Getting Started with TCP/IP on Packet Radio

by John Ackermann, AG9V
Miami Valley FM Association
Dayton, Ohio

18 January 2023

Introduction

This document is intended to help hams with some experience in packet radio get started with the TCP/IP software written by KA9Q and others. It is not intended to take the place of the software's reference manual, but rather to provide a quick-and-dirty introduction to the capabilities of TCP/IP and the mysteries of installing and using the software.

There are several different versions of the KA9Q software floating around. It was originally written for MS-DOS computers, but has been ported to Macintosh, Amiga, Atari and UNIX systems. The original program was called "**NET**" and its last formal version was issued in April, 1989. If someone talks about "**890421.1 NET**," that's what they're referring to.

Since 1989, work has concentrated on a rewritten program called "**NOS**" (for **N**etwork **O**perating **S**ystem, though confusingly the executable program for PCs is usually still called "**NET.EXE**"). NOS offers many new features that make using TCP/IP much more effective; you should use it instead of NET. However, NOS is a growing and changing creature; since there are several different versions, and they are being updated rapidly, I can't tell you precisely where to find the latest, greatest version. Your best bet is to check with a local user. If that doesn't work, there are several telephone BBS systems that carry various flavors of NOS:

N8EMR's Ham BBS	(614) 895-2553
ChowdaNet	(401) 331-0334
WB3FFV	(301) 335-0858

The version I'm using, and which is reflected in this document, is PA0GRI's adaptation of NOS version 061891, as modified and distributed by N1BEE and available as "**GRINOS**" from the ChowdaNet BBS. I try not to dwell on features that are specific to this version, but if something I say doesn't seem to match your software, that's probably why.

A last note before plunging in -- I said it before, and I'll say it again: this document barely scratches the surface of NOS. Nearly every command described here has options or parameters that I'm ignoring. My goal is to give you a feel for what TCP/IP does, and to get you on the air with NOS; to get beyond the novice stage you need to look at the reference manual and experiment with the software. Appendix A includes a list of organizations and individuals that can provide further information about TCP/IP and amateur radio.

TCP/IP and Ham Radio

TCP/IP is a set of communication protocols that have become a standard in the computer networking world. It is designed to link different kinds of computer systems together over dissimilar networks. TCP/IP software runs on nearly every kind of computer available, from IBM mainframes to PCs, Macs, Amigas, and Ataris. The KA9Q software (from now on, I'll call it "**NOS**") is special because it includes the features necessary to run TCP/IP over ham packet radio.

The TCP/IP protocol suite allows different kinds of computers to talk to one another across networks. The services it provides include terminal sessions, file transfer, electronic mail, and data routing services. Computers running TCP/IP (referred to as "**hosts**") can run some or all of these applications simultaneously; it's entirely possible to sit at a PC computer running NOS and carry on a keyboard-to-keyboard chat with one station, while another retrieves a file from your hard disk and you send electronic mail to a third.

It's also comforting to know that when you run TCP/IP, you don't give up the ability to carry on normal packet communications. You can use NOS just like a terminal program to establish connections with your local BBS or to chat with friends who don't run NOS (yet).

If you've looked at the size of the NOS documentation, you're probably asking yourself what the benefit is of mastering this fairly complex stuff. Well, NOS has several features that improve on regular packet radio. It has much more sophisticated file transfer and electronic mail capabilities than our present PBBS systems (and it's possible to feed PBBS messages into NOS in a way that makes it much easier to use them). It supports multiple simultaneous connections. It has new and better transport methods that improve the reliability and throughput of slow and congested channels.

NOS also has the ability to route transmissions to distant stations without the user needing to know every hop along the way; all you need to do is get your data to a "**gateway**" station that knows how to move it one hop closer to its destination. New work being done with NOS promises dynamic routing that automatically adjusts to changes in the network.

And, since it is directly adapted from the de facto standard system of interconnecting computers, NOS offers the possibility of sophisticated services far beyond anything available on regular packet radio. For example, in some areas ham TCP/IP users can log into multi-user UNIX computer systems and run applications as if they were directly connected to those machines.

What is TCP/IP?

As mentioned above, TCP/IP is actually a set of protocols for the transfer of data across networks of computers. Two of these protocols underlie most of the others, and they give the set its popular name:

TCP **T**ransport **C**ontrol **P**rotocol, a "reliable stream service" (which is a fancy way of saying it makes sure that all the data sent to a remote host actually gets there), and

IP **I**nternet **P**rotocol, which sets the basic rules for formatting packets of data to go out over a network. TCP rides on top of IP.

Now that you finally know what "**TCP/IP**" stands for, there are a few concepts that are critical because they address a basic problem in any communications system -- identifying the parties to the conversation. Simply using our ham callsigns to address TCP/IP packets doesn't work for two reasons. First, the protocols work across many different networks, and have to have a consistent address scheme. Second, and as important, ham callsigns don't contain enough information to allow TCP/IP's sophisticated routing mechanisms to work.

Names and Addresses

The first important concept is the "**IP Address.**" Since these protocols are used on lots of different computers, it is necessary to use an addressing system that works with all of them, that provides adequate routing information, and that doesn't take up a lot of space. The answer is to build addresses out of a four byte sequence of integers, with each byte providing information about the network and subnetwork(s) to which a host belongs.

IP addresses are "**hierarchical**" because the four bytes have decreasing significance from left to right. By looking at the leftmost byte(s) we can learn how to route a transmission to the host represented by the rightmost byte(s). We usually print these addresses using the numeric value of each byte, separated by a period, such as **[44.70.12.34]**. This is known as "**dotted notation**." The square brackets aren't strictly necessary, but they are convenient to set off IP addresses; I'll use them that way in this document.

I won't go into all the semantics of hierarchical addressing here, but as an example the address [44.70.12.34] breaks down as:

44. The network assigned to amateur radio TCP/IP.
70. The subnetwork for Ohio.
12. The Dayton/Cincinnati subnetwork.
34 A specific system address within that area.

IP addresses are assigned by coordinators who derive their authority from a central registry. The coordinator for the ham radio net is Brian Kantor, WB6CYT. He has delegated authority to assign addresses to various state and national coordinators. The folks in Appendix A can help you find your local coordinator.

The second important concept is the "**hostname**." Obviously, IP addresses aren't very intuitive. English-like hostnames make remembering addresses much easier, and TCP/IP programs, including NOS, have means (discussed below) to map between IP addresses and hostnames. A "**host**" is any computer running TCP/IP; even when you're using services from another computer, your system is still a host. When we talk about a "**remote host**," we're talking about a machine that you're communicating with via TCP/IP.

The convention in ham radio TCP/IP is to use your callsign as your hostname. To help reduce confusion, we usually print hostnames in lower case, and callsigns in capital letters -- my hostname is "**ag9v**," and my call is "**AG9V**" (though NOS isn't case sensitive and won't care if you don't do it this way).

Closely related to the hostname is the "**domain name**." A "**domain**" is a group of machines that are logically (though not necessarily physically) connected together. Domain names are like IP addresses; periods separate parts of the name, with each part representing a different level in the domain hierarchy. But the domain name is ordered in reverse -- its highest-level portion is at the right, the opposite of IP addresses.

The ham network's domain is "**ampr.org**"; "**org**" (short for "organizations") is the top level domain, and "**ampr**" (for AMateur Packet Radio) is the second level domain, containing all ham TCP/IP hosts.

When you combine a hostname with a domain name, you get something like "**ag9v.ampr.org**." This is called a "**Fully Qualified Domain Name**" ("**FQDN**" -- knowing this acronym allows you to sound like a real expert). If a host has multiple users, we can add the user's login name at the beginning of the address, separated from the FQDN by a "@" character. This combination is commonly known as an "**Internet address**" (the "**Internet**" is the general term for all the TCP/IP hosts that are interconnected) and is the address form used for most electronic mail in the real world. For example, if there is a user

"jra" at ag9v, "**jra@ag9v.ampr.org**" would be that user's full Internet address.

There's one last twist. Some services (such as Domain Name Service, discussed below) need to know whether an address they are processing is in fact an FQDN. To do so, they look for a trailing period at the end of the domain name. Some versions of NOS ignore this issue, but the PA0GRI versions (such as GRINOS) insist that you "anchor" all domain names with a period at the end of the name. In other words, GRINOS will barf if you issue the command "hostname ag9v.ampr.org" but "hostname ag9v.ampr.org." will make it happy.

This may seem like an overly complicated scheme to simply allow two hams to talk to each other, but we use it because the ham radio TCP/IP network can be tied to the worldwide TCP/IP network in a number of different ways, and using the full set of TCP/IP address conventions makes it possible for traffic to flow between the ham network and the real world.

Leaving aside legal issues about third-party traffic, there's no reason, for example, why electronic mail can't be automatically routed through a "**gateway**" (a computer that interconnects two or more networks) between a ham TCP/IP user and a non-ham who has access to the Internet. In fact, this service already exists in some areas.

The good news is that for traffic within the ham network, we only need to worry about hostnames, and NOS's "**domain suffix**" command will take care of adding the "ampr.org" extension for us; we only need to deal with the full details of addressing if we want to go outside the ham radio network.

TCP/IP Services

Now that we have those boring basics out of the way, the protocols that use TCP/IP to provide real, useful services include:

TELNET The terminal emulation program. In "real" networks, telnet lets a user at one host remotely access a remote host, just as if he was on a terminal directly connected to that computer. In NOS, the telnet function usually connects you to a remote host's mailbox, which acts very much like a personal PBBS. The NOS telnet command does allow you to remotely login to a host that supports that function; in some areas UNIX computers connected to the ham TCP/IP network provide that service.

FTP **F**ile **T**ransfer **P**rotocol. A means of transferring both ASCII (text) and binary (program, data, or compressed) files between hosts.

SMTP **S**imple **M**ail **T**ransfer **P**rotocol. A (mostly) invisible way of moving electronic mail from one host to another. If you create a message on your computer (using the BM program, discussed below), SMTP will automatically attempt to transfer it to the destination computer.

POP **P**ost **O**ffice **P**rotocol. SMTP is neat, but it's really designed to work with hosts that are available full time. Most ham TCP/IP stations aren't. POP is designed for them; it allows incoming mail to be stored at a host that acts as a "**mail server**;" when you come on the air, your system automatically asks the server to send you your mail.

PING **P**acket **I**nter**N**et **G**roper. A diagnostic that sends a packet to a specified host; if the host is accessible to you and on the air, it responds with another packet. PING tells you how long the round trip took.

FINGER A way of finding out information about the users at a host. The finger command can simply list all the users at a host, or spit out information (like the "brag tape" of RTTY days) about a specific user.

ARP **A**ddress **R**esolution **P**rotocol. IP addresses need to be matched with the correct hardware address (in our case, ham callsign) to allow packets to be sent to their destination -- NOS doesn't know what callsign goes with a given IP address. ARP does this by sending out a broadcast message when it needs to know the callsign that matches an address. The remote host (if it's on the air) will answer and provide its hardware address.

DNS **D**omain **N**ame **S**ervice. Remembering IP addresses isn't easy. NOS can use a file called "**DOMAIN.TXT**" to contain mappings between hostnames and IP addresses, but that means you need to know the hostname and address of any station you want to contact. Alternatively, a remote host may agree to serve as a "**domain name server**" that NOS can query when it needs to know the address of a host. Not all areas have a name server available to the ham community, but in those that do, life is a lot easier.

Installing NOS

Frankly, there's no completely painless way to get NOS running on your computer. NOS is somewhat picky about the directories used for its files, and there are a number of custom parameters that you must set to teach the program about your environment and your network. Those parameters are contained in a configuration file that most versions of NOS call "**AUTOEXEC.NET**" (PA0GRI versions use "**AUTOEXEC.NOS**;" our references to "AUTOEXEC.NET" mean whichever name is appropriate**)**.

Files and Directories

You should create the following directories on your disk (NOS can work from either a hard disk or a floppy; it's getting big enough, though, that working from a 360K floppy can be tough):

\spool	(holds NOS' working files)
\spool\help	(help files for the mbox)
\spool\mail	(mail messages go here)
\spool\mqueue	(mail workfiles)
\spool\rqueue	(incoming mail workfiles)
\finger	(home for finger info files)
\public	(file uploads/downloads)

These files need to go in the root directory of your default disk (it is possible to configure NOS to look for these files in other than the root directory; see the reference manual for details):

AUTOEXEC.NET	(the NOS configuration file))
FTPUSERS	(user ftp/mbox access)
DOMAIN.TXT	(hostnames if not using DNS)
BM.RC	(mail program configuration)
ALIAS	(used by smtp and BM)

NOS uses two executable files. These can be installed anywhere on your file path:

NET.EXE, **NOS.EXE**, **or GRINOS.EXE**	(main executable file)
BM.EXE	(mailer program)

Setting up AUTOEXEC.NET

Once the directories are created and the files copied, you need to edit the AUTOEXEC.NET file with a text editor to customize it. A sample file is included as Appendix B. Some of the things you'll have to put in the file are:

Your hostname (usually your callsign in lower case):

hostname ag9v.ampr.org.

Your IP address:

IP address [44.70.12.34]

Your callsign (optionally including an SSID; local customs vary on this):

ax25 mycall AG9V

"attach" commands to tell NOS how to talk to your hardware. These can get quite hairy; Appendix C has the details. For a TNC on COM 1 at 4800 baud serial port speed, use:

attach asy 0x3f8 4 ax25 ax0 1024 256 4800

The "**ax0**" in the middle of the command is the "**interface**" name -- you use it to identify this port to NOS when you set up routing commands and the like. You can use any (short) name you'd like, but the convention for COM ports is to use ax0, ax1, etc.

At least one routing command. NOS needs to know where to send packets. A default route that sends all packets out the ax0 interface is:

route add default ax0

If you have a gateway for packets going outside the local area, include a route like:

route add [44.70.13.0]/24 ax0 ag9v

This command would route packets addressed to any host with "44.70.13" as the first three bytes of its address out the ax0 interface to ag9v, which presumably knows how to get these packets to their destination. The "/24" means that the first 24 bits (three bytes) of the address are significant; NOS will ignore the last byte when making routing decisions.

If you have a domain name server, add a command near the beginning of your configuration file identifying its IP address:

domain addserver [44.70.12.34]

If you have a local mail server that knows how to route messages outside the area (see the discussion of electronic mail, below), add a command identifying it:

smtp gateway [44.70.12.34]

Storing Name/Address Matches in DOMAIN.TXT

If you don't have a local domain name server (**DNS**), you'll need to create "**DOMAIN.TXT**" in the root directory, with one entry for every hostname you want to communicate with. Appendix D shows how to set up this file. If you don't have an entry for a host in the file (or the name server doesn't know about it), you can use the IP address instead of the hostname in NOS commands.

If you're using DNS, NOS will save the hostname/address matches it gets from the server in **DOMAIN.TXT**, so you'll find that file existing (and growing) even if you didn't create it.

Giving the Finger

If you want users to be able to learn about your station with the finger command, you need to create a text file in the **\finger** directory called **<hostname>.txt** (by the way, when we use angle brackets like this, it means this is a value you'll need to insert yourself -- minus the angles -- based on your own configuration). You can use any ASCII text editor to create the file; it should contain basic info about your system. Don't go overboard... one screen of text is plenty.

You can also create additional files with information about specific aspects of your system. For example, you might have a list of the files available for downloading on your system in a finger file called "filelist.txt." A remote host who issues the command "**finger filelist@<myhost>**" will get that list.

Some Boring but Necessary Technical Stuff

Before we move on to the good stuff about how to make NOS do magic, we need to talk about three related commands that you may need to tweak depending on local custom and the quality of the RF paths you're using. Just as regular AX.25 uses the "**paclen**" command to limit the size of packets, TCP/IP has parameters defining how much data is moved in one chunk. In theory, the larger the **datagram** (TCP/IP's term for a single block of data), the higher the efficiency, because the protocol headers add a fixed amount of overhead; in

larger datagrams the overhead is a smaller percentage of the total data sent.

However, some networks (such as NetRom) can't handle large datagrams in one piece. More importantly, the larger the datagram, the longer it takes to transmit, and on a busy or flaky path, the greater the likelihood that something will corrupt it along the way. And, it takes longer to resend a large packet than a small one, so the cost of retries is greater. Because of these factors, a fast network with clear channels and solid paths can get away with sending much larger datagrams than a slow, unreliable one.

NOS provides three parameters that deal with datagram sizes. The most important one is the "**mtu"** (the sixth value in the "**attach asy**" command described above). It is similar to **paclen**; it sets the largest packet, including any headers, that can be sent on an interface. Datagrams larger than the mtu are **fragmented** into multiple pieces, which seriously reduces efficiency. Each interface has its own mtu, set as part of its attach command.

For 1200 baud channels that are shared with other packet users, an mtu value of **256** is reasonable; in fact, since that is the largest packet size most non-TCP/IP ham networks (like digipeaters and NetRom) are designed to handle, 256 is the largest mtu you should specify if any of your packets are going to travel via such a node.

Faster networks may use higher values. For good-quality channels with fast data rates (9600 baud or above), it may be reasonable to use an mtu ranging from 512 to 1500 (which matches the standard mtu used by ethernet systems).

The other two parameters that set datagram size are part of the TCP protocol. "**tcp mss**" (maximum segment size) is the largest chunk of data that TCP will send in a single frame. Because the TCP and IP headers attached to each datagram total 40 bytes, mss should be 40 bytes smaller than mtu; **216** is the correct value for an mtu of 256.

The "**tcp window**" parameter tells NOS how many datagrams it can have outstanding at once -- if it is twice the value of mss, NOS can receive two datagrams before sending an acknowledgment. This parameter is analogous to the "**maxframe**" parameter in AX.25. A large window improves efficiency because, among other things, multiple acknowledgments can be sent in a single packet.

Although using a large window has major benefits on full duplex networks, on typical ham networks best performance comes from smaller windows ranging from one to three times the mss. A good

starting point is to set the window equal to twice the value of mss (**432** for an mss of 216).

In summary, good starting points are:

	1200 baud shared channel	**9600 baud or higher clear channel**
mtu	256	1500
tcp mss	216	1460
tcp window	432	2920

Even more than in other parts of this manual, this discussion glosses over lots of subtleties. Throughput can be drastically affected by tuning these values, and both experimentation and local consensus are necessary to come up with settings that work well without stomping on other users of the channel.

Using NOS

To run NOS, first make sure you have your TNC configured for "**KISS**" mode (see Appendix F for details) and turned on. Then, type **NET**, **NOS**, or **GRINOS** (as appropriate). In a few seconds, you should see a "**net>**" prompt. Any error messages that appear first probably indicate a problem with one or more commands in your AUTOEXEC.NET file.

When you see the prompt, NOS is in "**command mode**." When you are communicating with another host, NOS is in "**converse mode**." To return to command mode from converse mode, press the **F10** function key (sometimes called the "**escape**" key, but not to be confused with the "ESC" key on your keyboard). All commands typed at the NOS prompt need to be followed by the return key.

Typing "**?**" in command mode will display a list of commands. Typing a command name followed by ? will display the valid subcommands. You can't really call it a help system, but it's better than nothing.

Some commands can be abbreviated to save typing; the degree of abbreviation allowed depends on the command set of the NOS version you're using. Experimentation is the best way to see what works and what doesn't. One minor annoyance in GRINOS is that **commands are case sensitive** -- "c ax0 n8acv" is fine, but "C ax0 n8acv" isn't. It's safest to do all your NOS keyboarding in lower case -- apart from case sensitive commands, in the Email world, typing in all upper case is considered shouting!

You can issue several commands from within NOS to deal with files and directories. "**pwd**" displays your current working directory, and "**cd**"

allows you to change directories. "**dir**" displays files in the current directory. "**mkdir <dirname>"** creates a new directory, and "**rmdir <dirname>**" removes one. "**delete <filename>**" erases a file.

You can also "shell out" to DOS from within NOS by entering either an exclamation mark (**!**) or the command "**shell**." To return to NOS, type "**exit**" at the DOS prompt.

From command mode, you can start a number of different types of **sessions** to communicate with remote hosts. Each session has its own display screen and you can switch between a session and command mode, or between sessions. The **se** command displays the active sessions with identifying numbers. To switch to a session, you can type "**se <session number>**." From command mode, you can return to the current (most recently displayed) session by entering a carriage return.

You can capture incoming data from the current session to a disk file by using the "**record <filename>**" command, and you can read in a data file from disk with the "**upload <filename>**" command.

To close a session, press F10 to return to command mode and enter "**close <session number>**." If there's only one session open, you can just enter "**close**." You can also end the session by issuing the appropriate exit or quit command at the remote host's prompt.

The most common NOS session types are probably "**telnet**," its cousin "**ttylink"**, "**ftp**," and a regular packet "**connect**" (technically called an "**ax25"** session). Telnet is used to "login" to a remote host, ttylink is a kind of telnet specially designed for keyboard-to-keyboard communications, ftp handles file transfers, and ax25 sessions allow you to carry on normal packet activity. We'll talk about ax25 sessions first, since they give you a chance to test your setup without having another TCP/IP station on the air.

AX.25 Mode

The "**connect**" command simply lets you do normal packet radio stuff. Establishing an ax25 connect through NOS is like using the standard TNC commands with a few small differences. First, since NOS can support several interfaces, each with its own hardware, you need to tell NOS which one to use.

So, to connect to N8ACV on interface ax0, enter "**connect ax0 N8ACV**." Once you get a "Connected" message, you'll be able to type to the station at the other end just as you would with normal packet. In addition to closing the session as described above, you can exit an

ax25 session by typing "**disconnect**" at the command mode prompt. (Just as with a TNC, these commands can be abbreviated; just how few of the letters are necessary will depend on each implementation of NOS and the commands it supports).

The other minor difference between the NOS connect command and a regular TNC is that the word "**via**" is not used when specifying digipeaters. To connect to N8ACV through N8KZA on interface ax0, you would enter "**connect ax0 N8ACV N8KZA.**"

Telnet

The "**telnet**" command logs you in to a remote TCP/IP host; depending on the capabilities of that host, you might find yourself chatting directly with the user at the other end, connecting to the NOS mailbox, "**mbox**" (which acts very much like a sophisticated personal PBBS), or getting a UNIX "login:" prompt. To establish a telnet session, enter "**telnet <hostname>**" at the command prompt.

Some versions of NOS offer a new type of session that improves on telnet for real-time keyboard-to-keyboard chats. It's called "**ttylink**," and it works just like telnet (for example, start a session with **"ttylink <hostname>**") except that it connects you directly to the remote host's chat mode, and uses a split-screen format to make things less confusing as you type to each other.

You'll get a message like "Telnet session 1 failed: Reset/Refused errno 9" if the remote host doesn't support ttylink. If the operator at the other end isn't available to chat, you'll get a message like "The system is unattended." You'll still be able to type, but there won't be anyone there to reply. You can change the status on your machine by setting the "**attended**" command to either **on** or **off**. You might want to put this command in your AUTOEXEC.NET file to set your default status. You exit from ttylink just as you would from telnet.

And now a note from Miss Manners: you should **never** simply exit from the NOS program when you have an open session. Doing so can cause great unpleasantness at the remote host. Unless you're in some sort of software or hardware lockup, or you **know** that the station on the other end has gone away, always close sessions and wait for confirmation before exiting the program.

You should also be aware that your system may have started sessions in the background, for example to transfer electronic mail, or someone else may have started a session with your system. You may not even know these sessions are running. Pulling the plug on them would be very impolite. Before exiting NOS, you should first use the **se**

command to make sure there are no current sessions running, and then the "**tcp status**" command to see if there are any background connections established. "**tcp status"** will show you a long and confusing list of information; the important stuff at the end is the list of **sockets** (which are services your system can either offer or request on the network). If anything other than "**Listening**" appears in the **Status** column, that means there's at least one remote host communicating with you.

File Transfers

You initiate a file transfer (ftp) session by entering "**ftp <hostname>**" at the command prompt. Once the session is established, the remote host will prompt you for a user name and a password. If your hostname and password have been added to the remote host's **FTPUSERS** file, you'll have the ability to download and perhaps upload files in the directories permitted you.

If you haven't arranged with the remote host for your own account, you can try to login as "**anonymous**" or "**guest**;" many systems support these user names and grant limited (usually download-only) privileges to them. If you login under one of these accounts, you should enter your hostname as the password; that allows the remote host to keep track of who's been using the system.

Once you've logged in, you'll see a new prompt: "**ftp>**." This will remind you that you're actually issuing commands to the remote computer. From the ftp> prompt, you can list the files in a directory, change directories, upload files, or download files.

To list files, enter "**dir**" at the ftp> prompt. You will get a listing that shows subdirectories (if any) and files together with their dates and sizes. To show the current directory name, type "**pwd**." To change directories, issue the "**cd <directory>**" command. Note that directories are displayed with a forward slash (/) instead of the usual MS-DOS backslash (\). That's because the UNIX operating system, which is TCP/IP's natural home, uses forward slashes. If the remote host is running NOS, you can use either character, but some other systems (particularly those running UNIX) will recognize only the forward slash.

Once you've found a file you want to upload or download, you need to make a decision. ftp can transfer the file either as an "**image**" file, byte for byte, or as an "**ascii**" file, converting the line-end character as necessary to compensate for different operating systems (UNIX uses only a linefeed character at the end of lines; MS-DOS uses carriage

return/linefeed). Before beginning a file transfer, enter "**type i**" for an image file, or "**type a**" for an ASCII file, at the ftp> prompt.

What are the consequences choosing the wrong transfer type? Well, transferring a binary file as type **"a**" will almost certainly fail. Transferring an ASCII file as type **"i"** will work, but you may find that the line-ends are screwed up. ASCII transfers are also quite a bit slower than image, because each line needs to be processed separately.

To actually start a file transfer, use the command "**put <local filename> <remote filename>**" to send a file, or "**get <remote filename> <local filename>**" to receive one. The file name can include a full path if you desire; remember to use the proper path separator character for the remote host.

If you only specify one filename, ftp will assume that both the local and remote hosts will use the same name. This can be dangerous if the remote host uses a different operating system than you do, as it may have filenames that are illegal on your system.

If a file transfer goes awry, you can terminate it by going to command mode via F10 and issuing the "**abort**" command. To end an ftp session, you can either type "**quit**" at the ftp> prompt (the preferred way), or you can close the session from the net> prompt.

If you want others to be able to access files on your system, you'll need to set up an **FTPUSERS** file in your root directory. Appendix E describes the contents of that file.

Another message from Miss Manners: transferring files via ftp is reliable, but can be slooooow, particularly at 1200 baud. Before you start downloading a 250 kilobyte file, consider how busy the channel is, and whether you want to tie things up for (perhaps) several hours by your download. NOS is polite and won't hog the channel, but don't doubt that a large file transfer will slow things down for everyone else.

Other Protocols

The "**ping**" protocol mentioned above is very useful to see if a remote host is on the air. Just enter the command "**ping <hostname>**" at the NOS prompt. If the host is available, you will see a response indicating what the round-trip time was to that host. The time may be many seconds if you're going through gateways, so be patient.

The "**finger**" protocol lets you see information about a remote host's users and services. Entering "**finger @<hostname>**" (note the

slightly different syntax -- the "**@**" symbol must immediately precede the remote hostname) will display a list of the finger files (described above) at that host. Entering "**finger <user@hostname>**" will display the text file for that user.

Electronic Mail

We've saved NOS's electronic mail capabilities for last because they are a bit more involved than some other parts of the program. You use two programs to handle mail: **BM** (a "**user mail agent**," in UNIX terms) to write and read messages, and NOS to send and receive them. First we'll talk about reading and writing messages, and then about using NOS to transport them.

Using BM.EXE to Read and Write Messages

BM.EXE is a program that reads and writes mail message in the format TCP/IP systems recognize. Contrary to popular belief, "BM" stands for "Bdale's Mailer" in honor of its creator, Bdale Garbee. You can run BM from the DOS prompt just like any other program, from within NOS by shelling to DOS with **!** or **shell**, or (in GRINOS) by typing the **mail** command from the net> prompt.

Before using BM, you need to create its configuration file, **BM.RC**, which must live in the root directory of your disk. An annotated BM.RC file is included as Appendix G. Only the first three commands in the sample file are absolutely necessary to make BM work.

There's a bit of controversy in some areas over the proper name to enter for "**user**" in BM.RC. Some folks recommend using either your first name, or your initials (for example, my address would be "john@ag9v.ampr.org") while other suggest using the callsign instead ("ag9v@ag9v.ampr.org").

While using the callsign may seem more impersonal, it has **major** advantages when mail is moving between TCP/IP and the packet BBS system, or when using the **POP** server; we strongly recommend that you use the "callsign@hostname" format unless the locals object even more strongly. It's important to be consistent within the area, so that everyone knows how to address mail to everyone else.

When you start BM, you'll see a prompt such as "**ag9v>"** showing the default mailbox (based on the "**user**" entry in BM.RC). As in NOS, you enter commands at the prompt, following them with a carriage return. Most BM commands are single letters, optionally followed by a mail addressee or a message number (or numbers).

To send mail, use the command "**m <addressee>**." The addressee will normally be a user at a remote host; for example, ag9v might send mail to k8gkh@k8gkh. The single biggest problem with BM is **forgetting to include the hostname** -- in other words, sending mail to <user> rather than <user>@<hostname>. Without the

hostname, BM will think the user is on your local system, and the mail will end up being stored in a mailbox under that user's name on your own system. That doesn't work too well.

One way to solve that problem, and do some other interesting things, is to create an **ALIAS** file in your root directory. When you send a message, BM will compare the addressee with the alias file, and if it finds a match will replace the alias with a full address from the file. An alias can point to a list of addresses, so it's possible to define an alias that will send a copy of the message to everyone in your local group. A sample alias file might look like:

```
greg  k8gkh@k8gkh.ampr.org
bill  n8kza@n8kza.ampr.org
club  k8gkh@k8gkh.ampr.org n8kza@n8kza.ampr.org
      n8acv@n8acv.ampr.org wb8gxb@wb8gxb.ampr.org
```

The alias for "club" demonstrates two things: a single alias can expand to several addresses, and you can continue a long address list on subsequent lines by indenting them with spaces or a tab character.

Now, if you send mail to "greg" it will automatically be expanded to the full address, and by sending a message to "club" all four users will get a copy.

By the way, you do <u>not</u> use a trailing dot after an FQDN (as discussed above) in Email addressing; doing so will screw things up.

If you use BM's built-in editor to compose messages, remember that it doesn't wrap lines; you have to hit the carriage return at the end of each line. Use the "**l**" command to list outbound mail; you can kill an outbound message with the **"k <msg#>**" command, using the message number obtained from the "**l**" command.

Several commands are used to deal with incoming mail. "**h**" displays the headers (summary info) about messages in your mailbox. It is the basic command you should use to check your incoming mail. Each header displayed includes a message number to use with the other message manipulation commands. Commands given without a message number act on the current message (the one marked with an "**>**" in the display from the "**h**" command); if there's only one message, it is always the current one.

BM can support multiple users at a single host; a separate mailbox is created for each user. Unfortunately, BM has no way of knowing if incoming mail addressed to <someuser>@<yourhost> is valid, so it will happily accept such mail and create a new mailbox for <someuser>. You may never know it's there, unless you use the **"n**"

command to display the list of mailboxes. You can also use "**n**" to change to a different mailbox: "**n <mbox>**."

The commonly used commands (which may be followed by one or more message numbers if appropriate) are:

msg# message number by itself will display that message and set it as the current message.
r reply to a message.
d delete a message.
s save a message; if a file name follows the message number(s), the message(s) will be saved in that file. Otherwise, they'll be saved in the default mbox file.
u undelete a message previously marked for deletion.
p print a message on the local printer.
w save a message to a file without including headers.
f forward a message to another recipient.
b bounce a message. Like forward, but keeps the original sender information intact (i.e., the message will not appear to have been sent by you).
$ update the mailbox. This deletes messages marked for deletion and reads in any new mail that may have arrived since you started BM.

There are two commands that exit from BM: "**x**" will exit without updating the mailbox. In other words, the same messages will be there the next time you run the program. "**q**" updates the mailbox (like "$") and then exits.

Outbound mail created by BM is stored in the \spool\mqueue directory, where it waits patiently until one of NOS's servers (SMTP or POP) attempts to send it to its destination.

Moving Mail With NOS

Now, to the mechanics of getting mail into and out of your system. All mail that you create is sent to its destination (or at least to the next stop on the way) by the "**smtp**" server in NOS. The "**smtp timer**" command (set in AUTOEXEC.NET) tells smtp how often to scan the \spool\mqueue directory for outgoing mail. When it finds some, it attempts to open an smtp session to the remote host in the address and send the mail there. There's no default for the smtp timer value, so your AUTOEXEC.NET file should include something like "**smtp timer 600**" (which scans for mail every ten minutes). You can manually force smtp to scan the queue by issuing the "**smtp kick**" command from the net> prompt.

If you have a local mail server with connections to the outside world, you can use it to route mail for hosts that aren't in your domain file with the "**smtp gateway <hostid>**" command.

Incoming mail can arrive at your station when a remote host does this and starts an smtp session with you. But if you don't keep your station up 24 hours a day, the remote host will be trying, and trying, and trying, to connect with you until you finally show up. A far better approach is to use **"POP**" -- the **P**ost **O**ffice **P**rotocol. If your system runs POP, and someone in the area has agreed to be a POP server, NOS will automatically contact that server when you come on the air; the server will respond by sending the mail waiting in your mailbox. You can then read it with BM just as if it had arrived via smtp.

To use POP, the server must establish a mailbox and password for you, and you need to add the appropriate commands to your AUTOEXEC.NET file (see the annotated AUTOEXEC.NET file in Appendix B).

Remember that smtp or POP sessions may be running in the background without your knowing about it. Always check for activity with the **"tcp status**" command before pulling the plug!

Additionally, smtp creates lock files in \spool\mqueue when it tries to send outgoing mail. If NOS is killed before the mail transfer has succeeded, these files (with the extension ".LCK") will be left behind and if they are not manually removed, they will prevent smtp from trying again to send those messages. To prevent this, you should always issue the command "erase \spool\mqueue*.LCK" before starting NOS. It's a good idea to launch NOS using a batch file that removes the locks before executing the program.

Conclusion

This has been a whirlwind tour of TCP/IP. Once you have the software installed, it's not hard to use, and NOS truly opens the door to enjoying packet radio in a whole new way.

To learn the subtleties of NOS, you should do two things: read the reference manual for the version you're using, and experiment with the program. Once you know the ins and outs, please share your knowledge with others. The ham radio TCP/IP community is still small, and we need all the Elmers we can get!

John Ackermann AG9V
2371 Stewart Road
Xenia, OH 45385

TCP/IP: ag9v@ag9v.ampr.org. [44.70.12.34]
PBBS: AG9V@N8ACV.OH.US.NA
Internet: jra@lawday.daytonOH.ncr.com
CompuServe: 72300,1160

APPENDIX A
Resources for NOS and TCP/IP

(Note: This is a very incomplete list; please feel free to provide additional resources to add for the next edition!)

TAPR
P.O. Box 22888
Tucson, AZ 85734

The New England TCP Association
3628 Acushnet Ave.
New Bedford, MA 02745

APPENDIX B
Sample AUTOEXEC.NOS File for GRINOS

```
# AUTOEXEC.NET
# This is a sample autoexec file for GRINOS version N1BEE 0.72.
# It doesn't have all the fancy features one might hope for, but
# the basics are there, with some hopefully useful comments.
# Any line beginning with a "#" character is treated as a comment.   To
# uncomment a line, delete the # character

# These are a couple of things for NOS to use internally.
mem eff on
watchdog on
nibufs 10

# NOS needs to know three things about you:   your hostname, your
# ham callsign, and your IP address.   By convention, the hostname
# is your callsign in lower case, followed by ".ampr.org".   The
# callsign is generally used in upper case to distinguish it.
# The IP address comes from a local area coordinator.   Note that there
are
# a minimum of three places in this file where you need to insert your
IP
# address -- here, in the ifconfig command, and at the end of each
attach
# command.
hostname nocall.ampr.org
ax25 mycall NOCALL
ip address [44.xx.xx.xx]

# This should match your IP address
ifconfig loopback ipaddress [44.xx.xx.xx]

# This makes short forms of the hostname work.
domain suffix ampr.org.

# NOS needs to know how to convert hostnames to IP addresses.
# You can do this manually via the "DOMAIN.TXT" file, or you can
# use a nameserver if one is available.   To enable the
# nameserver, uncomment this line and plug in its correct
# address.
#domain addserver [44.xx.xx.xx]
```

```
# Some additional commands for the domain service.  Don't turn
translate
# on unless you have a small domain file and/or a fast machine.
domain verbose off
domain cache size 40
domain translate off

# To use POP, uncomment these lines.  Fill in "pop mailhost" with the
# IP address of the station serving as your POP server.  Fill in the "pop
# mailbox" name with your hostname, i.e., your call.  The "pop
userdata"
# line needs to have your hostname, followed by a password
# (as negotiated with your mail server).  "pop timer" sets
# how often, in seconds, to query for mail.
#pop mailhost [44.xx.xx.xx]
#pop mailbox hostname
#pop userdata hostname password
#pop timer 1800

#.Attach commands are complex; these are samples for COM 1
# and 2.  See Appendix C for details.  Uncomment the
# appropriate line(s) for your hardware.
# COM1 -- 256 byte MTU, 4800 baud serial link as ax0
attach asy 0x3f8 4 ax25 ax0 2048 256 4800
# COM2 -- 256 byte MTU, 4800 baud serial link as ax1
#attach asy 0x2f8 3 ax25 ax1 2048 256 4800

# This is the basic route, sending everything out ax0
route add default ax0

# These are tcp parameters you shouldn't need to mess with.
ip ttl 16
ip rtimer 240
tcp irtt 3000

# On a shared channel, you may want to change timertype to
# exponential; that's more courteous, but will slow your
# retries down significantly.  mss and window should ordinarily be the
same
# value, equal to the largest mtu set in the attach command(s) above
minus
# 40.  With the common mtu for 1200 baud channels of 256, that
means
# both mss and window should be 216.
tcp timertype linear
tcp bblimit 16
tcp mss 216
tcp window 216
```

```
# These set up AX.25 parameters
ax25 digipeat off
ax25 maxframe 1
ax25 paclen 256
ax25 retry 20
ax25 window 4096
ax25 blimit 15
ax25 version 2

# as with tcp timertype, you may want to set this to
# exponential on a shared channel.
ax25 timertype linear

# These are netrom setup commands.   Don't turn them on
# unless you need them, and you know what you're doing.   You
# can really screw up the network by putting out netrom
# broadcasts that don't fit with the configuration of the
# "real" netrom nodes that can hear you.
#attach netrom
#netrom interface ax0 MYALIAS 192
#netrom obsotimer 1800
#netrom nodetimer 10800
#netrom verbose yes
#netrom bcnodes ax0
#netrom ttl 8

# These start the servers.
start smtp
start ftp
start echo
start discard
start telnet
start finger
start ax25

# Uncomment this line to enable logging.
#log \spool\net.log

# Default file type for ftp transfers.   Type image is for binary files; type
# ascii is for text; it's safest to set the default to image.
ftype image

# This makes telnet sessions to Unix systems work
# line-by-line, rather than character-by-character.
echo refuse

# Tell smtp how often to scan for outgoing mail
```

```
smtp timer 600
smtp batch on

# grinos can send a string of commands to the TNC on startup.   You
could
# use this to force the TNC into KISS mode.   Note that you need to
specify
# which interface to use.   This must be done <after>defining the
interface,
# and <before>any data is sent to the TNC (for example, by the smtp
and
# pop kick commands below) These commands will do that for a TNC2:
comm ax0 "kiss on"
comm ax0 "reset"

# kick the smtp and POP servers at startup.   Only uncomment the
"pop
# kick" line if you've defined a POP server above.
smtp kick
#pop kick

# GRINOS (but not other versions) can define the function
# keys with macros to make things a bit easier.   Here are a
# couple of examples.   Note that each command must end with
# a "\n" to signify a carriage return.   The numbers
# represent the keys; 59 - 68 for F1- F10 (though F10 can't
# be redefined; it's always the escape key), 84 - 93 for
# shiftF1 - shift F10, 94 - 103 for ctrlF1 - ctrlF10, 104 -
# 113 for altF1 - altF10.
fkey 59 "tcp status\n"
fkey 60 "mem status\n"
fkey 61 "status\n"

# THE END
```

APPENDIX C
Designing ATTACH Commands

NOS supports a number of versions of the **attach** command to deal with different hardware. We'll discuss three of them here: **asy**, used for serial port connections; **pi**, used to connect to the Ottawa PI card; and **packet**, used to interface to hardware supporting the FTP, Inc., packet driver protocol. As usual, this discussion covers the basics; see the NOS reference manual for details on all the many options.

Hosts normally have a separate IP address for each interface. If you are running more than one interface, you can include that interface's IP address (in [xx.xx.xx.xx] form) at the end of the attach command.

The asy version provides an interface to a standard PC serial port. The syntax is:

attach asy <ioaddr> <vector> <mode> <if> <bufsize> <mtu> <speed>

In English, these parameters are:

ioaddr -- the address of the COM port being used. COM1 is usually **0x3f8** and COM2 is usually **0x2f8**. COM3 and COM4 aren't standardized; using them will require looking at the documentation for your serial card, and probably some experimentation.

vector -- the IRQ used by the hardware. COM1 is usually **4**, and COM2 is usually **3**. Again, COM3 and COM4 vary.

mode -- this specifies the nature of the interface. **ax25** is for a connection to a KISS TNC, **slip** for a hardwired connection to another host, **ppp** for a dial-up connection, and **nrs** is for attaching a NOS station to a NetRom node.

if -- the interface name. The convention is to use **ax0**, **ax1**, etc., for KISS interfaces.

bufsize -- the buffer for incoming data, in bytes. Usually a value of **1024** is more than sufficient for a 1200 baud channel.

mtu -- the maximum transmission unit size, in bytes. See the discussion in the main text on this subject.

speed -- the speed of the serial (not radio) link, in baud. The best setting for this will depend on the speed of your

computer, but generally two to four times the radio speed is adequate.

Some sample **attach asy** commands are:

```
# COM1, KISS TNC as ax0, MTU 256, 4800 BAUD
attach asy 0x3f8 4 ax25 ax0 1024 256 4800

# COM2, KISS TNC as ax1, MTU 256, 2400 BAUD
attach asy 0x2f8 3 ax25 ax1 1024 256 2400

# SLIP link, COM1 as sl0, MTU 256, 9600 BAUD
attach asy 0x3f8 4 slip sl0 1024 256 9600
```

The Ottawa PI card is a plug-in board for PCs designed for high-speed performance. It has two ports, one DMA driven for high speed and the other interrupt driven. The attach syntax is:

attach pi <ioaddr> <vector> <DMA chn> <mode> <name> <bufsize> <mtu> <speed a> <speed b>

A sample attach command (using the PI's default jumper settings) is:

attach pi 380 7 1 ax25 pi0 1750 1024 0 1200

In this example, the interface name for the DMA port is "pi0a" and the second port is "pi0b". Because the port a speed is 0, the PI card expects the modem to provide its own clocking. The PI attach syntax is explained in the manual provided with the card.

Finally, the **packet** interface is used to connect to ethernet cards and other hardware that supports the FTP, Inc. "packet driver" standard. There's a packet driver for the PI card. The syntax is:

attach packet <ioaddr> <vector> <if> <bufsize> <mtu>

In this case, **ioaddr** and **vector** need to match those used for the packet TSR that supports the hardware. **bufsize** is the number of packets (not bytes) that may be outstanding. For ethernet, the standard **mtu** is 1500.

APPENDIX D
The DOMAIN.TXT File

```
# The domain.txt file contains mappings between hostnames
# and IP addresses.   The file can be quite complex, but
# basic entries usually resemble this.

# Fields are separated by tabs or spaces.

# These are normal address records.   The first field is the
# hostname.   The second field is a "time to   live" value
# returned by the name server.   If you manually create an
# entry, you can leave this field   blank.   The third field
# is always "IN" to signify these are internet addresses.
# The fourth field is "A" to signify an address record.   The
# last field is the address.

k8gkh.ampr.org.   3599886          IN        A         44.70.12.31
ag9v.ampr.org.    3585524          IN        A         44.70.12.34

# This is a "canonical name" (CNAME) record that maps an
# alias to an official hostname.

server.ampr.org.  3599886          IN        CNAME     ag9v.ampr.org.
```

APPENDIX E
Sample FTPUSERS File

```
# This file establishes ftp user permissions.   Fields are
# separated by exactly one space.   The privileges value is a
# bitmask.   The only values significant for ftp are:

# 1 - read only
# 3 - read/write
# 7 - read/write/overwrite/delete


anonymous * /pub 1          # no password, read only in /pub
friend foobar /pub 3        # read/write privileges in /pub
spouse snoogums / 7         # read/write/delete everywhere
```

APPENDIX F
Making Your TNC Talk in KISS MODE

Once NOS is installed and your configuration files set, you need to do one more thing: get your TNC talking to your computer in **KISS** (**K**eep **I**t **S**imple, **S**tupid) mode. KISS is a special protocol that lets your computer do the work of processing packets; the TNC does only the very low-level packet assembly and disassembly functions. Nearly all TNCs support KISS in one way or another.

Typically, you'll need to issue commands to the TNC to set the serial line baud rate to the same speed as you've specified in the attach command, to 8 bit data, and to no parity. Then, issue the KISS command (on a TNC2, **kiss on**), and the TNC's software reset command. After that, you won't be able to talk to your TNC via the terminal program, but NOS will be able to. (And don't worry, you can easily return the TNC to normal mode if you want to.) Once you've done this, you're set to run NOS.

One trick that grinos supports is the ability to send commands to the TNC during startup. The **comm** command will send a string of text to the named interface. For example, to force a Kantronics DataEngine or KAM into KISS mode every time you start NOS, include the following commands in AUTOEXEC.NOS (**after** you've defined the interface with the attach command):

```
comm ax0 "interface kiss"
comm ax0 "reset"
```

Note that surrounding the text with quote characters will preserve spaces in the command.

Appendix G
A Sample BM.RC File

```
# BM.rc

# your hostname -- note that for mail we <don't> put a trailing period
at
# the end of the FQDN.
host ag9v.ampr.org

# the user name (one host can receive mail for several users); usually
# your callsign
user ag9v

# your full name, for the message "From:" line
fullname John Ackermann

# if you want to have replies sent to another host, because, for
example,
# you are using a POP server, this line specifies where replies should
go
reply ag9v@ag9v.ampr.org

# for faster screen writes on the pc, use direct video, not bios
screen direct

# if you want to use an editor different than BM's built-in one
edit ed

# put saved messages here; note "/" instead of "\"
mbox c:/folder/mbox

# save a copy of outbound mail here
record c:/folder/outmail

# folder for your mail
folder c:/folder

# maximum number of messages that can be pending
maxlet 200
```