray), the server places a copy of the message in the incoming mail queue.

The important point to remember, then, is that it is the SMTP *server* which uses the *alias* file when handling mail. When you address a message to a local user name which matches an alias (like thegirls), the message leaves the outgoing mail queue under the control of the SMTP client, but immediately re-appears as incoming mail under the control of the local SMTP server.

It is only then that the server expands the alias and creates the necessary copies of the original message for forwarding.

Using *alias* to forward Incoming Messages

Because the SMTP server expands aliases for all messages, regardless of whether the messages originated locally or came from another host, you can use the *alias* file to forward incoming messages onwards to other hosts.

For example, if Ken addresses a message to thegirls@ns9bob, the SMTP server at ns9bob will automatically create new copies for onward transmission to all the recipients in the alias list for thegirls, plus a copy for Bob's outtray.

Finding out somebody else's aliases

If Ken sends a message to thegirls@ns9bob as just described, the end result may be totally unexpected if Ken thought that thegirls was simply a single local user at ns9bob, not an alias for several other users! Ideally Ken would like to interrogate Bob's *alias* file first, to see what would happen if he subsequently addressed a message to thegirls.

When NOS is set up as described in this book, it's not possible directly to read somebody else's *alias* file (by means of FTP or a BBS download for example). This is a security precaution; the *alias* file is at the NOS root level (*N:*\), and we certainly don't want other users snooping around with FTP or the BBS at this level.

The way around this small difficulty is for Ken to ask SMTP to do the snooping for him (Fig 23-15). He does this by giving a special **telnet** command:

```
net>  telnet ns9bob 25
```

The number 25 is the "well-known port" number for SMTP — most of the popular protocols have a fixed port number, and 25 is always the port number for SMTP.

When Ken's TELNET client connects with port 25 on Bob's system, he is actually talking to the SMTP server there. The server first gives him a friendly welcome:

```
220 ns9bob SMTP ready
```

From now on Ken has to talk in language which the server understands. Fortunately SMTP-speak is very simple, with a limited vocabulary of just 9 command words: HELO NOOP MAIL QUIT RCPT HELP DATA RSET EXPN.

To find out if Bob has an alias list for thegirls, all Ken has to do now is give the **EXPN** (expand) command:

```
expn thegirls
```

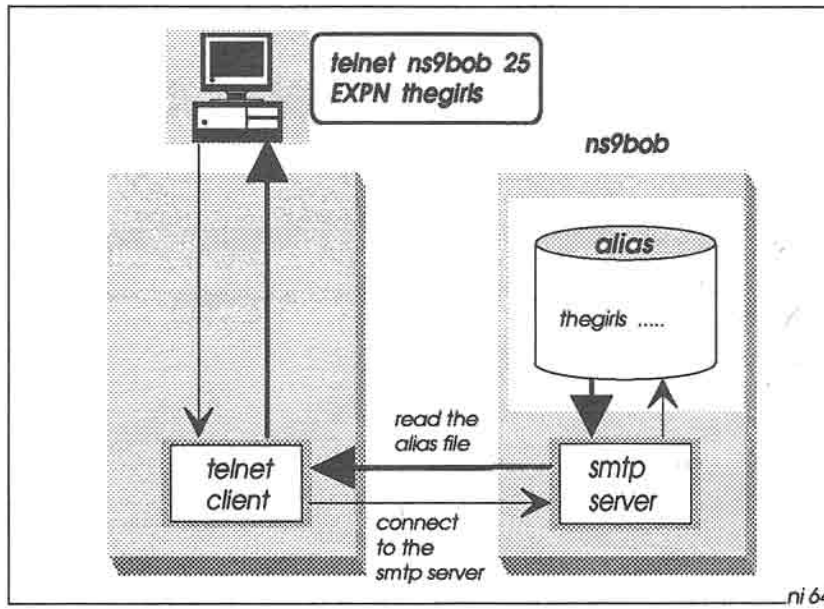


Fig 23-15: Using *telnet* to well-known port 25 allows you to talk direct to the SMTP server, which you can then interrogate with the *EXPN* command to get the *alias* file.

Bob's SMTP server will then respond with the *alias* file entry for thegirls:

```
250-outtray
250-ns9liz@ns9liz
250-ns9sue@ns9sue
250-ns9pam@ns9pam
```

Bingo! Now Ken knows what will happen if he sends a message to thegirls@ns9bob!

And finally, when he has finished with Bob's SMTP server, Ken simply gives the **quit** command to terminate the special TELNET session.

Using *rewrite* and *alias* to handle Incoming Bulletins

We've already seen how to set up a simple *rewrite* record to redirect incoming bulletins addressed to tcpip@anybody into the local tcpip area.

A reminder of the *rewrite* file entry:

```
tcpip@*  tcpip
```

If you also wish to forward such bulletins on to Ken, you can now set up an *alias* entry something like:

```
tcpip tcpip@ns9ken tcpip
```

Now, when a incoming message arrives addressed to tcpip@eu, the *rewrite* mapping will first change the address to the local name tcpip. This now matches the alias name (the first tcpip in the *alias* record), so the SMTP server will place a copy of the bulletin in the outgoing mail queue for tcpip@ns9ken, and a second copy in the local tcpip area for your own system.

Note the order in which the SMTP server manipulates destination addresses: *rewrite* mapping takes place before *alias* expansion.

Listing the SMTP Mail Queue

From time to time you may want to see what messages are still on the outgoing mail queue awaiting forwarding. The **smtp list** command gives you this information:

```
net> smtp list
S   Job Size  Date  Time   Host   From
L   123  244  09/13 07:13 ns9liz  ns9bob@ns9bob  ns9liz@ns9liz
    274 1987  09/13 08:26 ns9tom  ns9bob@ns9bob  tom@ns9tom
```

The left-most column (S) indicates the status of the message. If there is a letter L in this column, the message is locked. That is, SMTP is in the process of forwarding it. In this case, the lock file */spool/mqueue/n.lck* will exist (where *n* is the job number); e.g. */spool/mqueue/123.lck*.

The last three columns show the contents of the work file */spool/mqueue/n.wrk.*

To remove a job from the queue, use the **smtp kill** command. For example:

```
net>  smtp kill 123
```

24: POP MAIL COLLECTION

A NOS station can operate as a "poste restante" system; that is, it can hold mail indefinitely on behalf of other stations which are not normally operational. When those stations are ready to receive their mail, they connect with the poste restante host, which then forwards any outstanding mail, using POP (Post Office Protocol). See Fig 24-1.

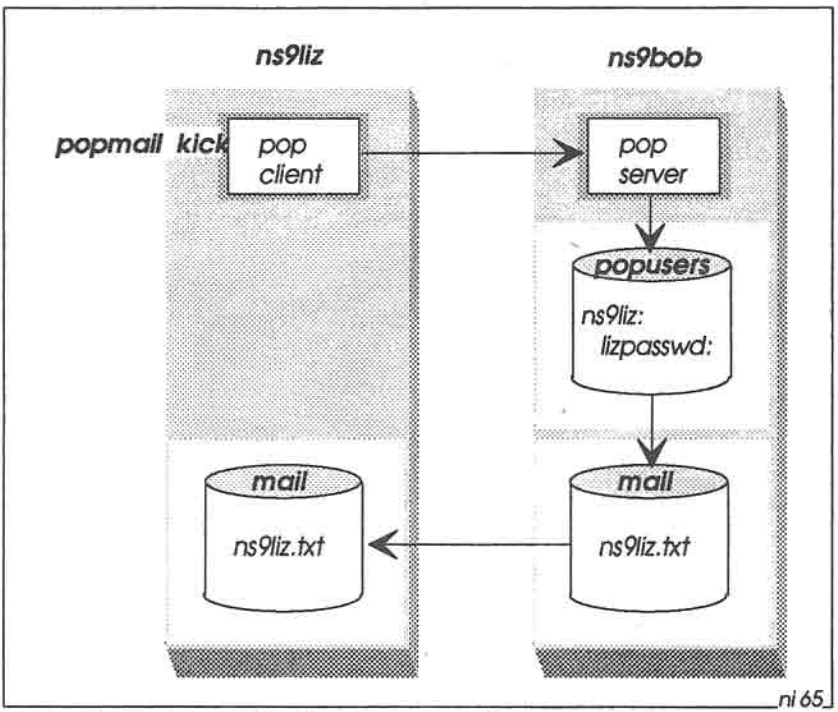


Fig 24-1: The POP client requests the server to forward any outstanding mail (assuming there is a suitable entry in the */popusers* file to authenticate the client).

Setting up the POP Server Host

There are two current versions of the POP protocol: POP2 and POP3. Both of these servers do essentially the same job.

When setting up a host, you should start both POP2 and POP3, so that the system will understand requests from clients running either of these protocols.

The place to start the servers is in *autoexec.nos*:

```
start pop2
start pop3
```

It is also necessary to set up the file */popusers*, which contains the name and POP password of every user for whom you will be holding mail.

For example:

```
#  SYNTAX:      username:password:
ns9liz:lizpasswd:
mary:poppins:
```

Note that both the username and the password terminate in a colon (:).

Finally, you need to ensure that all incoming mail addressed to Liz and Mary finishes up in the incoming mail directory (*/spool/mail*), as the text files *ns9liz.txt* and *mary.txt*. This is where the POP clients will collect it from.

In other words, if you want to send mail to ns9liz, you use a *local* address **(sp ns9liz)**, not a *remote* address **(sp ns9liz@ns9liz)**.

To handle mail coming in from other stations addressed to ns9liz@ns9liz, you will need an entry in the *rewrite* file to force NOS to put the mail into */spool/mail/ns9liz.txt*:

```
ns9liz@ns9liz  ns9liz
```

See Fig 24-2.

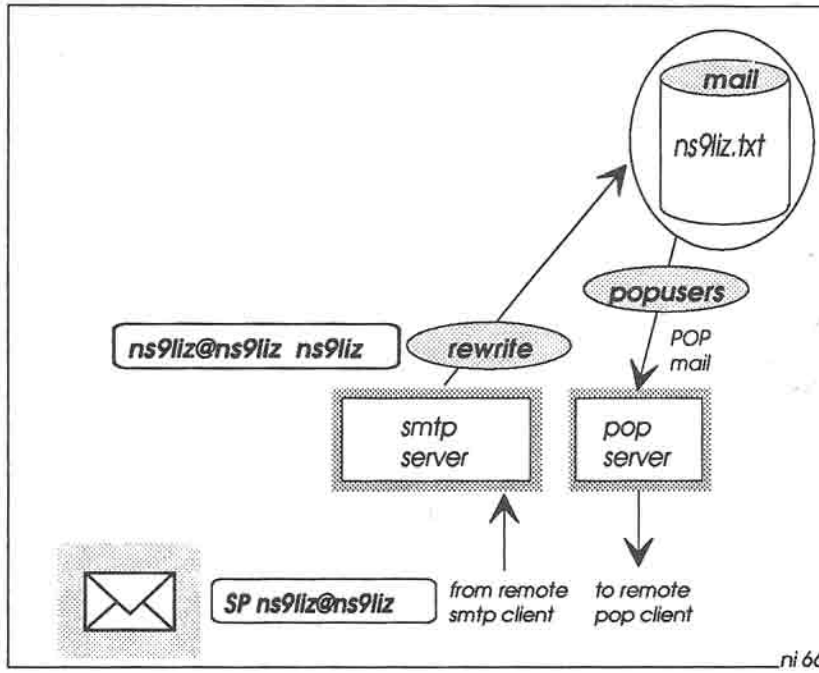


Fig 24-2: Incoming mail addressed to Liz must be directed to the *incoming* mail queue (*/spool/mail*), ready for the POP server. This means that an entry in */spool/rewrite* is needed, to map the remote address *ns9liz@ns9liz* to the local address *ns9liz*.

Setting up the POP Client

For a POP client to collect mail from a POP server, it is necessary on the client system to specify the server name (in *autoexec.nos*). The syntax is (all on one line):

```
popmail addserver host [seconds] [hh:mm-hh:mm] protocol
                                    mailbox username password
```

For example (all on one line):

```
popmail addserver ns9bob 1800 00:00-03:00 pop3 ns9liz ns9liz
                                                    lizpasswd
```

This means that ns9bob is running a POP3 server, which Liz's POP client is to contact at 1800-second (30-minute) intervals between midnight and 3am, to see if there is any mail in Bob's */spool/mail/ns9liz.txt* to be collected. The last two parameters (ns9liz and lizpasswd) must match an entry in Bob's *popusers* file.

The seconds and hh:mm-hh:mm parameters may be omitted. This means that the POP client never wakes up automatically to collect mail. In this case, to force the client to contact the server, Liz must give the command:

```
net> popmail kick ns9bob
```

You can define more than one POP server, by adding more **popmail addserver** commands.

To list all the current POP server hosts:

```
net> popmail list
```

and to remove a server from the list:

```
net> popmail dropserver ns9bob
```

When POP mail arrives, NOS will announce its arrival if you give the command:

```
net> popmail quiet no
```

You can change the setting to yes if you don't want to see the announcement.

Tracing POP Mail Collection

To check that POP mail collection is working properly, you can activate the POP trace.

The trace commands are:

```
net>  popmail trace 0     {no trace, turn trace off}
net>  popmail trace 1     {report serious errors}
net>  popmail trace 2     {report transient errors}
net>  popmail trace 3     {report complete POP sessions}
```

Trace output goes to the session log file (*/dump/session.log*). Note that for trace level 3, POP creates a large amount of data in the session log, so don't forget to turn the trace off if you don't need it.

25: PBBS MAIL FORWARDING

In Chapter 23 we saw how to launch mail onto AMPRnet using SMTP. This works well for sending mail to other stations on AMPRnet, and even to ordinary AX.25 PBBS stations, provided that your immediate neighbour understands SMTP and can somehow forward your mail to an SMTP/PBBS mail gateway for re-launch onto the PBBS network.

Unfortunately, if you don't have a neighbour that understands SMTP, you are forced to launch all of your mail directly onto the AX.25 PBBS network instead (Fig 25-1). Fortunately NOS is fully equipped to do this. Your station will appear to your PBBS neighbour just like any other PBBS, able to forward and reverse-forward private messages and public bulletins in exactly the same way as an ordinary PBBS.

Of course, you may choose to use both methods of launching your mail: SMTP mail goes to an SMTP neighbour and PBBS mail goes to an AX.25 PBBS neighbour. You could even choose to be an SMTP/PBBS mail gateway yourself, and forward SMTP mail received from other people onto the PBBS network (and vice versa).

The Principles of PBBS Forwarding

The basic procedure for forwarding mail onto the PBBS network is very straightforward (Fig 25-2). You address the mail in a similar way to SMTP mail, and through a combination of *rewrite* mappings and *alias* expansions you contrive to collect all the mail for PBBS forwarding into one or more *local* mailbox files in your system (i.e. in directory */spool/mail*).

You also set up a *forward* file (*/spool/forward.bbs*), in which you specify how each of these local mailbox files is to be forwarded. The *forward.bbs* file contains a list of neighbouring PBBS callsigns, and for each PBBS there are one or more commands specifying how and

when NOS is to connect to that PBBS, together with a list of the mailboxes to be forwarded there.

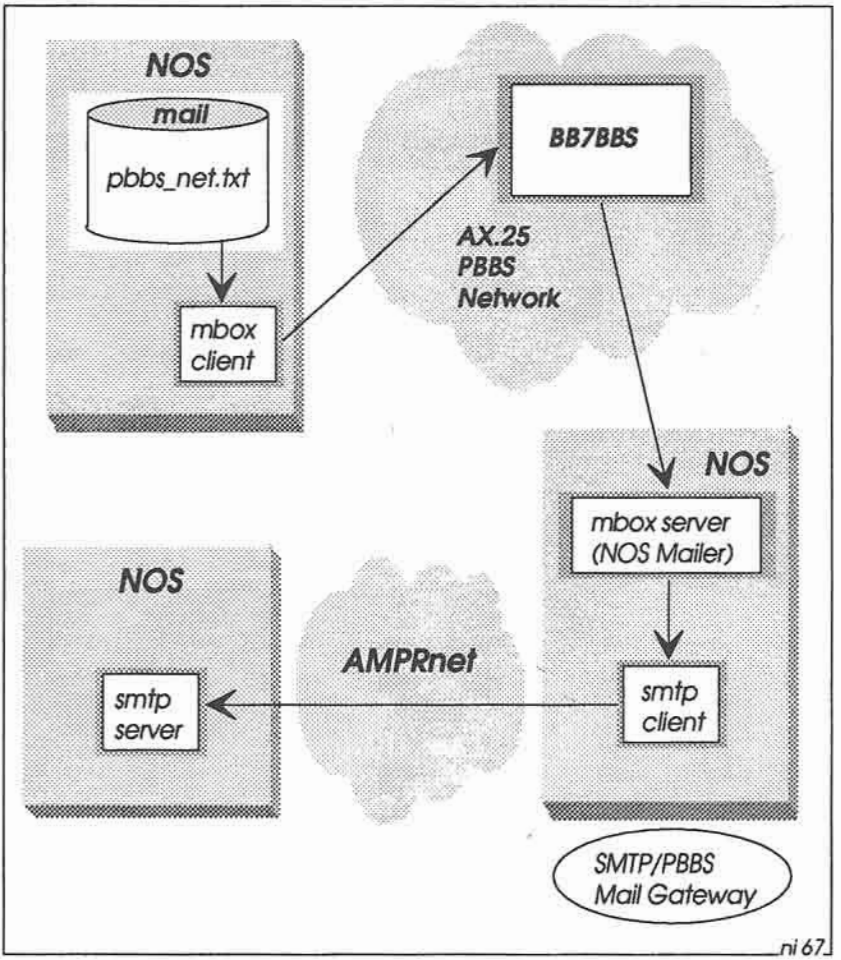


Fig 25-1: Forwarding mail to a NOS station via the PBBS network. The mailbox client forwards *.txt* files in the mail queue (*/spool/mail*) to a neighbouring AX.25 PBBS (*BB7BBS*). This BBS then forwards the mail to an SMTP/PBBS mail gateway. The gateway converts the mail to SMTP format, and forwards it via AMPRnet to the final destination.

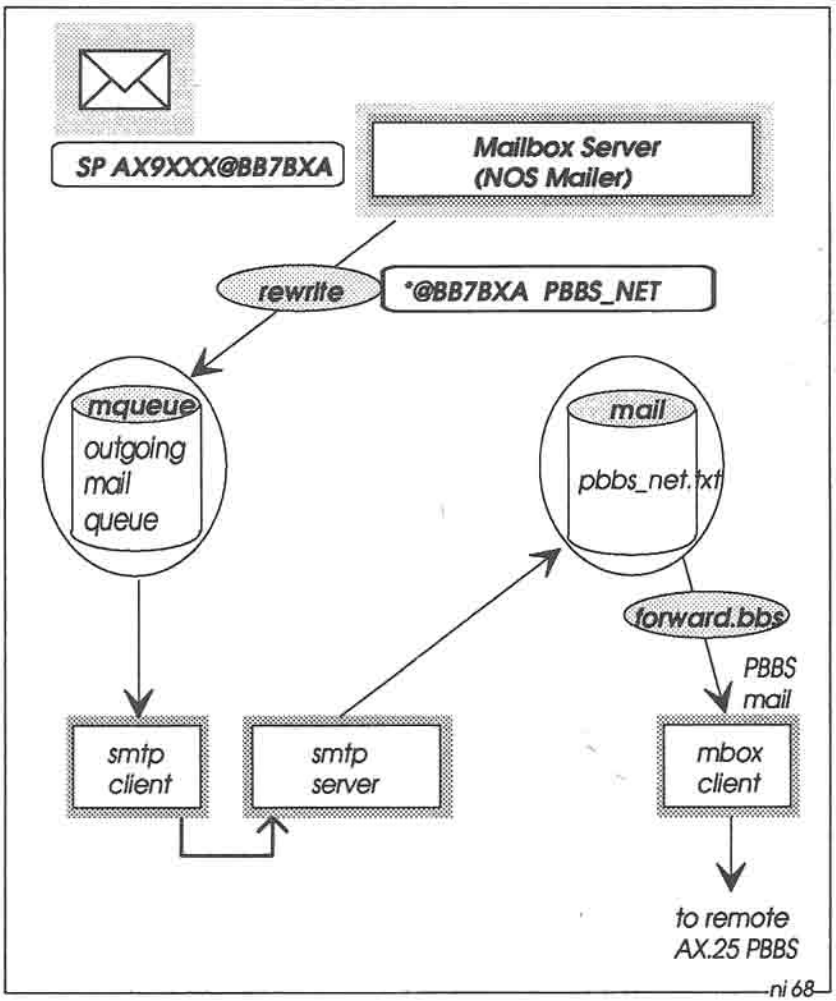


Fig 25-2: Forwarding mail to an AX.25 station. The *rewrite* file maps all messages addressed *@BB7BXA* to a local address (*PBBS_NET*). These messages finish up in the mail queue in *pbbs_net.txt*. From there the mailbox client forwards them to the PBBS specified in */spool/forward.bbs*.

The NOS mailbox client is responsible for reading the *forward.bbs* file and initiating the forwarding process — usually at regular intervals specified by the NOS **mbox timer** command, but you can force the

client to read the file immediately with the **mbox kick** command if you want.

For each PBBS listed in the *forward.bbs* file, the mailbox client attempts a connection, and then forwards the messages contained in the nominated local mailbox(es) to the PBBS. On completion, the PBBS then automatically reverse-forwards any outstanding mail which has been waiting at the PBBS for delivery to the NOS station.

Preparing for PBBS Forwarding

The biggest job in preparing for PBBS forwarding is setting up the *rewrite* and *alias* files, so that the PBBS network will recognise the destination addresses in your mail. These files will probably be quite long and complicated, as there are many and varied ways of addressing PBBS mail in different parts of the world.

And because there are so many ways of addressing PBBS mail, the potential for getting it totally wrong is enormous! So the *rewrite* and *alias* files have to catch and correct any *incorrectly* addressed mail as well.

By comparison, the *forward.bbs* file is straightforward to set up. All you have to decide is which PBBS mailhost(s) you want to forward to, what time(s) of day the forwarding is to take place, how to make the connections to the mailhosts (e.g. by an ordinary AX.25 **connect**, or via a NET/ROM node), and which NOS mailbox files are to be forwarded.

Finally, you have to tell the NOS mailbox client the hierarchical AX.25 address of your neighbouring PBBS. NOS automatically appends this to your own AX.25 callsign in the R: line of outgoing messages, so that any replies will be routed back to you correctly.

Let's Start Simple

To illustrate the basic mechanism for forwarding mail onto the PBBS network, let's look at what happens when you want to send a correctly addressed message to an ordinary AX.25 station (see Fig 25-3).

The message is addressed to AX9XXX@BB7BXA, somewhere across the PBBS Network, and BB7BBS is the callsign of the neighbouring

PBBS to which you will launch the message. For the moment, we'll assume that you can simply address the message with the command **sp AX9XXX@BB7BXA,** and that your neighbour BB7BBS knows how to add the necessary hierarchical address extension for the target mailhost BB7BXA.

Remembering that PBBS forwarding requires the message to be placed in a *local* mailbox, the first step is to set up a *rewrite* file record to translate the address AX9XXX@BB7BXA to a local address (i.e. an address without the @ symbol). Refer back to Fig 25-2. If we arbitrarily call the local mailbox PBBS_NET, the *rewrite* record looks like this:

```
*@BB7BXA PBBS_NET
```

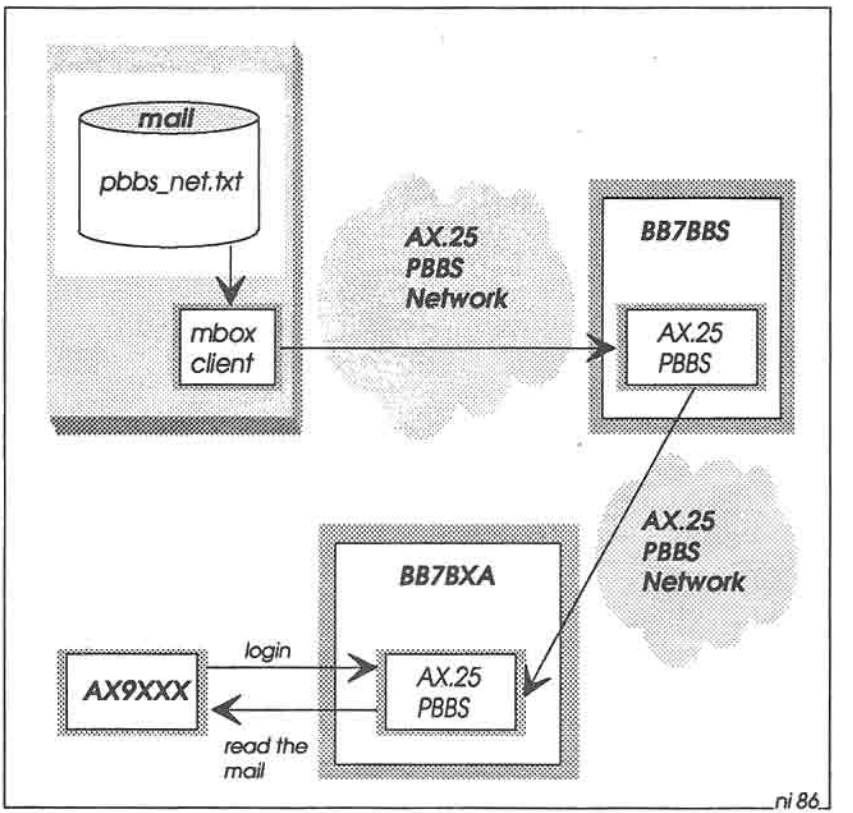


Fig 25-3: Forwarding to a remote AX.25 PBBS.

That is, any message addressed to anyone at BB7BXA goes into the local mailbox PBBS_NET (i.e. into the file */spool/mail/pbbs_net.txt*).

The second step in preparing for PBBS forwarding is to set up */spool/forward.bbs*. This needs an entry for your neighbour BB7BBS:

```
BB7BBS
connect tnc0 BB7BBS
PBBS_NET
```

The first line specifies the AX.25 callsign of the neighbouring PBBS to which NOS will forward the message.

The second line tells the mailbox client the method of connecting to the neighbour; in this case, it makes an ordinary AX.25 connection.

The last line states which local NOS mailbox file is to be forwarded to BB7BBS.

The PBBS Routing Header

The final step in preparing for PBBS forwarding is to tell the mailbox client certain information for inclusion in the PBBS routing header (the R: line) of your outgoing messages. Without this information you may not receive any replies to your messages and you will probably cause chaos to the PBBS forwarding network as well (not to mention thoroughly confusing any White Pages servers which happen to come across your messages...).

To ensure that the mailbox client sets up the R: line correctly, you should include lines like the following in your *autoexec.nos* file:

```
mbox qth          "[London]"
mbox haddress     "BB7BBS.#41.GBR.EU"
mbox utc          0
```

This will result in a line appearing automatically in the routing header of your PBBS messages; e.g.

```
R:920514/1543z @:NS9BOB.BB7BBS.#41.GBR.EU [London] #:234
```

That is, the R: line contains a date/timestamp for the message, the full hierarchical PBBS address of your station (i.e. your own callsign followed by the address of BB7BBS), your QTH and the message number.

The **mbox utc** command lets you specify a ± time offset from UTC. NOS uses this offset to calculate the UTC time which appears in the routing header.

You can add some more information to the routing header, with these commands in *autoexec.nos*:

```
mbox zipcode  123456
mbox fwdinfo  "HNLNET BBS"
mbox smtptoo  off
```

The first two of these are only for background information, and are not really required for forwarding. It's probably best to omit them altogether, as they only make the R: line longer than necessary.

The **mbox smtptoo** command specifies whether or not the full SMTP message headers are to be included in the message. These headers are often very long (sometimes running to dozens of lines) and are completely unnecessary if you are sending a message to an AX.25 station that doesn't understand SMTP. In that case, you should set this option to off.

On the other hand, if the addressee is running NOS and you expect a reply, you will need to set smtptoo to on.

Sending the Message

Having set up the necessary files, you are now ready to send the message. When you log into your NOS BBS and give the command **sp AX9XXX@BB7BXA**, the BBS scans the *rewrite* file and converts the address to PBBS_NET (but recall from Chapter 23 that this conversion only affects the *n.wrk* file in the outgoing message queue — the To: address in the *n.txt* file remains *exactly* as you typed it: AX9XXX@BB7BXA).

When you've finished composing your message, the NOS BBS places it in the outgoing message queue as usual. Then the SMTP client connects with the local SMTP server, which delivers the message to the

local PBBS_NET mailbox file (*/spool/mail/pbbs_net.txt*), in exactly the same way as described in Chapter 23.

The SMTP server now displays an alert on the screen, saying that new mail has arrived for PBBS_NET. You just ignore this — the server thinks this is ordinary local mail, and has no way of knowing that it's really PBBS mail waiting to be forwarded.

Finally, when the mailbox client wakes up, it examines the *forward.bbs* file and finds that local mailbox PBBS_NET is to be forwarded to BB7BBS. So the mailbox client connects to BB7BBS and transfers the message in PBBS_NET to BB7BBS. Your message is on its way at last! (If the mailbox client doesn't wake up, you can bring it to life with the command **mbox kick** to force it to connect to the PBBS).

More *rewrite* Records

Nice n' easy so far. Trouble is, you'll certainly want to send messages to PBBSs other than BB7BXA, so you'll need many more records in the *rewrite* file.

If you live in the United Kingdom, the situation is not too bad. Almost every UK PBBS has a GB7 callsign, with a few GB3 stations as well. In this case, the *rewrite* file looks like this:

```
*@GB7* PBBS_NET
*@GB3* PBBS_NET
```

In addition, the UK PBBS network already has a well-organised system of hierarchical addressing which is understood and maintained by all PBBS stations, so it's not necessary to think about hierarchical addresses when forwarding from NOS onto the PBBS network (and this is how it should be!).

Thus in the UK, a NOS user can simply address a message to G3NRW@GB7BIL, for example, without having to know the full hierarchical address of GB7BIL.

The situation in many other countries is much more irregular, and in the extreme you may need a separate *rewrite* record for every individual PBBS station you send messages to.

A Closer Look at Hierarchical Addressing

In general, a full hierarchical address is one of these:

```
CALLSIGN@PBBS_CALLSIGN.COUNTRY.CONTINENT
CALLSIGN@PBBS_CALLSIGN.STATE.COUNTRY.CONTINENT
CALLSIGN@PBBS_CALLSIGN.AREA.STATE.COUNTRY.CONTINENT
```

For example:

```
AX9YYY@BB7BBS.#41.GBR.EU
```

The Continent (EU) and Country (GBR) codes are internationally agreed.

The state/county/area codes (e.g. #41) are allocated locally within a country.

The problems arise when people use different codes for the same thing (e.g. some people use NA for North America, whereas others use NOAM), or when they only supply part of the full hierarchical address.

To overcome these difficulties, the *rewrite* file is a little more complicated. To handle incomplete/incorrect addresses for New York, for example, the *rewrite* file may look something like this:

```
*.NOAM              $1.NA           r
*.US                $1.USA.NA       r
*.USA               $1.USA.NA       r

*.NY                $1.NY.USA.NA    r

*@*.*.NY.USA.NA     $1%$2.$3.NY.USA.NA@BB7BBS   r

*@BB7BBS            PBBS_NET
```

The first three lines change NOAM to NA, and US to USA.

The fourth lines handles a message addressed to ANYONE.NY.

The fifth line re-addresses NY mail to the local PBBS.

The last line re-addresses local PBBS mail to PBBS_NET.

Get the idea? Note the letter **r** at the end of most lines; this forces NOS to re-scan the *rewrite* file after a match, so that you progressively work down the file as the address is refined for the PBBS_NET mailbox.

All you have to do now is finish off the file, adding all the remaining possible combinations of addressing that you are likely to encounter — that'll wipe the smile off your face for an hour or two!

One of the main objectives of the *rewrite* file is to re-address PBBS mail correctly.

This means that you have to get it right, which in turn means that you have to ***test*** it.

In other words, you should send dummy messages which exercise every record in the *rewrite* file, and check that the correct addresses are set up in the *n.wrk* files.

It hopefully goes without saying that you should turn off the tnc and the radio before you start testing. You should afterwards remove all the dummy messages from the outgoing mail queue (with the NOSview command **CLEANQ.BAT**) before turning the tnc and radio back on again!

Otherwise you'll set the network on fire ....

The *rewrite* file handles the re-addressing of ***all*** mail, irrespective of whether the mail originates locally or comes from a remote system, and irrespective of whether it is SMTP mail or PBBS mail. So there may be many address combinations you need to consider.

More on the *forward.bbs* File

Thus far we have only seen an example of a very simple *forward.bbs* file. Let's look at the more general form:

```
PBBS Callsign     {optionally followed by forwarding timeslots}
Connection Method
.Connection Command(s)              {zero or more lines}
NOS Mailbox(es) to be forwarded     {one per line}
----                                {end of record}
```

Each *forward.bbs* record contains at least three lines, plus an end-of-record line (a line starting with a minus character).

The first line contains the AX.25 callsign of the PBBS to which mail is to be forwarded. Optionally, you can also include one or more timeslots during which the forwarding is to take place.

Each time slot is expressed as a 4-digit number (e.g. 0003): the first two digits (00) specify the start hour, and the second two digits (03) specify the finish hour.

Thus the timeslot 0003 means that forwarding can take place between midnight and 0359am. You can specify several timeslots if you wish, separating the timeslots with commas. The default timeslot is 0023; i.e. from 0000 to 2359.

So, for example, to forward to BB7BBS during the timeslots 0000–0359 and 1100–1159 hours, the first line in the *forward.bbs* record will be:

```
BB7BBS 0003,1111
```

The second line in the *forward.bbs* file specifies how NOS is to make a connection to the PBBS. There are several possible methods, but the only two which you are likely to use are an ordinary AX.25 connect or a NET/ROM connect.

The syntax of an AX.25 connect in the *forward.bbs* file is:

```
connect interface callsign  {or}
ax25 interface callsign
```

For example:

```
connect tnc0 BB7BBS
```

To access a PBBS via the NET/ROM network, the syntax of the connection commands is:

```
netrom NET/ROM_alias | callsign
.connection_command(s)
```

For example, if Bob wants to forward mail from his local mailbox EURO_NET to BB7EUR via his local NET/ROM node NRA the complete *forward.bbs* file record looks like this:

```
BB7EUR
netrom NRA
.C BB7EUR
EURO_NET
---
```

That is, NOS will issue a NET/ROM connect request to NRA, and when connected will issue a connect request **(C BB7EUR)** to the NET/ROM node.

Note that the line containing this request starts with a dot. Note also that there must be an entry in the NET/ROM routing table for NRA; see Chapter 29 for details on how to set this up.

Bulletin IDs

The PBBS network uses bulletin identifiers for public bulletins, and uses these identifiers to check whether an incoming bulletin has already been received. NOS understands these identifiers, and you can add them yourself to your own bulletins if you wish (with the $ parameter; e.g. **SB TCPIP@WWW $1234**).

NOS maintains a list of bulletin IDs (BIDs) already received, in the file */spool/history*. See Fig 25-4. When a bulletin arrives, NOS checks to see if its BID is already in the *history* file. If it is, NOS tells the sending station that it already has the bulletin. Otherwise, NOS accepts the bulletin and adds its BID to the *history* file.

Note that bulletin IDs are a PBBS network phenomenon. SMTP mail does not understand the concept of bulletin IDs, and so doesn't do anything with the *history* file.

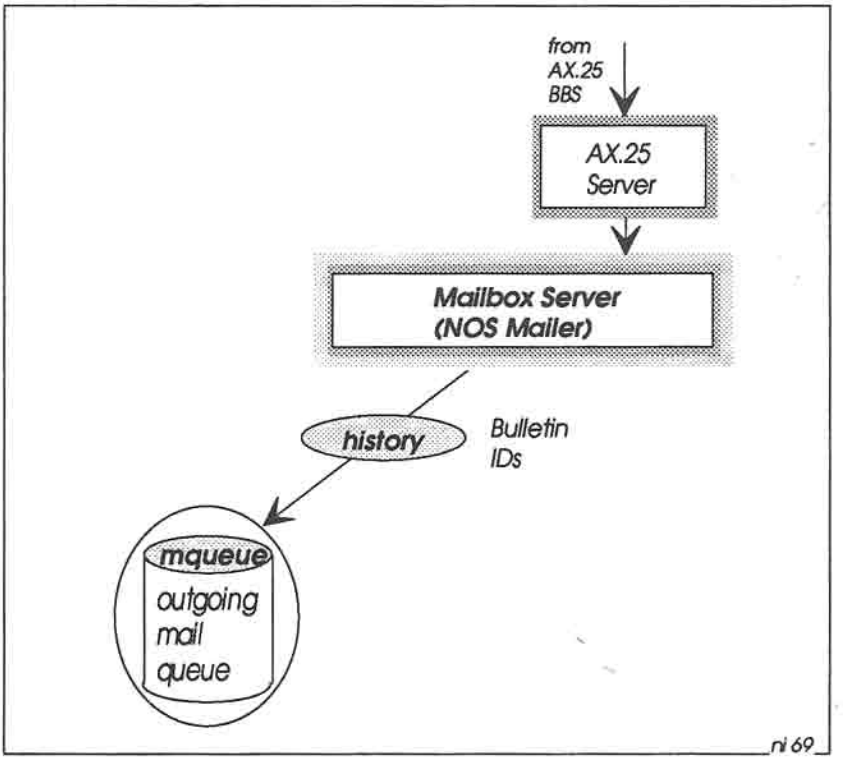


Fig 25-4: When a bulletin arrives at the NOS BBS, its bulletin ID is checked against the history file. If the BID exists in the history file, the BBS tells the originating station that it has got the bulletin already.

26: AX.25 ROUTING

Routing is to do with how NOS forwards packets from one station to another. The rules for forwarding packets are contained in routing tables — in general, these tables specify the next station "down the line" to which packets will go on their way to their eventual destination.

NOS provides three separate levels of routing, and thus three separate routing tables:

- AX.25 routing
- Internet Protocol (IP) routing
- NET/ROM routing

In this chapter we look in detail at AX.25 routing.

AX.25 Routing

AX.25 routing is perhaps the easiest of the three levels to understand, as you're probably familiar with it already, under its more common name: *digipeating*.

To set up the AX.25 routing table, all you need to do is include the **ax25 route add** command for each of your neighbours who is accessible via one or more digipeaters.

So, for the scenario in Fig 26-1, Bob needs the following commands in *autoexec.nos*:

```
ax25 route add AX9PAT AX9DGA AX9DGB
ax25 route add NS9PAM-5 AX9DGC
```

Note that there is no letter **v** (for **via**) in these commands.

If Bob wants to make an ordinary AX.25 connection to Pat, he simply gives the command:

```
net> connect tnc0 AX9PAT
```

and NOS will automatically make the connection via the digipeaters AX9DGA and AX9DGB.

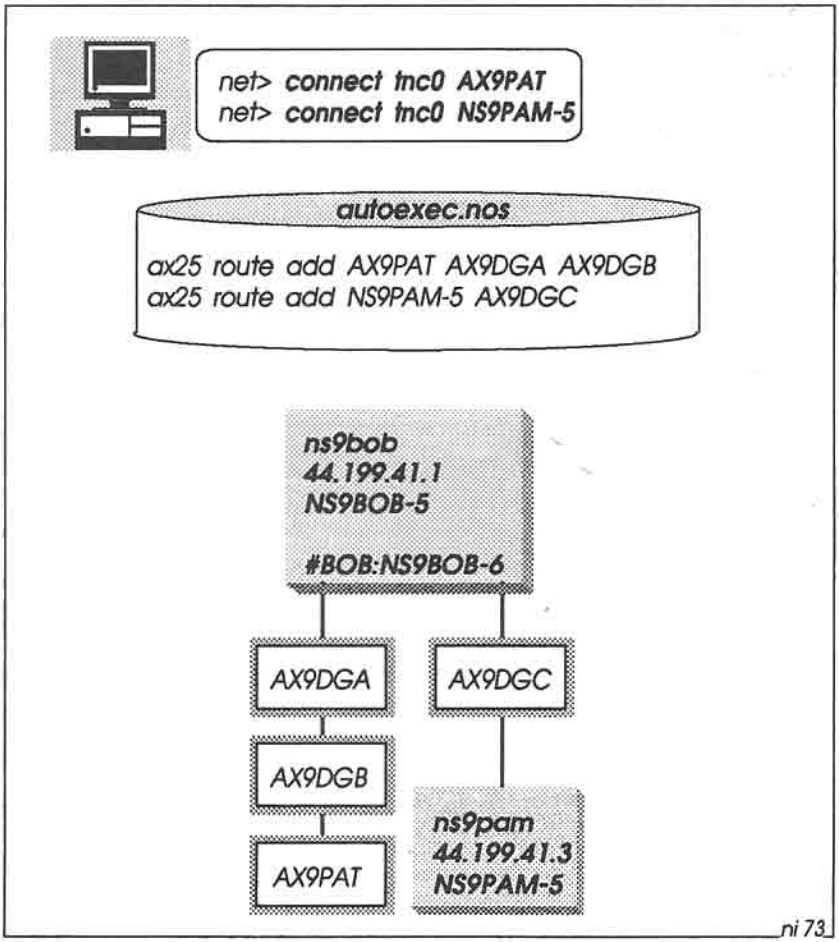


Fig 26-1: The AX.25 routing table specifies digipeater paths.

In the second routing table entry, NS9PAM-5 is the AX.25 callsign of the NOS station ns9pam. Bob could make an ordinary AX.25 connection via AX9DGC to Pam if he really wanted to, using **connect tnc0 NS9PAM-5** (but because Pam is running NOS, Bob is much more likely to use **telnet ns9pam**).

These two entries are permanent entries in the AX.25 routing table. They will remain there indefinitely unless you forcibly remove them (e.g. with the command **ax25 route drop AX9PAT**).

Another way of adding permanent entries to the table is to include the digipeater routing in the **connect** command:

```
net>  connect tnc0 AX9ZZZ AX9DGA
```

(N.B. Some versions of NOS don't allow you to include digipeaters in the **connect** command in this way; in that case you are forced to use **ax25 route add** to specify the digipeater chain).

In addition to permanent entries, it's also possible that temporary entries will find their way into the AX.25 routing table. These appear automatically if someone connects to you via a digipeater path (but if you already have an entry for that station in the table, the incoming connection will not change what you entered).

To find out the current status of the AX.25 routing table, use the **ax25 route** command:

```
net>  ax25 route

Target     Type  Mode Digipeaters
AX9PAT     Local  VC  AX9DGA AX9DGB
NS9PAM-5   Local  IF  AX9DGC
AX9XYZ     Auto   IF  AX9DGA
```

The first two entries are Local entries, corresponding to the permanent routes which Bob added to the table. The third entry appeared automatically when AX9XYZ connected to Bob via AX9DGA.

AX.25 Mode

In NOS it's possible to specify the mode for transmitting Level 2 AX.25 frames. For ordinary AX.25 connections — not using TCP/IP — the mode needs to be **vc** (virtual circuit). That is, when you connect to another AX.25 station, each information frame is numbered (from 0 through 7) to allow stations to check for missing or duplicate frames, and flow control is handled with RR (Receive Ready) frames.

With TCP/IP traffic however, there is no need to number each level 2 frame, because TCP/IP packets have their own numbering scheme. Likewise, there is no need to handle flow control at level 2, because the TCP/IP protocols do this at higher levels.

Because of this, the use of AX.25 virtual circuit mode for TCP/IP traffic is an overkill (and in fact leads to excessive and unnecessary level 2 traffic).

Instead, it's normal to use **datagram** mode at level 2 for TCP/IP traffic. This means that information is transmitted in UI (Unnumbered Information) frames, without frame sequence numbers, and there is no flow control at level 2. If a level 2 frame is lost in transit, the higher level protocols detect this and request re-transmission. Similarly, if the receiving station needs to impose flow control, it is the higher level protocols which handle this.

So, as most of our NOS traffic will be via TCP/IP rather than ordinary AX.25 connections, it's usual to set the default mode to **datagram**, in *autoexec.nos*:

```
mode tnc0 datagram
```

This won't do for ordinary AX.25 connections, however, which need to run in virtual circuit mode. So for each station with whom you normally connect in AX.25 mode, you need to add an **ax25 route mode** command. Thus Bob will have the following additional commands in his *autoexec.nos*:

```
ax25 route mode AX9PAT vc
ax25 route mode AX9DGA vc
ax25 route mode AX9DGB vc
ax25 route mode AX9DGC vc
```

27: ADDRESS RESOLUTION PROTOCOL

The question now arises: if Bob gives a command like **ftp ns9ken**, how does NOS relate the hostname in the **ftp** command (ns9ken) to Ken's AX.25 callsign (NS9KEN-5)? The answer is found in another table: the ARP (Address Resolution Protocol) table.

It's in the ARP table that you can enter the association between IP hostnames and AX.25 callsigns, for stations within radio range. Thus in Bob's case, there could be the **arp add** command in his *autoexec.nos*:

```
arp add ns9ken ax25 NS9KEN-5
```

The command **arp** shows the current state of the ARP table:

```
net> arp

rcvd 0 badtype 0 bogus addr 0 reqst in 0 replies 0 reqst out 0
IP addr         Type            Time Q Addr
44.199.41.2     AX.25           0      NS9KEN-5
```

Thus when Bob gives the command **ftp ns9ken** (see Fig 27-1), NOS first looks in *domain.txt* for Ken's IP address (44.199.41.2), and then uses this address when looking in the ARP table for Ken's AX.25 callsign (i.e. NS9KEN-5). Now that NOS knows the callsign, it can build the AX.25 frame for transmission to Ken.

Dynamic ARP Table Updates

The **arp add** command puts permanent entries into the ARP table; they will stay there for all time (unless you forcibly remove them with the **arp drop** command; e.g. **arp drop ns9ken**).

However, it's not always a good idea to put permanent entries in the ARP table — Ken may change his AX.25 callsign from NS9KEN-5 to NS9KEN-9 without telling Bob, and so the entry for ns9ken in Bob's ARP table will now be wrong (and **ftp ns9ken** will no longer work).

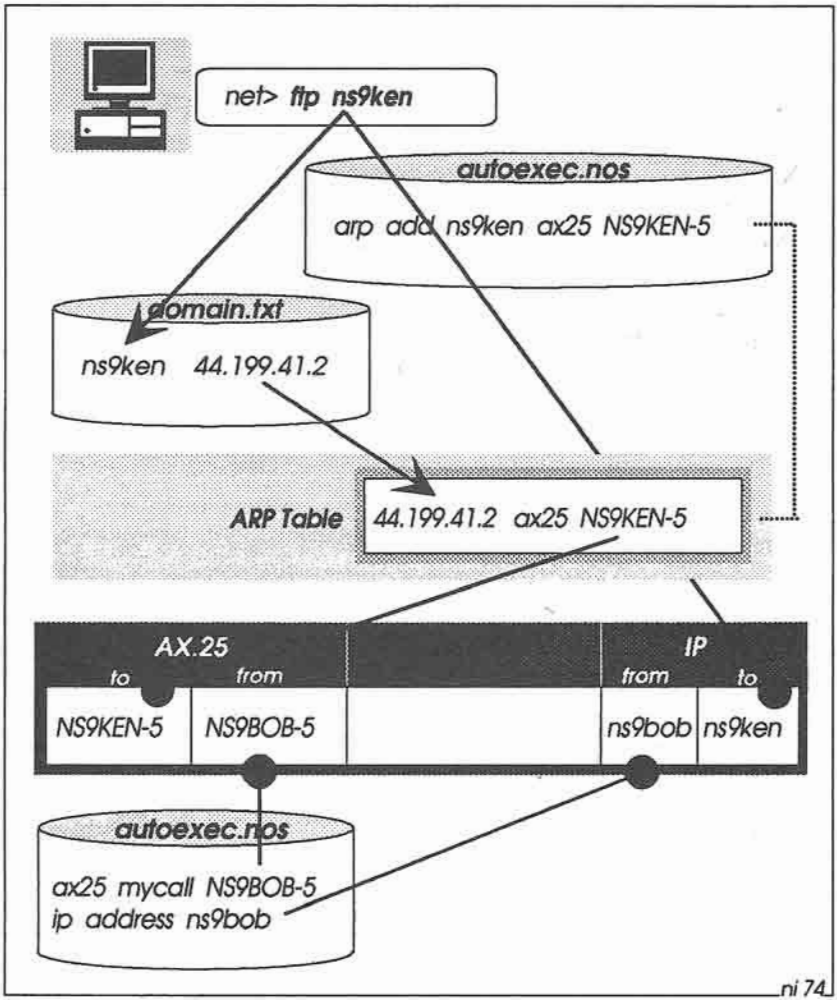


Fig 27-1: When building an AX.25 frame for transmission, NOS looks in *domain.txt* and then the ARP table to find out the AX.25 callsign of the target station. This callsign goes into the AX.25 "to" field in the frame. The *ax25 mycall* and *ip address* commands specify what goes into the "from" fields.

A better solution is to forget about putting permanent entries in the ARP table, and let NOS find out for itself the AX.25 callsign of any station you wish to contact. For example, if Bob gives the command **ping ns9mxa**, NOS will look in the ARP table as usual for an entry for ns9mxa, but won't find a match. NOS will then broadcast an *ARP Request* onto the network, asking for NS9MXA's callsign. See Fig 27-2.

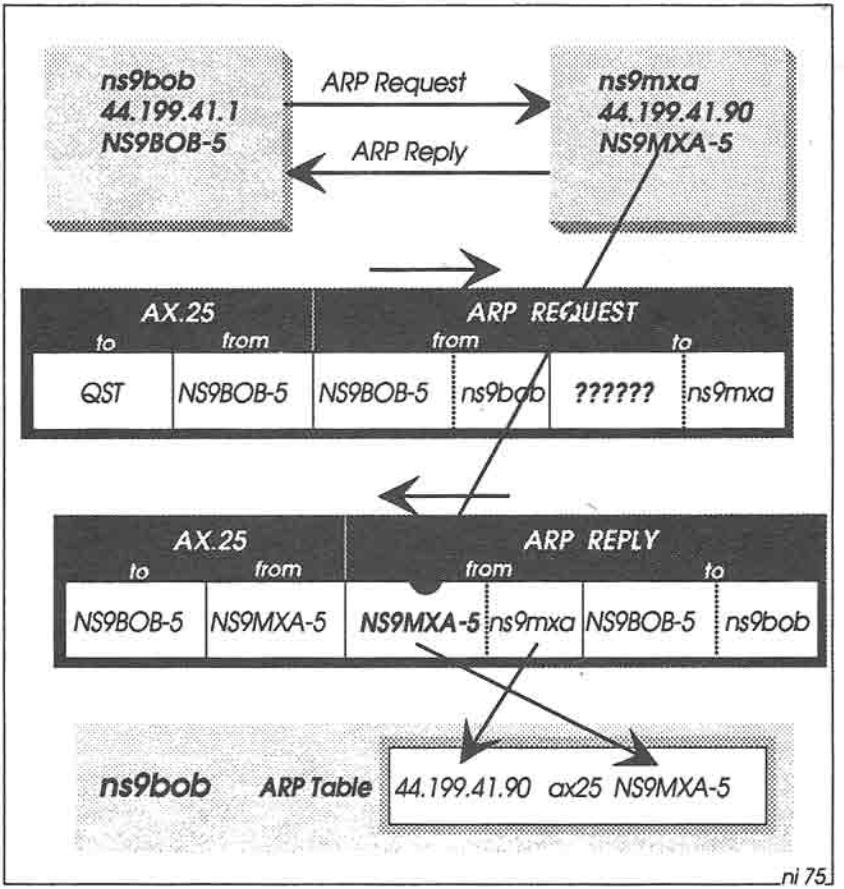


Fig 27-2: When there is no entry for a target station in the ARP table, NOS broadcasts an *ARP Request* (addressed to *QST*, the general AX.25 broadcast address). The target responds with an *ARP Reply* containing its AX.25 callsign. Using this reply, NOS then creates a dynamic ARP table entry.

The request packet contains NS9MXA's IP address (44.199.41.90). Assuming NS9MXA is in radio range, it will then automatically respond with an *ARP Reply* packet containing its AX.25 callsign (NS9MXA-5). Bob's NOS places the callsign his ARP table, and can now use it to address subsequent packets direct to NS9MXA.

The callsign that NOS places in the ARP table in response to an ARP Request is a temporary entry which has an initial lifetime of 900 seconds (15 minutes). You can see the current lifetime with the **arp** command:

```
net> arp

rcvd 1 badtype 0 bogus addr 0 reqst in 1 replies 0 reqst out 0
IP addr         Type            Time Q Addr
44.199.41.90    AX.25           879    NS9MXA-5
```

If no further packets are heard from the station within this lifetime, the entry will eventually disappear.

If you want to remove dynamic ARP table updates, use the command:

```
net> arp flush
```

This will not affect any entries which you set up manually.

The advantage, then, of allowing NOS to maintain the ARP table automatically is that you always know that the AX.25 callsigns in the table are correct. You don't have to find out these callsigns yourself, and you don't have to worry about people changing them.

A further benefit is that the ARP table now only contains temporary entries from stations that are *currently active*, and thus potentially contactable. If you try to make contact with a station that has a permanent entry in the table, you don't know if the station is actually reachable (it may be switched off). In consequence, NOS may waste a lot of air-time repeatedly trying to make contact with a non-existent station.

28: IP ROUTING

AX.25 routing, as described in the Chapter 26, is a form of *source routing*; that is, the entire route to the target is pre-defined at the sending end (the source). At the IP and NET/ROM layers, however, the sending end doesn't need to know the entire route. Instead, each station along the route forwards packets in roughly the right direction towards the target, on the assumption that the next station down the line does likewise.

This method of routing gives us much more flexibility in getting packets to their destination. The packet radio network is an ever-changing and unpredictable environment — nodes appear and disappear overnight — and so it's unrealistic to try to plan every hop along the route. Instead, we let individual stations along the way decide how to forward our packets, hoping that those stations can handle local circumstances which we may be blissfully unaware of!

Thus the only likely situation where we need to use AX.25 routing is for local links via digipeaters to neighbouring NOS (IP) stations. Once our packets reach a NOS neighbour, routing will then take place at the IP layer.

This chapter is concerned with how packets are routed at the IP layer. This means setting up the IP routing table to make sure that when NOS receives a packet addressed to another station, it knows where to forward it to.

IP Forwarding

It's important to recognise which packets NOS will forward. By default, when not actually transmitting, NOS listens on the channel, and will hear packets from all and sundry, addressed to all and sundry. For forwarding purposes, NOS will ignore all of these packets unless

they happen to be broadcasts packets, or they are addressed to itself *at the AX.25 level.*

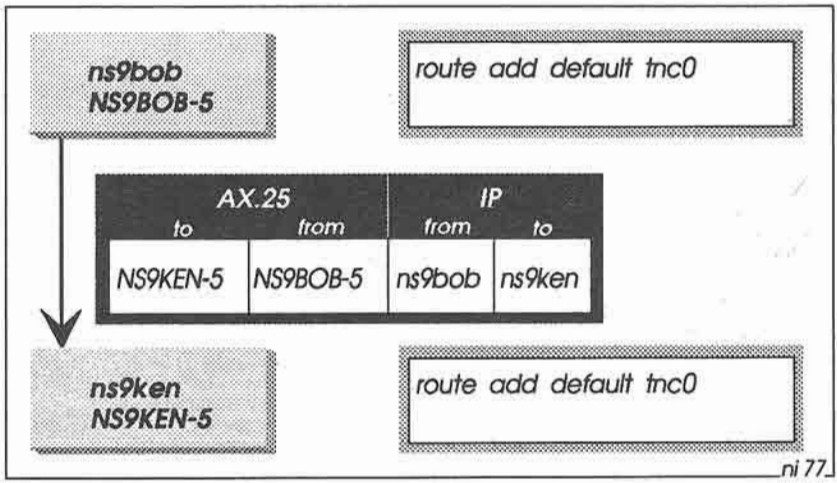


Fig 28-1: The *route add default* entry in the IP routing table handles all direct connections; i.e. all connections to stations within radio range and not passing through intervening IP gateways (routers).

For example, in Fig 28-1, the AX.25 destination address is NS9KEN-5, so only Ken's station will act on the packet — any other stations which hear the packet will ignore it. When the packet passes up to the IP layer at Ken's station, NOS discovers that the IP destination address is ns9ken, so the packet has reached the end of the line. NOS then passes the packet up to the higher protocol layers for further processing in Ken's system.

The generic command to add an entry into the IP routing table is **route add**, and the entry for this particular scenario is very simple:

```
route add default tnc0
```

That is, by default all IP packets are transmitted via the tnc0 interface and are intended for *local stations within radio range*. If you only want to talk to local NOS stations within radio range, and forwarding

by other stations is not required, this is the only entry you'll need in the IP routing table.

Forwarding packets via an IP Gateway

When the target system is not within local radio range, it will be necessary for an intermediate station, called an *IP Gateway* (or *router*) to forward your traffic in the right direction. Fig 28-2 shows this situation.

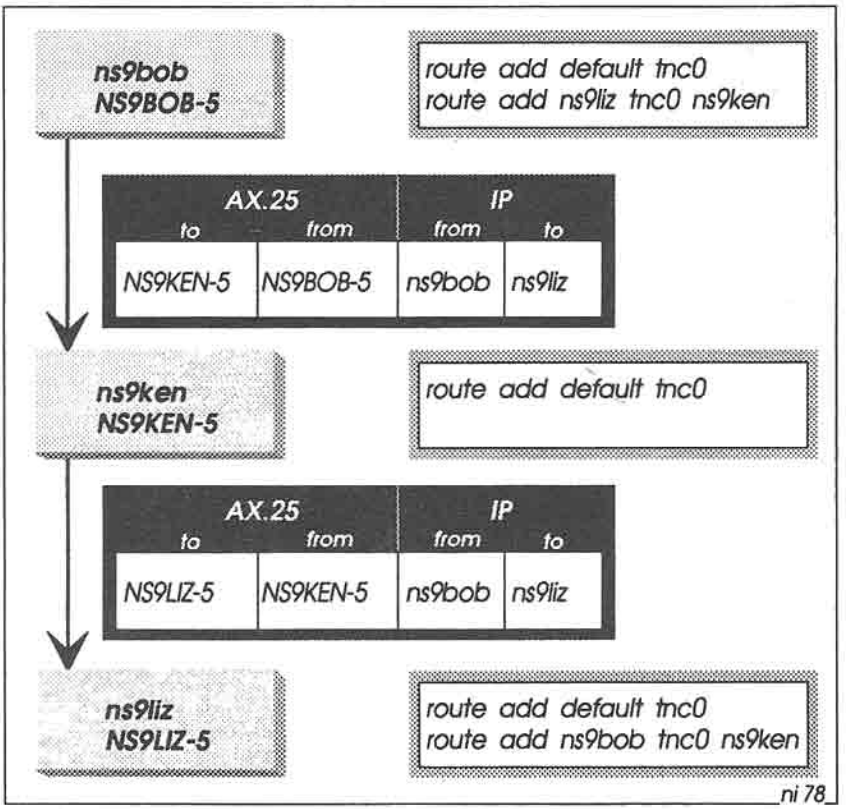


Fig 28-2: Forwarding via an IP gateway. The routing tables at *ns9bob* and *ns9liz* contain an entry to route traffic for each other via the gateway (*ns9ken*).

Bob wants to contact Liz, but as she is out of direct range, Bob uses Ken's station as a gateway. To launch packets on their way to Liz, Bob needs this line in *autoexec.nos*:

```
route add ns9liz tnc0 ns9ken
```

This tells NOS that the eventual target is ns9liz, but that the packets must initially be addressed to ns9ken. That is, the last parameter on the line (ns9ken) is the IP hostname of the gateway, which must be within radio range. Using ARP, NOS will discover that the AX.25 address of ns9ken is NS9KEN-5, and will use this address in the AX.25 "to" field.

When a packet for Liz arrive at Ken's station, it passes as usual up to the IP layer. Here, NOS discovers that the IP destination address is ns9liz. As Liz is within radio range of Ken, the existing default IP routing table entry is all that's required, so NOS just inserts Liz's callsign (NS9LIZ-5) into the AX.25 "to" field and sends the packet to her.

Liz also needs the following entry in her *autoexec.nos*:

```
route add ns9bob tnc0 ns9ken
```

so that if she wants to talk to Bob, her packets will go via ns9ken.

To summarise then, if a target station is out of direct radio range, you need to set up an IP routing table entry which incorporates the hostname of the next-door NOS station designated to forward your packets towards the target.

Multiple Hops

Taking this a stage further, it should now be clear how to forward packets through a chain of gateways. For example, in Fig 28-3, Bob is sending a packet to Jim via Ken and Liz. To do this, Bob needs another entry in his IP routing table:

```
route add ns9jim tnc0 ns9ken
```

This causes the packet to be addressed to NS9KEN-5. When the packet arrives at Ken's system, it has to be forwarded on towards Jim, but (unlike Liz), Jim is not within range.

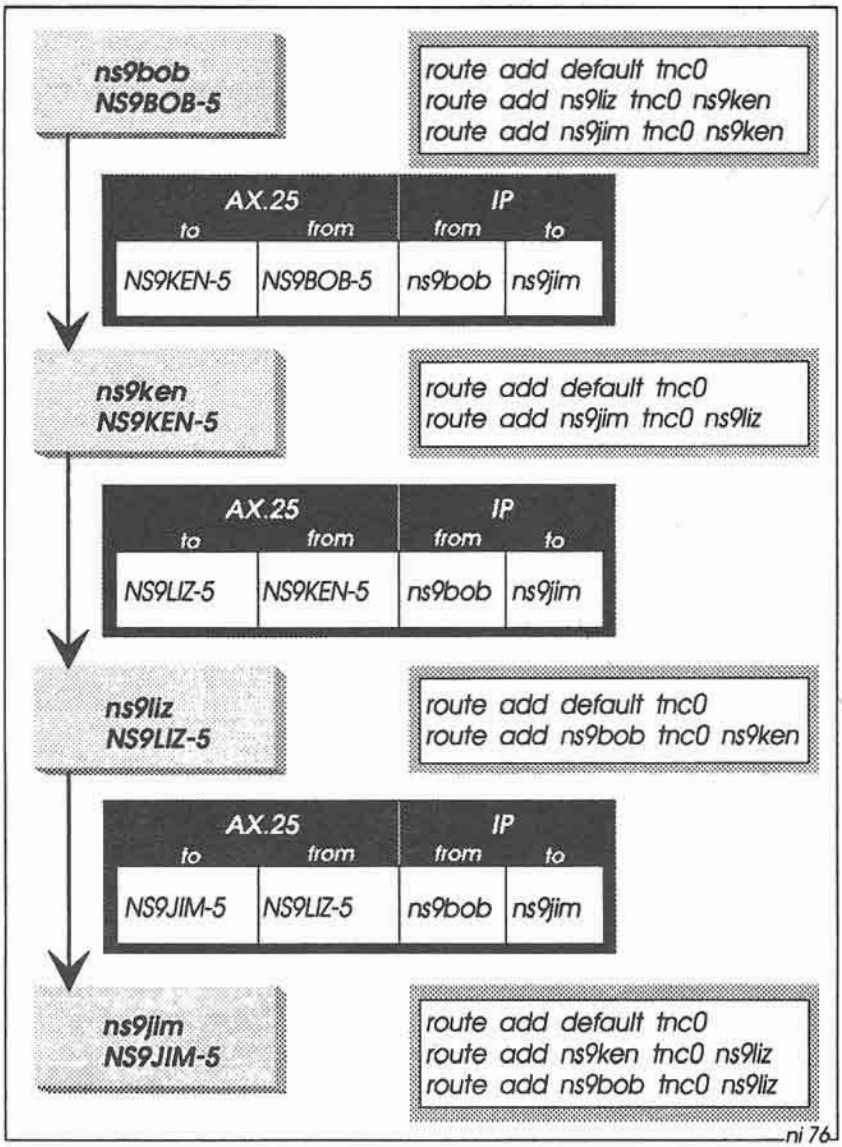


Fig 28-3: Multi-hop Forwarding. Each station needs IP routing table entries to forward to the next gateway.

So Ken needs this entry in *autoexec.nos*:

```
route add ns9jim tnc0 ns9liz
```

to forward the packet to Liz.

When the packet arrives at ns9liz, her default IP routing table entry is sufficient for the final leg of the journey to ns9jim.

In other words, as each station along the route forwards a packet, the AX.25 destination address in the packet is the address of the next station down the line, and this changes as it passes through each station. On the other hand, the IP destination address is the ultimate destination of the packet, and this remains the same all the way down the line.

Locally Generated Packets

Not only do all of the above forwarding rules apply when a station receives packets addressed to it at the AX.25 layer, they also apply when packets originate from within the station itself (for example, during an FTP transfer started at the keyboard). Thus when Ken gives the command **ftp ns9jim**, for example, NOS will forward his packets to Liz, on the assumption that Liz will then send them on to Jim.

Other IP Routing Table Commands

To see the current state of the IP routing table, use the **route** command:

```
net> route

Dest        Len    Interface  Gateway   Metric P   Timer  Use
ns9liz      32     tnc0       ns9ken     1         man    3
ns9jim      32     tnc0       ns9ken     1         man    4
default     0      tnc0                  1         man    5
```

The **route add** commands described earlier place entries in the IP routing table which NOS will include in routing table broadcasts if you have enabled broadcasting (e.g. with the RIP protocol). If you want to add some private entries that you don't want broadcast to the outside

world, you can use the **route addprivate** command instead. For example, to add an entry for a LAN station via interface en0:

```
route addprivate lanbox en0
```

In this case, the **route** command will display the entry with a capital P to show that it is private:

```
net> route

Dest        Len    Interface  Gateway   Metric P   Timer  Use
lanbox      32     en0                    1      P   man    3
```

To remove an entry from the IP routing table, use the **route drop** command. For example:

```
route drop ns9ken
```

Routing to a group of Stations

In practice you'll probably find that many of the remote stations you want to contact have similar IP addresses, and that most of them are contactable via just one or two gateways. See Fig 28-4.

NOS provides a convenient shorthand method of setting up the IP routing table for this situation. For example, the IP addresses for Liz and Jim are 44.199.45.17 and 44.199.45.18 respectively. The first 24 bits of their addresses are the same, and Bob forwards traffic for both Liz and Jim through ns9ken. So instead of having separate entries for Liz and Jim, Bob could set up an address mask for all addresses whose first 24 bits correspond to 44.199.45. The mask is given a meaningful name such as region45, and is defined in *domain.txt*:

```
region45.ampr.org.  IN A  44.199.45.0
```

Then Bob can use this mask in IP routing table entries in *autoexec.nos*. For example:

```
route add region45/24 tnc0 ns9ken
```

The /24 part of the target address indicates that only the first 24 bits of this address are significant. Liz's address and Jim's address both match this mask, so packets for them will be routed to ns9ken.

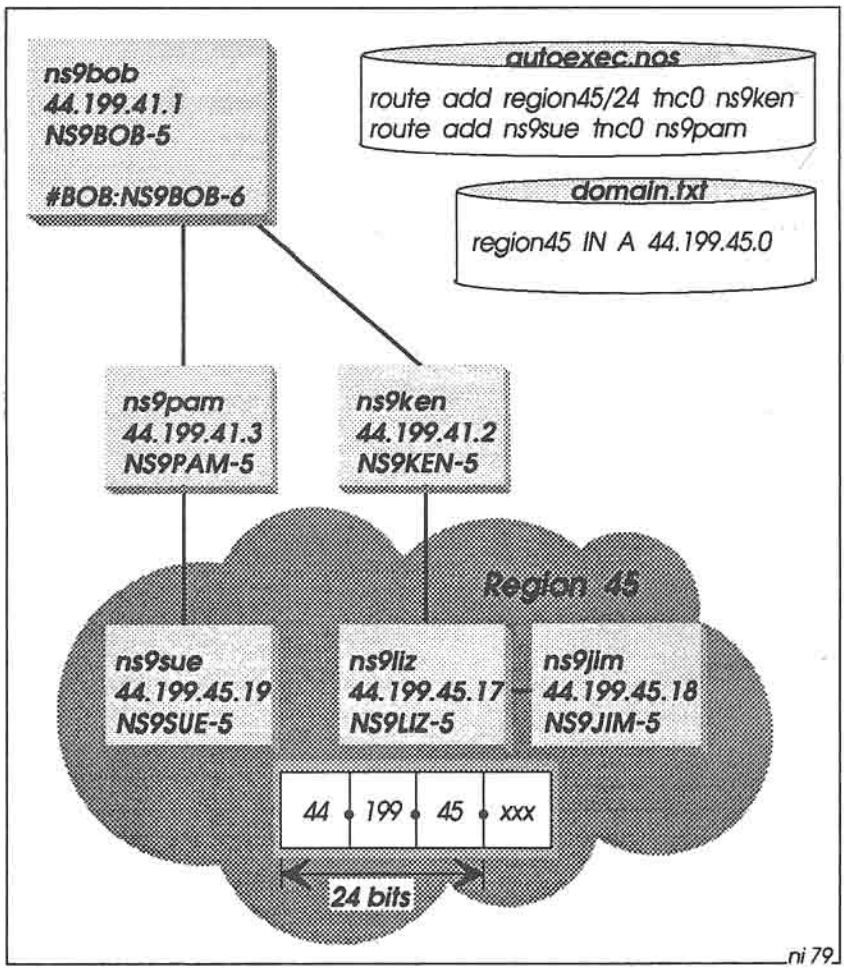


Fig 28-4: The *route add region45/24* command in *autoexec.nos* handles routing of packets to Liz and Jim via Ken. A separate entry is required for Sue, to force her traffic to be routed via Pam (otherwise it would also pass through Ken).

The **route** command shows this IP routing table entry as follows:

```
net>  route

Dest        Len     Interface  Gateway    Metric P   Timer  Use
region45    24      tnc0       ns9ken      1         man    3
```

Note that the Len (mask length) column now contains 24, instead of the usual 32.

Overriding General Routes

Another complication: what happens if Bob also wants to forward to ns9sue, whose IP address is 44.199.45.19, via ns9pam? (See Fig 28-4 again). If Bob only has the region45/24 entry as above, Sue's traffic will also go via ns9ken, not via ns9pam as intended.

The workaround here is to put a specific entry for Sue in *autoexec.nos*:

```
route add ns9sue tnc0 ns9pam
```

The routing table now looks like this:

```
net>  route

Dest        Len     Interface  Gateway    Metric P   Timer  Use
ns9sue      32      tnc0       ns9pam      1         man    1
region45    24      tnc0       ns9ken      1         man    3
default     0       tnc0                   1         man    5
```

Because Sue's entry refers specifically to her full 32-bit IP address, this takes precedence over the region45/24 entry.

In general, then, when NOS refers to the IP routing table to see how to route to a particular station, it attempts first to match the destination address against full 32-bit addresses in the table. If no match is found, NOS then takes the next lowest bit mask size in the table (24 bits in this example) and attempts the match again. This process continues with progressively shorter mask sizes until either a match is found, or until no match is found.

This latter case (i.e. no match at all) corresponds to the default entry in the routing table; i.e. this entry has a bit mask size of zero. Thus if there is no match, the packet will not be forwarded to another gateway, and will only be heard by local stations within radio range.

TheNet X1G

Until recently, the only practical way of forwarding IP packets has been to use NOS in the way just described. This relies on there being an IP route from end to end.

If there are any gaps in the route (i.e. no IP gateways to forward the traffic), it then becomes necessary to use NET/ROM links to bridge the gaps — in this case, IP packets are encapsulated inside NET/ROM packets whilst traversing the NET/ROM network, then decapsulated when they reach another IP gateway.

The way to use NET/ROM to do this is described in the next chapter.

A new development which has removed the need to descend to the NET/ROM level is the emergence of version X1G of **TheNet**, the NET/ROM "work-alike". See Fig 28-5 opposite.

This version of **TheNet** contains an IP router as well as a NET/ROM router, making it possible for NOS stations to use the NET/ROM network without encapsulating IP packets inside NET/ROM packets first. Version X1G of **TheNet** is now popping up all over the place, so it's likely that NET/ROM encapsulation of IP packets will virtually disappear in the fairly near future.

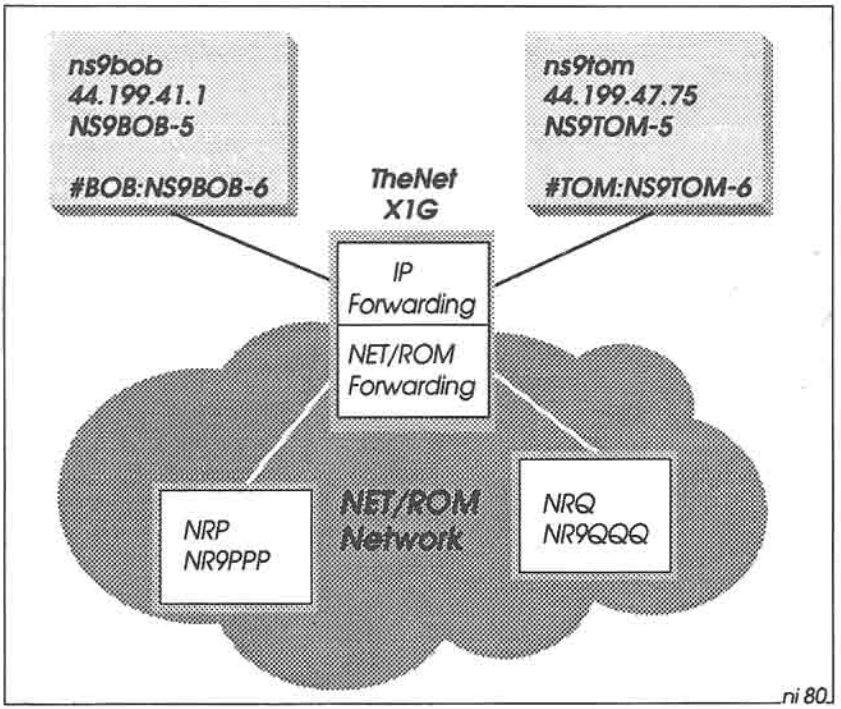


Fig 28-5: Version X1G of *TheNet* supports both IP and NET/ROM forwarding. This allows NOS stations to communicate with each other via the NET/ROM network, without having to encapsulate IP packets inside NET/ROM packets.

29: NET/ROM ROUTING

Everything we have covered thus far will be enough for setting up your AX.25 and IP routing tables, provided that you have a next-door neighbour who understands TCP/IP. If this is the case, and you are fortunate enough to live in an area where TCP/IP is well established, you should have no difficulty in IP routing, and you can skip this chapter on NET/ROM.

If, on the other hand, your nearest TCP/IP neighbour is out of direct radio range (or more than one or two digipeater hops away), you may need to consider using NET/ROM to transport your TCP/IP traffic. To do this, you need to configure NOS to act as a NET/ROM node.

You can configure NET/ROM to run as:

- a NET/ROM end-node, or
- a NET/ROM switch

When NET/ROM is configured as an *end-node* (see the top part of Fig 29-1), you are simply using it as a means of getting into the NET/ROM network, and thence into the AMPRnet, with the sole intent of transporting your own IP traffic. This is the probable way that you'll use NET/ROM under NOS.

When NET/ROM is configured as a *switch* (see the middle of Fig 29-1), you will be letting your station participate fully in the NET/ROM network, forwarding NET/ROM traffic for other people alongside your own NET/ROM and IP traffic. This is not really recommended, as third-party traffic through the switch may place an unacceptably high load on your NOS system.

Once NET/ROM is set up, you can then use it to transport your IP packets across the NET/ROM network to other TCP/IP stations; i.e. IP packets are encapsulated inside NET/ROM packets, which pass over the NET/ROM network as data — the NET/ROM network is unaware

that it is carrying IP traffic. When these NET/ROM packets arrive at their destination, NOS strips away the NET/ROM envelope, and passes the remaining IP data to the upper protocol layers for further processing.

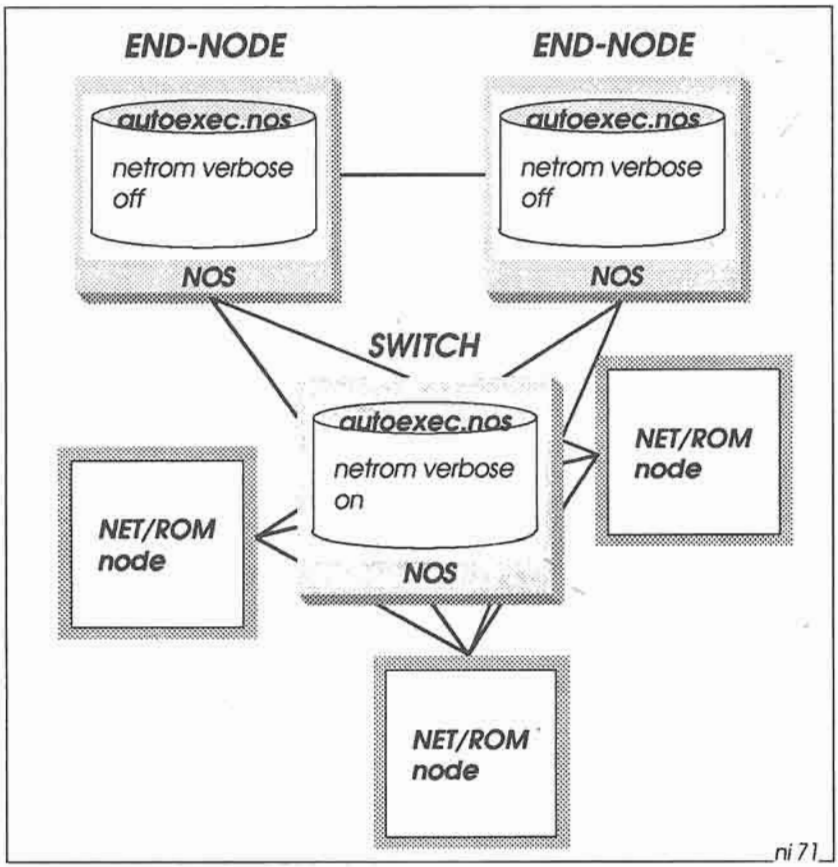


Fig 29-1: The *netrom verbose off* command configures NOS to run as a NET/ROM End Node. In this case, NOS only uses NET/ROM to encapsulate IP packets for transport over the NET/ROM network. The *netrom verbose on* command configures NOS to function as a network switch; i.e. it participates fully in the NET/ROM network. This is not recommended if traffic levels are likely to be high.

NET/ROM Initialisation

To tell NOS that you wish to use NET/ROM forwarding, you need several initialisation commands in *autoexec.nos*:

```
attach netrom
start netrom
mode netrom vc
```

Let's look at each of these commands.

attach netrom: This command tells NOS that it is to encapsulate higher level protocol packets inside NET/ROM packets when communicating with other NET/ROM stations.

start netrom: This starts the NET/ROM server (listener), so that other NET/ROM nodes can connect to this station.

mode netrom: This command sets the default mode for the packets which pass between this station and other NET/ROM nodes. This will usually be vc (virtual circuit); i.e. there is an error-corrected and flow-controlled link.

NET/ROM End-Node or Switch?

To set up your station as a NET/ROM end-node or a switch, you should include commands like the following in *autoexec.nos*:

```
netrom verbose off
netrom bcnodes tnc0
```

netrom verbose: This command specifies whether your NET/ROM node is to act as an *end-node* (**netrom verbose off**) or as a *switch* (**netrom verbose on**).

When operating as an end-node, your NET/ROM broadcasts will contain only your own NET/ROM callsign and alias; this is required by other stations in the NET/ROM network so that they know how to contact you.

When operating as a switch, NOS broadcasts the whole of your NET/ROM routing table to the NET/ROM network.

netrom bcnodes: This command broadcasts the current state of the NET/ROM routing table onto the network (just your own callsign and alias if verbose is off, or the whole of the table if verbose is on).

Thus when NOS starts up, your NET/ROM neighbours will know of your existence right away (To make sure of this, it's a good idea to include two or three **netrom bcnodes** commands in *autoexec.nos*, to ensure that your neighbours hear you). NOS will also re-broadcast the NET/ROM routing table at regular intervals thereafter; see the **netrom nodetimer** command below.

Timers and Counters

For NET/ROM, you need to set up a number of counters and timers. Most of the values assigned to these variables are system defaults which work for most situations, and it's unlikely you'll need to change them.

Thus in *autoexec.nos*:

```
netrom acktime 3000
netrom choketime 180000
netrom derate on
netrom irtt 15000
netrom minquality 10
netrom nodetimer 900
netrom obsotimer 1200
netrom qlimit 2048
netrom retries 10
netrom timertype linear
netrom ttl 10
netrom window 4
```

netrom acktime: This command sets the NET/ROM acknowledge timout period, in milliseconds. This is analogous to the AX.25 T2 timer.

netrom choketime: This specifies the time (in milliseconds) to wait before breaking a send choke (flow control) condition.

netrom derate: When derate is set to off, NET/ROM will only try alternative routes if a link completely fails. When derate is on, and retries on a particular route occur, the NET/ROM quality for that

route is progressively reduced. If the quality falls below that of another alternative route to the same destination, the alternative is used instead.

netrom irtt: This sets the Initial Round Trip Timer (in milliseconds).

netrom minquality: This sets the minimum acceptable quality of incoming NET/ROM routes. If the quality of a route is less than this value, it is not placed in the NET/ROM routing table (unless promiscuous is set to on; see below).

netrom nodetimer: This specifies the interval in seconds at which this station will broadcast the NET/ROM routing table to the NET/ROM network. If netrom verbose is set to off, only your own callsign and alias are broadcast. If verbose is on, the whole of the table is broadcast.

netrom obsotimer: This sets the obsolescence timer (in seconds) for NET/ROM routing table entries received in broadcasts from other NET/ROM stations. This indicates how long any received entries remain alive in the NET/ROM routing table.

A new or refreshed entry in the table has a lifetime of 6 times the obsotimer value. Thus, for example, if the obsotimer is set to 1200 seconds (20 minutes), entries will disappear after 2 hours (i.e. 6 × 20 minutes) if they are not refreshed.

Note that your own entries (added with the **netrom route add** command) remain in the NET/ROM routing table indefinitely, unless you forcibly remove them with the **netrom route drop** command.

netrom qlimit: This sets the maximum length (in bytes) of the NET/ROM receive queue. If this queue fills up, NET/ROM sends a choke request to suspend incoming data flow.

netrom retries: This sets the maximum number of connect and disconnect retry attempts.

netrom timertype: The timertype command lets you set the NET/ROM timer backoff mode. You can set it to linear or exponential; the linear setting is recommended for an amateur radio environment.

netrom ttl: This specifies the NET/ROM Time-to-Live; i.e. the maximum number of hops a packet can take before it is discarded. This prevents packets circulating for ever in an endless loop.

As a packet passes through a NET/ROM node, its TTL counter is decremented by one, and when the counter reaches zero the packet is discarded.

netrom window: This sets the maximum NET/ROM window size. This is the largest negotiable send and receive window.

Names and Addresses

To see how to identify your station to other NET/ROM users, let's look at the scenario in Fig 29-2. Bob and Tom are NOS stations without immediate TCP/IP neighbours, but they are each within range of conventional NET/ROM nodes (NR9AAA and NR9ZZZ respectively).

Bob needs the following commands in his *autoexec.nos* (Tom needs similar commands, of course):

```
netrom call NS9BOB-6
netrom alias #BOB
netrom interface tnc0 192
```

netrom call: This specifies the AX.25 callsign to be included in NET/ROM packets. The callsign will usually be the same as the ax25 mycall, but it doesn't have to be.

netrom alias: This is the NET/ROM alias for this station, and may be up to six characters long. There are no hard and fast rules for NOS station aliases — except, of course, they must not exist elsewhere on the NET/ROM network, otherwise some interesting routing problems may arise!

In some parts of the world the first letter of the alias is the # character (e.g. #BOB), whereas in other areas the alias starts with the letters IP, followed by the last part of the IP address in hexadecimal. For example, a station with IP address 44.199.32.10 could have the NET/ROM alias IP200A — i.e. 200A hexadecimal corresponds to 32.10 decimal.

In any case, it's a good idea to make sure that aliases for NOS stations have an obviously different format from regular NET/ROM

aliases, so that ordinary NET/ROM users can distinguish between them.

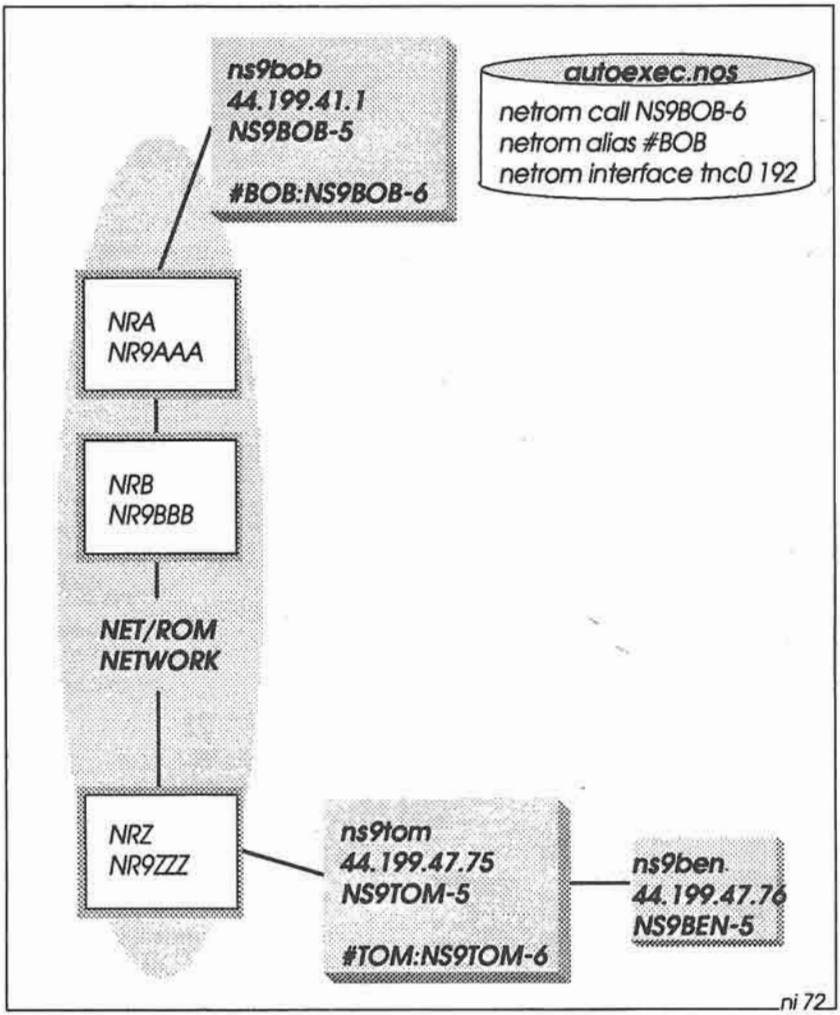


Fig 29-2: Bob and Tom communicate via the NET/ROM network, using the NET/ROM software built in to NOS.

netrom interface: This command specifies an interface through which NET/ROM packets pass. The value 192 is the quality coefficient (in the range 0-255) which this station contributes to the calculation of individual route qualities; 192 is a sensible value for 1200 bps links.

Note that in older versions of NOS, the **netrom call** and **netrom alias** commands do not exist. In that case, the netrom call is the same as the ax25 mycall, and the NET/ROM alias is included as a parameter in the **netrom interface** command. For example:

```
netrom interface tnc0 #BOB 192
```

Filtering Incoming NET/ROM Broadcasts

The NET/ROM routing table may contain permanent entries which you place there yourself (described below), plus temporary entries extracted from incoming NET/ROM broadcasts from other stations.

Normally you won't be interested in broadcasts having a low quality value — these entries will probably relate to long or unreliable routes. To control which incoming broadcasts you will or won't accept, you can include commands like the following in *autoexec.nos*:

```
netrom promiscuous off
netrom nodefilter mode accept
netrom nodefilter add NR9AAA tnc0
```

netrom promiscuous: When set to off, NET/ROM will ignore any incoming routing table broadcasts whose quality is less than the value set up in the **netrom minquality** command. When set to on, NET/ROM will accept all incoming NET/ROM routing table entries (irrespective of any nodefilters you may set up).

netrom nodefilter mode accept: This command allows you to build a list of NET/ROM nodes whose NET/ROM routing table broadcasts you are willing to accept. The list itself is built with individual **netrom nodefilter add** commands (see below).

On the other hand, you may wish to accept broadcasts from all stations *except* for one or two, in which case you can use the command **netrom nodefilter mode reject** instead. In that case, the

netrom nodefilter add list specifies which NET/ROM stations you *don't* want to hear from.

Yet another alternative is to say **netrom nodefilter mode none**, in which case there will be no filtering; i.e. you want to listen to NET/ROM broadcasts from all other stations.

netrom nodefilter add: This command specifies a NET/ROM station to be added to the **netrom nodefilter accept** or **reject** list. You need a separate **netrom nodefilter add** command for each station in the list.

Note that routing table broadcasts from stations in the list are *either* accepted *or* rejected, depending on the **netrom nodefilter mode**, described above; you can't accept some stations and reject others.

If you wish to remove a station from the filter list, you use the **netrom nodefilter drop** command. For example:

```
netrom nodefilter drop NR9AAA tnc0
```

Permanent Routing Table Entries

To communicate with other NOS stations over NET/ROM, you need to add entries to the IP routing table, the ARP (Address Resolution Protocol) table and the NET/ROM routing table. For example, Bob's entries will look like this:

```
route add region47/24 netrom ns9tom

arp add ns9tom netrom NS9TOM-6

netrom route add NRA NR9AAA tnc0 192 NR9AAA
netrom route add #TOM NS9TOM-6 tnc0 192 NR9AAA
```

route add ... netrom: An IP **route add** command with the netrom parameter is needed for every IP connection that is to be made over NET/ROM. As Tom happens to be in Region 47 (his IP address is 44.199.47.75), it's convenient to route all Region 47 traffic to him over NET/ROM.

This requires a suitable entry for Region 47 in *domain.txt*:

```
region47.ampr.org.  IN A  44.199.47.0
```

(If Tom were the only station in Region 47, then Bob could use the specific command **route add ns9tom netrom** instead).

arp add ... netrom: The ARP table is usually used to associate IP addresses with link addresses (such as AX.25 callsigns), as described in Chapter 27. However, when using NET/ROM, the **arp add ... netrom** command associates IP addresses with entries in the NET/ROM routing table instead.

netrom route add: This is the command to put an entry into the NET/ROM routing table. At least two entries are required here. The first entry is for the next-door neighbouring NET/ROM node (NR9AAA), and the second is for the remote node (NS9TOM-6).

The first two parameters in the **netrom route add** command (NRA and NR9AAA in the first example) are the NET/ROM alias and AX.25 callsign of the NET/ROM node, and the last parameter (NR9AAA) is the AX.25 callsign of the next-door NET/ROM neighbour to which NOS will send the NET/ROM traffic for that node.

Note that in the second example, the first two parameters apply to the NET/ROM node *inside Tom's NOS system*, not to the node NR9ZZZ at the end of the ordinary NET/ROM chain.

What's in a NET/ROM Packet

To see how NOS builds a NET/ROM packet, let's follow what happens when Bob gives the command **telnet ns9ben** (Fig 29-3). To reach Ben, the **telnet** connection request has to go via the NET/ROM network to Tom, then Tom will forward it at the IP level to Ben.

Starting at the top of Fig 29-3, NOS gets Ben's IP address from *domain.txt* (44.199.47.76) and then looks in the IP routing table for an entry for ns9ben. There isn't one, but the first 24 bits of Ben's IP address do match the region47/24 entry, so NOS determines that the request has to be routed to ns9tom via NET/ROM.

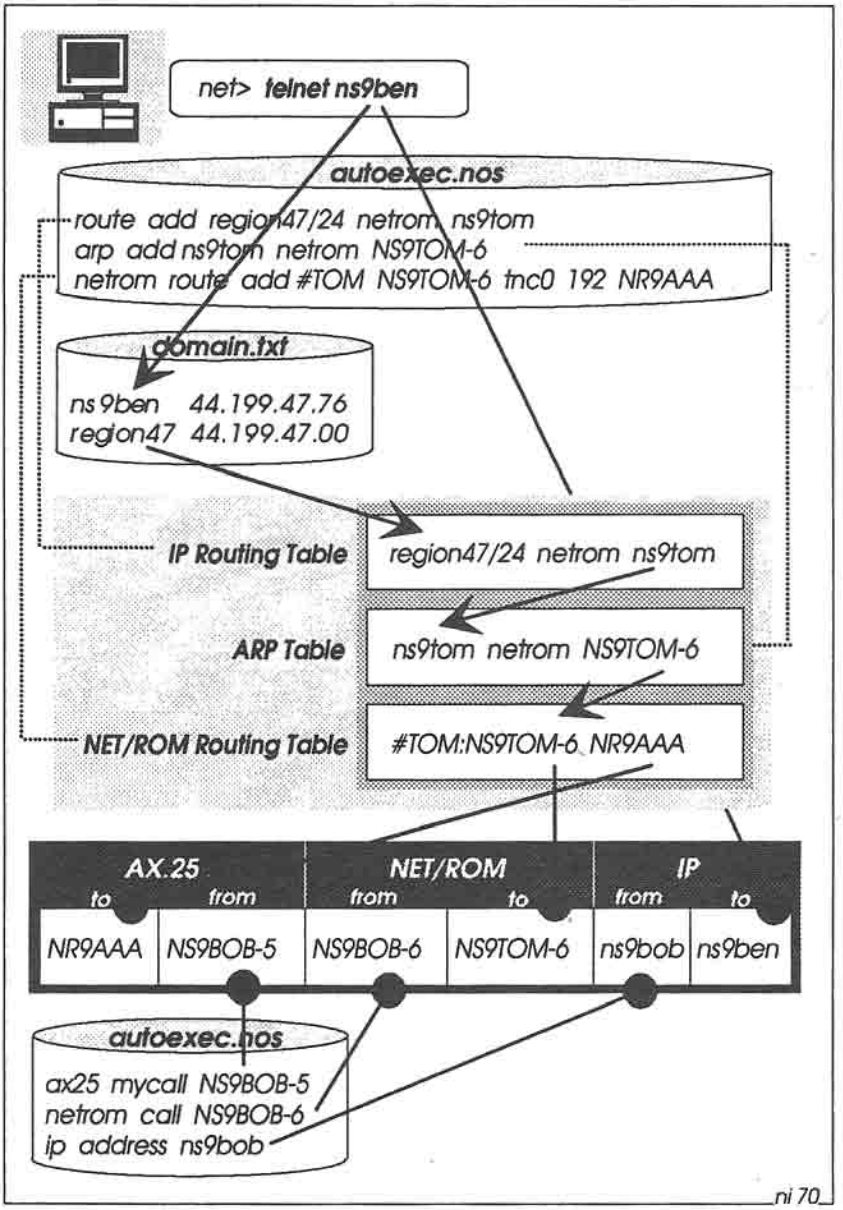


Fig 29-3: The AX.25, NET/ROM and IP addresses in a NET/ROM packet.

NOS then looks in the ARP table for a NET/ROM entry for ns9tom, and finds that Tom's AX.25 callsign is NS9TOM-6. Finally, NOS looks in the NET/ROM routing table for an entry for NS9TOM-6, and finds that traffic for Tom is to be sent to the local NET/ROM node NR9AAA (which knows how to forward it on towards Tom).

Thus the **telnet** connection request packet which leaves Bob's system contains a lot of address information for use at the three network levels: AX.25, NET/ROM and IP (again see Fig 29-3):

- The AX.25 destination and source fields apply to the local AX.25 connection between NS9BOB-5 and NR9AAA.
- The NET/ROM source and destination fields apply to the NET/ROM nodes inside Bob's and Tom's NOS systems.
- The IP source and destination fields apply to Bob's and Ben's IP addresses.

When this packet passes through an ordinary NET/ROM node, the node simply forwards it on to the next NET/ROM node, ignoring the IP information encapsulated within the packet. Eventually the packet reaches the NET/ROM destination (NS9TOM-6), where the IP information is extracted and passed up to the IP layer in Tom's system.

Because the packet is addressed at the IP level to Ben, Tom now forwards it to Ben (Tom's **route add default tnc0** command handles this). Finally, when the packet arrives at ns9ben, it will pass up to the TELNET server and the connection will (at last!) be established.

Ordinary NET/ROM connections

With NET/ROM routing in place, you can of course use NOS to make connections to ordinary NET/ROM nodes, using the **netrom connect** command. For example, Bob can define a NET/ROM route to NR9ZZZ in *autoexec.nos:*

```
netrom route add NRZ NR9ZZZ tnc0 192 NR9AAA
```

then give the keyboard command to connect to that node:

```
net>  netrom connect NRZ
```

30: GOING LIVE: PREPARING THE FILES

When you feel that you're familiar enough with NOS to use it in a live on-air environment, you need to modify many of the control files already described, before turning on the radio. Otherwise you'll pop up on the channel as NS9BOB, and then everyone will know which book *you've* been reading ..!

This chapter summarises the changes that you need to make. In several instances you may delete a file altogether if you don't need it — you still have a read-only master copy in */public/masters* to fall back on if you change your mind.

The Files to Change

/alias: *(see page 311)* Set up a list of aliases appropriate for your environment (or delete the file altogether if you don't need it).

/domain.txt: *(pages 318–319)* Replace *domain.txt* with a file containing real hostnames and IP addresses. To minimise address seek times, put the most frequently contacted stations at the front of the file, and remove all unnecessary comments. Make sure that the loopback address is included.

/ftpusers: *(pages 322–323)* Add real user entries to this file. For all users having access to sensitive directories, assign a login name which is longer than 6 characters, to prevent access by ordinary AX.25 stations. Check this file very carefully, and test it off-line with the **bbs** and **ftp loopback** commands, to make sure that nobody can do dangerous things to your filesystem.

/net.rc: *(page 324)* Remove this file if you don't need it. Otherwise replace the existing entries with real parameters.

/popusers: *(page 326)* Remove this file if you don't need it. Otherwise replace the existing entries with real names/passwords.

/signatur: *(page 327)* Change the text in this file to suit your own environment. Note that this file is used by PCElm, which appends it to the end of every message which you send. The file is ignored by the built-in NOS BBS.

/autoexec.nos: *(pages 312–317)* Change the following lines, or comment them out with the # character:

```
motd
domain addserver
ip address
hostname
ax25 mycall
ax25 bctext
ifconfig tnc0 description
netrom call
netrom alias
netrom route add
netrom nodefilter add
arp add netrom
route add
ax25 route add
third-party
smtp gateway
mbox motd
mbox qth
mbox zipcode
mbox fwdinfo
mbox haddress
mbox password
popmail addserver
popmail kick
```

Remember that you can create many different *autoexec.nos* files, with different names. Then you can specify which particular file to use in *STARTNOS.BAT* (or even create several versions of *STARTNOS.BAT* files for different scenarios).

/finger/sysop: *(page 329)* Delete this file, and replace with one or more files for users on your system. Filenames in this directory do not have an extension.

/scripts/kisson.dia: *(page 324)* If you permanently run your tnc in KISS mode, this script is superfluous. Otherwise change:

```
MYCALL
MID
```

/spool/areas: *(page 311)* Change this file as required. N.B. The first character on all comment lines must be a space.

/spool/forward.bbs: *(page 321)* Remove this file altogether if you are not planning to forward mail onto the AX.25 PBBS network. Otherwise replace the existing entries with real data.

/spool/rewrite: *(page 327)* Remove this file altogether if you don't need it. Otherwise replace the existing entries with real data. Test the file thoroughly with the radio switched off, to make sure that addresses are replaced correctly.

/spool/signatur/ns9bob.sig: Delete this file. Replace with similar *.sig* files for every NOS BBS login username (if you really need signatures, that is). N.B. The built-in NOS BBS uses these files; PCElm does not.

N:\NOSENV.BAT: *(page 325)* Change the timezone (TZ) variable to the local timezone abbreviation; e.g. **SET TZ=PST**

N:\PCELM.RC: Change the parameters as required if you plan to use PCElm.

The First Tests

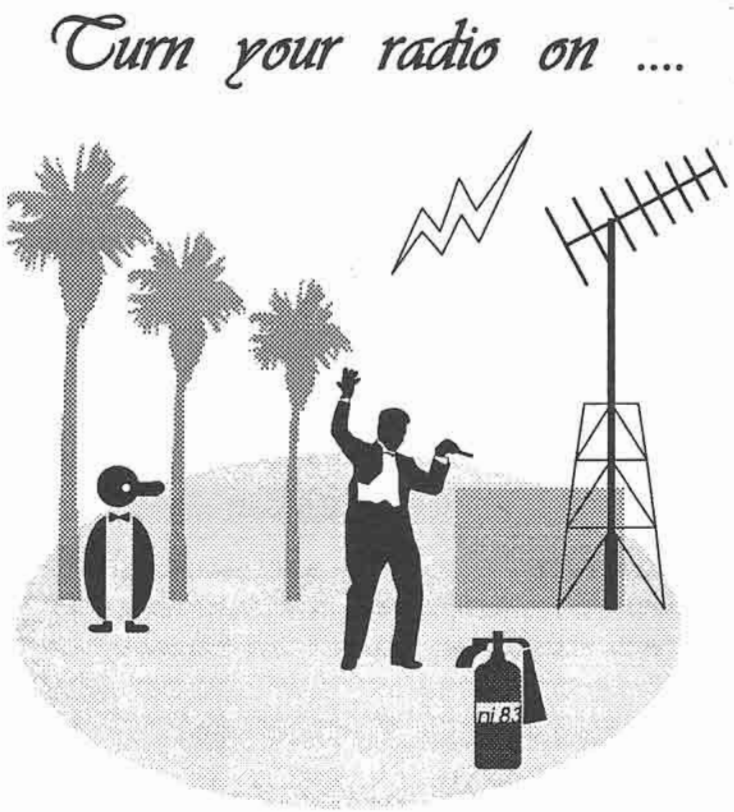
Having made the necessary changes to all these files, start NOS with the radio switched off, and make sure that no errors are reported during startup. If strange things happen when NOS reads *autoexec.nos*, it may be useful to start up with the -v (verbose) option, to get a detailed trace of what happens at startup time. For example:

```
N:\>  NOS_20M  -V  /autoexec.nos
```

Then try sending mail to yourself, and transferring files to and from yourself with **ftp**. In fact, try everything described so far in this book, to make absolutely sure that NOS is behaving.

If everything is looking good, you're ready to go live!

In fact, you can now, ... at last .....


Turn your radio on ....
pi 8.3

31: HANDS ON — AX.25

This chapter describes some recommended on-air tests which check that the tnc is working, and that basic AX.25 communication is possible with NOS. You need to make AX.25 work properly before you attempt to use TCP/IP — if AX.25 doesn't work, TCP/IP won't either.

We'll continue to use NS9BOB's parameters here, but you'll now be using real callsigns etc.

Start NOS as usual, and then try some commands to check that AX.25 is working.

The *trace* Command

One of the most useful commands in NOS is the **trace** command. This lets you monitor every incoming and/or outgoing packet at various levels of detail. You can display the trace output on the screen for immediate viewing, or direct it to a file for later analysis.

The syntax of the **trace** command is:

```
trace [ interface [ off | BTIO_flags [ tracefile ] ] ]
```

For example:

```
net>  trace tnc0 0211 /dump/trace/trace123
```

The digits 0211 here are the BTIO (Broadcast/Trace/Input/Output) flags. The full list of flags is as follows:

B=0:	Display broadcast packets
B=1:	Only display packets addressed to this node
T=0:	Decode protocol headers, but no data displayed
T=1:	Decode protocol headers, and display data
T=2:	Display decoded headers and entire packet data
I=0:	Ignore input packets
I=1:	Display input packets
O=0:	Ignore output packets
O=1:	Display output packets

Thus, in the example, 0211 means:

B=0:	Display broadcast packets
T=2:	Display decoded headers and entire packet data
I=1:	Display input packets
O=1:	Display output packets

You can omit leading zeros — 211 means the same as 0211.

For convenience in the **NOSview** distribution, five function key combinations are pre-programmed for tracing packets:

F9	**trace tnc0 0211**	to the screen (all packets)
SHIFT-F9	**trace tnc0 0211**	to a file (all packets)
CTRL-F9	**trace tnc0 0011**	to the screen (headers only)
ALT-F9	**trace tnc0 0011**	to a file (headers only)
ALT-F10	**trace tnc0 off**	

Try **CTRL-F9** (or **trace tnc0 11**). You should see something like Fig 31-1 on the screen (for clarity, transmitted packets are shown outdented in italic text).

Here you can see the header of every packet, showing how NOS decodes the packet at each protocol level (AX.25, IP, ARP, ICMP etc).

If you want to save the trace in a file, hit **ALT-F9**, and add a filename to the end of the command (alternatively, give the command **trace tnc0 11 /dump/trace/***filename*).

To stop tracing at any time (either to the screen or to a file), hit **ALT-F10**, or give the command **trace tnc0 off**.

```
Sun Sep 06 07:18:04 1992 - tnc0 recv:
KISS: Port 0 Data
AX25: GB7KHW-5->QST UI pid=ARP
ARP: len 30 hwtype AX.25 prot IP op REQUEST
sender IPaddr 44.131.5.68 hwaddr GB7KHW-5
target IPaddr 44.131.19.30 hwaddr

Sun Sep 06 07:18:13 1992 - tnc0 recv:
KISS: Port 0 Data
AX25: G7CDK-5->G4HPE-5 UI pid=IP
IP: len 44 44.131.5.86->44.131.19.13 ihl 20 ttl 24 prot TCP
TCP: 1027->3600 Seq xf2d72000 SYN Wnd 1024 MSS 512

Sun Sep 06 07:18:19 1992 - tnc0 sent:
KISS: Port 0 Data
AX25: NS9BOB-5->ID UI pid=Text

Sun Sep 06 07:18:25 1992 - tnc0 recv:
KISS: Port 0 Data
AX25: G4HPE-5->G7CDK-5 UI pid=IP
IP: len 56 44.131.19.96->44.131.5.86 ihl 20 ttl 9 prot ICMP
ICMP: type Unreachable code Host
ReturnedIP:len 44 44.131.5.86->44.131.19.13 ihl20 ttl 23 prot TCP
TCP: 1027->3600 Seq xf2d72000 Wnd 60192
```

Fig 31-1: A header trace (*trace tnc0 11*).

A Full Trace

The **trace tnc0 11** command just described usually gives enough detail about what's going on, but sometimes you may want to see every byte in every packet, including the data.

To do this, hit **F9**, or give the **trace tnc0 211** command. You will now see something like Fig 31-2.

To save the trace in a file instead, hit **SHIFT-F9**, and add a filename to the end of the command (alternatively, give the command **trace tnc0 211 /dump/trace/***filename*).

Needless to say, a full trace can produce an enormous amount of output in a very short time if the channel is busy. Just remember to turn the trace off with **ALT-F10**, or give the **trace tnc0 off** command, when you've finished. Otherwise you may find that everything stops because the disk has completely filled!

```
Sun Sep 06 08:35:21 1992 - tnc0 recv:
KISS: Port 0 Data
AX25: G4HPE-5->G8KVP-5 UI pid=IP
IP: len 40 44.131.19.96->44.131.19.196 ihl 20 ttl 9 prot TCP
TCP: 3600->1028 Seq x0 Ack x2c109001 ACK RST Wnd 0
0000  008e 7096 aca0 406a 8e68 90a0 8a40 6b03  ..p., @j.h. .@k.
0010  cc45 0000 2802 0b00 0009 062f 9c2c 8313  LE..(....../.,..
0020  602c 8313 c40e 1004 0400 0000 002c 1090  `,..D.......,..
0030  0150 1400 0061 8100 00                   .P...a...

Sun Sep 06 08:36:23 1992 - tnc0 recv:
KISS: Port 0 Data
AX25: G3NRW-5->ID UI pid=Text
0000  0092 8840 4040 40e0 8e66 9ca4 ae40 6b03  ...@@@@`.f.$.@k.
0010  f047 334e 5257 2d35 2054 4350 2f49 5020  pG3NRW-5 TCP/IP
0020  3434 2e31 3331 2e35 2e32 205b 4861 726c  44.131.5.2 [Harl
0030  696e 6774 6f6e 2c20 4265 6473 5d          ington, Beds]

Sun Sep 06 08:36:23 1992 - tnc0 recv:
KISS: Port 0 Data
AX25: GB7KHW-5->QST UI pid=ARP
ARP: len 30 hwtype AX.25 prot IP op REQUEST
sender IPaddr 44.131.5.68 hwaddr GB7KHW-5
target IPaddr 44.131.19.30 hwaddr
0000  00a2 a6a8 4040 40e0 8e84 6e96 90ae 6b03  ."&(@@@`..n...k.
0010  cd00 0300 cc07 0400 018e 846e 9690 ae6a  M...L......n...j
0020  2c83 0544 0000 0000 0000 002c 8313 1e     ,..D.......,....
```

Fig 31-2: A full trace (*trace tnc0 211*)

In Fig 31-2, the data in each packet is shown in bold text, in hexadecimal, in blocks of 16 bytes. The numbers in the left-most column (0000, 0010, 0020 etc) are block numbers.

To decode the data, you'll need the relevant protocol documentation describing the details of the data formats. Appendix 6 contains a list of suitable references.

As an example, let's decode part of the last packet shown above (the packet sent by GB7KHW-5 to QST). See Fig 31-3.

The first line of this packet reads:

```
00a2 a6a8 4040 40e0 8e84 6e96 90ae 6b03
```

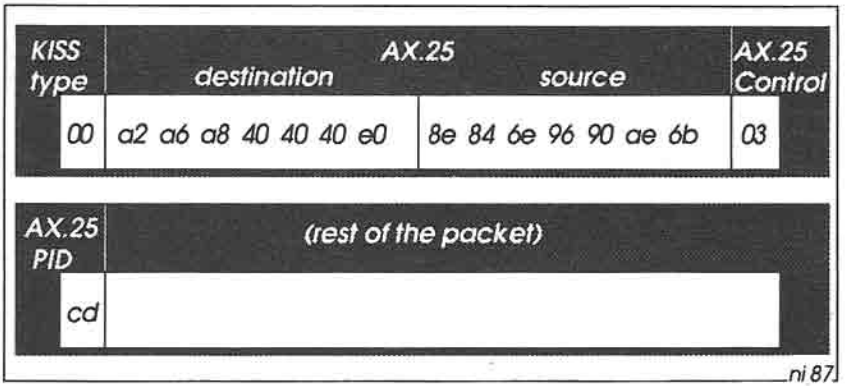


Fig 31-3: The first 17 bytes of a packet.

The first byte (00) is the KISS *type* byte. Type 00 means that this is a data block.

The next 7 bytes (a2 a6 a8 40 40 40 e0) are the AX.25 destination address, left-shifted by one bit. Appendix 4 contains a list of these left-shifted codes. Thus a2 means Q, a6 means S, a8 means T, and 40 means SPACE. Hence the packet is addressed to QST (the AX.25 broadcast address). The last of these 7 bytes (e0) contains the SSID.

The next 7 bytes (8e 84 6e 96 90 ae 6b) contain the AX.25 source callsign and SSID, in this case GB7KHW-5.

The last byte on the first line (03) is the AX.25 control field; this is a UI (Unnumbered Information) frame.

The second line reads:

```
cd00 0300 cc07 0400 018e 846e 9690 ae6a
```

The first byte (cd) is the AX.25 PID (Protocol ID) field. This byte decodes as follows:

```
cc    Internet Protocol (IP)
cd    Address Resolution Protocol (ARP)
cf    NET/ROM
f0    AX.25
```

So this is an ARP packet.

The next 2 bytes (00 03) are the ARP Hardware Code. The codes you're likely to come across are:

```
00 01 Ethernet
00 03 AX.25
00 06 IEEE
```

Thus this ARP packet contains AX.25 hardware addresses (callsigns). These follow later in the packet, and are again in bit-shifted form.

The rest of the packet contains the usual ARP information, which is summarised in the header part of the trace:

```
ARP: len 30 hwtype AX.25 prot IP op REQUEST
sender IPaddr 44.131.5.68 hwaddr GB7KHW-5
target IPaddr 44.131.19.30 hwaddr
```

Trying AX.25

Now that you can see things happening on the channel, you're ready to try a simple AX.25 connection. First, you may care to see who has been on the channel recently, using the **ax25 heard** command **(F4)**:

```
net> ax25 heard
Interface  Station    Time since send  Pkts sent
tnc0        NS9BOB-5      0:00:00:07         7
Station    Time since  Pkts  : Station   Time since  Pkts
             heard      rcvd                heard      rcvd
NS9BOB-5   0:00:00:07    2   : NR9XYZ-5  0:00:00:13    2
BB7BBS     0:00:00:18   17   : NS9KEN-5  0:00:00:21    2
```

Choose an ordinary AX.25 station from the list and connect to it:

```
net> connect tnc0 BB7BBS
```

Remember to include the interface name (tnc0) in the command.

The screen will clear, and a new session will start, with NOS trying to make the connection. Assuming this is successful, you will see:

```
Trying BB7BBS on tnc0...
AX25 session 1 connected to BB7BBS
```

You should now be able to talk to BB7BBS in the usual way. (N.B. if you connect to a PBBS as in this example, you may need to hit **CR** once or twice to bring the connection to life).

Now hit **F2**, or return to the Session Manager and give the **session** command. You will see:

```
  #  S#  Type    Rcv-Q Snd-Q  State       Remote Socket
*1   143 AX25        0     0  Connected   bb7bbs (BB7BBS on tnc0)
```

To disconnect from the AX.25 station, give the **disconnect** command:

```
net>  disconnect 1
```

Then hit **CR** again, to return to the AX.25 session. You should now see the message:

```
AX25 session 1 closed: Normal
Hit enter to continue
```

Hit **CR** again. This will finish the session, and take you back to the Session Manager.

If **disconnect** does not seem to work, you can try the **reset** command instead:

```
net>  reset 1
```

This is a brute-force reset, and should always work.

An AX.25 Connection to a NOS BBS

Now try another AX.25 connection, this time to a station known to be running NOS:

```
net>  connect tnc0 NS9KEN-5
```

Note that you use the *AX.25 callsign* of the NOS station in the **connect** command, not the IP hostname — vanilla AX.25 knows nothing about IP.

With luck you'll now be connected to NS9KEN's NOS BBS. Because you used AX.25, you won't be asked for a login name or password, and you should be able to use the BBS immediately. Give the **?** command at the BBS prompt to find out exactly what commands are available, and you're away! When you've finished, give the **B** command to say goodbye, then hit **CR** to terminate the session.

AX.25 Beacon Broadcasts

There are several command which control the broadcasting of AX.25 beacons (in *autoexec.nos*):

```
ax25 bc           tnc0 on
ax25 bcinterval   840
ax25 bctext       "NS9BOB-5 TCP/IP 44.191.41.1 [London]"
```

These commands enable the broadcasting of beacons on interface tnc0 every 840 seconds (14 minutes).

To force a beacon broadcast:

```
net>  ax25 bckick tnc0
```

32: HANDS ON — NET/ROM

If everything has worked so far, and if you don't need to use NET/ROM to talk to NOS stations, you can skip this chapter.

On the other hand, if you are forced to use NET/ROM to transport your IP traffic, you must first check that you can make *ordinary* NET/ROM connections, and that your NET/ROM alias is propagated properly to neighbouring nodes.

Making a NET/ROM Connection

You should first check your NET/ROM routing table, to make sure it has correct entries:

```
net>  netrom route
NRA:NR9AAA    #TOM:NS9TOM-6
```

NRA is a neighbouring conventional NET/ROM node, and #TOM is the NET/ROM node built in to Tom's NOS system.

Then use the **netrom connect** command to connect to the conventional NET/ROM node:

```
net>  netrom connect NRA
```

The screen will go blank, and NOS will attempt to make the connection. Assuming it is successful, the screen will look like this:

```
Trying NR9AAA @ NR9AAA...
NET/ROM session 1 connected to nra
```

You can check the connection with the **session** command (or **F2**):

```
net> session
 #  S#  Type     Rcv-Q Snd-Q  State       Remote socket
*1  143 NET/ROM      0     0  Connected   nra (NR9AAA @ NR9AAA)
```

Return to the NET/ROM session (with **CR**), and give the " **N *** " command to get the NET/ROM routing table from **NRA**. You should see something like this:

```
N *
NRA:NR9AAA> Nodes:

#BOB:NS9BOB-6    #NHM:G6LOH       #OXON:G6IKQ     #TOM:NS9TOM-6
AVN:G0DFP        AYLS4:G3OZF-4    BDMBX:GB7ZPU    BEDBOX:GB7ZPU-1
```

Look very carefully through the list, and verify that it contains both your own NET/ROM callsign (**NS9BOB-6**) *and* your alias (**#BOB**).

It's quite possible that it only contains your callsign, without the alias:

```
NRA:NR9AAA> Nodes:

  NS9BOB-6       #NHM:G6LOH       #OXON:G6IKQ     #TOM:NS9TOM-6
↑
{missing alias}
```

This means that the node has heard one or more NET/ROM transmissions from you, but hasn't heard your NET/ROM routing table broadcast. To correct this situation, escape to the Session Manager and give the command:

```
net> netrom bcnodes tnc0
```

This will re-broadcast your alias. Now return to the NET/ROM session, and give the " **N *** " command again, to get an up-to-date copy of NRA's routing table. This time you should find that both your alias and callsign are in place. If not, you'll have to repeat this procedure until they are (or maybe persuade the node operator to include your entry permanently in his routing table).

Finally, when you are satisfied that ordinary NET/ROM is working properly, disconnect from the node with the usual node **B** command. The session will terminate with the message:

```
NET/ROM session 1 closed: By Peer
Hit enter to continue
```

If this doesn't work, you can force the session to terminate with the NOS **reset** command.

Saving the NET/ROM Routing Table

When you first start up NOS, the entries in the NET/ROM routing table will be the ones you included in *autoexec.nos*. As time passes, you'll find that new entries appear, gleaned from broadcasts by other nodes.

At any time you can give the **netrom route** command, or **SHIFT-F7**, to find out the current state of the table.

If you wish you can save these new entries, by giving the command:

```
net> netrom save
```

This puts them in the file */netrom.sav*. Then, on some future occasion, you can fetch them back with the command:

```
net> netrom load
```

If you put the **netrom load** command in *autoexec.nos*, NOS will load these entries from */netrom.sav* into the NET/ROM routing table every time you start NOS. This means that you can start to use the entries immediately, without having to wait for perhaps half an hour for broadcasts from other nodes.

Making a NET/ROM Connection to a NOS BBS

You are now ready to try making a NET/ROM connection to a remote NOS BBS (e.g. #TOM). First you must check that your routing table is correct:

```
net> netrom route info #TOM
      Node            Neighbour        Port  PQual Obsocnt
#TOM:NS9TOM-6       NRA:NR9AAA         tnc0   192       6
```

In addition, check that your neighbour NRA also has an entry for #TOM; there's no point in forwarding a connect request through NRA if NRA doesn't know how to forward it onwards to #TOM.

Finally, check with Tom that he has actually set up his station to accept NET/ROM requests. In areas where IP is well established, many users don't start the NET/ROM server, so even if your connect request makes it through several NET/ROM nodes, it may fall at the last fence because the target doesn't want to talk NET/ROM!

So, assuming you're reasonably confident that the NET/ROM network knows how to find #TOM, and that Tom is willing to talk NET/ROM, you can now attempt a connection:

```
net> netrom connect #TOM
```

With luck you will eventually make a connection, and find yourself logged straight into Tom's NOS BBS. Again, as with an AX.25 connection, you don't need to provide a login name or password, and should be able to use the BBS immediately.

If you need to find out the current status of the NET/ROM system, you can give the **netrom status** command:

```
net> netrom status
   &CB   Snd-W Snd-Q Rcv-Q  LUser   RUser      @Node     State
8d8c0008     0     0     0  NS9BOB-6                     Listening
94b60008     0     0     0  NS9BOB-6 NS9TOM-6 NS9TOM-6   Connected
```

Finally, if all of this works as expected, you are now ready to try TCP/IP.

33: HANDS ON — PING AND HOP

One of the very first TCP/IP commands which you're likely to use on-air is **ping**, which sends test packets to another station. If that station is within range and operational, it should reply to those packets. Ping then displays the round-trip time (rtt) between sending each test packet and getting a reply.

If the rtt is low (of the order of a few seconds for a 1200 bps radio link), you know that all is well. Hence, if you're not certain that you can talk to another TCP/IP station, **ping** is the command to use.

The **ping** command tells you how long it took to send out a test packet and get a reply, but it doesn't tell you which route it took in doing so. Sometimes it's useful to know this, particularly if the rtt is unexpectedly long — someone could have changed their routing tables, so that your packets are now travelling over completely different terrain. The command to find out which way your packets are going is **hop**.

This chapter explains in detail how to use the **ping** and **hop** commands.

The *ping* Command

The syntax of the **ping** command depends on the version of NOS which you are using. It will be one of the following:

```
ping   host  [ length [ milliseconds [ incflag ] ] ]      {or}

ping   host  [ length [ seconds [ incflag ] ] ]
```

The only difference between them is the way you specify the time interval between sending out test packets. This should be about 5000 milliseconds, or 5 seconds. If you get the units wrong, you will either

send out packets every 5000 seconds or every 5 milliseconds — neither of these are particularly helpful!

To find out which version you have, try pinging yourself, with the **ping loopback** command:

```
net>  ping loopback 0 1000        (or)
net>  ping loopback 0 1
```

(The digit 0 is the number of additional bytes to include in each test packet — there is usually no need to increase this).

A new session will start, and you will see an output something like this:

```
Resolving loopback... Pinging loopback (127.0.0.1); data 0
interval 1000 ms:

 sent      rcvd     %       rtt    avg rtt      mdev
    1         1   100        32         32         0
    2         2   100        15         30         4
    3         3   100        14         28         7
    4         4   100        14         26         9
```

To stop the **ping**, escape back to the Session Manager and give the **reset** command.

Note that the rtt values are short (just tens of milliseconds). This is because the loopback path is entirely within your system.

If you **ping** a real station (Fig 33-1), you'll find the times are very much longer:

```
net>  ping ns9mxa 0 5000

Resolving ns9mxa... Pinging ns9mxa (44.199.41.90); data 0
interval 5000 ms:

 sent      rcvd     %       rtt    avg rtt      mdev
    1         1   100      4923       4923         0
    2         2   100      2305       4596       655
    3         3   100      2818       4374       936
    4         4   100      2399       4127      1196
```

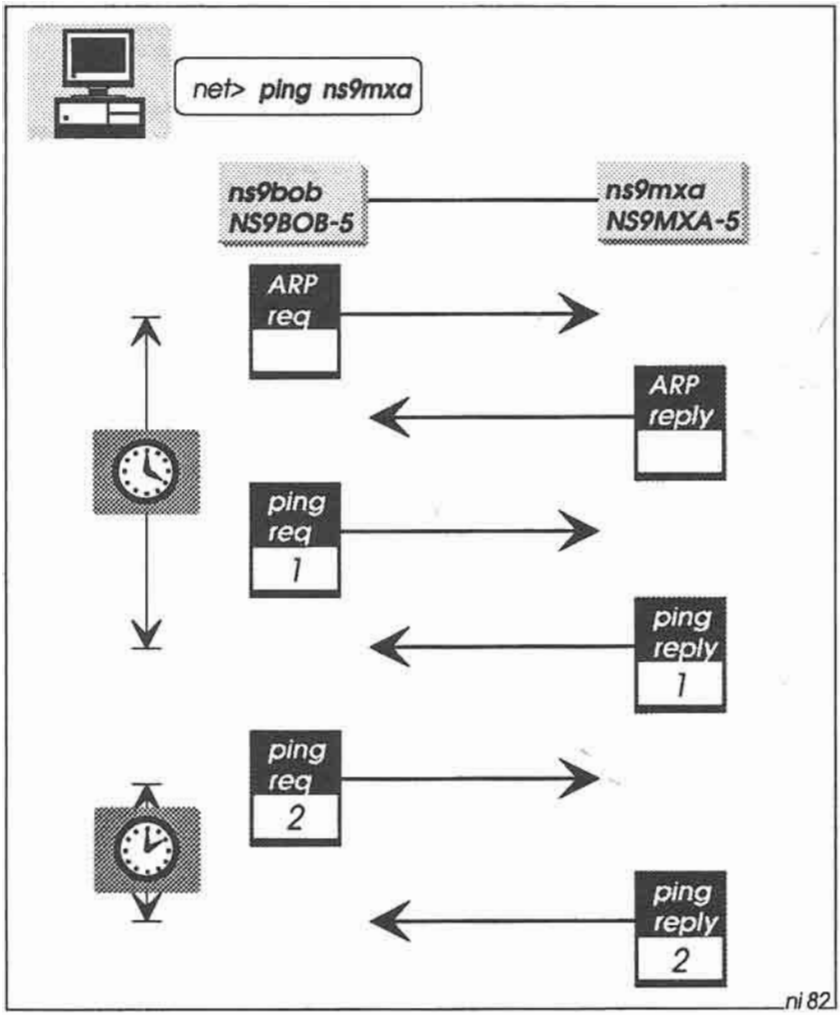


Fig 33-1: The *ping* command checks that a remote station is active. If the sending station does not know the AX.25 callsign of the target, an ARP request is sent out first to get the callsign.

Note here that the rtt for the first **ping** (4923 ms) is about twice as long as the remaining rtt values. This is because the AX.25 callsign for ns9mxa is unknown at the start (there is no ARP table entry), and so ns9bob needs to send out an ARP Request to get it. Station ns9mxa

responds with an ARP Reply containing its callsign (NS9MXA-5), giving ns9bob the necessary AX.25 destination address for the ping packets.

The incflag parameter in the **ping** command is an experimental feature to allow a group of stations to be pinged. If you set the flag to a non-zero value, the Internet address will be incremented by one after each ping.

The *hop* commands

There are several **hop** commands:

hop check: This command initiates a hop check session to the specified target host. It sends a series of UDP probe packets to determine the sequence of gateways in the path to the target. For example:

```
net>  hop check ns9liz
```

A new session starts, reporting the progress of the probe packets through the network to the target:

```
Resolving ns9liz... hopcheck to 44.199.45.17:33434
 1: 44.199.41.1      ns9bob.ampr.org.   (20 ms) (12 ms)
 2: 44.199.41.2      ns9ken.ampr.org.   (2047 ms) (1946 ms)
 3: 44.199.45.17     ns9liz.ampr.org.   (4237 ms) (4163 ms)
hopcheck done: normal (Unreachable Port)
```

This shows that there is a route from Bob to Liz via Ken. Exactly how this works is described in detail below.

hop maxttl: This command sets the limit of your search; i.e. the maximum number of IP gateways through which the probe packets pass on their way to the target. For example, to limit the search to 10 gateways:

```
net>  hop maxttl 10
```

hop maxwait: This command lets you set the maximum time in seconds that a hop check session will wait for responses at each stage of the trace. For example:

```
net>  hop maxwait 60
```

hop queries: This command sets the number of UDP probes that will be sent at each stage of the trace. For example:

```
net>  hop queries 2
```

hop trace: You can record the progress of the check with this command. For example:

```
net>  hop trace on
```

The trace goes into the session log file (*/dump/session.log*). Predictably, the command **hop trace off** turns the trace off.

How *hop check* works

The **hop check** command works by sending a series of UDP probe packets to the target, with progressively increasing values of the time-to-live (ttl) parameter — see Fig 33-2. That is, the first packet has a ttl value of 1, the second has a value of 2, and so on, up to the maxttl value. Each packet is addressed to a dummy UDP port number (33434) at the target.

As each probe packet passes through a gateway, the gateway decrements the ttl value by one. If this results in a value of zero, the gateway sends an ICMP *Time Exceeded* packet back to the originating station, ns9bob.

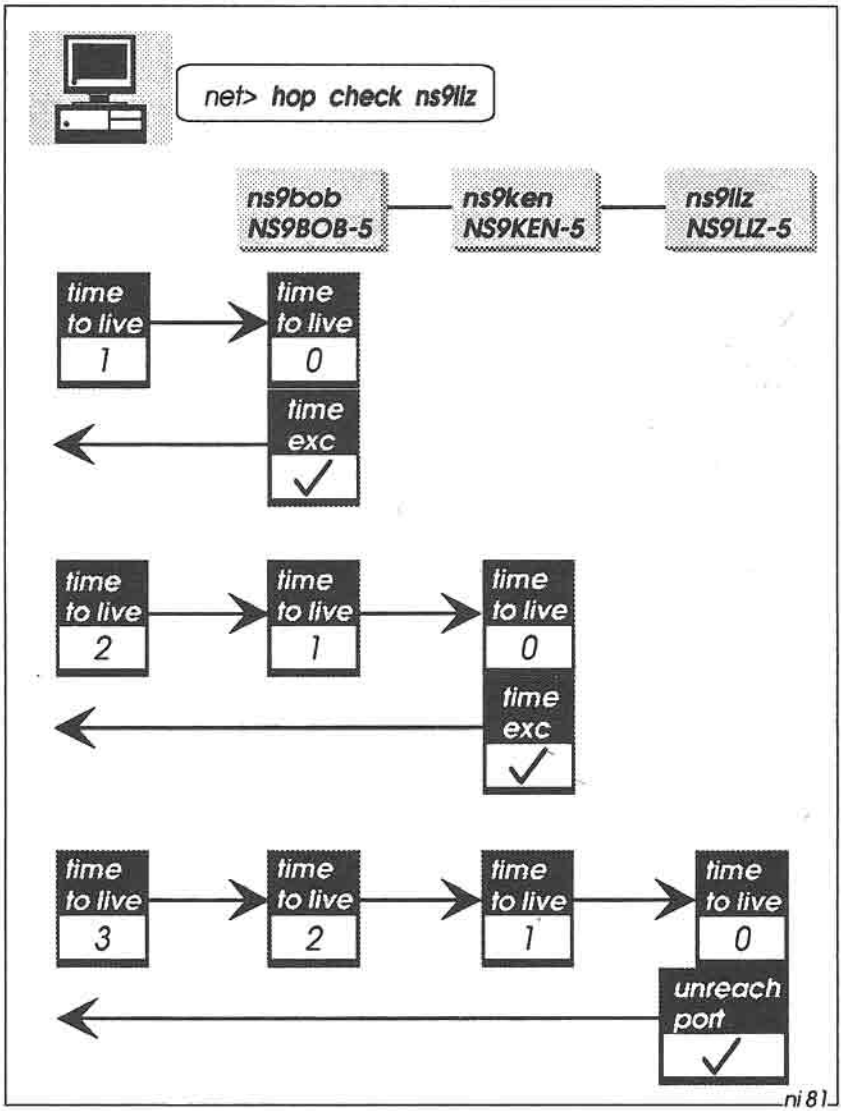


Fig 33-2: The *hop check* command sends a series of UDP probe packets towards the target. As each probe passes through a gateway, the time-to-live is decremented by one. If the ttl becomes zero, the gateway returns *Time Exceeded*. When the final target receives a probe, it returns *Unreachable Port*.

The first probe passes to ns9bob's own gateway, which decrements the ttl to zero and returns a Time Exceeded packet. The second probe reaches ns9ken, which likewise returns Time Exceeded.

The third probe gets as far as the target, ns9liz. As the packet is addressed to ns9liz, an attempt is now made to connect to UDP port 33434. This port doesn't actually exist, so ns9liz sends an ICMP *Unreachable Port* packet back to ns9bob. This tells ns9bob that the hop check is complete, and it can finally display the complete report.

34: HANDS ON — DOMAIN NAME SYSTEM

Most NOS users keep all of their IP name and address information in *domain.txt* — because there is no alternative. In some areas, however, certain NOS stations provide a Domain Name System (DNS) server that contains a much more comprehensive list of names and addresses.

If a DNS server exists in your area, you can make use of it (Fig 34-1).

The first step is to tell NOS the hostname of the DNS server; e.g.

```
net> domain addserver ns9dns
```

DNS Queries

Once you have defined a DNS server as above, you can virtually forget about it.

Now, when you make a request to talk to a particular station, NOS first looks in your *domain.txt* for the station's IP address. If NOS doesn't find it there, it then sends a query to the DNS server for the address. Hopefully the server will then respond with the address you need, and then NOS can go ahead with your original command.

To see what is happening when NOS sends a query to the DNS, you can turn on a trace:

```
net> domain trace on
```

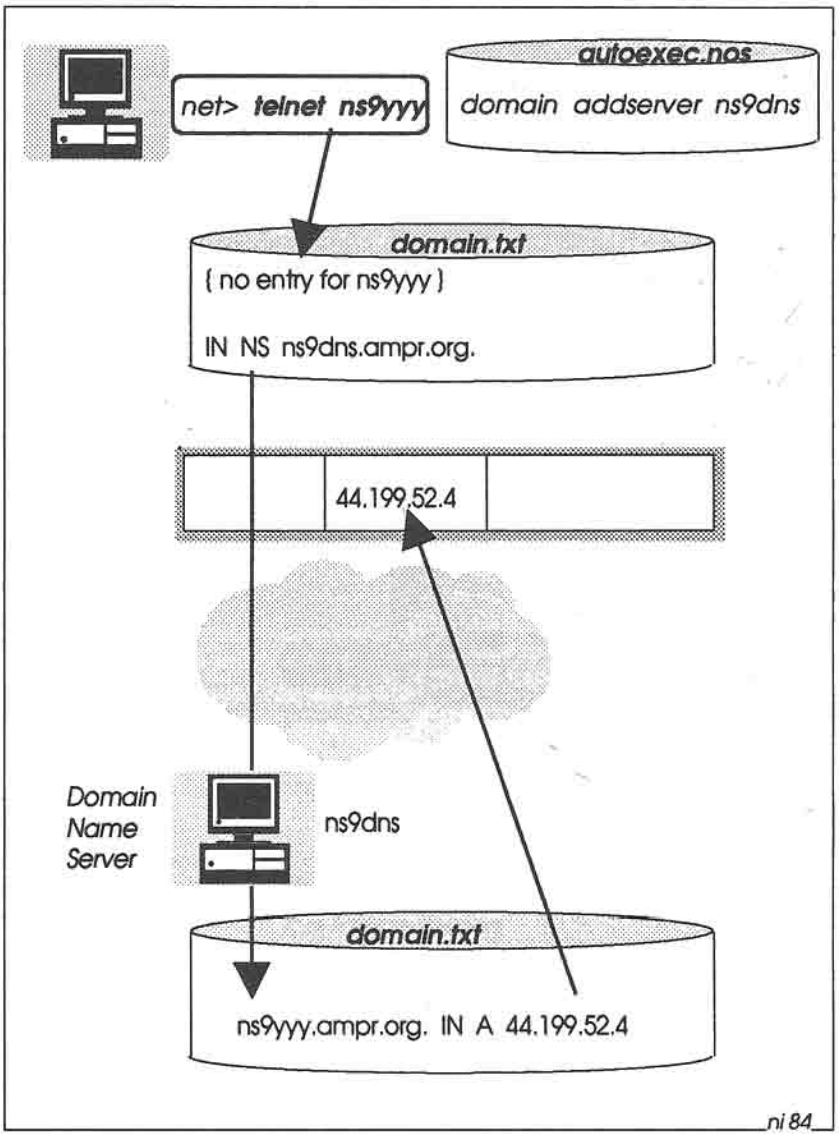


Fig 34-1: If there is no entry in the local *domain.txt* file for the target host, NOS makes a request to the DNS server (*ns9dns*). The server then provides the wanted IP address.

Let's see what happens when you give the command **ping ns9yyy**. Station ns9yyy is not in *domain.txt*. A new session starts, and the domain trace output appears on the screen:

```
Resolving ns9yyy...
dns_query: querying server ns9dns for ns9yyy.ampr.org.
dns_query: received message length 62, errno 0
response id 21208 (rtt 8026 ms) qr 1 opcode 0 aa 1 tc 0 rd 1
ra 0 rcode 0
1 questions:
ns9yyy.ampr.org. type A class 1
1 answers:
ns9yyy.ampr.org. 3600    IN      A       44.199.52.4
```

Now that NOS knows the IP address for ns9yyy (44.199.52.4), it can go ahead with the **ping** command. (If the DNS server doesn't have an entry for the station you want, it returns an IP address of 0.0.0.0).

The Domain Cache

As well as using the new-found address for the **ping** command, NOS also puts this new address into the *domain cache*, an area of memory which contains a list of currently used IP addresses.

To discover the state of the cache, you can give the **domain cache list** command:

```
net> domain cache list
ns9yyy.ampr.org. 3505    IN      A       44.199.52.4
ns9ken.ampr.org.         IN      A       44.199.41.2
ns9bob.ampr.org.         IN      A       44.199.41.1
```

The entries for ns9ken and ns9bob were derived from *domain.txt*, but the entry for ns9yyy came from the DNS server. Note that a time-to-live value (3505) also appears in this entry. By default, the entry has a time-to-live at birth of 3600 seconds (one hour).

When the time-to-live eventually falls to zero, NOS marks the entry as "old". The entry may then disappear from the list, depending on the setting of the *cache clean* flag.

You will normally set the flag to off (in *autoexec.nos*), which means that the entry will not be removed. But you can set the flag to on:

```
net>  domain cache clean on
```

In this case the entry will disappear when its time-to-live has expired.

Updating *domain.txt*

Another thing happens when NOS puts a DNS entry into the domain cache. After the entry has been there for a certain time, NOS will copy it into *domain.txt*. Thus you should look at *domain.txt* from time to time, when you may find something like this:

```
ns9yyy.ampr.org.   3193   IN A   44.199.52.4    {new DNS entry}
ns9qqq.ampr.org.   3024   IN A   0.0.0.0        {new DNS entry}
ampr.ampr.org.          IN A      44.0.0.0      {original entry}
nosland.ampr.org.       IN A      44.199.0.0    {original entry}

                                  {etc}
```

That is, new DNS entries have appeared at the front of the file. Note that this includes 0.0.0.0 entries for which the DNS server couldn't find a real IP address; you should remove these entries with a text editor.

You can control the time that NOS waits before updating *domain.txt*, using the **domain cache wait** command:

```
net>  domain cache wait 300
```

This sets the waiting time to 300 seconds (5 minutes) before NOS updates the file.

Other Domain Commands

There are several other domain commands which you may find useful.

- To list the server(s) which you can access, and certain related performance statistics:

```
net> domain list
Server address  srtt  mdev timeout queries responses timeouts
ns9dns          7540  2746   18524       6         4        2
```

- To remove a DNS server:

```
net> domain dropserver ns9dns
```

- To set the number of entries which the domain cache can hold:

```
net> domain cache size 30
```

- To set the timeout value (in seconds) for a DNS query:

```
net> domain maxwait 120
```

Note that you must give this command before adding any servers.

35: TRAILING FLAG

That's **NOSintro**. If this is the first time you've reached this chapter, you should now have a rough idea about what NOS and TCP/IP can do. Now return to the start, and collect a copy of **NOSview** on the way — you'll need it later to make sense of the detail next time around.

If this is the third or fourth time through, you're probably ready to try NOS on the air. After you've edited all the control files to suit your own environment, please check everything thoroughly once again before switching on the radio. You'll need all the help you can get to make everything work properly, and the last thing you want is to upset your friends by flooding the network with broadcasts from NS9BOB!

If this is the umpteenth time you've passed this way, you should be fairly fluent in using NOS. Now is the time to start experimenting. Set up or join a packet group to try things out together. Tell the world what you're doing, write articles for newsletters and magazines (or even write a book), and pledge to help others who are just starting out on this fascinating adventure.

And finally, if you can spare a few moments, please let me know what you think of this book. Have you found it helpful, was it clear to understand, were some topics too detailed or not detailed enough, how can the book be improved? There are almost certainly some (hopefully minor) mistakes in it — in a book of this nature, it's virtually impossible to guarantee 100 percent accuracy, however thorough the checking. Let me know what they are.

Your feedback is important to me, complimentary or not, and I will of course reply.

Happy pinging!

APPENDIX 1: WHERE TO GET THE SOFTWARE

NOS (and related versions such as **NET**) are available on many platforms today. The table below contains a list of people whom you can contact to get a copy.

PLATFORM	CONTACT
Amiga	G1YYH @ GB7NWP G4UFG @ GB7CRG
Apple Macintosh	PA2AGA @ PI8EAE.NLD.EU G0OAN @ GB3XP
Archimedes	G4KLX @ GB7HMZ
Atari	G1PLT @ GB3XP PE1CHL @ PI8UTR.NLD.EU
DEC VAX/VMS	SM0IJZ @ SM0ETV.A.SWE.EU G1OXB @ GB3XP
HP-UX (WAMPES)	DK5SG/N0PRA
Interactive UNIX	G8ZHR @ GB3XP
PC DOS (JNOS)	WG7J
PC DOS (PA0GRI)	G3NRW @ GB7BIL
PC DOS (WNOS)	G6DHU @ GB7IMB DB3FL @ DB0GV.DEU.EU

OS/2 (PMNOS)	KZ1F
SCO Xenix/Unix	G1PLT @ GB3XP G8ZHR @ GB3XP PE1CHL @ PI8UTR.NLD.EU DG7DAH @ DB0MWE.BA.DEU.EU
SunOS	G1PLT @ GB3XP

The Clarkson Packet Drivers

The Clarkson drivers (aka the Crynwr drivers) are distributed in three files: *drivers.zip* (executables and documentation), *drivers1.zip* (first half of the .ASM files) and *drivers2.zip* (second half of the .ASM files).

By Post Office Mail: 9-track 1600 bpi tapes in ANSI, tar, or OS SL format, or PC diskettes (360K 5¼" and 720 KB 3½"). Exact terms and conditions have yet to be worked out. Please call USA (212) 854-3703 for ordering information, or write to Kermit Distribution, Dept PD, Columbia University Center for Computing Activities, 612 West 115th Street, New York, NY 10025, USA, or send e-mail to *kermit@watsun.cc.columbia.edu* (Internet) or *KERMIT @CUVMA* (BITNET/EARN).

By FTP/email: The packet driver collection has its own directory devoted to it on *simtel20.army.mil, pd1:<msdos.pktdrvr>*. The drivers are there, along with many free programs that use the packet drivers.

For more details contact Russell Nelson at 11 Grant St, Potsdam, NY 13676, USA.

APPENDIX 2
NOS COMMAND SET REFERENCE

This appendix contains details of all of the commands to be found in the major NOS packages generally available. Note that not all commands are available in every package.

Command names and literal strings shown in **bold** type.

Parameters shown in *italics*.

Default parameters shown in braces; e.g. *{30}*.

NOS Startup Options

nos [**-b**] (console BIOS)
[**-d** *root_directory*]
[**-m** *heap_memory_in_KB*]
[**-s** socket_array_size]
[**-v**] (verbose — startup trace)
[*nos_autoexec_filename*]

NOS Command Set

? (help: list of top-level NOS commands)
! (break out to shell)
(comment line)
F10 (escape to NOS command level)

---

abort [*session_number*] (FTP)

arp
arp add *host* **ether | ax25 | netrom | arcnet** *ether_address | callsign*
arp drop *host* **ether | ax25 | netrom | arcnet**

arp flush
arp publish *host* **ether | ax25 | netrom | arcnet** *ether_address | callsign*

asystat

attach 3c500 *ioaddress vector* **arpa** *interface qlen mtu* [*ip_address*]

attach asy *ioaddress vector* **ax25 | nrs | ppp | slip | raw** *interface buffers mtu speed* [*options*]

option c: enable RTS/CTS
r: enable RLSD/CD
v: enable compression

attach axip *interface mtu their_host my_axip_callsign*

attach drsi *ioaddress vector* **ax25** *interface bufsize mtu chan_a_speed chan_b_speed* [*ip_address_a*] [*ip_address_b*]

attach eagle *ioaddress vector* **ax25** *interface buffers mtu speed* [*ip_address_a*] [*ip_address_b*]

attach hapn *ioaddress vector* **ax25** *interface rx_bufsize mtu* **csma | full** [*ip_address*]

attach hs *ioaddress vector* **ax25** *interface buffers mtu txdelay persistence* [*ip_address_a*] [*ip_address_b*]

attach kiss *existing_asy_interface port interface* [*mtu*]

attach netrom

attach packet *vector interface tx_queue_length mtu* [*ip_address*]

attach pc100 *ioaddress vector* **ax25** *interface buffers mtu speed* [*ip_address_a*] [*ip_address_b*]

attach pi

attach scc *devices* **init** *ioaddress spacing Aoff Boff Dataoff intack vector* [p]*clock* [*hardware_type*] [*param*]

attach scc *chan* **slip | kiss | nrs | ax25** *interface mtu speed bufsize* [*callsign*]

attach slfp

attended [**on | off**] *{on}*

autoroute [**on | off**] *{off}*

ax25 bc *interface* [**on | off**] *{off}*
ax25 bcinterval [*seconds*] *{0}*

ax25 bckick	*interface*	
ax25 bctext	["*broadcast_text*"]	
ax25 blimit	[*count*]	*{30}*
ax25 dest	[*interface*]	
ax25 digipeat	**[on | off]**	*{on}*
ax25 filter	**[0 | 1 | 2 | 3]**	*0=src+dest {0}* *1=dest only* *2=src only* *3=neither*
ax25 flush		
ax25 heard	[*interface*]	
ax25 hearddest	[*interface*]	
ax25 irtt	[*millisecs*]	*{5000}*
ax25 kick	*&AXB*	
ax25 maxframe	[*window_size*]	*{1}*
ax25 mycall	[*callsign*]	
ax25 paclen	[*bytes*]	*{256}*
ax25 pthresh	[*bytes*]	*{128}*
ax25 reset	*&AXB*	
ax25 retry	[*n*]	*{10}*
ax25 route		
ax25 route add	*target_callsign* [*digi_callsign ...*]	
ax25 route drop	*target_callsign*	
ax25 route mode	*target_callsign* **[vc | datagram | interface]**	
ax25 status	[*&AXB*]	
ax25 t3	[*millisecs*]	*{0}*
ax25 t4	[*seconds*]	*{300}*
ax25 timertype	**[original | linear | exponential]**	*{exponential}*
ax25 version	**[1 | 2]**	
ax25 window	[*bytes*]	*{2048}*

---

bbs

	Help	**?**	*(command list)*
	Area	**A** [*area_name*]	
	Bye	**B**	
	Chat	**C**	
or	*Connect*	**C** *netrom_node*	
		C *port callsign*	
	Download	**D** *filename*	*(ASCII file)*
		DU *filename*	*(uuencoded binary file)*
	Escape	**E** [*esc_char*]	*{^X}*
	Finger	**F** *@host*	
		F *username@host*	
	Gateway	**G** *interface callsign* [*digi_callsign...*]	
	Help	**H** [*command_name*]	
	Info	**I**	
	Heard	**J** [*interface*]	
	Kill	**K** *n ...*	
	List	**L** [*n ...*]	
	Mbxusers	**M**	

	Nodes	**N**	
or	*Netrom*	**N**	**c** *callsign* **i** **n** *callsign* **u**
	Operator	**O**	
	Ports	**P**	
	Read	**R**	*n ...*
	Send	**S**	*username*[**%***host*][**@***host*] [**<** *from_addr*] [**$***bulletin_id*]
	Forward	**SF**	*username*[**%***host*][**@***host*] [**<** *from_addr*] [**$***bulletin_id*]
	Reply	**SR**	[*n*]
	Telnet	**T**	*host* [*well_known_port_number*] *{23}*
	Upload	**U**	*filename* *(ASCII file)*
	Verbose	**V**	*n ...*
	What	**W**	[*directory*]
	Expert	**X**	
	Zap	**Z**	*filename*
	Sysop	**@**	

bootp defaultfile [*boot_file* | **default**]
bootp dns [*ip_address*]
bootp dyip [*interface* | *interface ip_addr1 ip_addr2* | *interface* **off**]
bootp homedir [*directory* | **default**]
bootp host [*host_addr hardware_type hardware_address*
ip_address [*boot_file*]]
bootp logfile [*filename* | **default**] [**on** | **off**]
bootp logscreen [**on** | **off**]
bootp rmhost *ip_address*
bootp start
bootp stop

bootpd

---

cd [*directory*]
close [*session_number*]
cls
comm *interface* **"***string***"**
connect *interface callsign* [*digi_callsign ...*]

---

delete *filename*
detach *interface*

dialer *interface* [*file* [*seconds* [*pings* [*host*]]]]
control up | down
send *string*
speed 9600 | 4800 | 2400 | 1200 | 300
status up | down
wait *millisecs* [*string* [*speed*]]

dialer *interface seconds target_host dialer_filename*

dialer *interface* **0** *(turns dialer off)*

dir [*directory* | *filename*]
disconnect [*session_number*] *(AX.25)*

domain addserver *host* [*host* ...]
domain cache clean [**on** | **off**] *{off}*
domain cache list
domain cache size [*entries*] *{20}*
domain cache wait [*seconds*] *{300}*
domain dropserver *host* [*host* ...]
domain list
domain maxwait [*seconds*] *{60}*
domain query *host*
domain remote add *host* [*host* ...]
domain remote drop *host* [*host* ...]
domain remote list
domain remote retry [*n*]
domain remote trace [**on** | **off**]
domain retry [*n*] *{2}*
domain suffix [*domain_suffix*] *{ampr.org.}*
domain trace [**on** | **off**] *{off}*
domain translate [**on** | **off**] *{off}*
domain verbose [**on** | **off**] *{on}*
domain xyzzy

drsistat

dump *hex_memory_address* | **.** [*decimal_range*]
(memory address is 8 hex chars without colon)

---

eaglestat
echo [**accept** | **refuse**] *(telnet) {accept}*
eol [**standard** | **null**] *(telnet) {standard}*

escape *literal_character* *{^X}*
(^[is the **ESC** *key)*

etherstat

exit

---

finger [*username*]@*host* *(no spaces between parameters)*

fkey [*key_number* [*value* | "*string*"]] *(use ^M for CR)*

key	*normal*	*shift*	*control*	*alt*		
F1	59	84	94	104	PgUp	73
F2	60	85	95	105	PgDn	81
F3	61	86	96	106	Home	71
F4	62	87	97	107	End	79
F5	63	88	98	108	↑	72
F6	64	89	99	109	↓	80
F7	65	90	100	110	←	75
F8	66	91	101	111	→	77
F9	67	92	102	112	Ins	82
F10	68	93	103	113	Del	83

ftp *host*

ascii		
batch	[**on** | **off**]	
binary		
cd	*remote_dir*	
dele	*remote_file*	
dir	[*remote_dir* | *remote_file* [*local_file*]]	
flow	[**on** | **off**]	
get	*remote_file* [*local_file*]	
hash		
list	[*remote_dir* | *remote_file* [*local_file*]]	
ls	[*remote_dir* | *remote_file* [*local_file*]]	
mget	*remote_file* [*remote_file* ...]	
mkdir	*remote_dir*	
mput	*local_file* [*local_file* ...]	
nlst	[*remote_dir* | *remote_file* [*local_file*]]	
pass	*password*	
put	*local_file* [*remote_file*]	
pwd		
quit		
rmdir	*remote_dir*	
type	[**a** | **i** | **l** *bytesize*]	*{a}*
user	*username*	
verbose	[*n*]	*n=0: errors only* *1: + summary* *2: + progress* *3: + hash*

ftype [**ascii** | **binary** | **image** | **logical** *size*] *{ascii}*

F10 *(to escape to NOS command level)*

---

hapnstat
help *(list of top-level NOS commands)*

hop check *host*
hop maxttl [*hops*] *{30}*
hop maxwait [*seconds*] *{5}*
hop queries [*count*] *{3}*
hop trace [**on** | **off**] *{off}*

hostname [*hostname*]
hs

---

icmp echo [**on** | **off**] *(must be on for one-shot ping)* *{on}*
icmp status
icmp trace [**on** | **off**] *(turn off for hop check)* *{off}*

ifconfig
ifconfig [*interface*]
ifconfig *interface* **broadcast** *bcast_ip_address*
ifconfig *interface* **description** [**"***description***"**]
ifconfig *interface* **encapsulation** **none** | **ax25** | **encap** | **ether** | **slip** | **netrom** | **ppp**
ifconfig *interface* **forward** *fwd_interface*
ifconfig *interface* **ip_address** *ip_address*
ifconfig *interface* **linkaddress** *callsign* | *ether_addr*
ifconfig *interface* **mtu** *bytes*
ifconfig *interface* **netmask** [**0x**]*hexmask*
ifconfig *interface* **rxbuf** *buffer_size*

info

ip access **permit** | **deny** | **delete** *ip_address*[/*bits*] *interface* [*low_port_no* [*high_port_no*]]

ip address [*ip_address*]
ip rtimer [*seconds*] *{30}*
ip status
ip ttl [*hops*] *{255}*

isat [**on** | **off**] *{off}*

---

kick [*session_number*]

---

lock	**[password** *"password_string"***]**	
log	[*log_filename* | **stop**]	
lzw	**mode [fast | compact]**	
lzw	**bits** [*n*]	*{9}*

---

mail		
mbox		
mbox attend	**[on | off]**	
mbox expert	**[on | off]**	*{off}*
mbox fwdinfo	[*"forward_info"*]	
mbox haddress	[*"home_address"*]	
mbox jumpstart	**[on | off]**	*{off}*
mbox kick		
mbox maxmsg	[*n*]	*{200}*
mbox motd	[*"string"*]	
mbox nrid	**[on | off]**	
mbox operator	[*ip_address*]	
mbox password	*"password_string"*	
mbox qth	[*"qth_string"*]	
mbox secure	**[on | off]**	
mbox smtptoo	**[on | off]**	
mbox status		
mbox timer	[*seconds*]	*{0}*
mbox tiptimeout	[*seconds*]	*{180}*
mbox trace	**[on | off]**	*{off}*
mbox utc	[+ | -][*n*]	
mbox zipcode	[*"zipcode"*]	
mem circular	**[on | off]**	*{off}*
mem debug	**[on | off]**	*{off}*
mem efficient	**[on | off]**	*{off}*
mem free		
mem garbage		
mem ifbufsize	[*bytes*]	*{2048}*
mem minheap	[*n*]	
mem nibufs	[*n*]	*{5}*
mem sizes		
mem status		
mem thresh	[*bytes*]	*{8192}*
mkdir	*directory*	
mode	*interface* **[vc | datagram]**	*(AX.25)*
mode	**netrom [vc | datagram]**	
more	*filename* [*filename ...*]	*(q: quit)* *(space: next page)* *(CR: next line)*

motd	["*string*"]	
multitask	**[on | off]**	*{on}*

---

netrom acktime	[*millisecs*]	*{3000}*
netrom alias	[*node_alias*]	
netrom bcnodes	*interface*	
netrom call	[*callsign*]	*{ax25 mycall}*
netrom connect	*node_callsign* | *node_alias*	
netrom choketime	[*millisecs*]	*{180000}*
netrom derate	**[on | off]**	*{on}*
netrom interface	*interface quality*	
netrom interface	*interface alias quality*	
netrom irtt	[*millisecs*]	*{15000}*
netrom kick	*&CB*	
netrom load		
netrom minquality	[*n*]	*{10}*
netrom nodefilter		
netrom nodefilter add	*neighbour_callsign interface* [*quality*]	
netrom nodefilter drop	*neighbour_callsign interface*	
netrom nodefilter mode	**[none | accept | reject]**	*{accept}*
netrom nodetimer	[*seconds*]	*{0}*
netrom obsotimer	[*seconds*]	*{0}*
netrom promiscuous	**[on | off]**	*{off}*
netrom qlimit	[*bytes*]	*{2048}*
netrom reset	*&CB*	
netrom retries	[*n*]	*{10}*
netrom route		
netrom route add	*alias target_callsign interface quality neighbour_callsign*	
netrom route drop	*target_callsign neighbour_callsign interface*	
netrom route info	*target_callsign* | *target_alias*	
netrom save		
netrom status	[*&CB*]	
netrom timertype	**[linear | exponential]**	
netrom ttl	[*hops*]	*{10}*
netrom user	[*username*]	
netrom verbose	**[on | off]**	*{off}*
netrom window	[*frames*]	*{4}*
nntp addserver	*nntpserver_host* [*interval_in_seconds*] [*time_range*] [*group* [*group* ...]]	
nntp directory	**[spool | control** *directory*]	
nntp dropserver	*nntpserver_host*	
nntp groups	[*newsgroup_name* ...]	
nntp kick	*nntpserver_host*	
nntp listservers		

nntp quiet [on | off]
nntp trace [*n*]
n=0: no trace
1: serious errors
2: transient errors
3: session progress
4: received articles
5: errors

nrstat

---

param *interface*
param *interface* *param* [*param* ...]

param *interface* **0** *data frame*
param *interface* **1** *TX_delay* *(10mS units)*
param *interface* **2** *persistence* *(0-255)*
param *interface* **3** *slot_time* *(10mS units)*
param *interface* **4** *TX_tail* *(10mS units)*
param *interface* **5** *n* *(n=0: HDX)*
(n>0: FDX)
param *interface* **6** *(hardware dependent)*
param *interface* **7** *(TX mute)*
param *interface* **8 | dtr** [*n*] *(n=0: DTR low)*
(n=1: DTR hi)
param *interface* **9 | rts** [*n*] *(n=0: RTS low)*
(n=1: RTS hi)
param *interface* **10** *(speed)*
param *interface* **11** *(end delay)*
param *interface* **12** *(group)*
param *interface* **13** *(idle)*
param *interface* **14** *(min)*
param *interface* **15** *(max key)*
param *interface* **16** *(wait)*
param *interface* **17** *parity* *(n=0: none)*
(n=1: even)
(n=2: odd)
param *interface* **129** *(down)*
param *interface* **130** *(up)*
param *interface* **254** *(return2)*
param *interface* **255** *(exit KISS)*

ping *host* [*length* [*seconds* [*incflag*]]]
ping *host* [*length* [*milliseconds* [*incflag*]]]

pop kick
pop mailbox *mbox_name*
pop mailhost [*host*]
pop quiet [on | off] *{off}*
pop timer [*seconds*] *{0}*
pop userdata [*username password*]

popmail addserver *host [seconds] [hh:mm-hh:mm] protocol mailbox username password*
popmail dropserver *host*
popmail list
popmail quiet **[on | off]**
popmail trace *[n]* *(n=0: no trace) (n=1: serious) (n=2: transient) (n=3: session)*

ppp *interface*

ppp *interface* **ipcp** **open active | passive**
ppp *interface* **ipcp** **timeout** *[seconds]*
ppp *interface* **ipcp** **try configure** *[count]*
ppp *interface* **ipcp** **try failure** *[count]*
ppp *interface* **ipcp** **try terminate** *[count]*
ppp *interface* **ipcp** **local | remote address** *[host | allow* **[on | off]** *]*
ppp *interface* **ipcp** **local | remote compress [tcp** *slots [flag* **| none | allow [on | off]]**

ppp *interface* **lcp** **close**
ppp *interface* **lcp** **local | remote**
ppp *interface* **lcp** **local | remote accm** *[bitmap* **| allow [on | off]]**
ppp *interface* **lcp** **local | remote authenticate [pap | none | allow [on | off]]**
ppp *interface* **lcp** **local | remote compress address | control [on | off | allow [on | off]]**
ppp *interface* **lcp** **local | remote compress protocol [on | off | allow [on | off]]**
ppp *interface* **lcp** **local | remote default**
ppp *interface* **lcp** **local | remote magic [on | off | allow [on | off]]**
ppp *interface* **lcp** **local | remote mru** *[size]* **| allow [on | off]]**
ppp *interface* **lcp** **open active | passive**
ppp *interface* **lcp** **timeout** *[seconds]*
ppp *interface* **lcp** **try configure** *[count]*
ppp *interface* **lcp** **try failure** *[count]*
ppp *interface* **lcp** **try terminate** *[count]*

ppp *interface* **pap** **user** *[username [password]]*

ppp *interface* **trace** *[flags]*

ps
pwd *[directory]*

---

rarp
rarp query *interface ether_addr | callsign [ether_addr | callsign...]*

record *[filename* **| off]** *{off}*

remote [**-p** *port*] [**-k** *key*] [**-a** *kickaddr*] *host* **exit | reset | kick**
remote **-s** *key*

rename *old_filename new_filename*
reset [*session_number*]

rip accept *incoming_gateway_host*
rip add *destination_host secs* [*flags*] *flags=1: include route to self*
2: split horizon
4: triggered update
rip drop *destination_host*
rip merge [**on | off**] *{off}*
rip refuse *incoming_gateway_host*
rip request *incoming_gateway_host*
rip status
rip trace [*n*] *n=0: no trace*
1: changes only
2: full trace
rip ttl [*seconds*] *{240}*

rlogin *host*
rmdir *directory*

route
route add *target_host*[/*bits*] | **default** *interface* [*gateway_host* [*metric*]]
route addprivate *target_host*[/*bits*] | **default** *interface* [*gateway_host* [*metric*]]
route drop *target_host*[/*bits*]
route flush
route lookup *target_host*

rspf interface [*interface quality horizon*]
rspf maxping [*n*] *{5}*
rspf message ["*message_string*"]
rspf mode [**vc | datagram | none**] *{none}*
rspf routes
rspf rrhtimer [*seconds*] *{0}*
rspf status
rspf suspecttimer [*seconds*]
rspf timer [*seconds*] *{0}*

---

sccstat

session
session [*session_number*]
session *session_number* **flowmode [on | off]** *{off}*

shell

skick	*socket_number*	
smtp batch	[**on** | **off**]	*{off}*
smtp gateway	[*host*]	
smtp kick		
smtp kill	*job_number*	
smtp list		
smtp maxclients	[*n*]	*{10}*
smtp mode	[**queue** | **route**]	*{route}*
smtp mxlookup	[**on** | **off**]	*{off}*
smtp quiet	[**on** | **off**]	*{off}*
smtp reclzw	[**on** | **off**]	*{on}*
smtp sendlzw	[**on** | **off**]	*{on}*
smtp timer	[*seconds*]	*{0}*
smtp trace	[*n*]	*n=0: trace off* *1: trace on*
smtp usemx	[**on** | **off**]	*{off}*
socket	[*socket_number*] [**flowmode** [**on** | **off**]]	
source	*script_filename*	
start	**ax25** | **discard** | **echo** | **finger** | **ftp** | **netrom** | **pop** | **pop2** | **pop3** | **remote** | **rip** | **smtp** | **telnet** | **ttylink**	
start tip	*sync_interface*	
status		
stop	**ax25** | **discard** | **echo** | **finger** | **ftp** | **netrom** | **pop** | **pop2** | **pop3** | **remote** | **rip** | **smtp** | **telnet** | **ttylink**	
stop tip	*sync_interface*	

---

tail	*filename*	
tcp irtt	[*millisecs*]	*{5000}*
tcp kick	*&TCB*	
tcp mss	[*bytes*]	*{512}*
tcp reset	*&TCB*	
tcp rtt	*&TCB millisecs*	
tcp status	[*&TCB*]	
tcp syndata	[**on** | **off**]	*{off}*
tcp timertype	[**linear** | **exponential**]	*{exponential}*
tcp trace	[**on** | **off**]	*{off}*
tcp view		
tcp window	[*bytes*]	*{2048}*
telnet	*host* [*well_known_port_number*]	*{23}*
telnet	*host* **25**	*(SMTP)*
telnet	*host* **87**	*(CHAT/TTYLINK)*

test
third-party **[on | off]** *{on}*
tip *async_interface*
ttylink *host [well_known_port_number]* *{87}*

trace
trace *interface* **[off |** *BTIO_flags [trace_filename]* **]]**

BTIO_flags:

B=0 Broadcast filter off (trace all packets)
B=1 Broadcast filter on (ignore broadcasts)

T=0 Display protocol headers only
T=1 Display headers + ASCII text
T=2 Display headers + ASCII text + hex

I=0 Ignore input packets
I=1 Trace input packets

O=0 Ignore output packets
O=1 Trace output packets

---

udp **status**

upload
upload *filename*

---

watch **[on | off]** *{off}*
watchdog **[on | off]** *{off}*

APPENDIX 3
NOS CONTROL FILES

alias

```
# ======                                                NOSview [244]
# /alias
# ======
#
# SMTP server ALIAS file.  This is for resolving a given
# target address into a single- or multiple-entry mail list.
#
# Format:
# ------------------------------------------------------------
# mail_list_name call_1@host_1 [call_2@host_2]... # comments
# ------------------------------------------------------------
# N.B. There must be exactly ONE SPACE between each field.
#
# ken ns9ken@ns9ken
# thegirls ns9pam@ns9pam ns9sue@ns9sue ns9liz@ns9liz outtray
```

areas

```
# ==========                                            NOSview [244]
# /spool/areas
# ==========

              NS9BOB Public Message Areas
              ===========================

all     ........  General chit-chat.
tcpip   ........  General TCP/IP and NOS messages.

 To enter an area, type "a" followed by the area name:
    e.g.  "a all"
```

AUTOEXEC.BAT

```
REM ================                                    NOSview [244]
REM C:\AUTOEXEC.BAT
REM ================

REM  All the usual AUTOEXEC.BAT commands here

REM  SET UP THE NOS ENVIRONMENT
CALL C:\NOS\NOSENV.BAT
```

autoexec.nos

```
# ============                                        NOSview [244]
# /autoexec.nos
# ============

#  This is the configuration file for NOS.
#  This file must be at the NOS root.

#  There are many commands which NOS needs to configure the
#  program each time it starts.  To save typing them by hand
#  each time, you can put them in this file.

#  AUTOEXEC.NOS is the default name for the configuration
#  file. You can also produce alternative versions with
#  different names.  This is a very convenient way of setting
#  up different scenarios.  To use one of these alternatives,
#  you include its name as the final parameter in the NOS
#  command line.
#    e.g.  nos_20m /autoexec.bob

#  Many of the commands below are commented out with the #
#  sign.  As you gain experience with NOS you can then remove
#  the # signs to try out these commands.

#  If NOS hangs when starting up, it could be due to errors in
#  this file.  To help discover what is wrong, you can trace
#  the startup using the -v option in the NOS command line.
```

```
# Miscellaneous setup ****************************************
attended          on
escape            ESC             # <ESC> character
isat              yes              # 286/386 clock
multitask         on
log               /dump/session.log
watchdog          off

memory ibufsize   256
memory nibufs     1

motd  "If I'm not here, please leave a message in mailbox."

# Set up domain defaults *************************************
domain cache size 30
domain suffix     ampr.org.
domain translate  on              # display host names
domain verbose    off             # do not display suffix
domain addserver  ns9dns

# Station Identification *************************************
ip address        ns9bob
hostname          ns9bob
ax25 mycall       NS9BOB-5        # This MUST precede 'attach'

# Set up the TNC *********************************************
  attach asy      0x3f8 4 ax25 tnc0 2048 256 4800       # COM1
# attach asy      0x2f8 3 ax25 tnc0 2048 256 4800       # COM2
# attach asy      0x3e8 4 ax25 tnc0 2048 256 4800       # COM3
# attach asy      0x2e8 3 ax25 tnc0 2048 256 4800       # COM4

# trace tnc0      211

# Initialise the tnc to KISS mode ****************************
dialer tnc0 /scripts/kisson.dia

param tnc0        1 20            # TX delay     (x 10mS)
param tnc0        2 63            # Persistence  (0-255)
param tnc0        3 10            # Slot Time    (x 10mS)
param tnc0        4 10            # TX tail      (x 10mS)
param tnc0        5 0             # 0=HDX
param tnc0        dtr 1
param tnc0        rts 1

# Baycom AX.25 Packet Driver  ********************************
attach packet     0x60 tnc0 5 512

# Set up AX.25  **********************************************
ax25 bc           tnc0 on
ax25 bcinterval   840
ax25 bctext       "NS9BOB-5 TCP/IP 44.199.41.1 [London]"
```

```
ax25 digipeat     on
ax25 irtt         2500
ax25 maxframe     2
ax25 paclen       256
ax25 pthresh      128
ax25 retry        10
ax25 t3           65000
ax25 t4           300
ax25 timertype    linear
ax25 version      2
ax25 window       2048

mode tnc0         datagram

# Set up the Ethernet interface ******************************
# attach packet 0x61 en0 8 1500
# arp add ns9wrm ether 00:00:C0:11:22:33
# route add ns9wrm en0

# Set up the AXIP wormhole ***********************************
# attach axip ai0 256 ns9wrm NS9BOB-11

# Set up ifconfig ********************************************
ifconfig tnc0      broadcast 44.255.255.255
ifconfig tnc0      netmask 0xFF000000
ifconfig tnc0      description "144.625 MHz port"

# Set up TCP/IP defaults *************************************
ip ttl            10
tcp mss           216
tcp irtt          65000
tcp window        216
tcp timertype     linear

# Start network services *************************************
start             ax25
start             discard
start             echo
start             finger
start             ftp
start             pop2
start             pop3
# start           remote
start             rip
start             smtp
start             telnet
start             ttylink

# Configure NET/ROM ******************************************
# start             netrom
# attach            netrom
# netrom call       NS9BOB-6
```

```
# netrom alias      #BOB
# netrom interface  tnc0 192
# netrom nodetimer  900
# netrom obsotimer  1200
# netrom verbose    off
# netrom minquality 10
# netrom ttl        15
# netrom acktime    3000
# netrom qlimit     1024
# netrom retries    5
# netrom irtt       5000
# netrom promiscuous off
# netrom timertype  linear
# mode netrom       vc

# Broadcast your alias for local nodes - 3 times for luck
# netrom bcnodes    tnc0
# netrom bcnodes    tnc0
# netrom bcnodes    tnc0

# Set up NET/ROM filtering ************************************
# netrom nodefilter mode accept
# netrom nodefilter add NR9AAA-2 tnc0
# netrom nodefilter add NS9KEN-6 tnc0

# Set up the IP routing table for NET/ROM *********************
# route add       region47/24 netrom ns9tom
# route add       region41/24 netrom ns9ken

# Set up the ARP table for NET/ROM ****************************
# arp add         ns9tom netrom NS9TOM-6

# Set up NET/ROM routing **************************************
# netrom route add #NRA NR9AAA tnc0 192 NR9AAA
# netrom route add #TOM NS9TOM-6 tnc0 192 NR9AAA

# Set up AX.25 routing ****************************************
ax25 route add  AX9TIM AX9DGA AX9DGB
ax25 route add  NS9PAM-5 AX9DGC

# Set up AX.25 modes ******************************************
ax25 route mode AX9AAA vc

# Set up the ARP table ****************************************
arp add         nosland ax25 QST-0
arp add         ns9pam ax25 NS9PAM-5

# Set up IP routing *******************************************
route addprivate default     tnc0
route addprivate nosland     tnc0
route add        ns9liz      tnc0 ns9ken
route add        ns9jim      tnc0 ns9ken
```

```
route add         region45/24 tnc0 ns9ken
route add         ns9sue      tnc0 ns9pam

# Set up RIP ************************************************
# rip merge       no
# rip add         nosland 900 6
# rip refuse      ns9bbb
# rip refuse      ns9ccc
# rip request     nosland

# Set up RSPF ***********************************************
# rspf interface    tnc0 8 32
# rspf rrhtimer     900
# rspf suspecttimer 2000
# rspf timer        900
# rspf message      "RSPF routing in use.  RIP disabled"

# Set up the hop check (route tracer) environment ************
hop maxwait         60
hop maxttl          10
hop queries         2

# Get every NOS node in range to emit their NET/ROM broadcasts
# to load our tables.  Very heavy on bandwidth!!!
# remote -s         dummy
# remote nosland    kick

# Set up the mailbox ****************************************
third-party       on

smtp                timer 600
smtp gateway        ns9sgw
smtp usemx          on
smtp mode           route
smtp kick

mbox attend         on
mbox motd "Please use sp ns9bob to leave a message for NS9BOB"
mbox expert         off
mbox qth            "[London]"
mbox utc            0
# mbox zipcode        "123456"
mbox nrid           on
# mbox fwdinfo      "HNLNET BBS"
mbox haddress       "BB7BBS.#41.GBR.EU"
mbox password       "Maximum 30-character password."
mbox kick

# Wake up POP to start shipping any outstanding mail *********
# popmail addserver ns9ken pop3 ns9bob bob bobspasswd
# popmail trace 3
# popmail quiet no
```

```
# popmail kick ns9liz

# Set up FTP defaults ****************************************
ftype             binary
eol               standard

# Set up function keys ***************************************
source            /scripts/fkeys.scr

cd /dump/record

# THE END
```

CLEANQ.BAT

```
REM ============                                    NOSview [244]
REM N:\CLEANQ.BAT
REM ============

@ECHO OFF

IF EXIST  Q:\*.lck          DEL Q:\*.lck
IF EXIST  Q:\*.txt          DEL Q:\*.txt
IF EXIST  Q:\*.wrk          DEL Q:\*.wrk
IF EXIST  M:\pbbs_net.txt   DEL M:\pbbs_net.txt
IF EXIST  N:\tmp\tmp*.$$$   DEL N:\tmp\tmp*.$$$
```

CONFIG.SYS

```
REM ============                                    NOSview [244]
REM C:\CONFIG.SYS
REM ============

REM  All the usual CONFIG.SYS commands

LASTDRIVE=Z
SHELL=C:\DOS\COMMAND.COM /e:1024 /p
```

domain.txt

```
# ==========                                          NOSview [244]
# /domain.txt
# ==========

# ------------------------------------------------------------
# SPECIAL ADDRESSES
# ------------------------------------------------------------
ampr.ampr.org.          IN A     44.0.0.0
nosland.ampr.org.       IN A     44.199.0.0

region41.ampr.org.      IN A     44.199.41.0
region45.ampr.org.      IN A     44.199.45.0
region47.ampr.org.      IN A     44.199.47.0

loopback.ampr.org.      IN A     127.0.0.1

# ------------------------------------------------------------
# RADIO REGION 41
# ------------------------------------------------------------
ns9bob.ampr.org.        IN A     44.199.41.1

ns9ken.ampr.org.        IN A     44.199.41.2
ken.ampr.org.           IN CNAME ns9ken.ampr.org. # Nickname

ns9pam.ampr.org.        IN A     44.199.41.3

ns9mxa.ampr.org.        IN A     44.199.41.90     # Mail exchange

# An MX record.  You need 'smtp usemx on' for this to work.
ns9zzz.ampr.org.        IN MX 0  ns9mxa.ampr.org.

# SMTP Gateway for unknown mail destinations
ns9sgw.ampr.org.        IN A     44.199.41.91

# Domain Name Server
                        IN NS    ns9dns.ampr.org.
ns9dns.ampr.org.        IN A     44.199.41.99

# ------------------------------------------------------------
# RADIO REGION 45
# ------------------------------------------------------------
ns9liz.ampr.org.        IN A     44.199.45.17
ns9jim.ampr.org.        IN A     44.199.45.18
ns9sue.ampr.org.        IN A     44.199.45.19
```

```
# ------------------------------------------------------------
# RADIO REGION 47
# ------------------------------------------------------------
ns9tom.ampr.org.      IN A      44.199.47.75
ns9ben.ampr.org.      IN A      44.199.47.76

# ------------------------------------------------------------
# LOCAL AREA NETWORK
# ------------------------------------------------------------
alpha.acme.com.         IN A      192.93.94.95
beta.acme.com.          IN A      192.93.94.96
```

fkeys.lst

```
# ==================                                NOSview [244]
# /scripts/fkeys.lst
# ==================


      NORMAL          SHIFT             CTRL             ALT

F1    fkey help       KISS ON           TNC RESET        session 1
F2    session         socket            mbox             session 2
F3    kick            smtp kick         mbox kick        session 3
F4    ax25 heard      ifconfig          close            session 4
F5    arp             hop check         reset            session 5
F6    ping            route             route lookup     session 6
F7    netrom status   netrom route  netrom route info     session 7
F8    smtp list       bbs               tcp view         session 8
F9    trace 211 scrn  trace 211 file    trace 011 scrn   trace 011
F10   Session Mgr     record to file    record off       trace off
```

fkeys.scr

```
# ═══════════════                                        NOSview [244]
# /scripts/fkeys.scr
# ═══════════════

# Format:
# ------------------------------------------------------------
# fkey <key_number> [<value> | "<string>"]
# ------------------------------------------------------------
#
# Use ^ for a control character; e.g. ^M = CR

# N.B. A line containing a # character ANYWHERE on
#      the line is a comment.          ^^^^^^^^


#  f1  59 | sf1  84 | cf1   94 | af1  104 | pgup  73 |
#  f2  60 | sf2  85 | cf2   95 | af2  105 | pgdn  81 |
#  f3  61 | sf3  86 | cf3   96 | af3  106 | home  71 |
#  f4  62 | sf4  87 | cf4   97 | af4  107 | end   79 |
#  f5  63 | sf5  88 | cf5   98 | af5  108 | arup  72 |
#  f6  64 | sf6  89 | cf6   99 | af6  109 | ardn  80 |
#  f7  65 | sf7  90 | cf7  100 | af7  110 | ar l  75 |
#  f8  66 | sf8  91 | cf8  101 | af8  111 | ar r  77 |
#  f9  67 | sf9  92 | cf9  102 | af9  112 | ins   82 |
#         | sf10 93 | cf10 103 | af10 113 | del   83 |


# DELETE KEY (to abort a keyboard command with CTRL-U)
fkey 83   "^U"

# DISABLE THE ARROW KEYS
fkey 72   "^B"
fkey 75   ""
fkey 77   ""
fkey 80   ""

# NORMAL
fkey 59   "^[tail /scripts/fkeys.lst^M"
fkey 60   "^[session^M"
fkey 61   "^[kick^M"
fkey 62   "^[ax25 heard^M"
fkey 63   "^[arp^M"
fkey 64   "^[ping "
fkey 65   "^[netrom status^M"
fkey 66   "^[smtp list^M"
fkey 67   "^[trace tnc0 211^M"
```

```
# SHIFT
fkey 84   "^[dialer tnc0 /scripts/kisson.dia"
fkey 85   "^[socket^M"
fkey 86   "^[smtp kick^M"
fkey 87   "^[ifconfig^M"
fkey 88   "^[hop check "
fkey 89   "^[route^M"
fkey 90   "^[netrom route^M"
fkey 91   "^[bbs^M"
fkey 92   "^[trace tnc0 211 /dump/trace/"
fkey 93   "^[record /dump/record/"

# CONTROL
fkey 94   "^[source /scripts/tncreset.scr"
fkey 95   "^[mbox^M"
fkey 96   "^[mbox kick^M"
fkey 97   "^[close "
fkey 98   "^[reset "
fkey 99   "^[route lookup "
fkey 100  "^[netrom route info "
fkey 101  "^[tcp view^M"
fkey 102  "^[trace tnc0 011^M"
fkey 103  "^[record off^M"

# ALT
fkey 104  "^[session 1^M"
fkey 105  "^[session 2^M"
fkey 106  "^[session 3^M"
fkey 107  "^[session 4^M"
fkey 108  "^[session 5^M"
fkey 109  "^[session 6^M"
fkey 110  "^[session 7^M"
fkey 111  "^[session 8^M"
fkey 112  "^[trace tnc0 011 /dump/trace/"
fkey 113  "^[trace tnc0 0^M"
```

forward.bbs

```
# ==========                                    NOSview [244]
# /spool/forward.bbs
# ==========

BB7BBS
connect tnc0 BB7BBS
PBBS_NET
---
```

ftpusers

```
# ========                                              NOSview [244]
# /ftpusers
# ========
#
# Format:
# ------------------------------------------------------------
# <login_name> <password> <root_dir> <permissions>
# ------------------------------------------------------------
#
# N.B.      EXACTLY ONE SPACE between fields.
#
# <password> is any string of characters, without spaces/tabs.
#   An asterisk in this field indicates that any password will
#   be accepted; by convention, users then give their callsign
#   as the password.
#
# <root_dir> is the highest directory level which the user is
#   permitted to access. This must be expressed as an absolute
#   full pathname from the DOS root, but without drive letter.
#
#   N.B.  In the examples included in NOSview, the DOS root is
#         N: (because of the SUBST N: command in NOSENV.BAT).
#         This removes the risk of accidentally allowing users
#         to access directories outside the scope of NOS.
#
# <permissions>
#    ftp and telnet
#    --------------
#       1      read file
#       2      create new file
#       4      write/delete file
#
#    telnet only
#    -----------
#       8      AX.25 Gateway access
#      16      Telnet Gateway access
#      32      NET/ROM Access
#      64      Remote control
#     128      Disallow access
#
#    ppp only
#    --------
#     256      PPP connection
#     512      peer ID/password lookup
#
```

```
#    misc
#    ----
#    1024      disallow send commands (except to sysop)
#    2048      disallow read commands
#    4096      disallow third-party mail
#    8192      this station is a known BBS

# Be very careful about giving access to sensitive
# directories.  # Although you can theoretically prevent
# unauthorised access through password protection, remember
# that anyone can monitor the channel and discover user
# passwords as they are being transmitted.
#
# If a user connects to the BBS using vanilla AX.25 or NET/ROM
# (not telnet), access is granted without having to provide a
# login username or password.  In this case the user name is
# assumed to be the AX.25 callsign (without SSID).
# IF THIS CALLSIGN MATCHES A USER NAME IN FTPUSERS, THE CALLER
# GAINS THE PERMISSIONS ASSIGNED TO THAT USER.  THUS ANY USER
# NAMES THAT LOOK LIKE CALLSIGNS SHOULD HAVE A SAFE SET OF
# PERMISSIONS.
#
# Therefore if you wish to prevent vanilla AX.25 users gaining
# directory access, user names should be 7 or 8 characters
# long.

# Miscellaneous accounts requiring no password:
anonymous * /public 3
anon * /public 3
bbs * /public 3
guest * /public 3

# Special accounts:
ns9bob bobby /public 7
roberto robertspw / 127          # Login name > 6 characters
superuser supasswd /public 67    # Remote sysop permission

# Friendly visitors
ns9ken kenneth /public 7
ns9liz lizzie /public 7

# Unwanted visitors:
NS9NRD * /public/tmp 128         # Sorry, no access
```

kisson.dia

```
# ═══════════════                                    NOSview [244]
# /scripts/kisson.dia
# ═══════════════
control down
control up
speed 4800
send "*"
wait 200
send "*"
wait 200
send "*"
wait 200
send "*"
wait 1000
send "\r"
wait 1000 cmd:
send "MYCALL NS9BOB-5\r"
wait 1000 cmd:
send "MID 84\r"
wait 1000 cmd:
send "XMITOK ON\r"
wait 1000 cmd:
send "KISS ON\r"
wait 1000
```

net.rc

```
# ═══════                                            NOSview [244]
# /net.rc
# ═══════
#
# NET.RC contains hostnames, user names and passwords for
# automatic FTP logins.
#
# Format:
# ------------------------------------------------------------
# <hostname> <username> <password>
# ------------------------------------------------------------
#
# N.B.   EXACTLY ONE SPACE between each field.
#
loopback ns9bob bobby
ns9ken ns9bob mypasswd
```

NOSENV.BAT

```
REM ═══════════                                          NOSview [244]
REM N:\NOSENV.BAT
REM ═══════════

@ECHO OFF

REM  This script is CALLed from AUTOEXEC.BAT and defines the
REM  NOS environment.

REM  Make sure you have sufficent DOS environment space
REM       (e.g. use "/E:1024" in CONFIG.SYS).

REM  All NOS files are relative to the NOS root directory, N:

SUBST N: C:\nos

SUBST M: N:\spool\mail
SUBST Q: N:\spool\mqueue
SUBST R: N:\dump\record
SUBST T: N:\dump\trace
SUBST V: N:\public\nosview

PATH=%PATH%;N:\

REM  The SET HOME command below specifies the directory
REM  containing the PCELM startup file (PCELM.RC).  This
REM  definition  M-U-S-T  be relative to a DOS drive letter,
REM  not to a SUBSTituted drive letter, otherwise PCELM
REM  complains that it cannot find its startup file.

SET HOME=C:\nos
SET MAILER=N:\pcelm.exe
SET TMP=N:\tmp
SET TZ=UTC
SET USER=ns9bob

REM  Start the file-viewer
N:\VIEW

REM  Install the Baycom AX.25 driver
REM N:\AX25 -B3f8 -I4

REM  Install the Clarkson WD8003E Ethernet Adaptor Driver
REM   (if fitted).
REM N:\wd8003e 0x61 2 0x240 0xd000
```

popusers

```
# ═══════                                                    NOSview [244]
# /popusers
# ═══════

# List of known POP users

# Format:
# ------------------------------------------------------------
# <username>:<password>:
# ------------------------------------------------------------
#
# Note that both the username AND the password are terminated
# with a colon.

ns9liz:lizpasswd:
mary:poppins:
```

REMOTE.BAT

```
REM ═══════════                                               NOSview [244]
REM N:\REMOTE.BAT
REM ═══════════

REM  This DOS batch file puts NOS in an endless loop, and is
REM  intended for use in a remote location.
REM  Thus if NOS is exited for any reason, it is automatically
REM  restarted.

:loop
   STARTNOS
   GOTO loop
```

rewrite

```
# ==========                                          NOSview [244]
# /spool/rewrite
# ==========
# Read the rewrite file for lines where the first word is a
# regular expression and the second word are rewriting rules.
#
# The special character '$' followed by a digit denotes the
# string that matched a '*' character.
#
# The '*' characters are numbered from 1 to 9.
#

# Example: the line "*@*.* $2@$1.ampr.org" would rewrite the
#          address "foo@bar.xxx" to "bar@foo.ampr.org".
#

*@gb7bil pbbs_net
*@gb7* $1%gb7$2@gb7bil r
*@ns9bob.ampr.org $1
*@ns9bob.ampr $1
*@ns9bob $1
```

signatur

```
# ========                                            NOSview [244]
# /signatur
# ========

73 de Bob
NS9BOB  [44.199.41.1]
```

STARTNOS.BAT

```
REM ═══════════════                                    NOSview [244]
REM N:\STARTNOS.BAT
REM ═══════════════

@ECHO OFF

REM   Before starting NOS, the PROMPT is changed to a special
REM   flashing prompt.  This is for when you shell out of NOS;
REM   the flashing prompt reminds you that you are not in the
REM   the top-level shell.
REM   The second PROMPT statement at the end of the script
REM   restores the prompt to its original setting when you
REM   exit NOS.

PROMPT $e[5mEXIT TO RETURN TO NOS $e[0m$e[44m$p$g$e[m
SET COMSPEC=c:\command.com

REM   For DR-DOS, make unused video RAM available:
MEMMAX +v > NUL

IF EXIST Q:\*.LCK DEL Q:\*.LCK

N:
N:\NOS_20M.EXE /autoexec.nos

REM   Finished with NOS.   For DR-DOS, restore video RAM:
MEMMAX -v > NUL

N:
CD N:\

PROMPT $e[41m$p$g $e[m
SET COMSPEC=c:\command.com
```

sysop

```
# ==========                                          NOSview [244]
# /finger/sysop
# ==========
                Hello and welcome to ns9bob

        User:           bob (NS9BOB)
        Real Name:      Robert R Roberts
        Class:          Extra
        Address:        12345 Anystreet
                        Anytown, Anystate, AnyZIP
        Telephone:      (111) BOB-3456
        System Config:  PK-88
                        Yaesu FT-27RB
                        144.625 MHz

        Occupation:     Professor of Anglo-Saxon
           Hobbies:     Sheepshearing
```

tncreset.dia

```
# ==================                                  NOSview [244]
# /scripts/tncreset.dia
# ==================
control down
control up
speed 4800
send "\r"
wait 1000 cmd:
send "\r"
wait 1000 cmd:
send "RESET\r"
wait 1000
```

```
send "*"
wait 200
send "*"
wait 200
send "*"
wait 200
send "*"
wait 200
send "*"
wait 200
send "*"
wait 200
send "*"
wait 200
send "*"
wait 1000
send "\r\r"
wait 1000 cmd:
send \r"
wait 1000 cmd:
send "XMITOK OFF\r"
wait 1000
send "XMITOK OFF\r"
wait 1000
```

tncreset.scr

```
# ========================                    NOSview [244]
# /scripts/tncreset.scr
# ========================

# param tnc0 254

param tnc0 255

dialer tnc0 /scripts/tncreset.dia
```

APPENDIX 4
CHARACTER CODES

Hexadecimal Conversion Table

Dec	Hex	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	a	1010
11	b	1011
12	c	1100
13	d	1101
14	e	1110
15	f	1111

ASCII Character Set

In addition to listing the ASCII character codes in their usual form, the hexadecimal codes for ASCII digits 0-9 and upper-case letters A-Z are also expressed in shifted form; i.e. shifted one bit left. This helps when decoding callsigns in AX.25 information frames and in ARP packets.

Dec	Hex	Char		
0	00	NUL		
1	01	SOH	CTRL-A	start of header
2	02	STX	CTRL-B	start of text
3	03	ETX	CTRL-C	end of text
4	04	EOT	CTRL-D	end of transmission
5	05	ENQ	CTRL-E	enquiry (poll)
6	06	ACK	CTRL-F	acknowledge
7	07	BEL	CTRL-G	bell
8	08	BS	CTRL-H	backspace
9	09	HT	CTRL-I	horizontal tab
10	0a	LF	CTRL-J	line feed
11	0b	VT	CTRL-K	vertical tab
12	0c	FF	CTRL-L	form feed
13	0d	CR	CTRL-M	carriage return
14	0e	SO	CTRL-N	shift out
15	0f	SI	CTRL-O	shift in
16	10	DLE	CTRL-P	data link escape
17	11	DC1/XON	CTRL-Q	device control 1/XON
18	12	DC2	CTRL-R	device control 2
19	13	DC3/XOFF	CTRL-S	device control 3/XOFF
20	14	DC4	CTRL-T	device control 4
21	15	NAK	CTRL-U	negative acknowledge
22	16	SYN	CTRL-V	synchronous idle
23	17	ETB	CTRL-W	end of transmission block
24	18	CAN	CTRL-X	cancel
25	19	EM	CTRL-Y	end of medium
26	1a	SUB	CTRL-Z	substitute
27	1b	ESC	CTRL-[	escape
28	1c	FS	CTRL-\	file separator
29	1d	GS	CTRL-]	group separator
30	1e	RS	CTRL-^	record separator
31	1f	US	CTRL-_	unit separator

Dec	Hex	Char	
32	20	SPACE	(shifted:40/41)
33	21	!	
34	22	"	
35	23	#	
36	24	$	
37	25	%	
38	26	&	
39	27	'	apostrophe
40	28	(	
41	29	)	
42	2a	*	
43	2b	+	
44	2c	,	comma
45	2d	-	minus
46	2e	.	
47	2f	/	
48	30	0	(shifted: 60/61)
49	31	1	(shifted: 62/63)
50	32	2	(shifted: 64/65)
51	33	3	(shifted: 66/67)
52	34	4	(shifted: 68/69)
53	35	5	(shifted: 6a/6b)
54	36	6	(shifted: 6c/6d)
55	37	7	(shifted: 6e/6f)
56	38	8	(shifted: 70/71)
57	39	9	(shifted: 72/73)
58	3a	:	
59	3b	;	
60	3c	<	
61	3d	=	
62	3e	>	
63	3f	?	
64	40	@	
65	41	A	(shifted: 82/83)
66	42	B	(shifted: 84/85)
67	43	C	(shifted: 86/87)
68	44	D	(shifted: 88/89)
69	45	E	(shifted: 8a/8b)
70	46	F	(shifted: 8c/8d)
71	47	G	(shifted: 8e/8f)
72	48	H	(shifted: 90/91)
73	49	I	(shifted: 92/93)
74	4a	J	(shifted: 94/95)
75	4b	K	(shifted: 96/97)
76	4c	L	(shifted: 98/99)
77	4d	M	(shifted: 9a/9b)
78	4e	N	(shifted: 9c/9d)
79	4f	O	(shifted: 9e/9f)
80	50	P	(shifted: a0/a1)
81	51	Q	(shifted: a2/a3)
82	52	R	(shifted: a4/a5)
83	53	S	(shifted: a6/a7)
84	54	T	(shifted: a8/a9)
85	55	U	(shifted: aa/ab)
86	56	V	(shifted: ac/ad)
87	57	W	(shifted: ae/af)
88	58	X	(shifted: b0/b1)
89	59	Y	(shifted: b2/b3)
90	5a	Z	(shifted: b4/b5)
91	5b	[	
92	5c	\	
93	5d	]	
94	5e	^	
95	5f	_	underscore
96	60	`	grave(back quote)
97	61	a	
98	62	b	
99	63	c	
100	64	d	
101	65	e	
102	66	f	
103	67	g	
104	68	h	
105	69	i	
106	6a	j	
107	6b	k	
108	6c	l	
109	6d	m	
110	6e	n	
111	6f	o	
112	70	p	
113	71	q	
114	72	r	
115	73	s	
116	74	t	
117	75	u	
118	76	v	
119	77	w	
120	78	x	
121	79	y	
122	7a	z	
123	7b	{	
124	7c	|	
125	7d	}	
126	7e	~	
127	7f	DEL	delete

Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
128	80	160	a0	192	c0	224	e0
129	81	161	a1	193	c1	225	e1
130	82	162	a2	194	c2	226	e2
131	83	163	a3	195	c3	227	e3
132	84	164	a4	196	c4	228	e4
133	85	165	a5	197	c5	229	e5
134	86	166	a6	198	c6	230	e6
135	87	167	a7	199	c7	231	e7
136	88	168	a8	200	c8	232	e8
137	89	169	a9	201	c9	233	e9
138	8a	170	aa	202	ca	234	ea
139	8b	171	ab	203	cb	235	eb
140	8c	172	ac	204	cc	236	ec
141	8d	173	ad	205	cd	237	ed
142	8e	174	ae	206	ce	238	ee
143	8f	175	af	207	cf	239	ef
144	90	176	b0	208	d0	240	f0
145	91	177	b1	209	d1	241	f1
146	92	178	b2	210	d2	242	f2
147	93	179	b3	211	d3	243	f3
148	94	180	b4	212	d4	244	f4
149	95	181	b5	213	d5	245	f5
150	96	182	b6	214	d6	246	f6
151	97	183	b7	215	d7	247	f7
152	98	184	b8	216	d8	248	f8
153	99	185	b9	217	d9	249	f9
154	9a	186	ba	218	da	250	fa
155	9b	187	bb	219	db	251	fb
156	9c	188	bc	220	dc	252	fc
157	9d	189	bd	221	dd	253	fd
158	9e	190	be	222	de	254	fe
159	9f	191	bf	223	df	255	ff

APPENDIX 5
AMPRnet IP ADDRESS COORDINATORS

The global coordinator for IP addresses for the AMPRnet (44.x.x.x) is Brian Kantor, WB6CYT. Brian can be reached on email (brian@ucsd.edu) or via the AX.25 PBBS network (WB6CYT @ WB6YMH.CA.USA).

In addition, there are separate address coordinators for the major conurbations in the United States, plus country coordinators for the rest of the world. A more-or-less complete list of coordinators throughout the world is included in Tables A5-1 and A5-2 below.

To get your own IP address, you should contact the coordinator closest to you. If you happen to live in a country which is not included in the list, you could try putting out a PBBS bulletin enquiring if there is a coordinator yet. If you get no reply, you can then contact Brian Kantor direct for help.

In the United Kingdom, IP address allocation is handled on a regional basis. The regional coordinators are listed in Table A5-3.

Table A5-1: United States IP Address Coordinators

44.002.x.x	Bob Meyer	K6RTV	Calif: Sacramento
44.004.x.x	Douglas Thom	N6OYU	Calif: Silicon Valley - San Francisco
44.006.x.x	Don Jacob	WB5EKU	Calif: Santa Barbara/ Ventura
44.008.x.x	Brian Kantor	WB6CYT	Calif: San Diego
44.010.x.x	Terry Neal	AA6TN	Calif: Orange County
44.012.x.x	Steven King	KD7RO	Eastern Washington,Idaho
44.014.x.x	John Shalamskas	KJ9U	Hawaii & Pacific Islands
44.016.x.x	Jeff Angus	WA6FWI	Calif: Los Angeles S F Valley
44.017.x.x	Dana Myers	KK6JQ	Calif: Antelope Valley/ Kern County
44.018.x.x	Geoffrey Joy	KE6QH	Calif: San Bernardino and Riverside
44.020.x.x	Fred Schneider	K0YUM	Colorado: Northeast
44.022.x.x	John Stannard	KL7JL	Alaska
44.024.x.x	Dennis Goodwin	KB7DZ	Washington state: Western (Puget Sound)
44.026.x.x	Ron Henderson	WA7TAS	Oregon
44.028.x.x	Don Adkins	KD5QN	Texas: Dallas
44.030.x.x	J Gary Bender	WS5N	New Mexico
44.032.x.x	Bdale Garbee	N3EUA	Colorado: Southeast
44.034.x.x	Jeff Pierce	WD4NMQ	Tennesee
44.036.x.x	Doug Drye	KD4NC	Georgia
44.038.x.x	Mike Abbott	N4QXV	South Carolina
44.040.x.x	Jeff Jacobsen	WA7MBL	Utah
44.042.x.x	Phil Akers	WA4DDE	Mississippi
44.044.x.x	Ed Thorne	WB1FEM	Massachusetts: western
44.046.x.x	William Simmons	WB0ROT	Missouri
44.048.x.x	Jacques Kubley	KA9FJS	Indiana
44.050.x.x	Ron Breitwisch	KC0OX	Iowa
44.052.x.x	Gary Grebus	K8LT	New Hampshire
44.054.x.x	Ralph Stetson	KD1R	Vermont
44.056.x.x	Don Hughes	KA1MF	Eastern Mass
44.058.x.x	Rich Clemens	KB8AOB	West Virginia
44.060.x.x	Howard Leadmon	WB3FFV	Maryland
44.062.x.x	Jim Dearras	WA4ONG	Virginia (not DC)
44.064.x.x	Dave Trulli	NN2Z	New Jersey: northern
44.065.x.x	Bob Applegate	WA2ZZX	New Jersey: southern
44.066.x.x	John DeGood	NU3E	Delaware
44.068.x.x	Bob Foxworth	K2EUH	New York: Long Island
44.069.x.x	Paul Gerwitz	WA2WPI	New York: upstate
44.070.x.x	Gary Sanders	N8EMR	Ohio
44.072.x.x	Ken Stritzel	WA9AEK	Chicago - North Ill.

Table A5-1: United States IP Address Coordinators (continued)

44.074.x.x	James Curran	KA4OJN	North Carolina (east)
44.075.x.x	Charles Layno	WB4WOR	North Carolina (west)
44.076.x.x	Kurt Freiberger	WB5BBW	Texas: south
44.077.x.x	Rod Huckabay	KA5EJX	Texas: west
44.078.x.x	Joe Buswell	K5JB	Oklahoma
44.080.x.x	John Gayman	WA3WBU	Pennsylvania: eastern
44.082.x.x	Steven Elwood	N7GXP	Montana
44.084.x.x	Bob Ludtke	K9MWM	Colorado: Western
44.086.x.x	Reid Fletcher	WB7CJO	Wyoming
44.088.x.x	Jon Bloom	KE3Z	Connecticut
44.090.x.x	Mike Nickolaus	NF0N	Nebraska
44.092.x.x	Pat Davis	KD9UU	Wisconsin, upper peninsula Michigan
44.094.x.x	Gary Sharp	WD0HEB	Minnesota
44.096.x.x	Don Bennett	K4NGC	District of Columbia
44.098.x.x	Garry Paxinos	(waiting)	Florida
44.100.x.x	Ken Adkisson	WB4FAY	Alabama
44.102.x.x	Jeff King	WB8WKA	Michigan (lwr peninsula)
44.104.x.x	Charles Greene	W1CG	Rhode Island
44.106.x.x	Tyler Barnett	N4TY	Kentucky
44.108.x.x	James Dugal	N5KNX	Louisiana
44.110.x.x	Richard Duncan	WD5B	Arkansas
44.112.x.x	Bob Hoffman	N3CVL	Pennsylvania: western
44.114.x.x	Steven Elwood	N7GXP	N&S Dakota
44.116.x.x	Tom Kloos	WS7S	Oregon: NW&Portland, Vancouver WA
44.118.x.x	Jon Andrews	WA2YVL	Maine
44.120.x.x	unassigned		
44.122.x.x	Dale Puckett	K0HYD	Kansas
44.124.x.x	David Dodell	WB7TPY	Arizona
44.125.x.x	Earl Petersen	KF7TI	Southern Nevada
44.126.x.x	Karl Wagner	KP4QG	Puerto Rico

44.128.x.x is reserved for testing. Do not use for operational networks.

Table A5-2: International IP Address Coordinators

44.129.x.x	Japan	JG1SLY	Tak Kushida, JH3XCU Joly Kanbayashi
44.130.x.x	Germany	DL4TA	Ralf D Kloth
44.131.x.x	United Kingdom	G6PWY	Chris Baron (G6PWY@GB7PWY)
44.132.x.x	Indonesia	YB1BG	Robby Soebiakto
44.133.x.x	Spain	EA4DQX	Jose Antonio Garcia Madrid (EA4DQX @ EA4DQX)
44.134.x.x	Italy	I2KFX	
44.135.x.x	Canada	VE3GYQ	David Toth
44.136.x.x	Australia	VK2ZXQ	John Tanner
44.137.x.x	Holland	PA0GRI	Gerard Van Der Grinten
44.138.x.x	Israel	4X6OJ	Ofer Lapid
44.139.x.x	Finland	OH1MQK	Matti Aarnio
44.140.x.x	Sweden	SM0IES	Lennart
44.141.x.x	Norway	LA4JL	Per Eotang
44.142.x.x	Switzerland	HB9CAT	Marco Zollinger
44.143.x.x	Austria	OE1KDA	Krzysztof Dabrowski
44.144.x.x	Belgium	ON7LE	
44.145.x.x	Denmark	OZ6QI	Karsten
44.146.x.x	Phillipines	DU1UJ	Eddie Manolo
44.147.x.x	New Zealand		
44.148.x.x	Ecuador	HC5K	Ted
44.149.x.x	Hong Kong	VS6EL	
44.150.x.x	Yugoslavia	YU3FK	Iztok Saje
44.151.x.x	France	FC1BQP	Pierre-Francois Monet
44.152.x.x	Venezuela	OA4KO/YV5	Luis Suarez
44.153.x.x	Argentina	LU7ABF	Pedro Converso
44.154.x.x	Greece	SV1IW	Manos
44.155.x.x	Ireland	EI9GL	Paul Healy
44.156.x.x	Hungary	HA5DI	Markus Bela
44.157.x.x	Chile	CE6EZB	Raul Burgos
44.158.x.x	Portugal	CT1DIA	Artur Gomes
44.159.x.x	Thailand	HS1JC	Kunchit Charmaraman
44.160.x.x	South Africa	ZS6BHD	John
44.161.x.x	Luxembourg	LX1YZ	Erny Tontlinger
44.162.x.x	Cyprus	5B4TX	C. Costis
44.163.x.x	Central America	TI3DJT	Chuck Hast
44.164.x.x	Surinam	PZ2AC	Otto Morroy
44.165.x.x	Poland	SP5WCA	Andrzej K. Brandt
44.166.x.x	Korea	HL9TG	Gary ?
44.167.x.x	India	VU2LBW	Lakshman ("Lucky") Bijanki
44.168.x.x	Taiwan	BV5AF	Bolon
44.193.x.x	Outer Space-AMSAT	W3IWI	Tom Clark
44.199.x.x	Nosland	G3NRW	Ian Wade ☺

Table A5-3: United Kingdom Regional IP Address Coordinators

44.131.1.x	Dave	G4TUP @ GB7BPL	Lancs, Cheshire, Merseyside, Cumbria, Gtr Manchester, IOM
44.131.2.x	Paul	G8NRY @ GB7NRY	N Humberside, N/S/W Yorks
44.131.3.x	Terry	G1FNQ @ GB7SAM	Hereford & Worcs, Staffs, Shropshire, Warks, W Midlands
44.131.4.x	Jonathan	G4KLX @ GB7HMZ	S Humberside, Derbys, Leics, Lincs, Notts
44.131.5.x	Ian	G3NRW @ GB7BIL	Beds, Cambs, Northants
44.131.6.x	Paul	G1PLT @ GB3XP	Berks, Bucks, Oxon
44.131.7.x	Bob	G8GGI @ GB3XP	Gtr London South, Surrey
44.131.8.x	Nick	G8ZHR @ GB3XP	Kent, E/W Sussex
44.131.9.x	Alan	G3FKN @ GB7PLY	Devon, Cornwall
44.131.10.x	Alan	GW4HDR @ GB7OAR	Dyfed, Gwent, Powys, Mid/S/W Glamorgan
44.131.11.x	Alan	GW4HDR @ GB7OAR	Clwyd, Gwynedd
44.131.12.x	Robin	GM4YED @ GB7EDN	Grampian, Highlands, Orkneys, Shetlands, Tayside, W Isles
44.131.13.x	Robin	GM4YED @ GB7EDN	Borders, Fife, Lothian
44.131.14.x	Susan	GM4SGB @ GB7SAN	Central, Dumfries & Galloway, Strathclyde
44.131.15.x	Dave	GI0PEZ @ GB7TED	Northern Ireland
44.131.16.x	Dave	G8KBB @ GB7MXM	Essex, Norfolk, Suffolk
44.131.17.x	Dave	G4WPT @ GB7BNM	Dorset, Hants, Wilts
44.131.18.x	Harry	G6AUC @ GB7NCL	Cleveland, Durham, Northumberland, Tyne & Wear
44.131.19.x	Andrew	G8FSL @ GB7HSN	Gtr London North, Herts
44.131.20.x	Mike	G6DHU @ GB7WRW	Avon, Gloucs, Somerset
44.131.21.x	Dave	G4WPT @ GB7BNM	IOW, Channel Is

APPENDIX 6
REFERENCES

This Appendix lists some worthwhile reference material for further study.

"**Internetworking with TCP/IP**"
by Douglas Comer
Published by Prentice-Hall
ISBN 0-13-470188-7

(The classic theoretical reference work on TCP/IP protocols).

"**TCP/IP Network Administration**"
by Craig Hunt
Published by O'Reilly and Associates
ISBN 0-937175-82-X

(A new and excellent book on the practical aspects of setting up, managing and troubleshooting a TCP/IP network. Good coverage of routing, domain name system and mail handling. Highly recommended).

"**Your Gateway to Packet Radio**"
by Stan Horzepa
Published by ARRL
ISBN 0-87259-26-34
(The bible for AX.25 "native mode" packet radio).

"**Computer Networking Conference Proceedings**"
Edited and published by ARRL
225 Main Street, Newington, CT 06111, USA

	11th Proceedings:	Teaneck, New Jersey (Nov 1992)
	10th Proceedings:	San Jose, California (Sep 1991)
	9th Proceedings:	London, Ontario (Sep 1990)
	8th Proceedings:	Colorado Springs (Oct 1989)
	7th Proceedings:	Columbia, Maryland (Oct 1988)
*	6th Proceedings:	Redondo Beach, CA (Aug 1987)
	5th Proceedings:	Orlando, Florida (Mar 1986)
	4th Proceedings:	San Francisco (Mar 1985)
	3rd Proceedings:	Trenton, New Jersey (Apr 1984)
	2nd Proceedings:	San Francisco (Mar 1983)
	1st Proceedings:	Gaithersburg, Maryland (Oct 1981)

(Major contributions from the leaders in packet radio).

* (This set of proceedings contains the paper on the KISS protocol by Mike Chepponis, K3MC and Phil Karn, KA9Q).

"**QEX: The ARRL Experimenters' Exchange**"
Edited by Jon Bloom, KE3Z.
Published monthly by ARRL (address as above)

(Incorporating the "Digital Communications" column by Harold Price, NK6K, QEX provides detailed technical articles on all aspects of radio communication, including packet).

"**Packet International — π**"
Edited by Ian Wade, G3NRW.
Published quarterly by BARTG. Details from Ian Brothwell, G4EAN, 56 Arnot Hill Road, Arnold, Nottingham NG5 6LQ, United Kingdom.

(Detailed technical articles and news of the packet radio scene).

INDEX

The following conventions apply in this index:

- Internet Protocol Names and DOS environment variables are in Times New Roman upper-case; e.g. TCP, TZ
- DOS commands are in Arial bold upper-case; e.g. **UUENCODE**
- NOS commands are in Arial bold lower-case; e.g. **status**
- NOS command parameters are in Arial italic; e.g. *status*
- NOS filenames and directory names are in Times New Roman italic lower-case; e.g. */domain.txt*
- DOS filenames and directory names are in Times New Roman italic upper-case; e.g. *AUTOEXEC.BAT*.

@ (see NOS BBS commands)
? (see **help** command, **NOS BBS commands**)
! (see **shell** command)
(see Comments)
% (see Intermediate_mailhost, **NOS BBS commands**)
π (see Packet International)

3c500 parameter (see **attach** command)
7-layer Reference Model: 17,49

A

Abbreviated Commands: 14,100
Abbreviations: 11
abort command: 110,111,140,297
Aborting an **ftp** command: 140
accept parameter (see **netrom** command)
access parameter (see **ip** command)
Access permissions: 126-127
Acknowledgements: 4
acktime parameter (see **netrom** command)
Acronyms: 11

add parameter (see **arp**, **ax25 route**, **netrom route**, **route** commands)
addprivate parameter (see **route** command)
address parameter (see **ip** command)
Address Coordinators: 335-339
Addressing Mail: 188
addserver parameter (see **domain**, **nntp**, **popmail** commands)
Alias: 88,149,202,204-209,217,263,311
alias parameter (see **netrom** command)
/alias (see Files)
Amiga: 295
AMPRnet: 12,19,335-339
Anon account: 123-127
Anonymous account: 123-127
ANSI Escape Sequences: 96
ANSI.SYS (see Files)
Apple Macintosh: 295
Application Layer: 50
Archimedes: 295
Area command (see **NOS BBS commands**)
Area (mailbox): 149-150
areas (see Files)
ARP, Address Resolution Protocol: 12,28, 235-238,281-282
arp command: 28,111,113,235,297-298
  add: 28,235,259-261,297
  drop: 235,297
  flush: 238,298
  publish: 238,298
ARP Hardware Code: 272
ARRL: 341-342
  Digital Communications Committee: 342
  QEX: 342
ASCII Character Set: 332-334
ascii subcommand (see **ftp** command)
asy parameter (see **attach** command)
asystat command: 111,298
Atari: 295
attach command: 52,112,298
  3c500: 298
  asy: 52,95-96,118-119,298
  axip: 298
  drsi: 298
  eagle: 298
  hapn: 298
  hs: 298
  kiss: 298
  netrom: 253,298
  packet: 298
  pc100: 298
  pi: 298
  scc: 298
  slfp: 298
attend parameter (see **mbox** command)
attended command: 109,116,298
ATTRIB command
AUTOEXEC.BAT (see Files)
/autoexec.nos (see Files)
Autobaud: 99
Automated FTP login: 137
autoroute command: 111,298
AWLEN command: 46,73
AX.25, Amateur X.25 Link Layer Protocol: 12
ax25 command: 29,298-299
  bc: 274,298
  bcinterval: 274,298
  bckick: 274,299
  bctext: 274,299
  blimit: 299
  dest: 299
  digipeat: 299
  filter: 299
  flush: 299
  heard: 272,299
  hearddest: 299
  irtt: 299
  kick: 299
  maxframe: 299
  mycall: 118,236,299
  paclen: 299
  pthresh: 299
  reset: 299
  retry: 299
  route: 29,111,233,299
  route add: 29,231-232,299
  route drop: 233,299
  route mode: 234,299
  status: 299
  t3: 299
  t4: 299
  timertype: 299
  version: 299
  window: 299
AX.25 Mail Forwarding (see PBBS Mail Forwarding)
AX.25 Routing: 231-234
AXIP, AX.25-over-IP protocol: 12
axip parameter (see **attach** command)

B

-*b* Startup Option: 97
Backslash (\): 14
batch parameter (see **smtp** command)
Batch files
Battery-backed RAM: 71
Baudrate: 119
BAYCOM AX.25 driver: 30,37
BBS (see NOS BBS)
BBS account: 123-127
bbs command: 70,110,112,145,161,299-300
BBS Commands (see **NOS BBS Commands**)
BBS Login: 143-146,161-162
bc parameter (see **ax25** command)
bcinterval parameter (see **ax25** command)
bckick parameter (see **ax25** command)
bcnodes parameter (see **netrom** command)
bctext parameter (see **ax25** command)
Beacons (AX.25): 274
binary subcommand (see **ftp** command)
Binary file transfers
  BBS: 167,168,175-177
  FTP: 138-139
blimit parameter (see **ax25** command)
BM, Bdale's Messy Mailer: 16,25
BOOTP: 12
bootp command: 113,300
  defaultfile: 300
  dns: 300
  dyip: 300
  homedir: 300
  host: 300
  logfile: 300
  logscreen: 300
  rmhost: 300
  start: 300
  stop: 300
bootpd command: 113,300
broadcast parameter (see **ifconfig** command)
BTIO flags (see **trace** command)
Bulletin: 15
Bulletin ID (BID): 228-229
Bye command (see **NOS BBS commands**)

C

Cache (domain): 289
cache clean parameter (see **domain** command)
cache list parameter (see **domain** command)
cache size parameter (see **domain** command)
cache wait parameter (see **domain** command)
call parameter (see **netrom** command)
Callsign: 14,15
Canonical Name (CNAME): (see Resource Records)
cd command: 106,110,300
cd subcommand (see **ftp** command)
CD-ROM: 126
Character Codes: 331-334
CHAT (see TTYLINK)
chat command: 112
Chat command (see **NOS BBS commands**)
check parameter (see **hop** command)
choketime parameter (see **netrom** command)
circular parameter (see **mem** command)
Clarkson Drivers: 30,53,296
CLEANQ.BAT (see Files)
Client: 63-70
Client/Server: 63-70
Clockwork View (see VIEW)
close command: 110,300
cls command: 109,300
Codes: 331-334
COM ports (see Serial port)
comm command: 53,113,300
Command ground rules: 13-14,115
Command interpreter: 38
CONFIG.SYS (see Files)
CONMODE command: 46
Connect command (see **NOS BBS commands**)
connect command: 52,110,146,232-233, 272-274,300
connect parameter (see **netrom** command)
Control Code (KISS): 43-44,271
Control Files: 311-330
Conventions: 13
Crynwr Packet Drivers: 296
CTS, Clear to Send: 72-73
CWID Interval (see MID command)

D

-*d* startup option: 97,98
Daemon: 64

DARPA, Defense Advanced Research Projects Agency: 23
Datagram: 234
Data Link Layer: 49,52-53
debug parameter (see **mem** command)
DEC VAX/VMS: 295
defaultfile parameter (see **bootp** command)
dele subcommand (see **ftp** command)
delete command: 110,300
derate parameter (see **netrom** command)
description parameter (see **ifconfig** command)
dest parameter (see **ax25** command)
Destination: 16,17
detach command: 112,300
dialer command: 52,53,113,300-301
digipeat parameter (see **ax25** command)
Digipeater routing: 29,231-234
dir command: 106,110,301
dir subcommand (see **ftp** command)
directory parameter (see **nntp** command)
Directories: 83-85
  /dump: 83
  /dump/record: 83,173
  /dump/trace: 83
  /finger: 83
  /public: 84
  /public/masters: 85
  /public/nosdocs: 85
  /public/nosview: 85
  /scripts: 85
  /spool: 85
  /spool/help: 85,164
  /spool/mail: 85,148,149,157,158,191
  /spool/mqueue: 85,147,191
  /spool/news: 85
  /spool/rqueue: 85,157,158
  /spool/signatur: 85
  /tmp: 85
disconnect command: 110,273,301
Distribution lists (see Alias)
dns parameter (see **bootp** command)
Domain cache: 289-290
Domain Name System (DNS): 28,61,287-291
DNS queries: 287-291
domain command: 113,117,301
  addserver: 117,287,301
  cache clean: 117,289-290,301
  cache list: 289,301
  cache size: 291,301
  cache wait: 290,301
  dropserver: 291,301
  list: 291,301
  maxwait: 291,301
  query: 301
  remote add: 301
  remote drop: 301
  remote list: 301
  remote retry: 301
  remote trace: 301
  retry: 301
  suffix: 58,117,301
  trace: 112,287,301
  translate: 117,301
  verbose: 117,118,301
  xyzzy: 301
Domain suffix: 58
/domain.txt (see Files)
Dotted-decimal notation: 55
Download command (see **NOS BBS commands**)
Download UUencode command (see **NOS BBS commands**)
Drivers
  AX.25 (Baycom): 30,37
  Clarkson: 30,53,296
  Ethernet: 314
drop parameter (see **arp**, **ax25 route**, **netrom route**, **route** commands)
dropserver parameter (see **domain**, **nntp**, **popmail** commands)
DRSI PCPA 8530 Card: 30
drsi parameter (see **attach** command)
drsistat command: 111,301
DTR, Data Terminal Ready: 119
dump command: 109,111,301
/dump (see Directories)
/dump/record (see Directories)
/dump/trace (see Directories)
/dump/record/.** (see Files)
/dump/session.log (see Files)
/dump/trace/.** (see Files)
dyip parameter (see **bootp** command)

E

Eagle 8530 card: 30
eagle parameter (see **attach** command)
eaglestat command: 111,301
echo command: 111,301
echo parameter (see **icmp** command)
efficient parameter (see **mem** command)
ELM Mailer: 16,25

Email (see Electronic mail)
Encap interface: 121
encapsulation parameter (see **ifconfig** command)
End Delay: 119
End Node (see NET/ROM)
Environment Variables: 81,87
eol command: 111,301
ESC key: 103,104,109
Escape command (see **NOS BBS commands**)
escape command: 104,109,116,301
Escape to DOS: 102
Escape to Session Manager: 103
Ethernet: 30
etherstat command: 111,301
EXIT command: 96,102
exit command: 109,301
expert parameter (see **mbox** command)
External Mailer: 16,145

F

F10 (Escape to Session Mgr): 103,104,109
FEND/FESC/TFEND/TFESC (KISS): 43-44
File Access Permissions: 123-128
Files: 83-94
DOS:
C:\AUTOEXEC.BAT: 76,86,312
C:\CONFIG.SYS: 76,86,317
C:\DOS\ANSI.SYS: 76,86
N:\AX25.COM: 86
N:\CLEANQ.BAT: 86,317
N:\NOSxxxxx.EXE: 86
N:\NOSENV.BAT: 76,81,86,265,325
N:\PCELM.EXE: 87
N:\PCELM.MSG: 87
N:\PCELM.RC: 87,265
N:\REMOTE.BAT: 87,116-117,326
N:\STARTNOS.BAT: 87,96-98,328
N:\UUDECODE.EXE: 88
N:\UUENCODE.EXE: 87
N:\VIEW.COM: 88
N:\VIEW.HLP: 88
N:\WD8003E.COM: 88
NOS:
/alias: 88,149,202,204-209,217,263,311
/autoexec.nos: 88-90,97,115-122, 155-160,264,312-317
/domain.txt: 28,55-62,90,263,318-319
/ftpusers: 91,123-128,161,263,322-323
/hosts.net: 90
/net.rc: 91,137,263,324
/netrom.sav: 91
/popusers: 91,151,211-212,263,326
/seqf: 91
/signatur: 91,264,327
/dump/record/.**: 91
/dump/session.log: 91,215,283
/dump/trace/.**: 91,268-269
/finger/sysop: 92,122,264,329
/public/masters/.**: 92
/public/nosdocs/.**: 92
/public/nosview/.** : 92
/scripts/fkeys.lst: 92,105,319
/scripts/fkeys.scr: 92,104,320-321
/scripts/kisson.dia: 92,98-100,118,264,324
/scripts/tncreset.dia: 92,106-107,329-330
/scripts/tncreset.scr: 92,106-107,330
/spool/areas: 92,150,155,164-165,265,311
/spool/forward.bbs: 92,151,217,219,222, 227-228,265,321
/spool/history: 92,147,148,228-229
/spool/mail.log: 93
/spool/rewrite: 93,147,148,151, 199-204,209,213,217-226,265,327
/spool/help/.hlp*: 93,164
/spool/mail/.txt*: 93,149,165,191,211
/spool/mqueue/.lck*: 93,209-210
/spool/mqueue/sequence.seq: 93,191
/spool/mqueue/.txt*: 93,191
/spool/mqueue/.wrk*: 93,191,209
/spool/news/.**: 93
/spool/rqueue/.txt*: 93
/spool/rqueue/.wrk*: 93
/spool/signatur/.sig*: 93,265
/tmp/.**: 93
filter parameter (see **ax25** command)
Filtering NET/ROM Broadcasts: 258-259
FINGER protocol: 12
Finger command (see **NOS BBS commands**)
finger command: 112,121,302
/finger (see Directories)
/finger/sysop (see Files)
fkey command: 104-105,109,302
fkeys.lst (see Files)
fkeys.scr (see Files)
flow subcommand (see **ftp** command)
Flow Control: 73
flush parameter (see **arp**, **ax25**, **route** commands)

Forward command (see **NOS BBS commands**)
forward parameter (see **ifconfig** command)
forward.bbs (see Files)
Forward Slash (/): 14
Forwarding
IP: 239-249
NET/ROM: 251-262
SMTP: 148,185-209
PBBS: 21,151-153
POP: 151,152
Frame End (see FEND)
Frame Escape (see FESC)
free parameter (see **mem** command)
FTP, File Transfer Protocol: 12
ftp command: 24,111,112,129-140,302
ascii: 24,302
batch: 302
binary: 24,302
cd: 133,302
dele: 136,302
dir: 133,302
flow: 133,302
get: 24,133,302
hash: 133,302
list: 302
ls: 133 ,302
mget: 135,302
mkdir: 302
mput: 302
nlst: 302
pass: 132,302
put: 24,133,138,302
pwd: 133,302
quit: 133,302
rmdir: 302
type: 138,302
user:132,302
verbose: 302
FTP Inc Packet Interface: 53
/ftpusers (see Files)
ftype command: 111,129,303
Full duplex: 119
Function Key Mapping: 104-105
fwdinfo parameter (see **mbox** command)

G

garbage parameter (see **mem** command)
Gateway: 17
Gateway command (see **NOS BBS commands**)
gateway parameter (see **smtp** command)
Gateway Server (NOS BBS)
get subcommand (see **ftp** command)
Graphics memory: 75,96-97
GRINOS, 295
groups parameter (see **nntp** command)
Guest account: 123-127

H

haddress parameter (see **mbox** command)
Half duplex: 119
hapn parameter (see **attach** command)
HAPN 8273 adapter: 30
hapnstat command: 111,303
Hardware Checkout: 71-73
hash subcommand (see **ftp** command)
Heard (see **NOS BBS commands**)
heard parameter (see **ax25** command)
hearddest parameter (see **ax25** command)
Help command (see **NOS BBS commands**)
help command: 100-101,109,303
help/.hlp* (see Files)
Hexadecimal Conversion Table: 331
HID command: 46
Hierarchical Addressing: 224-226
history (see Files)
HOME environment variable: 81,87
homedir parameter (see **bootp** command)
hop command: 27,111,279,282-285,303
check: 27,282-285,303
maxttl: 282,303
maxwait: 283,303
queries: 283,303
trace: 112,283,303
HOST command: 46
host parameter (see **bootp** command)
Host Mode (see TNC)
Hostid: 118
Hostname: 14,113,118
hostname command: 113,118,303
/hosts.net (see Files)
How to get NOSview: 9
HPOLL command: 46
HP-UX: 296
hs command: 303
hs parameter (see **attach** command)

I

IBM PC (see PC)
ICMP, Internet Control Message Protocol: 12,283-285
icmp command: 111,303
  echo: 303
  status: 303
  trace: 112,303
ifbufsize parameter (see **mem** command)
ifconfig command: 111,112,120-121,122,303
  broadcast: 121,303
  description: 120,155,156,170,303
  encapsulation: 120,303
  forward: 303
  ip_address: 120,303
  linkaddress: 121,303
  mtu: 120,303
  netmask: 121,303
  rxbuf: 121,303
Image mode: 129
IN A, IN CNAME, IN MX, IN NS (see Resource Records)
Incoming Bulletins: 203-204,206
Info command (see **NOS BBS commands**)
info command: 101,109,303
info parameter (see **netrom route** command)
Interactive UNIX: 295
Interface: 7,30
interface parameter (see **netrom** command)
Intermediate mailhost (%): 188
Internet Protocols: 23
Internet Stack: 49
Internetwork Layer: 50
I/O Address: 95
IP, Internet Protocol: 12,23
ip command: 111,303
  access: 303
  address: 118,236,303
  rtimer: 303
  status: 111,303
  ttl: 303
ip_address parameter (see **ifconfig** command)
IP Address: 27
IP Address Coordinators: 335-339
IP Gateway: 18,241-248
IP routing/forwarding: 239-249
IRQ: 95
irtt parameter (see **ax25, netrom, tcp** commands)
isat command: 109,116,303
ISO, International Standards Organisation: 17

J

JNOS: 11,295
jumpstart parameter (see **mbox** command)
Justheard command (see **NOS BBS commands**)

K

KA9Q (Phil Karn) 4, 11
Keyboard characters: 15
Keyboard mapping: 104,310
kick command: 110,112,140,303
kick parameter (see **ax25, mbox, netrom, nntp, pop, smtp, tcp** commands)
Kill command (see **NOS BBS commands**)
kill parameter (see **smtp** command)
KISS, "Keep It Simple, Stupid!" protocol: 12,31,43-45,342
kiss parameter (see **attach** command)
KISS command: 46,99
KISS Control Codes: 44,271
Kiss Mode (see TNC)
kisson.dia (see Files)

L

Label: 7
LASTDRIVE: 76
Link Address: 28
linkaddress parameter (see **ifconfig** command)
Link Layer (see Data Link Layer)
List command (see **NOS BBS commands**)
list parameter (see **domain, popmail, smtp** commands)
list subcommand (see **ftp** command)
Listener (see Client/Server)
listservers parameter (see **nntp** command)
load parameter (see **netrom** command)
Localhost (see loopback)
lock command: 109,304
Lock files: 97
log command: 109,116,304
logfile parameter (see **bootp** command)
Login Name: 124,181

logscreen parameter (see **bootp** command)
lookup parameter (see **route** command)
Loopback, Loopback address: 53,70,121,137
loopback (domain.txt): 55-56
ls subcommand (see **ftp** command)
lzw command: 110,304

M

Macintosh: 295
mail (see Files)
Mail: 15,142
mail command: 110,145,304
mailbox parameter (see **pop** command)
Mailbox password: 181-184
Mailbox server: 141
Mail Exchanger: 193-197
Mail Gateways: 186-187
MAILER environment variable: 81,87,145
Mailer: 15,25,141-142
Mail forwarding: 26,148,185-229
mailhost parameter (see **pop** command)
mail.log (see Files)
mail/.txt* (see Files)
maxclients parameter (see **smtp** command)
maxframe parameter (see **ax25** command)
maxmsg parameter (see **mbox** command)
maxttl parameter (see **hop** command)
maxwait parameter (see **domain, hop** commands)
mbox command: 110,111,304
attend: 159,304
expert: 159,304
fwdinfo: 159,304
haddress: 159,222,304
jumpstart: 304
kick: 152,153,159,220,224,304
maxmsg: 159,304
motd: 159,304
nrid: 159,304
operator: 304
password: 159-160,304
qth: 159,222,304
secure: 159,160,304
smtptoo: 159,223,304
status: 304
timer: 152,153,219,304
tiptimeout: 304
trace: 304
utc: 159,222-223,304
zipcode: 159,223,304
Mbxusers command (see **NOS BBS commands**)
mem command: 109,111,304
circular: 304
debug: 304
efficient: 304
free: 304
garbage: 304
ifbufsize: 304
minheap: 304
nibufs: 304
sizes: 304
status: 304
thresh: 304
MEMMAX: 75,96
Message: 15
mget subcommand (see **ftp** command)
MID command: 99
minheap parameter (see **mem** command)
minquality parameter (see **netrom** command)
mkdir command: 110,304
mkdir subcommand (see **ftp** command)
mode command: 112,234,253,304
mode parameter (see **ax25 route, smtp** commands)
Modem Control: 30,72-73
Monitoring FTP transfers (#): 134-135
more command: 102-105,110,304
Morse Identification (MID): 99,100
motd command: 110,117,305
motd parameter (see **mbox** command)
mput subcommand (see **ftp** command)
mqueue (see Files)
mqueue/.lck* (see Files)
mqueue/.txt* (see Files)
mqueue/.wrk* (see Files)
mss parameter (see **tcp** command)
mtu parameter (see **ifconfig** command)
multitask command: 109,116
MX, Mail Exchanger: 34,35,193-197
mxlookup parameter (see **smtp** command)
MYCALL command: 99
mycall parameter (see **ax25** command)

N

Native Mode (see TNC)
NET: 11,295
netmask parameter (see **ifconfig** command)
/net.rc (see Files)
NET/ROM: 11,12,21,251-262,275-278

NET/ROM Alias: 15,256,258,275-276
Netrom command (see **NOS BBS commands**)
netrom parameter (see **attach** command)
netrom command: 111,305
acktime: 254,305
alias: 256,258,305
bcnodes: 253,254,276,305
call: 256,258,261,305
connect: 262,275,278,305
choketime: 254,305
derate: 254,305
interface: 256,258,305
irtt: 254,255,305
kick: 305
load: 277,305
minquality: 254,255,258,305
nodefilter: 258-259,305
nodetimer: 254,255,305
obsotimer: 254,255,305
promiscuous: 258,305
qlimit: 254,255,305
reset: 305
retries: 254,255,305
route: 111,275,305
route add: 259-262,305
route drop: 305
route info: 278,305
save: 277,305
status: 278,305
timertype: 254,255,305
ttl: 254,256,305
user: 305
verbose: 252,253,305
window: 254,256,305
/netrom.sav (see Files)
Network Connectivity Services: 27
Network Layer: 49,50
Network Mask (see **ifconfig** command)
nibufs parameter (see **mem** command)
Nickname (see CNAME, Resource Records)
nlst subcommand (see **ftp** command)
NNTP, Network News Transfer Protocol: 12
nntp command: 111,112,305
addserver: 305
directory: 305
dropserver: 305
groups: 305
kick: 305
listservers: 305
quiet: 306
trace: 112,306
nodefilter parameter (see **netrom** command)
Nodes command (see **NOS BBS commands**)
nodetimer parameter (see **netrom** command)
NOS: 11,22
Command Set Summary: 109-113
Directory Tree: 78
File Tree: 79,84
Mailer: 141,162-166
Protocols: 51
Startup: 96-99
Where to get NOS: 295-296
NOS BBS: 16,25,141-184
File Server: 141-142,166-168,173-179
Finger Server: 141-143,171
Gateway Server: 141-143,168-170
Mailbox Server: 141-142,162-166
Remote Sysop Server: 141-143,171,181-184
Setup: 155-160
NOS BBS Commands: 161-171
?: 162
Area: 150,164-165
Bye: 163
Chat: 162
Connect: 168-169
Download: 166,174-175
Download UUencoded (DU): 167, 175-177
Escape: 163
Finger: 171
Gateway: 162
Help: 163
Info: 164
Justheard: 168
Kill: 165
List: 166
Mbxusers: 164
Netrom: 168
Nodes: 168
Operator: 158,163,168
Ports: 156,168,170
Read: 166
Send: 166
Send Forward: 166
Send Reply: 166
Telnet: 170
Upload: 168,177-178
Verbose: 166
What:168,173-174
Xpert: 164
Zap: 168,179
@ Remote Sysop: 171

NOSENV.BAT (see Files)
NOSgas, NOS Get-Away Special: 9
NOSland: 33
NOS root: 14,77,83
NOSview: 7,76
NOSVIEW.ZIP, NOSVW244.ZIP: 10
nrid parameter (see **mbox** command)
NRS, NET/ROM Serial Protocol: 12,31
nrstat command: 111,306

O

obsotimer parameter (see **netrom** command)
operator parameter (see **mbox** command)
Optimising *domain.txt*: 60
Optimising DOS: 75-76
$origin (domain.txt): 60,62
Origin: 16,17
OS/2: 296
OSI, Open Systems Interconnection: 17,49
OSI Router: 18
OSI stack: 50
Other Versions of NOS: 10,295-296

P

PA0GRI, Gerard van der Grinten: 11,295
PAC/COM PC100 card: 30
packet parameter (see **attach** command)
Packet International: 342
paclen parameter (see **ax25** command)
PAD, Packet Assembler/Disassembler: 37,38
param command: 47,106,111,113,119,306
PARITY command: 46
pass subcommand (see **ftp** command)
password parameter (see **mbox** command)
Password
  bbs: 161-162
  ftpusers: 123-125
  mailbox: 181-184
  popmail: 211-214
PBBS Mail Forwarding: 21,26,217-229
PBBS Reverse Forwarding: 21,220
pc100 parameter (see **attach** command)
PCElm Mailer: 16,25,144-145
PCELM.MSG (see Files)
PCELM.RC (see Files)
PCPA 8530 card: 30
Persistence: 119
Physical Layer: 49-52
pi parameter (see **attach** command)
PID, Protocol Identifier: 54,271
PING, Packet Internet Groper protocol: 12
ping command: 27,112,279-282,306
Platforms: 295-296
PMNOS, Presentation Mgr NOS (OS/2): 296
PMS, Personal Messaging System: 16,39
POP, POP2, POP3 protocols: 12
pop command: 110,112,306
  kick: 306
  mailbox: 306
  mailhost: 306
  quiet: 306
  timer: 306
  userdata: 306
popmail command: 110,112,307
  addserver: 213,307
  dropserver: 214,307
  kick: 211,214,307
  list: 214,307
  quiet: 214,307
  trace: 214-215,307
POP Mail Forwarding: 26,151,152,211-215
/popusers (see Files)
Port, port number: 64-66, 68-70
Ports command (see **NOS BBS commands**)
Poste Restante (see POP Mail Forwarding)
PPERSIST command: 46
PPP, Point-to-Point Protocol: 12,31
ppp command: 113,307
Preference value: 59,194-195
PROMPT symbol: 96,98
promiscuous parameter (see **netrom** command)
Protocol Stacks: 49-50
ps command: 109,111,307
pthresh parameter (see **ax25** command)
/public/masters (see Directories)
/public/nosdocs (see Directories)
/public/nosview (see Directories)
publish parameter (see **arp** command)
put subcommand (see **ftp** command)
pwd command: 106,110,307
pwd subcommand (see **ftp** command)

Q

QEX: 342
qlimit parameter (see **netrom** command)

qth parameter (see **mbox** command)
queries parameter (see **hop** command)
query parameter (see **domain** command)
quiet parameter (see **nntp**, **pop**, **popmail**, **smtp** commands)
quit subcommand (see **ftp** command)

R

Radio callsign (see Callsign)
rarp command: 112,307
RAWHDLC command: 46
RD, Receive Data: 72
Read command (see **NOS BBS commands**)
Read-only files: 127-128
reclzw parameter (see **smtp** command)
record command: 109,175-176,307
record/.** (see Files)
References: 341-342
remote command: 112,308
remote add parameter (see **domain** command)
remote drop parameter (see **domain** command)
remote list parameter (see **domain** command)
Remote login: 24
remote retry parameter (see **domain** command)
Remote Session Manager: 181-184
Remote Sysop command (see **NOS BBS commands**)
remote trace parameter (see **domain** command)
REMOTE.BAT (see Files)
rename command: 110,308
Repeat last command (**^B**): 105
Reply command (see **NOS BBS commands**)
reset command: 104,108,110,112,273,308
reset parameter (see **ax25**, **netrom**, **tcp** command)
Resource Records: 55-62
  Address (A): 56-58
  Canonical Name (CNAME): 58
  Mail Exchange (MX): 59,194-195
  Name Server (NS): 60-61
retries parameter (see **netrom** command)
retry parameter (see **ax25**, **domain** commands)
Reverse forwarding (AX.25): 21,152,153
rewrite (see Files)
RIP, Routing Information Protocol: 13, 30
rip command: 111,112,308
RLOGIN, Remote Login protocol: 13
rlogin command: 24,112,308
rmdir command: 110,308
rmdir subcommand (see **ftp** command)
rmhost parameter (see **bootp** command)
route parameter (see **ax25**, **netrom** commands)
route command: 111,240-247,308
  add: 240-247,259,260-261,308
  addprivate: 245,308
  drop: 245,308
  flush: 308
  lookup: 308
route add parameter (see **netrom** command)
route drop parameter (see **netrom** command)
route info parameter (see **netrom** command)
Routing
  AX.25: 29,231-234
  NET/ROM: 29,251-262
  IP: 29,239-249
  Mask: 245-247
  TheNet X1G: 248-249
  Updates: 29
  Wormhole: 30
Routing Header (PBBS R:): 222-223
rqueue (see Directories)
rqueue/.txt* (see Files)
rqueue/.wrk* (see Files)
RS-232 cable: 72-73
RSPF, Radio Shortest Path First protocol: 13,30
rspf command: 111,308
rtimer parameter (see **ip** command)
RTS, Request to Send: 72-73,119
rtt parameter (see **tcp** command)
rxbuf parameter (see **ifconfig** command)

S

-s Startup Option: 97
save parameter (see **netrom** command)
scc parameter (see **attach** command)
sccstat command: 111,308
SCO: 296
/scripts/fkeys.lst (see Files)
/scripts/fkeys.scr (see Files)
/scripts/kisson.dia (see Files)
/scripts/tncreset.dia (see Files)

/scripts/tncreset.scr (see Files)
secure parameter (see **mbox** command)
Send command (see **NOS BBS commands**)
Send Forward command (see **NOS BBS commands**)
Send Reply command (see **NOS BBS commands**)
sendlzw parameter (see **smtp** command)
/seqf (see Files)
sequence.seq (see Files)
Serial port: 30,95
Session
  escape to the Session Manager: 103
  starting a session: 102-103
  terminating a session: 104
session command: 103-105,110,111,308
Session Layer: 50
session.log (see Files)
Session Manager: 31,32,54,95-108
SET command: 81,145
SHELL variable: 76
shell command: 102,109,308
Shell Escape (see Escape to DOS)
Shifted ASCII Character Codes: 332,333
/signatur (see Files)
sizes parameter (see **mem** command)
skick command: 110,309
Slash conventions: 14
SLIP, Serial Link Internet Protocol: 13,31
SLFP, Serial Link Frame Protocol: 13
slfp parameter (see **attach** command)
Slottime: 119
SMTP, Simple Mail Transfer Protocol: 13
smtp command: 110,112,309
  batch: 309
  gateway: 158,309
  kick: 148,149,158,192,309
  kill: 210,309
  list: 209,309
  maxclients: 309
  mode: 157,158,309
  mxlookup: 309
  quiet: 309
  reclzw: 309
  sendlzw: 309
  timer: 148,149,158,192,309
  trace: 112,309
  usemx: 158,309
SMTP Client/Server: 190
SMTP Gateway: 196-197
SMTP Mail Forwarding: 26,148-149,185-209
SMTP/PBBS mail gateways: 186-187,218
smtptoo parameter (see **mbox** command)
Socket: 66-67, 97
socket command: 111,309
Software Installation: 75-81
Source: 16,17
source command: 104,109,309
/spool/areas (see Files)
/spool/forward.bbs (see Files)
/spool/history (see Files)
/spool/mail.log (see Files)
/spool/rewrite (see Files)
/spool/help/.hlp* (see Files)
/spool/mail/.txt* (see Files)
/spool/mqueue/.lck* (see Files)
/spool/mqueue/sequence.seq (see Files)
/spool/mqueue/.txt* (see Files)
/spool/mqueue/.wrk* (see Files)
/spool/news/.** (see Files)
/spool/rqueue/.txt* (see Files)
/spool/rqueue/.wrk* (see Files)
/spool/signatur/.sig* (see Files)
Stack (see Protocol Stacks)
START command: 46,73
start parameter (see **bootp** command)
start command: 110,112,156,212,253,309
Starting NOS: 96-98
STARTNOS.BAT (see Files)
Startup Options: 97-98
Station Identification: 14,118
status command: 102,109,309
status parameter (see **ax25**, **icmp**, **ip**, **mbox**, **mem**, **netrom**, **tcp** commands)
STOP command: 46,73
stop parameter (see **bootp** command)
stop command: 110,112,309
SUBST command: 14,77,80-81,86
suffix parameter (see **domain** command)
Superuser account: 123,181
Switch (see NET/ROM)
syndata parameter (see **tcp** command)
sysop permission: 181
sysop (see Files)

T

t3 parameter (see **ax25** command)
t4 parameter (see **ax25** command)
tail command: 105,110,309
Target: 16,17
Target mailhost: 189

TBAUD command: 73
TCP, Transmission Control Protocol: 13,23
tcp command,309
irtt: 309
kick: 309
mss: 309
reset: 309
rtt: 309
status: 111,309
syndata: 309
timertype: 309
trace: 112,309
view: 309
window: 309
TD, Transmit Data: 72
TELNET protocol: 13
Telnet command (see **NOS BBS commands**)
Telnet client: 64-65
telnet command: 24,64-70,110,112,146,309
Telnet server: 64-68
telnet 25 command (SMTP): 112, 207-208,309
telnet 87 command (CHAT/TTYLINK): 69,112,309
Terminal Node Controller (see tnc)
test command: 109,310
TheNet: 11: 248-249
TheNet X1G: 248-249
third-party command: 110,156,310
thresh parameter (see **mem** command)
timer parameter (see **mbox, pop, smtp** commands)
timertype parameter (see **ax25, netrom, tcp** commands)
TIP, Terminal Interchange Protocol: 13
tip command: 53,107-108,112,113,310
tiptimeout parameter (see **mbox** command)
TMP environment variable: 81,87
TNC, Terminal Node Controller: 16,37,71
Control Codes (param): 44,119
Host mode: 16,38,40-42
KISS mode: 16,38,42-43
Native mode: 16,38-40,73,107-108
Reset: 46-47,106-107
Switching to KISS mode: 45-46,98-100
Switching to native mode: 46-47,106-107
tncreset.dia (see Files)
tncreset.scr (see Files)
ttl parameter (see **ip, netrom** commands)
TTYLINK (Chat) protocol: 13,24
ttylink command: 24,112,310
TRACE command: 46
trace command: 52,109,267-272,310
trace parameter (see **domain, hop, icmp, mbox, nntp, popmail, smtp, tcp** commands)
trace/.** (see Files)
traceroute command (see **hop** command)
translate parameter (see **domain** command)
Transport Layer: 50
Transpose Frame End (TFEND): 44
Transpose Frame Escape (TFESC): 44
TXDELAY command: 119
TXMUTE command: 119
TXTAIL command: 119
Type byte (KISS): 44
type subcommand (see **ftp** command)
TZ environment variable: 81,87

U

ucsd.edu: 9
UDP, User Datagram Protocol: 13
UDP Port 33434 (hop): 283-285
udp command: 111,310
UNIX: 295,296
Unsupervised Operation: 184
Upload command (see **NOS BBS commands**)
upload command: 110,178,310
Uploading Files (NOS BBS): 178-179
usemx parameter (see **smtp** command)
USER environment variable: 81,87
user parameter (see **netrom** command)
user subcommand (see **ftp** command)
userdata parameter (see **pop** command)
utc parameter (see **mbox** command)
UUDECODE command: 13,138-139,177
UUENCODE comamnd: 13,138-139,179
UUencoded Download command (see **NOS BBS commands**)

V

-v Startup Option: 97,265
VAX: 295
Verbose command (see -v, **NOS BBS commands**)
verbose parameter (see **domain, netrom** commands)

verbose subcommand (see **ftp** command)
version parameter (see **ax25** command)
VIDRAM, Video RAM: 75,96-97
VIEW 9,80
view parameter (see **tcp** command)
Virtual Circuit (vc): 234
VMS: 295

W

WAMPES: 295
watch command: 109,310
watchdog command: 109,116,184,310
Well-known Port Numbers: 68
What command (see **NOS BBS commands**)
window parameter (see **ax25**, **netrom**, **tcp** commands)
WNOS: 11,295
Wormhole Routing: 30,31

X

X1G (see TheNet)
XENIX: 296
XFLOW command: 46, 73
XMITOK command: 99,106,107
XOFF command: 46,73
XON command: 46, 73
Xpert command (see **NOS BBS commands**)
xyzzy parameter (see **domain** command)

Z

Zap command (see **NOS BBS commands**)
zipcode parameter (see **mbox** command)