NET User Reference Manual (NOS Version)

Phil Karn, KA9Q

1. The NET.EXE Program

The MS-DOS executable file **net.exe** provides Internet (TCP/IP), NET/ROM and AX.25 facilities. Because it has an internal multitasking operating system, **net.exe** can act simultaneously as a client, a server and a packet switch for all three sets of protocols. That is, while a local user accesses remote services, the system can also provide those same services to remote users while also switching IP, NET/ROM and AX.25 packets and frames between other client and server nodes.

The keyboard and display is used by the local operator to control both host and gateway level functions, for which a number of commands are provided.

1.1. Installation

Net.exe uses the following directory structure:

```
/net
/net/spool
/net/spool/help
/net/spool/mail
/net/spool/mqueue
/net/spool/rqueue
/net/spool/news
```

The "/net" directory is not strictly necessary, and defaults to the root of the current drive. Any name may be chosen using the **-d** command-line option (described below). The **alias, autoexec.net, dialer, domain.txt** and **ftpusers** configuration files are located here.

The "/spool" directory and its sub-directories are used by the bbs, SMTP and NNTP services. The **areas, forward.bbs, history, mail.log, rewrite** and **signatur** configuration files are located here.

The **alias, forward.bbs** and **rewrite** files are described in the document "maildoc", which should be found at the same location as this file.

1.2. net [-b] [-s <sockets>] [-d <directory>] [<startup file>]

1.2.1. -b

The **-b** option specifies the use of BIOS for console output; the default is to write directly to the video display buffer. Use this option if you are running under a windowing package and have trouble with output "bleeding through" on top of other windows.

1.2.2. -s

The **-s** option specifies the size of the *socket* array to be allocated within **net.exe**. This limits the number of network connections that may exist simultaneously. The default is 40.

1.2.3. -d

The **-d** option allows the user to specify a directory for the configuration and spool files; it defaults to the root directory of the system.

1.2.4. Startup file

After all command-line options, the name of a startup file may be specified. If no startup file is specified, **net.exe** attempts to open a file named **autoexec.net** in the configuration directory of the current drive. If the file exists, it is read and executed as though its contents were typed on the console as commands. (See the **Commands** chapter.) This feature is useful for attaching communication interfaces, configuring network addresses, and starting the various services.

2. Console modes

The console may be in one of two modes: *command mode* and *converse mode*. In *command mode*, the prompt **net>** is displayed and any of the commands described in the **Commands** chapter may be entered. In *converse mode*, keyboard input is processed according to the *current session*.

Sessions come in many types, including *Telnet, FTP, AX25, NETROM, Ping, More, Hopcheck* and *Tip*. In a Telnet, AX25, NETROM, or Tip session, keyboard input is sent to the remote system and any output from the remote system is displayed on the console. In a FTP session, keyboard input is first examined to see if it is a known local command; if so it is executed locally. If not, it is "passed through" to the remote FTP server. (See the **FTP Subcommands** chapter). In a Ping session the user may test the path to a remote site, and in a More session, the user may examine a local file. A Hopcheck session is used to trace the path taken by packets to reach a specified destination.

The keyboard also has *cooked* and *raw* states. In *cooked* state, input is line-at-a-time; the user may use the line editing characters ˆU, ˆR and backspace to erase the line, redisplay the line and erase the last character, respectively. Hitting either return or line feed passes the complete line up to the application. In *raw* state, each character is immediately passed to the application as it is typed.

The keyboard is always in *cooked* state in command mode. It is also *cooked* in converse mode on an AX25, FTP or NET/ROM session. In a Telnet session it depends on whether the remote end has issued (and the local end has accepted) the Telnet WILL ECHO option (see the **echo** command).

On the IBM-PC, the user may escape back to *command mode* by hitting the F10 key. On other systems, the user must enter the *escape* character, which is by default control-] (hex 1d, ASCII GS). (Note that this is distinct from the ASCII character of the same name). The *escape* character can be changed (see the **escape** command).

In the IBM PC version, each session (including the command "session") has its own screen. When a new session is created, the command display is saved in memory and the screen is cleared. When the command escape key (usually F10) is hit, the current session screen is saved and the command screen is restored. When a session is resumed, its screen is restored exactly as it appeared when it was last current.

3. Commands

This chapter describes the commands recognized in command mode, or within a startup file such as **autoexec.net**. These are given in the following notation:

command
command literal_parameter
command subcommand <parameter>
command [<optional_parameter>]
command a | b

Many commands take subcommands or parameters, which may be optional or required. In general, if a required subcommand or parameter is omitted, an error message will summarize the available subcommands or required parameters. (Giving a '?' in place of the subcommand will also generate the message. This is useful when the command word alone is a valid command.) If a command takes an optional value parameter, issuing the command without the parameter generally displays the current value of the variable. (Exceptions to this rule are noted in the individual command descriptions.)

Two or more parameters separated by vertical bar(s) denote a choice between the specified values. Optional parameters are shown enclosed in [brackets], and a parameter enclosed in <angle brackets> should be replaced with an actual value or string. For example, the notation <**hostid**> denotes an actual host or gateway, which may be specified in one of two ways: as a numeric IP address in dotted decimal notation (eg. 44.0.0.1), or as a symbolic name listed in the file **domain.txt**.

All commands and many subcommands may be abbreviated. You only need type enough of a command's name to distinguish it from others that begin with the same series of letters. Parameters, however, must be typed in full.

Certain FTP subcommands (eg. **put, get, dir**, etc) are recognized only in converse mode with the appropriate FTP session; they are not recognized in command mode. (See the **FTP Subcommands** chapter.)

3.1. <CR>

Entering a carriage return (empty line) while in command mode puts you in converse mode with the current session. If there is no current session, **net.exe** remains in command mode.

3.2. !

An alias for the **shell** command.

3.3.

Commands starting with the hash mark (#) are ignored. This is mainly useful for comments in the **autoexec.net** file.

3.4. abort [<session #>]

Abort a FTP **get, put** or **dir** operation in progress. If issued without an argument, the current session is aborted. (This command works only on FTP sessions.) When receiving a file, **abort** simply resets the data connection; the next incoming data packet will generate a TCP RST (reset) response to clear the remote server. When sending a file, **abort** sends a premature end-of-file. Note that in both cases **abort** will leave a partial copy of the file on the destination machine, which must be removed manually if it is unwanted.

3.5. arp

Display the Address Resolution Protocol table that maps IP addresses to their subnet (link) addresses on subnetworks capable of broadcasting. For each IP address entry the subnet type (eg. Ethernet, AX.25), subnet address and time to expiration is shown. If the link address is currently unknown, the number of IP datagrams awaiting resolution is also shown.

3.5.1. arp add <hostid> ethernet | ax25 <ethernet address> | <ax25_address>

Add a permanent entry to the table. It will not time out as will an automatically-created entry, but must be removed with the **arp drop** command.

3.5.2. arp publish <hostid> ethernet | ax25 <ethernet address> | <ax25_address>

This command is similar to the **arp add** command, but system will also respond to any ARP request it sees on the network that seeks the specified address. (Use this feature with great care.)

3.5.3. arp drop <hostid> ax25 | ethernet

Remove the specified entry from the ARP table.

3.5.4. arp flush

Drop all automatically-created entries in the ARP table; permanent entries are not affected.

3.6. asystat

Display statistics on attached asynchronous communications interfaces (8250 or 16550A), if any. The display for each port consists of three lines. The first line gives the port label and the configuration flags; these indicate whether the port is a 16550A chip, the *trigger character* if any, whether CTS flow control is enabled, whether RLSD (carrier detect) line control is enabled, and the speed in bits per second. (Receiving the *trigger character* causes the driver to signal upper layer software that data is ready; it is automatically set to the appropriate frame end character for SLIP, PPP and NRS lines.)

The second line of the status display shows receiver (RX) event counts: the total number of receive interrupts, received characters, receiver overruns (lost characters) and the receiver *high water mark*. The high water mark is the maximum number of characters ever read from the device during a single interrupt. This is useful for monitoring system interrupt latency margins as it shows how close the port hardware has come to overflowing due to the inability of the CPU to respond to a receiver interrupt in time. 8250 chips have no FIFO, so the high water mark cannot go higher than 2 before overruns occur. The 16550A chip, however, has a 16-byte receive FIFO which the software programs to interrupt the CPU when the FIFO is one-quarter full. The high water mark should typically be 4 or 5 when a 16550A is used; higher values indicate that the CPU has at least once been slow to respond to a receiver interrupt.

When the 16550A is used, a count of FIFO timeouts is also displayed on the RX status line. These are generated automatically by the 16550A when three character intervals go by with more than 0 but less than 4 characters in the FIFO. Since the characters that make up a SLIP or NRS frame are normally sent at full line speed, this count will usually be a lower bound on the number of frames received on the port, as only the last fragment of a frame generally results in a timeout (and then only when the frame is not a multiple of 4 bytes long.)

Finally, the software fifo overruns and high water mark are displayed. These indicate whether the <bufsize> parameter on the attach command needs to be adjusted (see the **Attach Commands** chapter).

The third line shows transmit (TX) statistics, including a total count of transmit interrupts, transmitted characters, the length of the transmit queue in bytes, the number of status interrupts, and the number of THRE timeouts. The status interrupt count will be zero unless CTS flow control or RLSD line control has been enabled. The THRE timeout is a stopgap measure to catch lost transmit interrupts, which seem to happen when there is a lot of activity (ideally, this will be zero).

3.7. attach <hw type> ...

Configure and attach a hardware interface to the system. Detailed instructions for each driver are in the **Attach Commands** chapter. An easy way to obtain a summary of the parameters required for a given device is to issue a partial attach command (eg. **attach packet**). This produces a usage message giving the complete command format.

3.8. ax25 ...

These commands are used for AX25 interfaces.

3.8.1. ax25 blimit [<count>]

Display or set the AX25 retransmission backoff limit. Normally each successive AX25 retransmission is delayed by twice the value of the previous interval; this is called *binary exponential backoff*. When the backoff reaches the blimit setting it is held at that value, which defaults to 30. To prevent the possibility of "congestive collapse" on a loaded channel, blimit should be set at least as high as the number of stations sharing the channel. Note that this is applicable only on actual AX25 connections; UI frames will never be retransmitted by the AX25 layer.

3.8.2. ax25 digipeat [on | off]

Display or set the digipeater enable flag.

3.8.3. ax25 flush

Clear the AX.25 "heard" list (see **ax25 heard**).

3.8.4. ax25 heard

Display the AX.25 "heard" list. For each interface that is configured to use AX.25, a list of all callsigns heard through that interface is shown, along with a count of the number of packets heard from each station and the interval, in hr:min:sec format, since each station was last heard. The local station always appears first in the listing; the packet count actually reflects the number of packets transmitted. This entry is always present even if no packets have been sent.

3.8.5. ax25 irtt [<milliseconds>]

Display or set the initial value of smoothed round trip time to be used when a new AX25 connection is created. The value is in milliseconds. The actual round trip time will be learned by measurement once the connection has been established.

3.8.6. ax25 kick <axcb>

Force a retransmission on the specified AX.25 control block.

3.8.7. ax25 maxframe [<count>]

Establish the maximum number of frames that will be allowed to remain unacknowledged at one time on new AX.25 connections. This number cannot be greater than 7.

3.8.8. ax25 mycall [<call>]

Display or set the local AX.25 address. The standard format is used (eg. KA9Q-0 or WB6RQN-5). This command must be given before any **attach** commands using AX.25 mode are given.

3.8.9. ax25 paclen [<size>]

Limit the size of I-fields on new AX.25 connections. If IP datagrams or fragments larger than this are transmitted, they will be transparently fragmented at the AX.25 level, sent as a series of I frames, and reassembled back into a complete IP datagram or fragment at the other end of the link. To have any effect on IP datagrams, this parameter should be less than or equal to the MTU of the associated interface.

3.8.10. ax25 pthresh [<size>]

Display or set the poll threshold to be used for new AX.25 Version 2 connections. The poll threshold controls retransmission behavior as follows. If the oldest unacknowledged I-frame size is less than the poll threshold, it will be sent with the poll (P) bit set if a timeout occurs. If the oldest unacked I-frame size is equal to or greater than the threshold, then a RR or RNR frame, as appropriate, with the poll bit set will be sent if a timeout occurs.

The idea behind the poll threshold is that the extra time needed to send a "small" I-frame instead of a supervisory frame when polling after a timeout is small, and since there's a good chance the I-frame will have to be sent anyway (i.e., if it were lost previously) then you might as well send it as the poll. But if the I-frame is large, send a supervisory (RR/RNR) poll instead to determine first if retransmitting the oldest unacknowledged I-frame is necessary; the timeout might have been caused by a lost acknowledgement. This is obviously a tradeoff, so experiment with the poll threshold setting. The default is 128 bytes, one half the default value of **paclen**.

3.8.11. ax25 reset <axcb>

Delete the AX.25 connection control block at the specified address.

3.8.12. ax25 retry [<count>]

Limit the number of successive unsuccessful retransmission attempts on new AX.25 connections. If this limit is exceeded, link re-establishment is attempted. If this fails **retry** times, then the connection is abandoned and all queued data is deleted.

3.8.13. ax25 route

Display the AX.25 routing table that specifies the digipeaters to be used in reaching a given station.

3.8.13.1. ax25 route add <target> [digis ...]

Add an entry to the AX.25 routing table. An automatic **ax25 route add** is executed if digipeaters are specified in an AX25 **connect** command, or if a connection is received from a remote station via digipeaters. Such automatic routing table entries won't override locally created entries, however.

3.8.13.2. ax25 route drop <target>

Drop an entry from the AX.25 routing table.

3.8.14. ax25 status [<axcb>]

Without an argument, display a one-line summary of each AX.25 control block. If the address of a particular control block is specified, the contents of that control block are dumped in more detail. Note that the send queue units are frames, while the receive queue units are bytes.

3.8.15. ax25 t3 [<milliseconds>]

Display or set the AX.25 idle "keep alive" timer. Value is in milliseconds.

3.8.16. ax25 version [1 | 2]

Display or set the version of the AX.25 protocol to attempt to use on new connections. The default is 1 (the version that does not use the poll/final bits).

3.8.17. ax25 window [<size>]

Set the number of bytes that can be pending on an AX.25 receive queue beyond which I frames will be answered with RNR (Receiver Not Ready) responses. This presently applies only to suspended interactive AX.25 sessions, since incoming I-frames containing network (IP, NET/ROM) packets are always processed immediately and are not placed on the receive queue. However, when an AX.25 connection carries both interactive and network packet traffic, an RNR generated because of backlogged interactive traffic will also stop network packet traffic from being sent.

3.9. cd [<dirname>]

Change the current working directory, and display the new setting. Without an argument, **cd** simply displays the current directory without change. The **pwd** command is an alias for **cd**.

3.10. close [<session>]

Close the specified session; without an argument, close the current session. On an AX.25 session, this command initiates a disconnect. On a FTP or Telnet session, this command sends a FIN (i.e., initiates a close) on the session's TCP connection. This is an alternative to asking the remote server to initiate a close (**QUIT** to FTP, or the logout command appropriate for the remote system in the case of Telnet). When either FTP or Telnet sees the incoming half of a TCP connection close, it automatically responds by closing the outgoing half of the connection. Close is more graceful than the **reset** command, in that it is less likely to leave the remote TCP in a "half-open" state.

3.11. connect <iface> <callsign> [<digipeater> ...]

Initiate a "vanilla" AX.25 session to the specified call sign using the specified interface. Data sent on this session goes out in conventional AX.25 packets with no upper layer protocol. The de-facto presentation standard format is used, in that each packet holds one line of text, terminated by a carriage return. A single AX.25 connection may be used for terminal-to-terminal, IP and NET/ROM traffic. The three types of data are automatically separated by their AX.25 Level 3 Protocol IDs.

Up to 7 optional digipeaters may be given; note that the word **via** is NOT needed. If digipeaters are specified, they are automatically added to the AX25 routing table as though the **ax25 route add** command had been given before issuing the **connect** command.

3.12. delete <filename>

Delete a **filename** in the current working directory.

3.13. detach <iface>

Detach a previously attached interface from the system. All IP routing table entries referring to this interface are deleted, and forwarding references by any other interface to this interface are removed.

3.14. dialer <iface> <seconds> <hostid> <pings> <dialer-file>

Setup an autodialer session for the interface. Whenever the interface is idle for the interval in <seconds>, the autodialer will ping the <hostid>. If there is no answer after <pings> attempts, the autodialer will execute the special commands contained in the <dialer-file>.

If the interval in <seconds> is zero, a previous dialer command process will be removed. If the number of <pings> is zero, the <dialer-file> will be executed without pinging the <hostid>.

The file may have any valid name, and must be located in the configuration directory (see the **Installion** section). The commands in the file are described in the **Dialer Subcommands** chapter.

3.15. dir [<dirname>]

List the contents of the specified directory on the console. If no argument is given, the current directory is listed. Note that this command works by first listing the directory into a temporary file, and then creating a **more** session to display it. After this completes, the temporary file is deleted.

3.16. disconnect [<session #>]

An alias for the **close** command (for the benefit of AX.25 users).

3.17. domain ...

These commands are used for access to the Domain Name Service.

3.17.1. domain addserver <hostid>

Add one or more domain name server(s) to the list of name servers.

3.17.2. domain dropserver <hostid>

Remove one or more domain name server(s) from the list of name servers.

3.17.3. domain listservers

List the currently configured domain name servers, along with statistics on how many queries and replies have been exchanged with each one, response times, etc.

3.17.4. domain query <hostid>

Send a query to a domain server asking for all resource records associated with this <hostid>, and list the records.

3.17.5. domain retry [<count>]

Display or set the number of attempts to reach each server on the list during one call to the resolver. If this count is exceeded, a failure indication is returned. If set to 0, the list will cycle forever; this may be useful for unattended operation. The default is 3.

3.17.6. domain suffix [<domain suffix>]

Display or specify the default domain name suffix to be appended to a host name when it contains no periods. For example, if the suffix is set to **ampr.org** and the user enters **telnet ka9q**, the domain resolver will attempt to find **ka9q.ampr.org**. If the host name being sought contains one or more periods, however, the default suffix is NOT applied (eg. **telnet foo.bar** would NOT be turned into **foo.bar.ampr.org**).

3.17.7. domain trace [on|off]

Display or set the flag controlling the tracing of domain server requests and responses. Trace messages will be seen only if a domain name being sought is not found in the local cache file, **domain.txt**.

3.17.8. domain cache ...

These commands are used for the use of the resource record file **domain.txt**, and the local memory cache.

3.17.8.1. domain cache clean [on|off]

Display or set the flag controlling the removal of resource records from the **domain.txt** file whose time-to-live has reached zero.

When clean is off (the default), expired records will be retained; if no replacement can be obtained from another domain name server, these records will continue to be used.

When clean is on, expired records will be removed from the file whenever any new record is added to the file.

3.17.8.2. domain cache list

List the current contents of the local memory cache.

3.17.8.3. domain cache size [<count>]

Display or set the nominal maximum size of the local memory cache. The default is 20.

(Note: The cache may be temporarily larger when waiting for new records to be written to the **domain.txt** file.)

3.17.8.4. domain cache wait [<seconds>]

Display or set the interval in seconds to wait for additional activity before updating the **domain.txt** file. The default is 300 seconds (5 minutes).

3.18. echo [accept | refuse]

Display or set the flag controlling client Telnet's response to a remote WILL ECHO offer.

The Telnet presentation protocol specifies that in the absence of a negotiated agreement to the contrary, neither end echoes data received from the other. In this mode, a Telnet client session echoes keyboard input locally and nothing is actually sent until a carriage return is typed. Local line editing is also performed: backspace deletes the last character typed, while control-U deletes the entire line.

When communicating from keyboard to keyboard the standard local echo mode is used, so the setting of this parameter has no effect. However, many timesharing systems (eg. UNIX) prefer to do their own echoing of typed input. (This makes screen editors work right, among other things). Such systems send a Telnet WILL ECHO offer immediately upon receiving an incoming Telnet connection request. If **echo accept** is in effect, a client Telnet session will automatically return a DO ECHO response. In this mode, local echoing and editing is turned off and each key stroke is sent immediately (subject to the Nagle tinygram algorithm in TCP). While this mode is just fine across an Ethernet, it is clearly inefficient and painful across slow paths like packet radio channels. Specifying **echo refuse** causes an incoming WILL ECHO offer to be answered with a DONT ECHO; the client Telnet session remains in the local echo mode. Sessions already in the remote echo mode are unaffected. (Note: Berkeley Unix has a bug in that it will still echo input even after the client has refused the WILL ECHO offer. To get around this problem, enter the **stty -echo** command to the shell once you have logged in.)

3.19. eol [unix | standard]

Display or set Telnet's end-of-line behavior when in remote echo mode. In standard mode, each key is sent as-is. In unix mode, carriage returns are translated to line feeds. This command is not necessary with all UNIX systems; use it only when you find that a particular system responds to line feeds but not carriage returns. Only SunOS release 3.2 seems to exhibit this behavior; later releases are fixed.

3.20. escape [<char>]

Display or set the current command-mode escape character in hex. (This command is not provided on the IBM-PC; on the PC, the escape char is always F10.)

3.21. etherstat

Display 3-Com Ethernet controller statistics (if configured).

3.22. exit

Exit the **net.exe** program and return to MS-DOS.

3.23. finger <user@hostid> [<user@hostid> ...]

Issue a network finger request for user **user** at host **hostid**. This creates a client session which may be interrupted, resumed, reset, etc, just like a Telnet client session.

3.24. ftp <hostid>

Open an FTP control channel to the specified remote host and enter converse mode on the new session. Responses from the remote server are displayed directly on the screen. See the **FTP Subcommands** chapter for descriptions of the commands available in a FTP session.

3.25. help

Display a brief summary of top-level commands.

3.26. hop ...

These commands are used to test the connectivity of the network.

3.26.1. hop check <hostid>

Initiate a *hopcheck* session to the specified host. This uses a series of UDP "probe" packets with increasing IP TTL fields to determine the sequence of gateways in the path to the specified destination. This function is patterned after the UNIX *traceroute* facility.

ICMP message tracing should be turned off before this command is executed (see the **icmp trace** command).

3.26.2. hop maxttl [<hops>]

Display or set the maximum TTL value to be used in hop check sessions. This effectively bounds the radius of the search.

3.26.3. hop maxwait [<seconds>]

Display or set the maximum interval that a hopcheck session will wait for responses at each stage of the trace. The default is 5 seconds.

3.26.4. hop queries [<count>]

Display or set the number of UDP probes that will be sent at each stage of the trace. The default is 3.

3.26.5. hop trace [on | off]

Display or set the flag that controls the display of additional information during a hop check session.

3.27. hostname [<name>]

Display or set the local host's name. By convention this should be the same as the host's primary domain name. This string is used only in the greeting messages of the various network servers; note that it does NOT set the system's IP address.

If <name> is the same as an <iface> (see the **Attach commands** chapter), this command will search for a CNAME domain resource record which corresponds to the IP address of the <iface>.

3.28. hs

Display statistics about the HS high speed HDLC driver (if configured and active).

3.29. icmp ...

These commands are used for the Internet Control Message Protocol service.

3.29.1. icmp echo [on | off]

Display or set the flag controlling the asynchronous display of ICMP Echo Reply packets. This flag must be on for one-shot pings to work (see the **ping** command.)

3.29.2. icmp status

Display statistics about the Internet Control Message Protocol (ICMP), including the number of ICMP messages of each type sent or received.

3.29.3. icmp trace [on | off]

Display or set the flag controlling the display of ICMP error messages. These informational messages are generated by Internet routers in response to routing, protocol or congestion problems. This option should be turned off before using the **hop check** facility because it relies on ICMP Time Exceeded messages, and the asynchronous display of these messages will be mingled with **hop check** command output.

3.30. ifconfig

Display a list of interfaces, with a short status for each.

3.30.1. ifconfig <iface>

Display an extended status of the interface.

3.30.2. ifconfig <iface> broadcast <address>

Set the broadcast address for the interface. The <address> takes the form of an IP address with 1's in the host part of the address. This is related to the **netmask** sub-command. See also the **arp** command.

3.30.3. ifconfig <iface> encapsulation <name>

Not fully implemented.

3.30.4. ifconfig <iface> forward <forward-iface>

Set a forwarding interface for multiple channel interfaces. To remove the forward, set <forward-iface> to <iface>.

3.30.5. ifconfig <iface> ipaddress <hostid>

Set the IP address for this interface. It is standard Internet practice that each interface has its own address. For hosts with only one interface, the interface address is usually the same as the host address. See also the **hostname** and **ip address** commands.

3.30.6. ifconfig <iface> linkaddress <hardware-dependant>

Set the hardware dependant address for this interface.

3.30.7. ifconfig <iface> mtu <mtu>

Set the MTU for this interface. See the **Setting ... MTU, MSS and Window** chapter for more information.

3.30.8. ifconfig <iface> netmask <address>

Set the sub-net mask for this interface. The <address> takes the form of an IP address with 1's in the network and subnet parts of the address, and 0's in the host part of the address. This is related to the **broadcast** sub-command. See also the **route** command.

3.30.9. ifconfig <iface> rxbuf <?>

Not yet implemented.

3.31. ip ...

These commands are used for the Internet Protocol service.

3.31.1. ip address [<hostid>]

Display or set the default local IP address. This command must be given before an **attach** command if it is to be used as the default IP address for the interface.

3.31.2. ip rtimer [<seconds>]

Display or set the IP reassembly timeout. The default is 30 seconds.

3.31.3. ip status

Display Internet Protocol (IP) statistics, such as total packet counts and error counters of various types.

3.31.4. ip ttl [<hops>]

Display or set the time-to-live value placed in each outgoing IP datagram. This limits the number of switch hops the datagram will be allowed to take. The idea is to bound the lifetime of the packet should it become caught in a routing loop, so make the value slightly larger than the number of hops across the network you expect to transit packets. The default is set at compilation time to the official recommended value for the Internet.

3.32. isat [on | off]

Display or set the AT flag. Currently, there is no sure-fire way to determine the type of clock-chip being used. If an AT type clock is in use, this command will allow measurement of time in milliseconds, rather than clock ticks (55 milliseconds per clock tick).

3.32.1. kick [<session>]

Kick all sockets associated with a session; if no argument is given, kick the current session. Performs the same function as the **ax25 kick** and **tcp kick** commands, but is easier to type.

3.33. log [stop | <filename>]

Display or set the **filename** for logging server sessions. If **stop** is given as the argument, logging is terminated (the servers themselves are unaffected). If a file name is given as an argument, server session log entries will be appended to it.

3.34. mbox

Display the status of the mailbox server system (if configured).

3.35. memory ...

These commands are used for memory allocation.

3.35.1. memory free

Display the storage allocator free list. Each entry consists of a starting address, in hex, and a size, in decimal bytes.

3.35.2. memory sizes

Display a histogram of storage allocator request sizes. Each histogram bin is a binary order of magnitude (i.e., a factor of 2).

3.35.3. memory status

Display a summary of storage allocator statistics. The first line shows the base address of the heap, its total size, the amount of heap memory available in bytes and as a percentage of the total heap size, and the amount of memory left over (i.e., not placed on the heap at startup) and therefore available for **shell** subcommands.

The second line shows the total number of calls to allocate and free blocks of memory, the difference of these two values (i.e., the number of allocated blocks outstanding), the number of allocation requests that were denied due to lack of memory, and the number of calls to free() that attempted to free garbage (eg. by freeing the same block twice or freeing a garbled pointer).

The third line shows the number of calls to malloc and free that occurred with interrupts off. In normal situations these values should be zero. The fourth line shows statistics for the special pool of fixed-size buffers used to satisfy requests for memory at interrupt time. The variables shown are the number of buffers currently in the pool, their size, and the number of requests that failed due to exhaustion of the pool.

3.36. mkdir <dirname>

Create a sub-directory in the current working directory.

3.37. mode <iface> [vc | datagram]

Control the default transmission mode on the specified AX.25 interface. In **datagram** mode, IP packets are encapsulated in AX.25 UI frames and transmitted without any other link level mechanisms, such as connections or acknowledgements.

In **vc** (virtual circuit) mode, IP packets are encapsulated in AX.25 I frames and are acknowledged at the link level according to the AX.25 protocol. Link level connections are opened if necessary.

In both modes, ARP is used to map IP to AX.25 addresses. The defaults can be overridden with the type-of-service (TOS) bits in the IP header. Turning on the "reliability" bit causes I frames to be used, while turning on the "low delay" bit uses UI frames. (The effect of turning on both bits is undefined and subject to change).

In both modes, IP-level fragmentation is done if the datagram is larger than the interface MTU. In virtual circuit mode, however, the resulting datagram (or fragments) is further fragmented at the AX.25 layer if it (or they) are still larger than the AX.25 **paclen** parameter. In AX.25 fragmentation, datagrams are broken into several I frames and reassembled at the receiving end before being passed to IP. This is preferable to IP fragmentation whenever possible because of decreased overhead (the IP header isn't repeated in each fragment) and increased robustness (a lost fragment is immediately retransmitted by the link layer).

3.38. more <file> [<file> ...]

Display the specified file(s) a screen at a time. To proceed to the next screen, press the space bar; to cancel the display, hit the 'q' key. The **more** command creates a session that you can suspend and resume just like any other session.

3.39. param <iface> [<param> ...]

Invoke a device-specific control routine. On a KISS TNC interface, this sends control packets to the TNC. Data bytes are treated as decimal. For example, **param ax0 1 255** will set the keyup timer (type field = 1) on the KISS TNC configured as ax0 to 2.55 seconds (255 x .01 sec). On a SLIP interface, the **param** command allows the baud rate to be read (without arguments) or set. The implementation of this command for the various interface drivers is incomplete and subject to change.

3.40. ping <hostid> [<length> [<seconds> [<incflag>]]]

Ping (send ICMP Echo Request packets to) the specified host. By default the data field contains only a small timestamp to aid in determining round trip time; if the optional **length** argument is given, the appropriate number of data bytes (consisting of hex 55) are added to the ping packets.

If interval is specified, pings will be repeated indefinitely at the specified number of seconds; otherwise a single, "one shot" ping is done. Responses to one-shot pings appear asynchronously on the command screen, while repeated pings create a session that may be suspended and resumed. Pinging continues until the session is manually reset.

The **incflag** option causes a repeated ping to increment the target IP address for each ping; it is an experimental feature for searching blocks of IP addresses for active hosts.

3.41. ppp ...

These commands are used for Point to Point Protocol interfaces.

This implementation of PPP is designed to be as complete as possible. Because of this, the number of options can be rather daunting. However, a typical PPP configuration might include the following commands:

```
attach asy 0x3f8 4 ppp pp0 4096 1500 9600
dial pp0 30 <hostid> 3 dialer.pp0
#
ppp pp0 lcp local accm 0
ppp pp0 lcp local compress address on
ppp pp0 lcp local compress protocol on
ppp pp0 lcp local magic on
ppp pp0 lcp open active
#
ppp pp0 ipcp local compress tcp 16 1
ppp pp0 ipcp open active
#
route add default pp0
```

3.41.1. ppp <iface>

Display the status of the PPP interface.

3.41.2. ppp <iface> lcp ...

These commands are used for the LCP [Link Control Protocol] configuration.

3.41.2.1. ppp <iface> lcp close

Shutdown the PPP interface.

3.41.2.2. ppp <iface> lcp local ...

These commands control the configuration of the local side of the link. If an option is specified, the parameters will be used as the initial values in configuration requests. If not specified, that option will not be requested.

For each of these options, the **allow** parameter will permit the remote to include that option in its response, even when the option is not included in the request. By default, all options are allowed.

3.41.2.2.1. ppp <iface> lcp local accm [<bitmap> | allow [on | off]]

Display or set the Async Control Character Map. The default is 0xffffffff.

3.41.2.2.2. ppp <iface> lcp local authenticate [pap | none | allow [on | off]]

Display or set the authentication protocol. The default is **none**.

3.41.2.2.3. ppp <iface> lcp local compress address/control [on | off | allow [on | off]]

Display or set the option to compress the address and control fields of the PPP HLDC-like header. This is generally desirable for slow asynchronous links, and undesirable for fast or synchronous links. The default is off.

3.41.2.2.4. ppp <iface> lcp local compress protocol [on | off | allow [on | off]]

Display or set the option to compress the protocol field of the PPP HLDC-like header. This is generally desirable for slow asynchronous links, and undesirable for fast or synchronous links. The default is off.

3.41.2.2.5. ppp <iface> lcp local magic [on | off | <value> | allow [on | off]]

Display or set the initial Magic Number. The default is off (zero).

3.41.2.2.6. ppp <iface> lcp local mru [<size> | allow [on | off]]

Display or set the Maximum Receive Unit. The default is 1500.

3.41.2.2.7. ppp <iface> lcp local default

Reset the options to their default values.

3.41.2.3. ppp <iface> lcp open active | passive

Wait for the physical layer to come up. If **active**, initiate configuration negotiation. If **passive**, wait for configuration negotiation from the remote.

3.41.2.4. ppp <iface> lcp remote ...

These commands control the configuration of the remote side of the link. The options are identical to those of the local side. If an option is specified, the parameters will be used in responses to the remote's configuration requests. If not specified, that option will be accepted if it is allowed.

For each of these options, the **allow** parameter will permit the remote to specify that option in its request. By default, all options are allowed.

3.41.2.5. ppp <iface> lcp timeout [<seconds>]

Display or set the interval to wait between configuration or termination attempts. The default is 3 seconds.

3.41.2.6. ppp <iface> lcp try ...

These commands are used for the various counters.

3.41.2.6.1. ppp <iface> lcp try configure [<count>]

Display or set the number of configuration requests sent. The default is 10.

3.41.2.6.2. ppp <iface> lcp try failure [<count>]

Display or set the number of bad configuration requests allowed from the remote. The default is 5.

3.41.2.6.3. ppp <iface> lcp try terminate [<count>]

Display or set the number of termination requests sent before shutdown. The default is 2.

3.41.3. ppp <iface> ipcp ...

These commands are used for the IPCP [Internet Protocol Control Protocol] configuration.

The **close, open, timeout** and **try** sub-commands are identical to the LCP (described above).

3.41.3.1. ppp <iface> ipcp local ...

These commands control the configuration of the local side of the link. If an option is specified, the parameters will be used as the initial values in configuration requests. If not specified, that option will not be requested.

For each of these options, the **allow** parameter will permit the remote to include that option in its response, even when the option is not included in the request. By default, all options are allowed.

3.41.3.1.1. ppp <iface> ipcp local address [<hostid> | allow [on | off]]

Display or set the local address for negotiation purposes. If an address of 0 is specified, the other side of the link will supply the address. By default, no addresses are negotiated.

3.41.3.1.2. ppp <iface> ipcp local compress [tcp <slots> [<flag>] | none | allow [on | off]]

Display or set the compression protocol. The default is **none**.

The **tcp** <slots> specifies the number of "conversation" slots, which must be 1 to 255. (This may be limited at compilation time to a smaller number.) A good choice is in the range 4 to 16.

The **tcp** <flag> is 0 (don't compress the slot number) or 1 (OK to compress the slot number). KA9Q can handle compressed slot numbers, so the default is 1.

3.41.3.2. ppp <iface> ipcp remote ...

These commands control the configuration of the remote side of the link. The options are identical to those of the local side. If an option is specified, the parameters will be used in responses to the remote's configuration requests. If not specified, that option will be accepted if it is allowed.

For each of these options, the **allow** parameter will permit the remote to specify that option in its request. By default, all options are allowed.

3.41.4. ppp <iface> pap ...

These commands are used for the PAP [Password Authentication Protocol] configuration.

The **timeout** and **try** sub-commands are identical to the LCP (described above). However, the terminate counter is unused.

3.41.4.1. ppp <iface> pap user [<username> [<password>]]

Display or set the username (the password may be set, but not displayed). When the username is specified, but no password is supplied, the **ftpusers** file is searched for the password. When a username/password is unknown or rejected, a session will appear at the console to prompt for a new username/password.

3.41.5. ppp <iface> trace [<flags>]

Display or set the flags that control the logging of information during PPP link configuration.

The flag value is 0 for none, 1 for basic, and 2 for general. Values greater than 2 are usually not compiled, and are described in the appropriate source files where they are defined.

3.42. ps

Display all current processes in the system. The fields are as follows:

PID - Process ID (the address of the process descriptor).

SP - The current value of the process stack pointer.

stksize - The size of the stack allocated to the process.

maxstk - The apparent peak stack utilization of this process. This is done in a somewhat heuristic fashion, so the numbers should be treated as approximate. If this number reaches or exceeds the stksize figure, the system is almost certain to crash; the **net.exe** program should be recompiled to give the process a larger allocation when it is started.

event - The event this task is waiting for, if it is not runnable.

fl - Process status flags. There are three: I (Interrupts enabled), W (Waiting for event) and S (Suspended). The I flag is set whenever a task has executed a pwait() call (wait for event) without first disabling hardware interrupts. Only tasks that wait for hardware interrupt events will turn off this flag; this is done to avoid critical sections and missed interrupts. The W flag indicates that the process is waiting for an event; the **event** column will be non-blank. Note that although there may be several runnable processes at any time (shown in the **ps** listing as those without the W flag and with blank event fields) only one process is actually running at any one instant (The Refrigerator Light Effect says that the **ps** command is always the one running when this display is generated.)

3.43. pwd [<dirname>]

An alias for the **cd** command.

3.44. record [off | <filename>]

Append to **filename** all data received on the current session. Data sent on the current session is also written into the file except for Telnet sessions in remote echo mode. The command **record off** stops recording and closes the file.

3.45. remote [-p <port>] [-k <key>] [-a <kickaddr>] <hostid> exit | reset | kick

Send a UDP packet to the specified host commanding it to exit the **net.exe** program, reset the processor, or force a retransmission on TCP connections. For this command to be accepted, the remote system must be running the **remote** server and the port number specified in the **remote** command must match the port number given when the server was started on the remote system. If the port numbers do not match, or if the remote server is not running on the target system, the command packet is ignored. Even if the command is accepted there is no acknowledgement.

The **kick** command forces a retransmission timeout on all TCP connections that the remote node may have with the local node. If the -a option is used, connections to the specified host are kicked instead. No key is required for the kick subcommand.

The **exit** and **reset** subcommands are mainly useful for restarting the **net.exe** program on a remote unattended system after the configuration file has been updated. The remote system should invoke the **net.exe** program automatically upon booting, preferably in an infinite loop. For example, under MS-DOS the boot disk should contain the following in **autoexec.net**:

```
:loop
net
goto :loop
```

3.46. remote -s <key>

The **exit** and **reset** subcommands of remote require a password. The password is set on a given system with the **-s** option, and it is specified in a command to a remote system with the **-k** option. If no password is set with the **-s** option, then the **exit** and **reset** subcommands are disabled.

Note that **remote** is an experimental feature in NOS; it is *not* yet supported by any other TCP/IP implementation.

3.47. rename <oldfilename> <newfilename>

Rename **oldfilename** to **newfilename**.

3.48. reset [<session>]

Reset the specified session; if no argument is given, reset the current session. This command should be used with caution since it does not reliably inform the remote end that the connection no longer exists. (In TCP a reset (RST) message will be automatically generated should the remote TCP send anything after a local reset has been done. In AX.25 the DM message performs a similar role. Both are used to get rid of a lingering half-open connection after a remote system has crashed.)

3.49. rip ...

These commands are used for the RIP service.

3.49.1. rip accept <gateway>

Remove the specified gateway from the RIP filter table, allowing future broadcasts from that gateway to be accepted.

3.49.2. rip add <hostid> <seconds> [<flags>]

Add an entry to the RIP broadcast table. The IP routing table will be sent to **hostid** every interval seconds. If **flags** is specified as 1, then "split horizon" processing will be performed for this destination. That is, any IP routing table entries pointing to the interface that will be used to send this update will be removed from the update. If split horizon processing is not specified, then all routing table entries except those marked "private" will be sent in each update. (Private entries are never sent in RIP packets).

Triggered updates are always done. That is, any change in the routing table that causes a previously reachable destination to become unreachable will trigger an update that advertises the destination with metric 15, defined to mean "infinity".

Note that for RIP packets to be sent properly to a broadcast address, there must exist correct IP routing and ARP table entries that will first steer the broadcast to the correct interface and then place the correct link-level broadcast address in the link-level destination field. If a standard IP broadcast address convention is used (eg. 128.96.0.0 or 128.96.255.255) then chances are you already have the necessary IP routing table entry, but unusual subnet or cluster-addressed networks may require special attention. However, an **arp add** command will be required to translate this address to the appropriate link level broadcast address. For example,

arp add 128.96.0.0 ethernet ff:ff:ff:ff:ff:ff

for an Ethernet network, and

arp add 44.255.255.255 ax25 qst-0

for an AX25 packet radio channel.

3.49.3. rip drop <dest>

Remove an entry from the RIP broadcast table.

3.49.4. rip merge [on | off]

This flag controls an experimental feature for consolidating redundant entries in the IP routing table. When rip merging is enabled, the table is scanned after processing each RIP update. An entry is considered redundant if the target(s) it covers would be routed identically by a less "specific" entry already in the table. That is, the target address(es) specified by the entry in question must also match the target addresses of the less specific entry and the two entries must have the same interface and gateway fields. For example, if the routing table contains

Dest	Len	Interface	Gateway	Metric	P	Timer	Use
1.2.3.4	32	ethernet0	128.96.1.2	1	0	0	0
1.2.3	24	ethernet0	128.96.1.2	1	0	0	0

then the first entry would be deleted as redundant since packets sent to 1.2.3.4 will still be routed correctly by the second entry. Note that the relative metrics of the entries are ignored.

3.49.5. rip refuse <gateway>

Refuse to accept RIP updates from the specified gateway by adding the gateway to the RIP filter table. It may be later removed with the **rip accept** command.

3.49.6. rip request <gateway>

Send a RIP Request packet to the specified gateway, causing it to reply with a RIP Response packet containing its routing table.

3.49.7. rip status

Display RIP status, including a count of the number of packets sent and received, the number of requests and responses, the number of unknown RIP packet types, and the number of refused RIP updates from hosts in the filter table. A list of the addresses and intervals to which periodic RIP updates are being sent is also shown, along with the contents of the filter table.

3.49.8. rip trace [0 | 1 | 2]

This variable controls the tracing of incoming and outgoing RIP packets. Setting it to 0 disables all RIP tracing. A value of 1 causes changes in the routing table to be displayed, while packets that cause no changes cause no output. Setting the variable to 2 produces maximum output, including tracing of RIP packets that cause no change in the routing table.

3.50. rmdir <dirname>

Remove a sub-directory from the current working directory.

3.51. route

With no arguments, **route** displays the IP routing table.

3.51.1. route add <dest_hostid>[/bits] | default <iface> [<gateway_hostid> [<metric>]]

This command adds an entry to the routing table. It requires at least two more arguments, the hostid of the target destination and the name of the interface to which its packets should be sent. If the destination is not local, the gateway's hostid should also be specified. (If the interface is a point-to-point link, then **gateway_hostid** may be omitted even if the target is non-local because this field is only used to determine the gateway's link level address, if any. If the destination is directly reachable, **gateway_hostid** is also unnecessary since the destination address is used to determine the interface link address).

The optional **/bits** suffix to the destination host id specifies how many leading bits in the host id are to be considered significant in the routing comparisons. If not specified, 32 bits (i.e., full significance) is assumed. With this option, a single routing table entry may refer to many hosts all sharing a common bit string prefix in their IP addresses. For example, ARPA Class A, B and C networks would use suffixes of /8, /16 and /24 respectively; the command

route add 44/8 sl0 44.64.0.2

causes any IP addresses beginning with "44" in the first 8 bits to be routed to 44.64.0.2; the remaining 24 bits are "don't-cares".

When an IP address to be routed matches more than one entry in the routing table, the entry with largest **bits** parameter (i.e., the "best" match) is used. This allows individual hosts or blocks of hosts to be exceptions to a more general rule for a larger block of hosts.

The special destination **default** is used to route datagrams to addresses not matched by any other entries in the routing table; it is equivalent to specifying a **/bits** suffix of /0 to any destination hostid. Care must be taken with default entries since two nodes with default entries pointing at each other will route packets to unknown addresses back and forth in a loop until their time-to-live (TTL) fields expire. (Routing loops for specific addresses can also be created, but this is less likely to occur accidentally).

Here are some examples of the **route** command:

```
# Route datagrams to IP address 44.0.0.3 to SLIP line #0.
# No gateway is needed because SLIP is point-to point.
route add 44.0.0.3 sl0

# Route all default traffic to the gateway on the local Ethernet
# with IP address 44.0.0.1
route add default ec0 44.0.0.1

# The local Ethernet has an ARPA Class-C address assignment;
# route all IP addresses beginning with 192.4.8 to it
route add 192.4.8/24 ec0

# The station with IP address 44.0.0.10 is on the local AX.25 channel
route add 44.0.0.10 ax0
```

3.51.2. route addprivate <dest hostid>[/bits] | default <iface> [<gateway hostid> [<metric>]]

This command is identical to **route add** except that it also marks the new entry as private; it will never be included in outgoing RIP updates.

3.51.3. route drop <dest hostid>

route drop deletes an entry from the table. If a packet arrives for the deleted address and a default route is in effect, it will be used.

3.52. session [<session #>]

Without arguments, displays the list of current sessions, including session number, remote TCP or AX.25 address and the address of the TCP or AX.25 control block. An asterisk (*) is shown next to the current session; entering a blank line at this point puts you in converse mode with that session. Entering a session number as an argument to the **session** command will put you in *converse* mode with that session. If the Telnet server is enabled, the user is notified of an incoming request and a session number is automatically assigned. The user may then select the session normally to converse with the remote user as though the session had been locally initiated.

3.53. shell

Suspends **net.exe** and executes a sub-shell ("command processor" under MS-DOS). When the sub-shell exits, **net.exe** resumes (under MS-DOS, enter the **exit** command). Background activity (FTP servers, etc) is also suspended while the subshell executes. Note that this will fail unless there is sufficient unused memory for the sub-shell and whatever command the user tries to run.

3.54. smtp ...

These commands are used for the Simple Message Transport Protocol service (that is, mail).

3.54.1. smtp gateway [<hostid>]

Displays or sets the host to be used as a "smart" mail relay. Any mail sent to a host not in the host table will instead be sent to the gateway for forwarding.

3.54.2. smtp kick

Run through the outgoing mail queue and attempt to deliver any pending mail. This command allows the user to "kick" the mail system manually. Normally, this command is periodically invoked by a timer whenever **net.exe** is running.

3.54.3. smtp maxclients [<count>]

Displays or sets the maximum number of simultaneous outgoing SMTP sessions that will be allowed. The default is 10; reduce it if network congestion is a problem.

3.54.4. smtp timer [<seconds>]

Displays or sets the interval between scans of the outbound mail queue. For example, **smtp timer 600** will cause the system to check for outgoing mail every 10 minutes and attempt to deliver anything it finds, subject of course to the **smtp maxclients** limit. Setting a value of zero disables queue scanning altogether, note that this is the default! This value is recommended for stand alone IP gateways that never handle mail, since it saves wear and tear on the floppy disk drive.

3.54.5. smtp trace [<value>]

Displays or sets the trace flag in the SMTP client, allowing you to watch SMTP's conversations as it delivers mail. Zero (the default) disables tracing.

3.55. socket [<socket #>]

Without an argument, displays all active sockets, giving their index and type, the address of the associated protocol control block and the and owner process ID and name. If the index to an active socket is supplied, the status display for the appropriate protocol is called. For example, if the socket refers to a TCP connection, the display will be that given by the **tcp status** command with the protocol control block address.

3.56. start ax25 | discard | echo | ftp | netrom | remote | smtp | telnet | ttylink

Start the specified Internet server, allowing remote connection requests.

3.57. stop ax25 | discard | echo | ftp | netrom | remote | smtp | telnet | ttylink

Stop the specified Internet server, rejecting any further remote connect requests. Existing connections are allowed to complete normally.

3.58. tcp ...

These commands are used for the Transmission Control Protocol service.

3.58.1. tcp irtt [<milliseconds>]

Display or set the initial round trip time estimate, in milliseconds, to be used for new TCP connections until they can measure and adapt to the actual value. The default is 5000 milliseconds (5 seconds). Increasing this when operating over slow channels will avoid the flurry of retransmissions that would otherwise occur as the smoothed estimate settles down at the correct value. Note that this command should be given before servers are started in order for it to have effect on incoming connections.

TCP also keeps a *cache* of measured round trip times and mean deviations (MDEV) for current and recent destinations. Whenever a new TCP connection is opened, the system first looks in this cache. If the destination is found, the cached IRTT and MDEV values are used. If not, the default IRTT value mentioned above is used, along with a MDEV of 0. This feature is fully automatic, and it can improve performance greatly when a series of connections are opened and closed to a given destination (eg. a series of FTP file transfers or directory listings).

3.58.2. tcp kick <tcb_addr>

If there is unacknowledged data on the send queue of the specified TCB, this command forces an immediate retransmission.

3.58.3. tcp mss [<size>]

Display or set the TCP Maximum Segment Size in bytes that will be sent on all outgoing TCP connect request (SYN segments). This tells the remote end the size of the largest segment (packet) it may send. Changing MSS affects only future connections; existing connections are unaffected.

3.58.4. tcp reset <tcb_addr>

Deletes the TCP control block at the specified address.

3.58.5. tcp rtt <tcb_addr> <milliseconds>

Replaces the automatically computed round trip time in the specified TCB with the rtt in milliseconds. This command is useful to speed up recovery from a series of lost packets since it provides a manual bypass around the normal backoff retransmission timing mechanisms.

3.58.6. tcp status [<tcb_addr>]

Without arguments, displays several TCP-level statistics, plus a summary of all existing TCP connections, including TCB address, send and receive queue sizes, local and remote sockets, and connection state. If **tcb_addr** is specified, a more detailed dump of the specified TCB is generated, including send and receive sequence numbers and timer information.

3.58.7. tcp window [<size>]

Displays or sets the default receive window size in bytes to be used by TCP when creating new connections. Existing connections are unaffected.

3.59. telnet <hostid>

Creates a Telnet session to the specified host and enters converse mode.

3.60. tip <iface>

Creates a **tip** session that connects to the specified interface in "dumb terminal" mode. The interface must have already been attached with the **attach** command. Any packet traffic (IP datagrams, etc) routed to the interface while this session exists will be discarded. To close a **tip** session, use the **reset** command. It will then revert to normal **slip, nrs** or **kiss** mode operation.

This feature is primarily useful for manually establishing SLIP connections. At present, only the built-in "com" ports can be used with this command.

3.61. trace [<iface> [off | <btio> [<tracefile>]]]

Controls packet tracing by the interface drivers. Specific bits enable tracing of the various interfaces and the amount of information produced. Tracing is controlled on a per-interface basis; without arguments, **trace** gives a list of all defined interfaces and their tracing status. Output can be limited to a single interface by specifying it, and the control flags can be change by specifying them as well. The flags are given as a hexadecimal number which is interpreted as follows:

```
O - Enable tracing of output packets if 1, disable if 0
I - Enable tracing of input packets if 1, disable if 0
T - Controls type of tracing:
    0 - Protocol headers are decoded, but data is not displayed
    1 - Protocol headers are decoded, and data (but not the
        headers themselves) are displayed as ASCII characters,
        64 characters/line. Unprintable characters are displayed
        as periods.
    2 - Protocol headers are decoded, and the entire packet
        (headers AND data) is also displayed in hexadecimal
        and ASCII, 16 characters per line.
B - Broadcast filter flag. If set, only packets specifically addressed
    to this node will be traced; broadcast packets will not be displayed.
```

If **tracefile** is not specified, tracing will be to the console.

3.62. udp status

Displays the status of all UDP receive queues.

3.63. upload [<filename>]

Opens **filename** and sends it on the current session as though it were typed on the terminal.

3.64. watch

Displays the current software stopwatch values, with min and max readings for each. This facility allows a programmer to measure the execution time of critical sections of code with microsecond resolution. This command is supported only on the IBM PC, and the meaning of each stopwatch value depends on where the calls have been inserted for test purposes; the distribution copy of **net.exe** usually has no stopwatch calls.

3.65. ?

Same as the **help** command.

4. Attach Commands

This chapter details the attach commands for the various hardware interface drivers. Not all of these drivers may be configured into every **net.exe** binary; a list of the available types may be obtained by entering the command **attach ?**.

Some parameters are accepted by several drivers. They are:

4.0.1. <bufsize>

For asynchronous devices (eg. COM ports operating in SLIP or NRS mode) this parameter specifies the size of the receiver's ring buffer. It should be large enough to hold incoming data at full line speed for the longest time that the system may be busy in MS-DOS or the BIOS doing a slow I/O operation (eg. to a floppy disk). A kilobyte is usually more than sufficient.

For synchronous devices (eg. the **scc, hs, pc100, hapn** and **drsi** interfaces operating in HDLC mode), the bufsize parameter specifies the largest packet that may be received on the interface. This should be set by mutual agreement among stations sharing the channel. For standard AX.25 with a maximum I-frame data size of 256 bytes, a value of 325 should provide an adequate safety margin. On higher speed channels (eg. 56kb/s) larger values (eg. 2K bytes) will provide much better performance and allow full-sized Ethernet packets to be carried without fragmentation.

4.0.2. <ioaddr>

The base address of the interface's control registers, in hex.

4.0.3. <vector>

The interface's hardware interrupt (IRQ) vector, in hex.

4.0.4. <iface>

The name (an arbitrary character string) to be assigned to this interface. It is used to refer to the interface in **ifconfig** and **route** commands and in **trace** output.

4.0.5. <mtu>

The Maximum Transmission Unit size, in bytes. Datagrams larger than this limit will be fragmented at the IP layer into smaller pieces. For AX.25 UI frames, this limits the size of the information field. For AX.25 I frames, however, the **ax25 paclen** parameter is also relevant. If the datagram or fragment is still larger than **paclen**, it is also fragmented at the AX.25 level (as opposed to the IP level) before transmission. (See the **ax25 paclen** command for further information).

4.0.6. <speed>

The speed in bits per second (eg. 2400).

4.1. attach 3c500 <ioaddr> <vector> arpa <iface> <qlen> <mtu> [<ip_addr>]

Attach a 3Com 3C501 Ethernet interface. **qlen** is the maximum allowable transmit queue length. If the **ip_addr** parameter is not given, the value associated with a prior **ip address** command will be used.

The use of this driver is not recommended; use the packet driver interface with the loadable 3C501 packet driver instead.

4.2. attach asy <ioaddr> <vector> ax25 | nrs | ppp | slip <iface> <bufsize> <mtu> <speed> [<crv>]

Attach a standard PC "com port" (asynchronous serial port), using the National 8250 or 16550A chip. Standard values on the IBM PC and clones for **ioaddr** and **vector** are 0x3f8 and 4 for COM1, and 0x2f8 and 3 for COM2. If the port uses a 16550A chip, it will be detected automatically and the FIFOs enabled.

4.2.1. ax25

Similar to **slip**, except that an AX.25 header and a KISS TNC control header are added to the front of the datagram before SLIP encoding. Either UI (connectionless) or I (connection-oriented) AX.25 frames can be used; see the **mode** command for details.

4.2.2. nrs

Use the NET/ROM asynchronous framing technique for communication with a local NET/ROM TNC.

4.2.3. ppp

Point-to-Point-Protocol. Encapsulates datagrams in an HDLC-like frame. This is a new Internet standard for point-to-point communication, compatible with CCITT standards.

4.2.4. slip

Serial Line Internet Protocol. Encapsulates IP datagrams directly in SLIP frames without a link header. This is for operation on point-to-point lines and is compatible with 4.2BSD UNIX SLIP.

4.2.5. <crv>

The optional flags are a string of characters "crv": **c** enables RTS/CTS detection, **r** enables RLSD (Carrier Detect) physical line sensing, **v** enables Van Jacobson TCP/IP Header Compression, and is valid only for SLIP.

4.3. attach drsi <ioaddr> <vector> ax25 <iface> <bufsize> <mtu> <ch_a_speed> <ch_b_speed>

N6TTO driver for the Digital Radio Systems PCPA 8530 card. Since there are two channels on the board, two interfaces are attached. They will be named **iface** with 'a' and 'b' appended. **bufsize** is the receiver buffer size in bytes; it must be larger than the largest frame to be received. **ch_a_speed** and **ch_b_speed** are the speeds, in bits/sec, for the A and B channels, respectively.

4.4. attach eagle <ioaddr> <vector> ax25 <iface> <bufsize> <mtu> <speed>

WA3CVG/NG6Q driver for the Eagle Computer card (Zilog 8530).

4.5. attach hapn <ioaddr> <vector> ax25 <iface> <bufsize> <mtu> csma|full

KE3Z driver for the Hamilton Amateur Packet Network adapter (Intel 8273). The **csma | full** parameter specifies whether the port should operate in carrier sense multiple access (CSMA) mode or in full duplex.

4.6. attach hs <ioaddr> <vector> ax25 <iface> <bufsize> <mtu> <keyup_delay> <p>

Attach a DRSI PCPA or Eagle Computer interface card using a special "high speed" 8530 driver. This driver uses busy-wait loops to send and receive each byte instead of interrupts, making it usable with high speed modems (such as the WA4DSY 56kb/s modem) on slow systems. This does have the side effect of "freezing" the system whenever the modem transmitter or receiver is active. This driver can operate only in CSMA mode, and it is recommended that no other interfaces requiring small interrupt latencies be attached to the same machine.

The **keyup_delay** parameter specifies the transmitter keyup delay in byte time intervals. The **p** value specifies the transmitter persistence value in the range 1-255; the corresponding slot time is fixed at one hardware clock tick, about 55 ms on the PC.

As with the other 8530 drivers, this driver actually attaches two interfaces, one for each 8530 channel.

4.7. attach packet <intvec> <iface> <txqlen> <mtu>

Driver for use with separate software "packet drivers" meeting the FTP Software, Inc, Software Packet Driver specification. The driver must have already been installed before the **attach** command is given. Packet drivers in the Ethernet, ARCNET, SLIP, SLFP, and KISS/AX25 classes are supported.

intvec is the software interrupt vector used for communication to the packet driver, and **txqlen** is the maximum number of packets that will be allowed on the transmit queue.

4.8. attach pc100 <ioaddr> <vector> ax25 <iface> <bufsize> <speed>

Driver for the PACCOMM PC-100 (Zilog 8530) card. Only AX.25 operation is supported.

4.9. attach scc <devices> init <addr> <spacing> <Aoff> <Boff> <Dataoff> <intack> <vec> [p|r]<clock> [<hdwe>] [<param>]

PE1CHL driver to initialize a generic SCC (8530) interface board prior to actually attaching it. The parameters are as follows:

4.9.1. <devices>

The number of SCC chips to support.

4.9.2. <addr>

The base address of the first SCC chip (hex).

4.9.3. <spacing>

The spacing between the SCC chip base addresses.

4.9.4. <Aoff>

The offset from a chip's base address to its channel A control register.

4.9.5. <Boff>

The offset from a chip's base address to its channel B control register.

4.9.6. <Dataoff>

The offset from each channel's control register to its data register.

4.9.7. <intack>

The address of the INTACK/Read Vector port. If none, specify 0 to read from RR3A/RR2B.

4.9.8. <vec>

The CPU interrupt vector for all connected SCCs.

4.9.9. <clock>

The clock frequency (PCLK/RTxC) of all SCCs in hertz. Prefix with 'p' for PCLK, 'r' for RTxC clock (for baudrate gen).

4.9.10. <hdwe>

Optional hardware type. The following values are currently supported: 1 - Eagle card, 2 - PACCOMM PC-100, 4 - PRIMUS-PC card (DG9BL), 8 - DRSI PCPA card.

4.9.11. <param>

Optional extra parameter. At present, this is used only with the PC-100 and PRIMUS-PC cards to set the modem mode. The value 0x22 is used with the PC-100 and 0x2 is used with the PRIMUS-PC card.

The **attach scc ... init** command must be given before the interfaces are actually attached with the following command.

4.10. attach scc <chan> slip | kiss | nrs | ax25 <iface> <mtu> <speed> <bufsize> [<call>]

Attach an initialized SCC port to the system. The parameters are as follows:

4.10.1. <chan>

The SCC channel number to attach, 0 or 1 for the first chip's A or B port, 2 or 3 for the second chip's A or B port, etc.

4.10.2. slip | kiss | nrs | ax25

The operating mode of the interface. **slip, kiss** and **nrs** all operate the port hardware in asynchronous mode; **slip** is Internet-standard serial line IP mode, **kiss** generates SLIP frames containing KISS TNC commands and AX.25 packets and **nrs** uses NET/ROM local serial link framing conventions to carry NET/ROM packets. Selecting **ax25** mode puts the interface into synchronous HDLC mode that is suitable for direct connection to a half duplex radio modem.

4.10.3. <speed>

The interface speed in bits per second (eg. 1200). Prefix with 'd' when an external divider is available to generate the TX clock. When the clock source is PCLK, this can be a /32 divider between TRxC and RTxC. When the clock is at RTxC, the TX rate must be supplied at TRxC. This is needed only for full duplex synchronous operation. When this arg is given as 'ext', the transmit and receive clocks are external, and the internal baud rate generator (BRG) and digital phase locked loop (DPLL) are not used.

4.11. Attach Examples

Here are some examples of the attach command:

```
# Attach a 3Com Ethernet controller using the standard 3Com address and
# vector (i.e., as it comes out of the box) to use ARPA-standard encapsulation.
# The receive queue is limited to 5 packets, and outgoing packets larger
# than 1500 bytes will be fragmented
attach 3c500 0x300 3 arpa ec0 5 1500

# Attach the PC asynch card normally known as "com1" (the first controller)
# to operate in point-to-point slip mode at 9600 baud, calling it "sl0".
# A 1024 byte receiver ring buffer is allocated. Outgoing packets larger
# than 256 bytes are fragmented.
attach asy 0x3f8 4 slip sl0 1024 256 9600

# Attach the secondary PC asynch card ("com2") to operate in AX.25 mode
# with an MTU of 576 bytes at 9600 baud with a KISS TNC, calling it "ax0".
# By default, IP datagrams are sent in UI frames
attach asy 0x2f8 3 ax25 ax0 1024 576 9600

# Attach the packet driver loaded at interrupt 0x7e
# The packet driver is for an Ethernet interface
attach packet 0x7e ethernet 8 1500
```

5. FTP Subcommands

During converse mode with an FTP server, everything typed on the console is first examined to see if it is a locally-known command. If not, the line is passed intact to the remote server on the control channel. If it is one of the following commands, however, it is executed locally. (Note that this generally involves other commands being sent to the remote server on the control channel.)

5.1. dir [<file> | <directory> [<local file>]]

Without arguments, **dir** requests that a full directory listing of the remote server's current directory be sent to the terminal. If one argument is given, this is passed along in the LIST command; this can be a specific file or subdirectory that is meaningful to the remote file system. If two arguments are given, the second is taken as the local file into which the directory listing should be put (instead of being sent to the console). The PORT command is used before the LIST command is sent.

5.2. get <remote file> [<local file>]

Asks the remote server to send the file specified in the first argument. The second argument, if given, will be the name of the file on the local machine; otherwise it will have the same name as on the remote machine. The PORT and RETR commands are sent on the control channel.

5.3. hash

A synonym for the **verbose 3** command.

5.4. ls [<file> | <directory> [<local file>]]

ls is identical to the **dir** command except that the "NLST" command is sent to the server instead of the "LIST" command. This results in an abbreviated directory listing, i.e., one showing only the file names themselves without any other information.

5.5. mget <file> [<file> ...]

Fetch a collection of files from the server. File names may include wild card characters; they will be interpreted and expanded into a list of files by the remote system using the NLST command. The files will have the same name on the local system that they had on the server.

5.6. mkdir <remote directory>

Creates a directory on the remote machine.

5.7. mput <file> [<file> ...]

Send a collection of files to the server. File names may include wild card characters; they will be expanded locally into a list of files to be sent. The files will have the same name on the server as on the local system.

5.8. put <local file> [<remote file>]

Asks the remote server to accept data, creating the file named in the first argument. The second argument, if given, will be the name of the file on the remote machine; otherwise it will have the same name as on the local machine. The PORT and STOR commands are sent on the control channel.

5.9. rmdir <remote directory>

Deletes a directory on the remote machine.

5.10. type [a | i | l <bytesize>]

Tells both the local client and remote server the type of file that is to be transferred. The default is 'a', which means ASCII (i.e., a text file). Type 'i' means *image*, i.e., binary. In ASCII mode, files are sent as varying length lines of text in ASCII separated by cr/lf sequences; in IMAGE mode, files are sent exactly as they appear in the file system. ASCII mode should be used whenever transferring text between dissimilar systems (eg.

UNIX and MS-DOS) because of their different end-of-line and/or end-of-file conventions. When exchanging text files between machines of the same type, either mode will work but IMAGE mode is usually faster. Naturally, when exchanging raw binary files (executables, compressed archives, etc) IMAGE mode must be used. Type 'l' (logical byte size) is used when exchanging binary files with remote servers having oddball word sizes (eg. DECSYSTEM-10s and 20s). Locally it works exactly like IMAGE, except that it notifies the remote system how large the byte size is. **bytesize** is typically 8. The type command sets the local transfer mode and generates the TYPE command on the control channel.

5.11. verbose [0 | 1 | 2 | 3]

Set or display the level of message output in file transfers. **Verbose 0** gives the least output, and **verbose 3** the most, as follows:

0 - Display error messages only.
1 - Display error messages plus a one-line summary after each transfer
    giving the name of the file, its size, and the transfer time and rate.
2 - Display error and summary messages plus the progress messages generated
    by the remote FTP server. (This setting is the default.)
3 - Display all messages. In addition, a "hash mark" (#) is displayed for
    every 1,000 bytes sent or received.

If a command is sent to the remote server because it is not recognized locally, the response is always displayed, regardless of the setting of **verbose**. This is necessary for commands like **pwd** (display working directory), which would otherwise produce no message at all if **verbose** were set to 0 or 1.

6. Dialer Subcommands

Each dialer command may (should) have a different dialer file. The file resides in the configuration directory, as specified in the **Installation** section (see chapter 1). A typical dialer file might be:

```
# Set the speed, and toggle DTR to ensure modem is in command mode.
control down
wait 3000
speed 2400
control up
wait 3000
# Dial, and wait for connection
send "atdt555-1212
wait 45000 "CONNECT " speed
wait 2000
# PAD specific initialization
send "
wait 15000 "Terminal ="
send "ppp    0
wait 10000 "
```

6.0.1. control down | up

Control **asy** interface. The **down** option drops DTR and RTS. The **up** option asserts DTR and RTS.

6.0.2. send "string"

This dialer command will write the specified string to the interface. The string quote marks are required, and the string may not contain embedded control characters. However, the standard C string escape sequences are recognized (\0 should not be used).

6.0.3. speed [9600 | 4800 | 2400 | 1200 | 300]

This dialer command will set the speed of the interface to one of the available speeds. If the speed is missing, the speed will be displayed in the dialer session window.

6.0.4. wait <milliseconds> ["test string"] [speed]

If only the time is specified, the dialer pauses for the desired number of milliseconds.

Otherwise, the dialer reads until the test string is detected on the interface. If the string is not detected within the desired time, the autodialer will reset. The string quote marks are required, and the string may not contain embedded control characters. However, the standard C string escape sequences are recognized (\0 should not be used).

Finally, if the *speed* parameter is specified, the dialer will continue to read characters until a non-digit is detected. The string read is converted to an integer, and used to set the interface speed. If the trailing non-digit is not detected within the desired time, or the integer value is not a valid speed, the autodialer will reset.

7. The /ftpusers File

Since MS-DOS is a single-user operating system (some might say it is a glorified bootstrap loader), it provides no access control; all files can be read, written or deleted by the local user. It is usually undesirable to give such open access to a system to remote network users. **Net.exe** therefore provides its own access control mechanisms.

The file **/ftpusers** controls remote FTP and mailbox access. The FTP default is *no* access; if this file does not exist, the FTP server will be unusable. A remote user must first "log in" to the system with the USER and PASS commands, giving a valid name and password listed in **/ftpusers**, before he or she can transfer files.

Each entry in **/ftpusers** consists of a single line of the form

username password /path permissions

There must be exactly four fields, and there must be exactly one space between each field. Comments may be added after the last field. Comment lines begin with '#' in column one.

username is the user's login name.

password is the required password. Note that this is in plain text; therefore it is not a good idea to give general read permission to the root directory. A password of '*' (a single asterisk) means that any password is acceptable.

/path is the allowable prefix on accessible files. Before any file or directory operation, the current directory and the user- specified file name are joined to form an absolute path name in "canonical" form (i.e., a full path name starting at the root, with "./" and "../" references, as well as redundant /'s, recognized and removed). The result MUST begin with the allowable path prefix; if not, the operation is denied. This field must always begin with a "/", i.e., at the root directory.

permissions is a decimal number granting permission for read, create and write operations. If the low order bit (0x1) is set, the user is allowed to read a file subject to the path name prefix restriction. If the next bit (0x2) is set, the user is allowed to create a new file if it does not overwrite an existing file. If the third bit (0x4) is set, the user is allowed to write a file even if it overwrites an existing file, and in addition he may delete files. Again, all operations are allowed subject to the path name prefix restrictions. Permissions may be combined by adding bits, for example, 0x3 (= 0x2 + 0x1) means that the user is given read and create permission, but not overwrite/delete permission.

For example, suppose **/ftpusers** on machine pc.ka9q.ampr.org contains the line

friendly test /testdir 7

A session using this account would look like this:

```
net> ftp pc.ka9q.ampr.org
Resolving pc.ka9q.ampr.org... Trying 128.96.160.1...
FTP session 1 connected to pc.ka9q.ampr.org
220 pc.ka9q.ampr.org FTP version 900418 ready at Mon May 7 16:27:18 1990
Enter user name: friendly
331 Enter PASS command
Password: test [not echoed]
230 Logged in
ftp>
```

The user now has read, write, overwrite and delete privileges for any file under /testdir; he may not access any other files.

Here are some more sample entries in **/ftpusers**:

```
karn foobar / 7          # User "karn" with password "foobar" may read,
                         # write, overwrite and delete any file on the
                         # system.

guest bletch /g/bogus 3  # User "guest" with password "bletch" may read
                         # any file under /g/bogus and its subdirectories,
                         # and may create a new file as long as it does
                         # not overwrite an existing file. He may NOT
                         # delete any files.

anonymous * /public 1    # User "anonymous" (any password) may read files
                         # under /public and its subdirectories; he may
                         # not create, overwrite or delete any files.
```

This last entry is the standard convention for keeping a repository of public files; in particular, the username "anonymous" is an established ARPA convention.

8. The domain.txt File

Net.exe translates domain names (eg. "pc.ka9q.ampr.org") to IP addresses (eg. 128.96.160.3) through the use of an Internet Domain Name resolver and a local "cache" file, **domain.txt**. Whenever the user specifies a domain name, the local cache is searched for the desired entry. If it is present, it is used; if not, and if domain name server(s) have been configured, a query is sent over the network to the current server. If the server responds, the answer is added to the **domain.txt** file for future use. If the server does not respond, any additional servers on the list are tried in a round-robin fashion until one responds, or the retry limit is reached (see the **domain retry** command). If **domain.txt** does not contain the desired entry and there are no configured domain name servers, then the request immediately fails.

If a domain name server is available, and if all references to host-ids in your /**autoexec.net** file are in IP address format, then it is possible to start with a completely empty **domain.txt** file and have **net.exe** build it for you. However, you may wish to add your own entries to **domain.txt**, either because you prefer to use symbolic domain names in your /**autoexec.net** file or you don't have access to a domain server and you need to create entries for all of the hosts you may wish to access.

Each entry takes one line, and the fields are separated by any combination of tabs or spaces. For example:

```
pc.ka9q.ampr.org. IN      A       128.96.160.3
```

IN is the *class* of the record. It means *Internet*, and it will be found in all entries. **A** is the *type* of the record, and it means that this is an *address* record. Domain name **pc.ka9q.ampr.org** therefore has Internet address 128.96.160.3.

Another possible entry is the **CNAME** (Canonical Name) record. For example:

```
ka9q.ampr.org.          IN      CNAME   pc.ka9q.ampr.org.
```

This says that domain name "ka9q.ampr.org" is actually an alias for the system with (primary, or *canonical*) domain name "pc.ka9q.ampr.org." When a domain name having a **CNAME** record is given to **net.exe**, the system automatically follows the reference to the canonical name and returns the IP address associated with that entry.

Entries added automatically by **net.exe** will have an additional field between the domain name and the class (**IN**) field. For example:

```
pc.ka9q.ampr.org. 3600  IN      A       128.96.160.3
```

This is the *time-to-live* value, in seconds, associated with the record received from the server. Clients (such as **net.exe**) caching these records are supposed to delete them after the time-to-live interval has expired, allowing for the possibility that the information in the record may become out of date.

This implementation of **net.exe** will decrement the TTL to zero, but will not delete the record unless the "clean" flag is on (see the **domain cache clean** command). When a remote server is not available, the old entry will be used.

When the *TTL* value is missing (as in the examples above), the record will never expire, and must be managed by hand. Since **domain.txt** is a plain text file, it may be easily edited by the user to add, change or delete records.

Additional types of records, include NS (name server) and SOA (start of authority) may appear in **domain.txt** from remote server responses. These are not currently used by **net.exe** but are retained for future development (such as the incorporation of a domain name server into **net.exe** itself).

9. Setting Bufsize, Paclen, Maxframe, MTU, MSS and Window

Many **net.exe** users are confused by these parameters and do not know how to set them properly. This chapter will first review these parameters and then discuss how to choose values for them. Special emphasis is given to avoiding interoperability problems that may appear when communicating with non-**net.exe** implementations of AX.25.

9.1. Hardware Parameters

9.1.1. Bufsize

This parameter is required by most of **net.exe**'s built-in HDLC drivers (eg. those for the DRSI PCPA and the Paccomm PC-100). It specifies the size of the buffer to be allocated for each receiver port. HDLC frames larger than this value cannot be received.

There is no default **bufsize**; it must be specified in the **attach** command for the interface.

9.2. AX25 Parameters

9.2.1. Paclen

Paclen limits the size of the data field in an AX.25 I-frame. This value does *not* include the AX.25 protocol header (source, destination and digipeater addresses).

Since unconnected-mode (datagram) AX.25 uses UI frames, this parameter has no effect in unconnected mode.

The default value of **paclen** is 256 bytes.

9.2.2. Maxframe

This parameter controls the number of I-frames that **net.exe** may send on an AX.25 connection before it must stop and wait for an acknowledgement. Since the AX.25/LAPB sequence number field is 3 bits wide, this number cannot be larger than 7.

Since unconnected-mode (datagram) AX.25 uses UI frames that do not have sequence numbers, this parameter does *not* apply to unconnected mode.

The default value of **maxframe** in **net.exe** is 1 frame.

9.3. IP and TCP Parameters

9.3.1. MTU

The MTU (Maximum Transmission Unit) is an interface parameter that limits the size of the largest IP datagram that it may handle. IP datagrams routed to an interface that are larger than its MTU are each split into two or more *fragments*. Each fragment has its own IP header and is handled by the network as if it were a distinct IP datagram, but when it arrives at the destination it is held by the IP layer until all of the other fragments belonging to the original datagram have arrived. Then they are reassembled back into the complete, original IP datagram. The minimum acceptable interface MTU is 28 bytes: 20 bytes for the IP (fragment) header, plus 8 bytes of data.

There is no default MTU in **net.exe**; it must be explicitly specified for each interface as part of the **attach** command.

9.3.2. MSS

MSS (Maximum Segment Size) is a TCP-level parameter that limits the amount of data that the *remote* TCP will send in a single TCP packet. MSS values are exchanged in the SYN (connection request) packets that open a TCP connection. In the **net.exe** implementation of TCP, the MSS actually used by TCP is further reduced in order to avoid fragmentation at the local IP interface. That is, the local TCP asks IP for the MTU of the interface that will be used to reach the destination. It then subtracts 40 from the MTU value to allow for the overhead of the TCP and IP headers. If the result is less than the MSS received from the remote TCP, it is used instead.

The default value of **MSS** is 512 bytes.

9.3.3. Window

This is a TCP-level parameter that controls how much data the local TCP will allow the remote TCP to send before it must stop and wait for an acknowledgement. The actual window value used by TCP when deciding how much more data to send is referred to as the *effective window*. This is the smaller of two values: the window advertised by the remote TCP minus the unacknowledged data in flight, and the *congestion window*, an automatically computed time-varying estimate of how much data the network can handle.

The default value of **Window** is 2048 bytes.

9.4. Discussion

9.4.1. IP Fragmentation vs AX.25 Segmentation

IP-level fragmentation often makes it possible to interconnect two dissimilar networks, but it is best avoided whenever possible. One reason is that when a single IP fragment is lost, all other fragments belonging to the same datagram are effectively also lost and the entire datagram must be retransmitted by the source. Even without loss, fragments require the allocation of temporary buffer memory at the destination, and it is never easy to decide how long to wait for missing fragments before giving up and discarding those that have already arrived. A reassembly timer controls this process. In **net.exe** it is (re)initialized with the **ip rtimer** parameter (default 30 seconds) whenever progress is made in reassembling a datagram (i.e., a new fragment is received). It is not necessary that all of the fragments belonging to a datagram arrive within a single timeout interval, only that the interval between fragments be less than the timeout.

Most subnetworks that carry IP have MTUs of 576 bytes or more, so interconnecting them with subnetworks having smaller values can result in considerable fragmentation. For this reason, IP implementors working with links or subnets having unusually small packet size limits are encouraged to use *transparent fragmentation*, that is, to devise schemes to break up large IP datagrams into a sequence of link or subnet frames that are immediately reassembled on the other end of the link or subnet into the original, whole IP datagram without the use of IP-level fragmentation. Such a scheme is provided in AX.25 Version 2.1. It can break a large IP or NET/ROM datagram into a series of **paclen**-sized AX.25 segments (not to be confused with TCP segments), one per AX.25 I-frame, for transmission and reassemble them into a single datagram at the other end of the link before handing it up to the IP or NET/ROM module. Unfortunately, the segmentation procedure is a new feature in AX.25 and is not yet widely implemented; in fact, **net.exe** is so far the only known implementation. This creates some interoperability problems between **net.exe** and non-**net.exe** nodes, in particular, standard NET/ROM nodes being used to carry IP datagrams. This problem is discussed further in the section on setting the MTU.

9.4.2. Setting paclen and bufsize

The more data you put into an AX.25 I frame, the smaller the AX.25 headers are in relation to the total frame size. In other words, by increasing **paclen**, you lower the AX.25 protocol overhead. Also, large data packets reduce the overhead of keying up a transmitter, and this can be an important factor with higher speed modems. On the other hand, large frames make bigger targets for noise and interference. Each link has an optimum value of **paclen** that is best discovered by experiment.

Another thing to remember when setting **paclen** is that the AX.25 version 2.0 specification limits it to 256 bytes. Although **net.exe** can handle much larger values, some other AX.25 implementations (including digipeaters) cannot and this may cause interoperability problems. Even **net.exe** may have trouble with certain KISS TNCs because of fixed-size buffers. The original KISS TNC code for the TNC-2 by K3MC can handle frames limited in size only by the RAM in the TNC, but some other KISS TNCs cannot.

Net.exe's built-in HDLC drivers (SCC, PC-100, DRSI, etc) allocate receive buffers according to the maximum expected frame size, so it is important that these devices be configured with the correct **bufsize**. To do this, you must know the size of the largest possible frame that can be received. The **paclen** parameter controls only the size of the data field in an I-frame and not the total size of the frame as it appears on the air. The AX.25 spec allows up to 8 digipeaters, so the largest possible frame is (**paclen** + 72) bytes. So you should make **bufsize** at least this large.

Another important consideration is that the more recent versions of NOS improve interrupt response by maintaining a special pool of buffers for use by the receive routines. These buffers are currently fixed in size to 2048 bytes and this can be changed only by editing config.h and recompiling NOS. This limits **bufsize**; in fact, attempting to set a larger value may cause the driver not to work at all. This situation can be detected by running the **memory status** command and looking for a non-zero count of **Ibuffail** events, although these events can also occur occasionally during normal operation.

One of the drawbacks of AX.25 that there is no way for one station to tell another how large a packet it is willing to accept. This requires the stations sharing a channel to agree beforehand on a maximum packet size. TCP is different, as we shall see.

9.4.3. Setting Maxframe

For best performance on a half-duplex radio channel, **maxframe** should always be set to 1. The reasons are explained in the paper *Link Level Protocols Revisited* by Brian Lloyd and Phil Karn, which appeared in the proceedings of the ARRL 5th Computer Networking Conference in 1986.

9.4.4. Setting MTU

TCP/IP header overhead considerations similar to those of the AX.25 layer when setting **paclen** apply when choosing an MTU. However, certain subnetwork types supported by **net.exe** have well-established MTUs, and these should always be used unless you know what you're doing: 1500 bytes for Ethernet, and 508 bytes for ARCNET. The MTU for PPP is automatically negotiated, and defaults to 1500. Other subnet types, including SLIP and AX.25, are not as well standardized.

SLIP has no official MTU, but the most common implementation (for BSD UNIX) uses an MTU of 1006 bytes. Although **net.exe** has no hard wired limit on the size of a received SLIP frame, this is not true for other systems. Interoperability problems may therefore result if larger MTUs are used in **net.exe**.

Choosing an MTU for an AX.25 interface is more complex. When the interface operates in datagram (UI-frame) mode, the **paclen** parameter does not apply. The MTU effectively becomes the **paclen** of the link. However, as mentioned earlier, large packets sent on AX.25 *connections* are automatically segmented into I-frames no larger than **paclen** bytes. Unfortunately, as also mentioned earlier, **net.exe** is so far the only known implementation of the new AX.25 segmentation procedure. This is fine as long as all of the NET/ROM nodes along a path are running **net.exe**, but since the main reason **net.exe** supports NET/ROM is to allow use of existing NET/ROM networks, this is unlikely.

So it is usually important to avoid AX.25 segmentation when running IP over NET/ROM. The way to do this is to make sure that packets larger than **paclen** are never handed to AX.25. A NET/ROM transport header is 5 bytes long and a NET/ROM network header takes 15 bytes, so 20 bytes must be added to the size of an IP datagram when figuring the size of the AX.25 I-frame data field. If **paclen** is 256, this leaves 236 bytes for the IP datagram. This is the default MTU of the **netrom** pseudo-interface, so as long as **paclen** is at least 256 bytes, AX.25 segmentation can't happen. But if smaller values of **paclen** are used, the **netrom** MTU must also be reduced with the **ifconfig** command.

On the other hand, if you're running IP directly on top of AX.25, chances are all of the nodes are running **net.exe** and support AX.25 segmentation. In this case there is no reason not to use a larger MTU and let AX.25 segmentation do its thing. If you choose an MTU on the order of 1000-1500 bytes, you can largely avoid IP-level fragmentation and reduce TCP/IP-level header overhead on file transfers to a very low level. And you are still free to pick whatever **paclen** value is appropriate for the link.

9.4.5. Setting MSS

The setting of this TCP-level parameter is somewhat less critical than the IP and AX.25 level parameters already discussed, mainly because it is automatically lowered according to the MTU of the local interface when a connection is created. Although this is, strictly speaking, a protocol layering violation (TCP is not supposed to have any knowledge of the workings of lower layers) this technique does work well in practice. However, it can be fooled; for example, if a routing change occurs after the connection has been opened and the new local interface has a smaller MTU than the previous one, IP fragmentation may occur in the local system.

The only drawback to setting a large MSS is that it might cause avoidable fragmentation at some other point within the network path if it includes a "bottleneck" subnet with an MTU smaller than that of the local interface. (Unfortunately, there is presently no way to know when this is the case. There is ongoing work within the Internet Engineering Task Force on a "MTU Discovery" procedure to determine the largest datagram that may be sent over a given path without fragmentation, but it is not yet complete.) Also, since the MSS you specify is sent to the remote system, and not all other TCPs do the MSS-lowering procedure yet, this might cause the remote system to generate IP fragments unnecessarily.

On the other hand, a too-small MSS can result in a considerable performance loss, especially when operating over fast LANs and networks that can handle larger packets. So the best value for MSS is probably 40 less than the largest MTU on your system, with the 40-byte margin allowing for the TCP and IP headers. For example, if you have a SLIP interface with a 1006 byte MTU and an Ethernet interface with a 1500 byte MTU, set MSS to 1460 bytes. This allows you to receive maximum-sized Ethernet packets, assuming the path to your system does not have any bottleneck subnets with smaller MTUs.

9.4.6. Setting Window

A sliding window protocol like TCP cannot transfer more than one window's worth of data per round trip time interval. So this TCP-level parameter controls the ability of the remote TCP to keep a long "pipe" full. That is, when operating over a path with many hops, offering a large TCP window will help keep all those hops busy when you're receiving data. On the other hand, offering too large a window can congest the network if it cannot buffer all that data. Fortunately, new algorithms for dynamic controlling the effective TCP flow control window have been developed over the past few years and are now widely deployed. **Net.exe** includes them, and you can watch them in action with the **tcp status <tcb>** or **socket <sockno>** commands. Look at the **cwind** (congestion window) value.

In most cases it is safe to set the TCP window to a small integer multiple of the MSS (eg. 4 times), or larger if necessary to fully utilize a high bandwidth*delay product path. One thing to keep in mind, however, is that advertising a certain TCP window value declares that the system has that much buffer space available for incoming data. **Net.exe** does not actually preallocate this space; it keeps it in a common pool and may well "overbook" it, exploiting the fact that many TCP connections are idle for long periods and gambling that most applications will read incoming data from an active connection as soon as it arrives, thereby quickly freeing the buffer memory. However, it is possible to run **net.exe** out of memory if excessive TCP window sizes are advertised and either the applications go to sleep indefinitely (eg. suspended Telnet sessions) or a lot of out-of-sequence data arrives. It is wise to keep an eye on the amount of available memory and to decrease the TCP window size (or limit the number of simultaneous connections) if it gets too low.

Depending on the channel access method and link level protocol, the use of a window setting that exceeds the MSS may cause an increase in channel collisions. In particular, collisions between data packets and returning acknowledgements during a bulk file transfer may become common. Although this is, strictly speaking, not TCP's fault, it is possible to work around the problem at the TCP level by decreasing the window so that the protocol operates in stop-and-wait mode. This is done by making the window value equal to the MSS.

9.5. Summary

In most cases, the default values provided by **net.exe** for each of these parameters will work correctly and give reasonable performance. Only in special circumstances such as operation over a very poor link or experimentation with high speed modems should it be necessary to change them.