Morning Star X.25

Daemon Version 1.1

User Guide

Revision 1.7
June 23, 1992

Published by
Morning Star Technologies, Inc.
1760 Zollinger Road
Columbus, Ohio 43221
USA

+1 614 451 1883 (voice)
+1 800 558 7827 (voice)
+1 614 459 5054 (FAX)
support@MorningStar.Com (e-mail)

Morning Star X.25
User Guide

1. Introduction to X.25

X.25 was first proposed by the CCITT in 1979 as a general purpose packet switch data format for use between data processing equipment. The proposal has been modified and updated over the years, and has steadily gained support. It is rapidly becoming the protocol of choice for efficient wide-area data communications, many large networks having been built around it. Telenet, Tymnet and Accunet are well known examples in the United States, and most European governments have established their own, such as the German Post Office network and the French Datapac network.

Morning Star X.25 provides an easy and flexible means of transferring data between the host UNIX[1] system and an X.25 network. Users of the UNIX system have complete access to X.25 networks for wide-area data exchange, or simply for system-to-system file transfer in a two node "network." Using the integral X.3/X.28 PAD and X.29 user interfaces, remote login and interactive terminal sessions across large networks are possible. The UNIX host's native TCP/IP implementation can use X.25 virtual circuits as described in RFC-877 to create a wide-area internetwork, providing access to remote hosts using the same TCP/IP protocols and familar commands as on the local area network.

2. Using this Manual

The portions of this manual following this introductory User Guide have been organized in the style of the standard Berkeley UNIX manual pages. User commands are described in Section 1. Section 3 describes the SunLink[2]-compatible socket library. Section 5 describes the format of the configuration files. Miscellaneous interfaces are described in Section 7. The X.25 daemon itself is described in Section 8.

[1] UNIX is a registered trademark of AT&T Bell Laboratories.
[2] SunLink is a registered trademark of Sun Microsystems Inc.

3. Related Manuals

Hardware installation, when necessary, is covered in the *SnapLink Hardware Installation Guide*. Not all Morning Star X.25 installations require additional hardware, so if you are using a serial-port-only product, you won't have any sort of hardware installation guide.

See *x25-socket*(3) for detailed information on using the SunLink-compatible socket interface. See *x25-config*(5) for detailed information on configuring the X.25 daemon.

4. Sample Configuration

We'll go through the steps necessary to configure a small but illustrative installation. Our system, *robin*, is connected to a Tymnet-supplied 9600 baud synchronous modem and to an adjacent machine, *sparrow*, via a synchronous modem eliminator. We want to support 1) logins from remote machines on the network and from *sparrow*, 2) connecting to other machines using the internal *PAD* (packet assembler/disassembler) interface, 3) UUCP to and from *sparrow* and machines on Tymnet, and 4) TCP/IP networking. Most of the installation and configuration process can be accomplished as a normal user, without need to become **root**.

Unpacking the distribution

You received the software distribution as a *tar*(1) archive, either compressed and delivered electronically or delivered on a tape or floppy disk. Create a working directory and unpack your distribution.

If you received a compressed tar file over the network:

```
% ls mstx25*
mstx25.tar.Z
% mkdir /usr/local/src/x25
% cp mstx25.tar.Z /usr/local/src/x25
% cd /usr/local/src
% zcat x25/mstx25.tar.Z | tar xvf –
...lots of output...
%
```

If you have a floppy or tape distribution:

```
% mkdir /usr/local/src/x25
% cd /usr/local/src
```

```
% tar xvf /dev/rst8
...lots of output...
%
```

You might use a device named something other than /dev/rst8, depending upon what type of workstation you're using and what distribution media you received.

There's nothing special about the selection of /usr/local/src/x25 as a place to put the Morning Star X.25 software distribution. You can put it anywhere you like, depending upon your system's local conventions and disk space constraints.

Configuring the installation

Become root and run the installation shell script:

```
% cd x25
% su
Password: (enter your root password here)
# ./Install
...lots of output...
#
```

The **Install** script will move the daemon, library, header files, and manual pages into the appropriate directories. It will move the object modules and/or device drivers into place for building your new kernel, and will ask you to edit your kernel configuration files as needed. It will create HDLC and IP tunnel driver interfaces (e.g. **/dev/hdlc0** and **/dev/tun0**) if possible. It will also put the cleanup script into an accessible place (usually **/etc/x25.uninstall**) so that the X.25 software can be easily removed from the system and a non-X.25 kernel built if desired.

Building and installing a kernel

The contents of the file /usr/local/src/x25/README will describe how to configure and build a new kernel, move it into place, and reboot your system to ensure that it runs correctly. If your system supports dynamically loadable device drivers, you may need only to reboot for them to become available.

Making X.25 start during a normal boot

Whenever *robin* is booted, the X.25 software needs to be started. We add the following lines to the appropriate spot in our machine's boot-time shell script (**/etc/rc.local** on our example machine):

```
if [ –x /usr/etc/x25d –a –r /etc/x25.config –a –r /etc/x25.auth \
   –a –d /usr/adm –a –w /usr/adm/x25.log –a –w /usr/adm/x25.trace ]
then
  /usr/etc/x25d –f /etc/x25.config –a /etc/x25.auth \
   –l /usr/adm/x25.log –t /usr/adm/x25.trace ; \
  (echo –n ' x25d') > /dev/console
fi
```

Creating the X.25 configuration file

Now we can start building our configuration file. See *x25-config*(5) for a description of all the configuration options, and see the Examples/ subdirectory for some simple example configurations.

```
# cd /etc
# cat > x25.config
; X.25 configuration parameters

; Line number aliases
(define Tymnet 0)
(define Sparrow 1)

^D
# chmod 644 x25.config
#
```

Next, we add commands to configure the *PAD* and *X.29* interfaces we require. The daemon can manage a large number of interfaces, but we'll limit ourselves to five of each to simplify the example. We'll also define a TCP port on which the daemon should expect connections from socket library clients.

```
# cat >> x25.config
; PAD devices are used to call other machines
(create-group PAD-group (type PAD) (pty-names "/dev/mcux[0-4]"))

; X.29 devices allow logins from other machines
(create-group Login-group (type X.29) (max-ptys 5))
```

```
; X.25 devices listen to socket library clients
(create-group Socket-group (type X.25) (two-way-svcs 5) (tcp-port 87))
^D
#
```

[3]

Now that we have attached "semantic content" to our devices, move (or link, if you wish) them to more meaningful names. Since the daemon works through the UNIX system's virtual terminal driver, we'll allocate a few pty devices for *PAD* and *X.29* use. It's best to pick pty names from the end of the list that *inetd* would assign to incoming *telnet* sessions. First, make sure that the names we want are not currently in use, then get the pty slave-sides and master-sides for our own use:

```
# cd /dev
# ls mcux* cux*
mcux* not found
cux* not found
#
# mv ttyqb cux0
# mv ttyqc cux1
# mv ttyqd cux2
# mv ttyqe cux3
# mv ttyqf cux4
#
# mv ptyqb mcux0
# mv ptyqc mcux1
# mv ptyqd mcux2
# mv ptyqe mcux3
# mv ptyqf mcux4
#
```

Make sure that *uucico* and friends can get at the "ACU" devices:

```
# chown uucp cux0 cux1 cux2 cux3 cux4
#
```

Next, we define the routing functions that are evaluated whenever a call is sent or received (again, see *x25-config*(5)). We can distinguish

[3] The request for this application's assigned port number is pending with the DDN Network Information Center. In the interim, we recommend using port number 87, described in RFC1060 as "any private terminal link."

X.29 calls from other calls by examining the protocol ID field. In X.29 calls, the first four bytes of the*call-user-data* field are 01 00 00 00:

```
# cd /etc
# cat >> x25.config
; All inbound calls get attached to logins,
;        but collect calls are rejected.
(define in-route-table
        (or (and (is-line Tymnet)
                (facility "01?1")
                (clear 25 65))
                ; Reject collect calls
                ; 25 = "Reverse charging acceptance
                ;      not subscribed"
                ; 65 = "Facility/registration code
                ;      not allowed"
            (and (call-user-data "01000000*")
                (route-to Login-group))
            (route-to Socket-group)))

; All outbound non-Tymnet calls get routed to Sparrow.
(define out-route-table
        (or (and (called-address "3106*")
                (route-to Tymnet))
            ; 3106 is Tymnet's DNIC
            (route-to Sparrow)))

^D
#
```

It is also possible to provide access to *sparrow* from Tymnet and vice versa. The following alternative routing function sends inbound calls from Tymnet with subaddress (the last two digits of the called address) 99 to *sparrow*, and send all of *sparrow's* non-Tymnet addressed calls to *robin*:

```
(define in-route-table
        (or (and (is-line Tymnet)
                (or (and (facility "01?1")
                        (clear 25 65))
                    (and (called-address "*99")
                        (route-to Sparrow))
```

```
                (and (call-user-data "01000000*")
                     (route-to Login-group))
                (route-to Socket-group)))
        (and (is-line Sparrow)
             (or (and (called-address "3106*")
                      (route-to Tymnet))
                 (and (call-user-data "01000000*")
                      (route-to Login-group))
                 (route-to Socket-group)))))
```

The last X.25 items to configure are the communications lines themselves. We have a Morning Star SnapLink SCSI-attached communications adaptor, configured as **/dev/rsd1a**, and the modem is connected to its port number 0. We are a DTE on Tymnet, a 1980 X.25 network; we have 16 two-way virtual circuits; and our assigned X.121 address is 3106555121. See *x25-config*(5) for a description of the *address* subcommand of *activate-line*. *Dte* is the default and therefore could be left out, but we include it below anyway for clarity:

```
# cat >> x25.config
; Tymnet connection
(activate-line Tymnet
               (device "/dev/rsd1a" 0) ; SnapLink port 0
               (dte)
               (address "3106555121")
               (standard-year 1980)
               (vc-ranges nil nil (1 16) nil))

^D
#
```

HDLC drivers for use with the native serial ports are provided with the Sun-4c and NeXT workstation versions of Morning Star X.25. Since *robin* is a Sun SPARCstation 1, we can use the supplied HDLC device driver, although we could just as well use another port on the SnapLink. *Sparrow* has been configured as a DTE, so we must be the DCE end of that circuit. Also, *sparrow* is configured for 1984 X.25 conformance (the default), so we must be as well. The rest of the configuration parameters must also match what the other side of the link is using:

```
# cat >> x25.config
; Sparrow
```



```
(activate-line Sparrow
               (device "/dev/hdlc0") ; Sun or NeXT serial port
               (dce)
               (vc-ranges nil nil (1 32) nil))
^D
#
```

To establish a TCP/IP network between the two machines, it is nessary to configure the X.25 link as an IP interface. If robin has been assigned the IP address [137.175.4.4] and the network is using subnetting, the following should tell the X.25 daemon that the line is to be used for IP traffic:

```
# cat >> /etc/x25.config
(define ddn-mtu-size 576)
(define ddn-address 137.175.4.4)
(define ddn-netmask 255.255.255.0)
(define ddn-address-table ((137.175.4.2 "3106555121")))

(create-group DDN-group (type DDN) (two-way-svcs 10))
^D
#
```

Since the RFC877 encapsulation protocol ID field for IP traffic in the call-user data field is 0xCC, we must also modify our in-route-table to look like

```
(define in-route-table
     (or (and (is-line Tymnet)
              (or (and (facility "01?1")
                       (clear 25 65))
                  (and (called-address "*99")
                       (route-to Sparrow))
                  (and (call-user-data "CC*")
                       (route-to DDN-group))
                  (and (call-user-data "01000000*")
                       (route-to Login-group))
                  (route-to Socket-group)))
         (and (is-line Sparrow)
              (or (and (called-address "3106*")
                       (route-to Tymnet))
                  (and (call-user-data "CC*")
                       (route-to DDN-group))
```

```
(and (call-user-data "01000000*")
      (route-to Login-group))
(route-to Socket-group)))))
```

Establishing the SunLink-compatible socket service

For testing, use the environment variables X25SERVER and X25SERVERPORT to instruct the client code where to find the socket service.[4]

```
# setenv X25SERVER localhost
# setenv X25SERVERPORT 87
#
```

For the long term, you need to tell your system where to find the socket service. Create an alias called **x25-server** in your hosts table, name service database, or YP/NIS or NetInfo hosts map. This alias should be equivalent to the machine on which the daemon will run. Also, create an entry in your *services*(5) database for the "x25" service on TCP port 87:

```
# cat >> /etc/services
x25        87/tcp
^D
#
```

Setting up an initial authorization file

For testing, set up an authorization file that will allow socket access only to clients running on this host. Later, consult *x25-auth*(5) for details of a more complete security setup.

```
# cat > /etc/x25.auth
127.0.0.1
^D
#
```

Testing the installation

[4] The request for this application's assigned port number is pending with the DDN Network Information Center. In the interim, we recommend using port number 87, described in RFC1060 as "any private terminal link."

Our X.25 configuration is now complete, so we can go ahead and start it up for testing:

```
# /usr/etc/x25d –f /etc/x25.config –a /etc/x25.auth –l /usr/adm/x25.log
#
```

And make sure it got started:

```
# cat /usr/adm/x25.log
06/23/92-16:20:47 X.25 started, pid 240
06/23/92-16:20:47
06/23/92-16:20:47 Version 1.1 [23-Jun-1992 11:41:09]
06/23/92-16:20:47
06/23/92-16:20:47 Copyright (c) 1988, 1990 Morning Star Technologies, Inc.
06/23/92-16:20:47 All rights reserved
06/23/92-16:20:47
06/23/92-16:20:47 PAD started on /dev/mcux0, fd 2
06/23/92-16:20:47 PAD started on /dev/mcux1, fd 5
06/23/92-16:20:47 PAD started on /dev/mcux2, fd 6
06/23/92-16:20:47 PAD started on /dev/mcux3, fd 7
06/23/92-16:20:47 PAD started on /dev/mcux4, fd 8
06/23/92-16:20:47 Line 0 activated on SnapLink /dev/rsd1a port 0
06/23/92-16:20:47 Line 1 activated on HDLC serial port /dev/hdlc0 (ttya)
06/23/92-16:20:47 ifconfig ddn1 137.175.4.4 netmask 255.255.255.0 mtu 576
06/23/92-16:20:47 /dev/rsd1a:
06/23/92-16:20:47     Copyright (C) 1991 Morning Star Technologies
06/23/92-16:20:47     All Rights Reserved
06/23/92-16:20:47     Version 1.325 of 8/1/91-16:39:25
06/23/92-16:20:47     MC68302 Rev 2, 256K RAM, 160 buffers
06/23/92-16:20:47 Line 0 activated on SnapLink /dev/rsd1a port 0
06/23/92-16:20:47 Line 1 activated on HDLC serial port /dev/hdlc0 (ttya)
06/23/92-16:20:48 Frame level 0 is connected
06/23/92-16:20:48 Frame level 1 is connected
06/23/92-16:20:49 Packet level 0 is connected
06/23/92-16:20:52 Packet level 1 is connected
#
```

X.25 is installed, configured, running and connected on both of our links. You should be able to login to *robin* from a remote machine via a *PAD* session. You should also be able to *telnet*, *ftp*, etc. between the two.

Configuring UUCP

If you want to want to use UUCP (both *uucp*/*uucico* and *cu*(1)/*tip*(1), over your X.25 connections, it must be configured as described here. *Robin* has the HDB (HoneyDanBer, also known as BNU, Basic Networking Utilities) version of UUCP, but we will also show how to configure the older style of UUCP. Change directory to **/usr/lib/uucp** (or perhaps **/etc/uucp**) and add the following script to the **Dialers** file:

```
X25   =x   * prof\suucp\r\dc\s\T \r \c
```

Add these lines to the **Devices** file:

```
Direct  cux0  -  9600  direct
Direct  cux1  -  9600  direct
Direct  cux2  -  9600  direct
Direct  cux3  -  9600  direct
Direct  cux4  -  9600  direct
X25     cux0  -  9600  X25
X25     cux1  -  9600  X25
X25     cux2  -  9600  X25
X25     cux3  -  9600  X25
X25     cux4  -  9600  X25
```

Last, add entries to the **Systems** file for *sparrow*, as well as for *pigeon*, which is connected to Tymnet:

```
sparrow  Any  X25  9600  0  in:  Urobin  word:  2eet
pigeon  Any  X25  9600  3106123456  in:  Urobin  word:  2itter
```

Now UUCP should work to *sparrow* and *pigeon*. Configure the **Permissions** file according to local security procedures.

Cu should also work:

```
robin 11% cu –l cux0
Morning Star PAD v1.0:  Device 0

*stat

FREE:  Device 0 / Line 0 / X.25 Line Up / FRM Up / PKT Up
                / Line 1 / X.25 Line Up / FRM Up / PKT Up

*c 0

Outgoing call to:   0
```



```
*
CALL CONNECTED

sparrow login: karl
Password:

Terminal set to ansi.

 .. are the STEWED PRUNES still in the HAIR DRYER?

sparrow 1% exit
Mon Jan 7 15:31:23 EDT 1991

CLR Cause: DTE ORIGINATED (0x00), Diagnostic: 0x00
*
Lost Carrier
Disconnected
robin 12%
```

Older UUCP systems don't allow for user-written dialing scripts; dialers were hard-coded into *uucico*. Dialing commands on these systems must be put directly into **L.sys**, the older equivalent of **Systems**:

```
sparrow Any DIR 9600 cux0 * prof\suucp \r c\s0 in: Urobin word: 2eet
pigeon Any DIR 9600 cux0 * prof\suucp \r c\s3106123456 in: Urobin word: 2itter
```

The **L-devices** file should then be configured as if the PAD "ACU" devices were directly-connected lines.

5. Using the X.25 socket library

If you have an application that uses SunLink X.25 sockets, it can be converted to using Morning Star's emulation library simply by relinking it with **-lx25**. For example, if your program source is in file **foo.c**, compile it with the command

```
% cc –o foo foo.c –lx25
```

If your application is in several source modules, they can be linked together like

```
% cc –c foo.c
% cc –c bar.c
% cc –o foo foo.o bar.o –lx25
```

Each C source file that calls an emulated routine (see *x25-socket*(3)) must include at least one of the X.25 include files. If your application was provided as object modules without corresponding source, it cannot be converted to use the Morning Star X.25 socket library, since a key part of the emulation is done by the X.25 include files.

NAME

x25trace – format X.25 line protocol tracing information

SYNOPSIS

x25trace [**–f**] [**–F**] [**–e**] [**–l**line] [**–t**type] [**–o**type] [**–f**] [files]

DESCRIPTION

X25trace reads binary X.25 communication line trace data from the specified files (or the standard input if none are provided) and prints a formatted description of it to the standard output. The options are used to select which portions of the data should be printed; the default is to print all data.

Trace data is obtained from the Morning Star X.25 trace file, by default *x25.trace* in the present working directory of the daemon. The raw trace data is **much** smaller than the processed output of *x25trace*.

The line tracing facility is useful for diagnosing X.25 protocol problems at installations where special line monitoring equipment may not be available. The activation of this feature causes additional processing overhead on the host computer, and trace files can consume large amounts of disk space in a short period of time if high line speeds are used and much data traffic occurs. It is recommended that line tracing not be used when not needed.

X25trace is generally useful only with a knowledge of the format and use of the X.25 protocol, and is used primarily under the direction of the Morning Star support personnel for solving communication problems.

OPTIONS

–f Watch forever, or until receiving a SIGINT or SIGQUIT from the controlling terminal.

–F Like **–f**, but start at the end of the input file.

–e Initialize the state of all communication lines to *extended* (modulus 128 mode, as opposed to modulus 8 mode). Sending or receiving a *SABM* or *SARM* on a particular line will clear this flag; sending or receiving a *SABME* will likewise set it. The **–e** flag is useful when examining trace files of connections using modulus 128 mode where recording was begun after the frame levels became connected.

–l*line* Print trace information from one of the controller's communication lines. Default is all lines and **–l** may be given multiple times to select any subset of the lines on the controller.

−t *frame_type* Only print transmitted or received frames of the specified type. Default is **−t all**. The **−t** option may be specified several times or used in conjunction with the **−o** option to select any set of frame types to print.

−o *frame_type* Omit frames of the specified type. Note that options are parsed left to right so that later **−t** options may undo the effect of a **−o**.

The following frame types may be specified with the **−t** or **−o** options:

all Print all frames (U-frames or S-frames) and packets (I-frames) of all types.

err Equivalent to **−t u_frmr −t u_bad −t s_rnr −t s_rej −t s_bad −t i_rnr −t i_rej −t i_restart −t i_diagnostic −t i_bad**.

i_bad Print unrecognized I-frames.

i_call Print call request, incoming call, call accepted and call connected packets.

i_clear Print clear request, clear indication and clear confirmation packets.

i_data Print all modulus-8 or modulus-128 data packets.

i_diagnostic Print diagnostic packets.

i_frame Print all I-frames.

i_interrupt Print interrupt and interrupt confirmation packets.

i_nondata Equivalent to **−t i_rr −t i_rnr −t i_rej −t i_call −t i_clear −t i_interrupt −t i_reset −t i_restart −t i_registration −t i_diagnostic.**

i_registration Print registration request and registration confirmation packets.

i_rej Print *REJ* packets.

i_reset Print reset request, reset indication and reset confirmation packets.

i_restart Print restart request, restart indication and restart confirmation packets.

i_rnr Print *RNR* (receive not ready) packets.

i_rr Print *RR* (receive ready) packets.

s_bad Print unrecognized S-frames (supervisory).

s_frame Print all S-frames (supervisory).

s_rej Print *REJ* (reject) frames.

s_rnr Print *RNR* (receive not ready) frames.

s_rr Print *RR* (receive ready) frames.

u_bad Print unrecognized U-frames (unnumbered).

u_disc Print *DISC* (disconnect) frames.

u_dm Print *DM* (disconnected mode) frames.

u_frame Print all U-frames.

u_frmr Print *FRMR* (frame reject) frames.

u_sabm Print *SABM* (set asynchronous balanced mode) and *SABME* (set asynchronous balanced mode, extended) frames.

u_sarm Print *SARM* (set asynchronous response mode) frames.

u_ua Print *UA* (unnumbered acknowledgment) frames.

EXAMPLE

To print all I-frames or error-type frames transmitted or received on line 1 except for RR packets:

x25trace -l1 -t i_frame -t err -o i_rr x25.trace

SEE ALSO

x25(7).

COPYRIGHT INFORMATION

NAME

x25-socket, socket, connect, bind, listen, accept, getsockname, getpeername, recv, send, read, write, close, ioctl, perror – SunLink emulation socket interface for Morning Star X.25

SYNOPSIS

```
#include <sundev/syncstat.h>
#include <netx25/x25_pk.h>
#include <netx25/x25_ctl.h>
#include <netx25/x25_ioctl.h>

int socket(domain, type, protocol)
int domain, type, protocol;

int connect(s, addr, addrlen)
int s;
CONN_DB *addr;
int addrlen;

int bind(s, addr, addrlen)
int s;
CONN_DB *addr;
int addrlen;

int listen(s, backlog)
int s, backlog;

int accept(s, addr, addrlenp)
int s;
CONN_DB *addr;
int *addrlenp;

int getsockname(s, addr, addrlenp)
int s;
CONN_DB *addr;
int *addrlenp;

int getpeername(s, addr, addrlenp)
int s;
CONN_DB *addr;
int *addrlenp;
```



```
int recv(s, buf, count, flags)
int s;
char *buf;
int count, flags;

int send(s, buf, count, flags)
int s;
char *buf;
int count, flags;

read(s, buf, count)
int s;
char *buf;
int count;

write(s, buf, count)
int s;
char *buf;
int count;

close(s)
int s;

int ioctl(s, command, arg)
int s, command;
caddr_t arg;

perror(message)
char *message;
```

DESCRIPTION

The Morning Star socket emulation interface is designed to be source code compatible with Sun Microsystems' SunLink X.25 packet interface. This is accomplished by 1) header files that contain data structures and defined constants compatible with SunLink X.25, plus macros that redefine several common system calls, replacing them with internal emulation routines; 2) a subroutine library containing those internal emulation routines; and 3) a TCP port on the X.25 daemon providing communication between the main X.25 process and the emulation software.

To build a program using the socket interface, first #include one or more of the following files:

```
# include <sundev/syncstat.h>
# include <netx25/x25_ctl.h>
# include <netx25/x25_ioctl.h>
# include <netx25/x25_pk.h>
```

Each of these #includes **<netx25/x25_emul.h>** which contains definitions for constants and data structures used by the X.25 socket interface as well as redefinitions of several system calls (for example, *socket* is renamed to *__x25_socket*) so that their functions can be emulated by library routines in the *−lx25* function library.

The functions emulated are:

```
int socket(domain, type, protocol)
int domain, type, protocol;
```

If *domain* is **AF_X25** and *type* is **SOCK_STREAM** and *protocol* is **0**, a connection will be established to **x25d**, the X.25 daemon, and the returned descriptor will reference an X.25 socket. Otherwise, the arguments will be passed to the normal system *socket* call.

A TCP connection will be established to the X.25 daemon using an **AF_INET** domain socket of type **SOCK_STREAM**, protocol **0**. The address called is **x25-server**, or *getenv*("X25SERVER") if it exists in the environment, and the port number is **x25** or *getenv*("X25SERVERPORT"), if it exists. If the system *socket* call fails, the emulated *socket* call will fail, with *errno* containing the system error number.

In addition to any system errors possible for this type of connection, the following errors can occur:

[ENOMEM] Memory could not be allocated for internal data structures.

[ENXIO] The *connect* to the X.25 daemon failed.

[EIO] The X.25 daemon is acting strangely.

The descriptions of the system calls below refer to their behavior when passed an X.25 socket descriptor. Their behavior will remain unchanged for any other type of socket. The single exception to this is that the first of any of these calls (including *socket*) tries to do some one-time initialization, and can return **−1** (*errno* = **ENOMEM**) if unable to allocate space for internal data structures.

In all cases, the possible error conditions listed are in addition to those generated by the operating system.

```
typedef struct conn_db_s
  {
  u_char hostlen;                      /* Addr length in BCD digits */
  u_char host[(MAXHOSTADR + 1) / 2];   /* DTE addr (packed BCD) */
  u_char datalen;                      /* Length of CUD in bytes */
  u_char data[MAXDATA];                /* Call User Data */
  } CONN_DB;

int connect(s, addr, addrlen)
int s;
CONN_DB *addr;
int addrlen;
```

Place an outgoing X.25 call to the X.121 address specified by *addr −> host* and *addr −> hostlen*. Include *addr −> datalen* bytes of data from *addr −> data* in the call-user-data field of the call request packet.

S is the socket to call with, *addr* is a pointer to a CONN_DB structure, and *addrlen* is the size of the structure pointed to by the second argument, typically *sizeof (CONN_DB)*.

The ANY_LINK bit may be set in the *addr −> hostlen* field, but it has no effect; all routing of outgoing calls is controlled by the X.25 daemon.

To include more than MAXDATA bytes in the call-user-data field, precede the *connect* call with one or more calls to the X25_WR_USER_DATA *ioctl*. The *addr −> data* bytes are included first, followed by any X25_WR_USER_DATA bytes.

To force the D-bit to be set in the call request, precede the *connect* call with an X25_SEND_TYPE *ioctl* with the (1<<D_BIT) mask set.

To include include non-default facilities in the call request, precede the *connect* call with an X25_WR_FACILITY *ioctl*.

If an error occurs, **−1** is returned and *errno* contains one of the following:

[ENOMEM] The call request packet being built was too large to send.

[EIO] An indecipherable message was received from the X.25 daemon.

[EFASTDATA] A fast-select clear request packet was received. Use the X25_RD_USER_DATA *ioctl* command to retrieve the clear-user-data. This can only happen if the X25_WR_FACILITY *ioctl* was previously called with the fast_select_type field set to either FAST_ACPT_CLR or FAST_CLR_ONLY.

[ENOTCONN] A normal clear request packet was received. Use the X25_RD_CAUSE_DIAG *ioctl* command to retrieve the clearing cause and diagnostic bytes contained in the clear request packet.

int bind(s, addr, addrlen)
int s;
CONN_DB *addr;
int addrlen;

Prepare for a *listen* call by defining the X.121 addresses and protocol ID fields that we are looking for in an incoming call request packet.

S is the socket to bind, *addr* is a pointer to a CONN_DB structure, and *addrlen* is the size of the structure pointed to by the second argument, typically (and no greater than) *sizeof (CONN_DB)*.

The *addr* –> *host* field contains X.121 address digits (if any) for matching against an incoming call. The *addr* –> *hostlen* field contains the number of digits in *addr* –> *host*, plus the optional ANY_LINK and ANY_SUBADDRESS bits. Note: Setting *addr* –> *hostlen* to zero with neither ANY_LINK nor ANY_SUBADDRESS set is the same as supplying a zero address length with both ANY_LINK and ANY_SUBADDRESS set.

The *addr* –> *data* field may contain bytes to be matched against the call-user-data field of an incoming call, potentially modified by mask bytes defined with the X25_WR_MASK_DATA *ioctl* command. The *addr* –> *datalen* field contains the number of bytes contained in *addr* –> *data*, plus the optional EXACT_MATCH bit. The description of the *listen* call below explains how all these parameters are used to select an incoming call.

If an error occurs, **–1** is returned and *errno* contains one of the following:

[EINVAL] The addr or addrlen argument didn't make sense.

int listen(s, backlog)
int s, backlog;

Request that incoming X.25 calls be screened and queued for future *accept* calls if they match a set of criteria. Use *bind* and the X25_WR_MASK_DATA, X25_SET_LINK and X25_WR_FACILITY *ioctl* commands to define selection criteria.

The following algorithm is used to decide whether a given incoming call should be delivered to a waiting *listen*.

if (the X25_SET_LINK *ioctl* command has been called but the call was not received on the specified line)
no match

if (the received X.121 called-address is shorter than the *addr* –> *host* digit string passed to *bind*)
no match

if (the *bind* ANY_LINK bit is set and the received called-address field does not end with the *bind* addr –> host digit string)
no match

if (the *bind* ANY_SUBADDRESS bit is set and the received called-address field does not begin with the *addr* –> *host* digit string passed to *bind*)
no match

if (the received call request contains the Reverse Charging facility but the X25_WR_FACILITY *ioctl* command was not issued with the reverse_charge field set to a non-zero value)
no match

if (the received call request contains the Fast Select facility, either With Restriction on Response or With No Restriction on Response, and if Fast Select has not been requested (i.e. X25_WR_FACILITY has not been done, or it was done with fast_select_type = FAST_OFF))
no match

if (X25_WR_FACILITY has been performed with fast_select_type = FAST_CLR_ONLY and the received call request does not contain the Fast Select facility With Restriction on Response)
no match

if (*bind* has been called with the EXACT_MATCH bit set in *addr* –> *datalen* but the received call request is not exactly *addr* –> *datalen* bytes long)
no match

if (*addr* –> *datalen* is greater than the length of the call-user-data field of the received call request)
no match

if (any of the first *addr* –> *datalen* bytes of the call-user-data field (masked with any X25_WR_MASK_DATA bytes) differ from their *addr* –> *data[]* counterparts)
no match

match

If an error occurs, **–1** is returned; otherwise **0** is returned.

int accept(s, addr, addrlenp)
int s;
CONN_DB *addr;
int *addrlenp;

Wait for an incoming call to be selected as a result of an earlier *listen* call.

S is the socket to wait on, *addr* points to a CONN_DB structure, and *addrlenp* points to an integer containing the length of the second argument, *sizeof (CONN_DB)*.

The CONN_DB structure will be filled in with the calling-address and call-user-data fields of the received call request. A call accept packet will be sent unless the X25_CALL_ACPT_APPROVAL *ioctl* command has been called with a non-zero value, or unless the received call is a fast select call. If the incoming call is cleared before we can accept it, **–1** will be returned and *errno* will be set to **EIO**. If the received clear request contained clear-user-data, *errno* will instead be set to **EFASTDATA** and the resulting facilities and clear-user-data can be retrieved with the X25_RD_FACILITIES and

X25_RD_USER_DATA *ioctl* commands.

If a call has been successfully received, a new socket descriptor is returned connected to the new X.25 virtual circuit. If neither the X25_CALL_ACPT_APPROVAL *ioctl* has been issued, nor the call contained a fast select facility field (with or without restriction on response), the call accept packet will have already been sent. Otherwise, use the X25_SEND_CALL_ACPT *ioctl* command to accept the call, or simply *close* the descriptor to clear it.

If an error occurs, **–1** is returned and *errno* is set to one of the following:

[EIO] The incoming call was cleared before we could receive it, or an indecipherable message was received from the X.25 daemon.

[EFASTDATA] The incoming call was cleared before we could receive it, and the clear packet contained a non-empty clear-user-data field. The data can be retrieved with the X25_RD_USER_DATA *ioctl* command on s (the original, listening socket).

int getsockname(s, addr, addrlenp)
int s;
CONN_DB *addr;
int *addrlenp;

Fetch the X.121 address of the local end of the virtual circuit associated with the specified X.25 socket.

S is the socket, *addr* points to a CONN_DB structure, and *addrlenp* points to an integer containing the length of the second argument, *sizeof (CONN_DB)*.

The *addr* –> *host* field will be filled in with the X.121 address digits, and the *addr* –> *hostlen* field will contain the number of digits and the value of the (unused) SUBADR_ONLY bit. The *addr* –> *datalen* field is set to zero.

If successful, **0** is returned; else **–1** is returned and *errno* is set to one of the following:

[EINVAL] The CONN_DB structure specified by *addr* and **addrlenp* didn't look valid.

int getpeername(s, addr, addrlenp)
int s;

CONN_DB *addr;
int *addrlenp;

Fetch the X.121 address of the remote end of the virtual circuit associated with the specified X.25 socket.

S is the socket, *addr* points to a CONN_DB structure, and *addrlenp* points to an integer containing the length of the second argument, *sizeof (CONN_DB)*.

The *addr* –> *host* field will be filled in with the X.121 address digits, and the *addr* –> *hostlen* field will contain the number of digits and the value of the ANY_LINK bit. The *addr* –> *datalen* field is set to zero.

If successful, zero is returned; else **–1** is returned and *errno* is set to one of the following:

[EINVAL] The CONN_DB structure specified by *addr* and **addrlenp* didn't look valid.

int recv(s, buf, count, flags)
int s;
char *buf;
int count, flags;

Read data from the specified X.25 socket. If *flags* is zero, normal in-band data is read; if *flags* is MSG_OOB, out-of-band data (interrupt request packet data) is read.

Recv will normally block until a complete packet sequence (see *send*, below) has arrived. If the packet sequence is larger than the supplied buffer, the remainder is silently discarded. See the X25_HEADER and X25_RECORD_SIZE *ioctl* commands below for alternative receive modes that can be used to avoid data loss. Use the X25_WR_SBHIWAT *ioctl* command to avoid deadlock when complete packet sequences are larger than the client-to-daemon flow-control window size.

int send(s, buf, count, flags)
int s;
char *buf;
int count, flags;

Write data to the specified X.25 socket. If *flags* is zero, normal in-band data is sent; if *flags* is MSG_OOB, the out-of-band data is sent in the data field of an interrupt request packet. The amount of out-of-band data given in a single *send* should not

exceed 1 byte for X.25 networks conforming to the 1980 or earlier X.25 specifications, or 32 bytes for 1984 or later networks.

The data in *buf* will normally be sent as a complete packet sequence. A complete packet sequence is a sequence of data packets, all of which have the M bit set except for the last, and all of which must be the maximum negotiated size except for the last one.

To send data packets with arbitrary combinations of the Q, D and M bits set, use the X25_SEND_TYPE *ioctl* command.

read(s, buf, count)
int s;
char *buf;
int count;

Read in-band data from the specified X.25 socket. Calling *read(s, buf, count)* is identical to calling *recv(s, buf, count, 0).*

write(s, buf, count)
int s;
char *buf;
int count;

Write in-band data to the specified X.25 socket. Calling *write(s, buf, count)* is identical to calling *send(s, buf, count, 0)*.

close(s)
int s;

Discontinue use of the specified socket and clear any X.25 virtual circuit associated with it. The X25_WR_USER_DATA *ioctl* command may be issued before closing to specify data to be included in the clear-user-data field of the clear request packet.

int ioctl(s, command, arg)
int s, command;
caddr_t arg;

Perform a special operation on the given X.25 socket. The type of *arg* depends on the value of command. Consult the following table for a description of each X.25 *ioctl* command and argument type.

Command	*Argument Type*
X25_VERSION	**int *arg;**

Fetch the version number of the X.25 socket implementation. The value returned for the current version is **42**.

SIOCSPGRP **int *arg;**

If a signal handler routine is associated with the SIGURG signal, the SIOCSPGRP command should be used to associate the specified socket with a process group ID. *Arg* should point to an integer containing the process group number of the process to receive SIGURG signals.

X25_GET_LINK **int *arg;**

Fetch the number of the X.25 link carrying the virtual circuit associated with the specified X.25 socket.

X25_SET_LINK **int *arg;**

Require that any incoming calls delivered to a following *accept* call arrive on the specified X.25 link number.

This command has no effect on the routing of outgoing calls; all outbound routing of outgoing calls is controlled by the X.25 daemon.

X25_RD_LCGN **int *arg;**

Fetch the LCN (Logical Channel Number) and LCGN (Logical Channel Group Number) of the virtual circuit associated with this socket. The LCN occupies the lowest eight bits of the integer pointed to by *arg* (mask 0xFF), and the LCGN occupies the four bits above that (mask 0xF00).

X25_SEND_TYPE **int *arg;**

Specify the values of the Q, D and M bits for data packets sent with *send* or *write*. The integer pointed to by *arg* must contain a combination of (1<<M_BIT), (1<<D_BIT) and (1<<Q_BIT).

The M (More) bit indicates that the current packet is not the last in a complete packet sequence. Once set, all subsequent *send* calls contribute data to a single packet sequence until another X25_SEND_TYPE turns it off and allows another *send* to supply the final packet in the sequence. Since X.25 requires that all M-bit packets must be the maximum negotiated size, *send* will not send an M-bit data packet until it accumulates enough data to

entirely fill one.

The M bit will stay set until turned off, but the Q and D bits are automatically reset after sending the last packet in a complete packet sequence.

Note: Changing the values of the Q and D bits within a complete packet sequence is a violation of X.25 and will elicit a clear or reset request from the X.25 daemon.

X25_SEND_TYPE can also be used to set the D bit in the call request packet sent by *connect*.

X25_HEADER **int *arg;**

If the integer pointed to by *arg* is non-zero, header mode is enabled. If *arg* points at a zero, header mode is disabled.

When header mode is enabled, each call to *recv* or *read* will have a one-byte header preceding the data. The header contains a combination of (1<<M_BIT), (1<<D_BIT) and (1<<Q_BIT), indicating the values of those bits in the received data packet.

This command may be issued at any time.

X25_RECORD_SIZE **int *arg;**

The *recv* and *read* calls normally return a complete packet sequence (see *send*, above). If the buffer passed to *recv* is not large enough to accommodate the entire packet sequence, the overflow data is silently discarded. To avoid this behavior, and to reduce the potential buffering requirements, it is possible to limit the amount of data delivered to a single *recv* or *read*.

The integer pointed to by *arg* should be the maximum number of data packets to be read by a *recv* or *read* call. The supplied receive buffer should then be at least as large as this number times the negotiated packet size to keep from losing data.

Note, however, that a single *recv* or *read* will only return data from a single packet sequence, even though it can take several *recv* or *read* calls to read the entire sequence. This means that shorter packet sequences will be returned in a single read less than the maximum size, and the last *recv* in a packet sequence will usually also not fill the buffer.

The setting of record size has no effect when header mode is enabled.

X25_OOB_TYPE **int *arg;**

Determine whether out-of-band data has been received. The integer pointed to by *arg* will be set to either zero, if no out-of-band messages have been received; INT_DATA, if an interrupt request has been received; or VC_RESET, if a reset request has been received.

X25_GET_NLINKS **int *arg;**

Fetch the number of X.25 links configured by the daemon.

X25_CALL_ACPT_APPROVAL **int *arg;**

Set (if *arg* points to a non-zero value) or reset (if *arg* points to a zero) call approval mode, which is initially reset. When *accept* receives an incoming call, a call accept packet is automatically sent unless in call approval mode (or unless an X25_SET_FACILITIES *ioctl* has set fast_select_type to FAST_ACPT_CLEAR or FAST_CLR_ONLY). Use the X25_SEND_CALL_ACPT *ioctl* to send the call accept, or use *close* to clear the call.

X25_SEND_CALL_ACPT **void *arg;**

Send a call accept packet in response either to a call request received in call approval mode or to a fast select call request received after having issued an X25_SET_FACILITIES *ioctl* with fast_select_type = FAST_ACPT_CLEAR. The X25_WR_USER_DATA *ioctl* command may be used to specify the call-user-data to be included in the fast select call accept.

```
typedef struct conn_adr_s
  {
  u_char hostlen;                  /* Addr length in BCD digits */
  u_char host[(MAXHOSTADR + 1) / 2]; /* Addr (packed BCD) */
  } CONN_ADR;
```

X25_RD_LOCAL_ADR **CONN_ADR *arg;**

Fetch the X.121 address of the local end of the virtual circuit associated with the specified X.25 socket. The *host* field will be filled in with the X.121 address digits, and

the *hostlen* field will contain the number of digits and the value of the (unused) SUBADR_ONLY bit. See also *getsockname*.

X25_RD_REMOTE_ADR **CONN_ADR *arg;**

Fetch the X.121 address of the remote end of the virtual circuit associated with the specified X.25 socket. The *addr* −> *host* field will be filled in with the X.121 address digits, and the *addr* −> *hostlen* field will contain the number of digits and the value of the ANY_LINK bit. See also *getpeername*.

X25_WR_LOCAL_ADR **CONN_ADR *arg;**

Specify the calling-address field of a call request packet to be generated by a later call to *connect*. The SUBADR_ONLY bit may be set in the *hostlen* field, but is ignored. If the specified address is two bytes long, however, the X.25 daemon will assume it to be a subaddress and will insert the DNIC and NTN of the X.25 link selected by the routing algorithm onto the front of the subaddress.

X25_RD_HOSTADR **CONN_ADR *arg;**

Fetch the configured X.121 address (DNIC plus NTN) of the X.25 link specified by an earlier call to the X25_SET_LINK *ioctl* command, or the zero link if X25_SET_LINK has not been called.

```
typedef struct mask_data_db_s
  {
  u_char masklen;
  u_char mask[MAXMASK];
  } MASK_DATA_DB;
```

X25_WR_MASK_DATA **MASK_DATA_DB *arg;**

Specify up to MAXMASK bytes of *mask* bytes to be used in screening the call-user-data fields of incoming calls with *listen* and *accept*. Each byte of the call-user-data field of an incoming call is logically ANDed with the corresponding *mask[]* byte before being compared with the *data[]* bytes supplied to *bind*. See the above description of *listen* and *bind*.

typedef struct so_hiwat_db_s

```
    {
    short sendhiwat;
    short recvhiwat;
    } SO_HIWAT_DB;
```

X25_RD_SBHIWAT **SO_HIWAT_DB *arg;**

Fetch the current values of the send and receive flow control high water marks. These values define the maximum amount of unacknowledged data that is allowed between the X.25 daemon and the user process (analogous to the TCP window size). The *sendhiwat* field contains the window size in the user-to-daemon direction, and *recvhiwat* contains the window size in the daemon-to-user direction.

X25_WR_SBHIWAT **SO_HIWAT_DB *arg;**

Set the send and receive flow control high water marks (see X25_RD_SBHIWAT, above). The default value for each direction is 2048 bytes.

```
typedef struct x25_cause_diag_s
    {
    u_char flags;
# define RECV_DIAG            0  /* 0<<0: no cause/diag */
                                 /* 1<<0: recv cause/diag */
# define DIAG_TYPE            1  /* 0<<1: reset cause/diag */
                                 /* 1<<1: clear cause/diag */
# define WAIT_CONFIRMATION    2  /* 0<<2: no wait after */
                                 /* X25_WR_DIAG_CODE */
                                 /* 1<<2: wait after */
                                 /* X25_WR_DIAG_CODE */
    u_char datalen;
    u_char data[64];
    } X25_CAUSE_DIAG;
```

X25_RD_CAUSE_DIAG **X25_CAUSE_DIAG *arg;**

Fetch up to 64 bytes from the data field (the entire packet data field, not just the clear-user-data field) of a received clear request or reset request packet. The *datalen* field will be set to the number of bytes actually copied. The RECV_DIAG bit will be set in flags if two or more bytes (the cause and diagnostic) are copied to *data[]*, and the DIAG_TYPE bit will be set if the packet was a clear

request instead of a reset request.

X25_WR_CAUSE_DIAG **X25_CAUSE_DIAG *arg;**

If the DIAG_TYPE bit is on in *flags*, a clear request packet will be sent, otherwise a reset request will be sent. Up to 64 bytes of data field may be specified in *data* and *datalen*, including the clearing or resetting cause (*data*[0]) and diagnostic (*data*[1]). If the WAIT_CONFIRMATION bit is set, this call will block until the corresponding clear confirm or reset confirm packet is received.

Note that sending a malformatted message, or one that conflicts with the X.25 daemon's configuration, will not return an error but may elicit a reset or clear request in response.

```
typedef struct facility_db_s
    {
    u_char reverse_charge;      /* 0 = don't request or allow */
                                /* reverse charging */
                                /* 1 = request or allow reverse */
                                /*    charging */
    u_short recvpktsize;        /* 0 = default receive pkt size */
                                /* Other sizes: 16, 32, 64, 128, */
                                /* 256, 512, 1024, 2048 or 4096 */
    u_short sendpktsize;        /* Same values as for recvpktsize */
    u_char recvwndsize;         /* 0 = default receive window size */
                                /* Other values: 1–7 or 1–127 */
    u_char sendwndsize;         /* Same values as for recvwndsize */
    u_char recvthruput;         /* 0 = default receive throughput */
                                /* class.  Other values are: */
                                /* 3 = 75   7 = 1200 11 = 19200 */
                                /* 4 = 150  8 = 2400 12 = 48000 */
                                /* 5 = 300  9 = 4800 */
                                /* 6 = 600 10 = 9600 */
    u_char sendthruput;         /* Same values as for recvthruput */
    u_char cug_req;             /* 0 = no CUG; 1 = CUG in */
                                /* cug_index */
    u_char cug_index;           /* Only valid when cug_req == 1 */
                                /* Values from 0x00 to 0x99 */
                                /* (BCD) */
    u_char fast_select_type;    /* Fast select options */
# define FAST_OFF             0 /* No fast select sent or accepted*/
```

```
# define FAST_CLR_ONLY  1  /* Clear fast select calls */
# define FAST_ACPT_CLR  2  /* Clear or accept fast select calls */

  u_char rpoa_req;          /* 0 = no RPOA transit request */
                            /* 1 = use RPOA transit req in */
                            /* rpoa */
  u_short rpoa;             /* Only valid when rpoa_req == 1 */
                            /* Values from 0x0000 to 0x9999 */
                            /* (BCD) */
  u_char stdservice;        /* 0: DDN Basic Service */
                            /* 1: DDN Standard Service */
  u_char osiservice;        /* Must be 0 */
  u_char precedence_req;    /* If != 0, use DDN Call */
                            /* Precedence */
  u_char precedence;        /* DDN Call Precedence; */
                            /* ignored if precedence_req */
                            /* == 0 */
  } FACILITY_DB;
```

X25_RD_FACILITY **FACILITY_DB *arg;**

Read the current facility settings associated with a virtual circuit. After call setup, the negotiated facility values are returned.

X25_WR_FACILITY **FACILITY_DB *arg;**

Set facilities to be used with inbound or outbound calls. Setting a field to zero means that the default values should be used.

For outbound calls, facilities set before calling *connect* will be included in the outgoing call request packet. When waiting for incoming calls with *listen* and *accept*, the *reverse_charge* and *fast_select_type* fields are used when screening incoming calls, and the *recvpktsize*, *sendpktsize*, *recvwndsize*, *sendwndsize*, *recvthruput* and *sendthruput* fields are used to negotiate the flow control parameters sent in the call accept.

If *fast_select_type* is set to FAST_ACPT_CLR or FAST_CLR_ONLY before *listen*/*accept*, then when the call packet arrives and *accept* returns, the *fast_select_type* field will be set to match the value of the fast select facility field contained in the incoming call request. The

X25_WR_USER_DATA *ioctl* may then be used to specify the call-user-data field to be sent in the responding call accept packet (sent with the X25_SEND_CALL_ACPT *ioctl*) or clear request packet (sent with *close*).

If the *stdservice* field is set to a non-zero value, the facility field of an outgoing call request sent by *connect* will contain a facility marker (two octets of zero), followed by the two-octet DDN Standard Service facility (0x0401).

If the *precedence_req* field is set to a non-zero value, the facility field of an outgoing call request sent by *connect* will contain a facility marker (two octets of zero), followed by the two-octet DDN Call Precedence facility (0x080**X** where the call's requested precedence increases as **X** goes from 0 to 3), as defined in the *precedence* field. If *precedence_req* is set to zero, the call will be established at the caller's default precedence.

If both *stdservice* and *precedence_req* are specified, only one facility marker will be sent, followed by the Standard Service facility, followed immediately by the Call Precedence facility.

```
typedef struct user_data_db_s
  {
  u_char datalen;
  u_char data[MAX_USER_DATA];
  } USER_DATA_DB;
```

X25_RD_USER_DATA **USER_DATA_DB *arg;**

Retrieve call-user-data or clear-user-data from a received call request, call confirm or clear request packet. Multiple calls will return up to MAX_USER_DATA bytes each until the entire amount has been extracted. The *datalen* field will be set to zero if there is none left.

When using *accept*, the first MAXDATA bytes will be returned in the *accept* CONN_DB structure. The remaining data must be retrieved with X25_RD_USER_DATA.

X25_WR_USER_DATA **USER_DATA_DB *arg;**

Supply data to be included either in the call-user-data field of a call request packet generated by *accept*, or in the clear-user-data of a clear request generated by a call to *close*. Multiple calls may be used to build fields larger than MAX_USER_DATA.

```
typedef struct link_adr_s
  {
  int linkid;                    /* Physical link/line number */
  u_char hostlen;                /* Addr length in BCD digits */
  u_char host[(MAXHOSTADR + 1) / 2]; /* Local addr in BCD */
  } LINK_ADR;
```

X25_RD_LINKADR **LINK_ADR *arg;**

Fetch the configured X.121 address (DNIC plus NTN) for the link number specified in the *linkid* field. The address will be returned in the *host* field packed two digits per byte, and the number of digits copied will be returned in the *hostlen* field.

perror(message) char *message;

Print the supplied message to stderr, followed by a colon, a space, the text of the error message associated with the global integer *errno*, and followed by a newline. The only difference between the emulated *perror* and the normal *perror* is that the emulated version knows about the **EFASTDATA** *errno* value.

BUGS

The ANY_LINK bit in connect()'s addr.hostlen field is ignored. All call routing is done by the out-route-table symbol defined in the X.25 server's config file.

When an emulated socket is put into non-blocking mode, select() can indicate that the socket is ready to be read from and yet have the subsequent read() fail with errno set to EWOULDBLOCK. User code must ignore read errors with errno set to EWOULDBLOCK when using non-blocking mode.

Sun's product only works as a DTE. This version also can work as a DCE, but will deal improperly with packet and window size negotiation if all the following conditions are met:

1) An outgoing call is routed to a DCE-configured link,

2) The packet and/or window sizes in the initial call request packet are not symmetrical (e.g. receive packet size of 128, send packet size of 1024), and

3) There are multiple physical links configured, at least one of which is a DTE.

SEE ALSO

x25(7), x25-auth(5), x25-ddn(7), x25d(8).

CREDITS

SunLink is a registered trademark of Sun Microsystems Inc.

COPYRIGHT INFORMATION

NAME

tun – IP network tunnel driver

CONFIG

pseudo-device tun[*n*]

SYNOPSIS

#include <sys/tun.h>
open("/dev/tun*n*", mode);

DESCRIPTION

When IP packets are written to **/dev/tun***n*, they will be received by the kernel's IP layer on the network interface **tun***n*. When the kernel's IP layer sends packets to the IP interface **tun***n*, they will be available for reading on **/dev/tun***n*.

Instead of having hardware and an associated kernel interface that support network functions, the **tun** driver allows a network interface to be implemented as a user-space process. While talking to the same set of tunnel drivers on the same system, different network interface processes can implement different IP encapsulation methods, such as RFC 877 for use over CCITT X.25-based public data networks, or RFC 1055 SLIP or RFC 1331/1332 PPP for use over dedicated lines and dialup modems.

The **tun** driver provides support for a pair of devices collectively known as an *IP tunnel*. The two devices comprising a tunnel are known as the *inbound* and *outbound sides*, similar to the pairing between **/dev/tty***n* (the inbound terminal) and **/dev/cu***n* (the outbound 'auto-call unit' available on many systems). The outbound side's minor device number is that of the inbound side plus 128, though they together appear to IP as one interface. If both the inbound and outbound sides of a tunnel device are open, packets received from IP are delivered to only the inbound side.

If a TCP packet received from IP is part of a telnet, rlogin, or FTP command stream, it will be put in a *fast queue*. All packets in the fast queue are delivered to the user before any packets in the normal queue.

IOCTLS

A few special **ioctl**s are provided for use on the **/dev/tun***n* devices to supply the functionality needed by applications programs to emulate real hardware interfaces. The complete list of supported **ioctl**s is:

TUIOSPTPT Set or clear the IFF_POINTOPOINT in the associated network interface. **TUIOSADRMD** Set or clear

'address mode', in which packets read are prefaced with four octets containing the destination IP address in network byte order. The third argument is a pointer to an integer containing either a zero or a one, indicating whether 'address mode' should be cleared or set, respectively. If both 'address mode' and 'packed buffer mode' are set, each packet's length will come first, followed by the packet's destination address, followed by the packet itself.

TUIOGADRMD Get the current status of 'address mode'.

TUIOSPKBMD Set or clear 'packed buffer mode' where multiple packets are encoded in single read/write buffers. The third argument is a pointer to an integer containing either a zero or a one, indicating whether 'packed buffer mode' should be cleared or set, respectively. If set (1), each packet is preceded by four octets representing the next packet's length in octets. The following packet will then be aligned to the next multiple of four octets. If cleared (0), packets will be delivered one per read(3) from the tunnel device. If both 'address mode' and 'packed buffer mode' are set, each packet's length will come first, followed by the packet's address, followed by the packet itself.

TUIOGPKBMD Get the current status of 'packed buffer mode'.

TUIOSPKMAX Set the max number of IP frames to send back in a packet buffer read.

TUIOGPKMAX Get the PKMAX value.

TUIOSPKPAD Set the number of long word zeroes to put on the front of each packet read in packed buffer mode.

TUIOGPKPAD Get the number of pad words.

TUIOSNAME Set the interface name (may only be invoked by the superuser).

TUIOGNAME Get the interface name.

FIONBIO Set or clear non-blocking mode for I/O operations.

EXAMPLES

```
#include <sys/tun.h>
int tun_fd = -1, len;
```

```
char *packet;

tun_fd = open("/dev/tun0", O_RDWR);
ioctl(tun_fd, TUIOSNAME, "du");
len = read(tun_fd, packet, size);
write(tun_fd, packet, len);
```

ERRORS

If a packet is delivered to the interface for an address family other than AF_INET, EAFNOSUPPORT will be returned.

FILES

/dev/tun0 through **/dev/tun127** ‘inbound’ tunnel devices

/dev/tun128 through **/dev/tun255** ‘outbound’ tunnel devices

SEE ALSO

if(4n), ppp.Auth(5), ppp.Devices(5), ppp.Dialers(5), ppp.Filter(5), ppp.Systems(5), pppd(8), RFC 1331, RFC 1332, RFC 1144, RFC 1055, RFC 877.

COPYRIGHT INFORMATION

NAME

x25-auth – X.25 authorization file format

DESCRIPTION

The Morning Star X.25 daemon offers the system manager extreme flexibility in designing a network configuration. With it, any user on any IP-connected workstation may make full use of the X.25 connection in networked applications.

The daemon’s **-a** option is used to designate an authorization file containing patterns that, when matched against a client host name, domain name or internet address, can allow or disallow X.25 packet service. This can be useful when X.25 service should not be provided to every IP-connected host; for example, a company attached to the Internet that wants X.25 service to be available to its own users, and maybe even to a limited set of outside users, but not to the entire world. The company can use this facility to ensure that its X.25 costs are incurred only by its own users; it is also a useful aid in managing security considerations.

OPTIONS

−a authorization-file File describing constraints on and priviliges of users of the SunLink compatibility facilities provided by the daemon. The default in the absence of an authorization file is the appliation of no constraints. The default in the presence of an empty (all white space or comments) or zero-length authorization file is to reject all connection requests.

IDENTITY VERIFICATION

Before the authorization file is consulted, the client’s identity is verified by

using gethostbyaddr() to get the client’s host/domain name and any aliases,

using gethostbyname() to look up the resulting hostname, and

verifying that the client’s address exists in the resulting nameserver record.

Any connection failing these tests is rejected. The only way to disable this test is to not supply an authorization file, which is most strenuously discouraged.

FORMAT

Patterns are one to a line; blank lines are ignored.

Comments begin with a '#', a space, or a tab, and extend to the end of the line. Upper/lower case distinctions are ignored. Each pattern can optionally begin or end (but not both) with a '*' character, which matches any string (even an empty string).

Patterns can also optionally begin with a '!' character, which indicates that a match means to reject, rather than accept, the connection.

The client's fully qualified domain name (such as vax27.eng.bigcorp.com), any aliases discovered by a gethostbyaddr(), and the numeric address (such as 123.45.67.89) are all compared against each pattern in the list.

When the first match is found, the search is complete. If the matching pattern began with a '!' then the connection is rejected, otherwise it is accepted. The rest of the authorization file is ignored until the next connection request.

If no matches are found, the connection is rejected. If it is desired that connections be instead accepted by default, put a line containing the single character '*' at the end of the file.

EXAMPLE

```
#
#      Morning Star X.25 SunLink-X.25-emulation client
#      authorization file
#
# block Joe in the Sales department
!isuzu.sales.bigcorp.com
# block the manufacturing subnet
!123.45.68.*
# allow all other users in our company
*.bigcorp.com
# block everyone else
```


RECOMMENDATIONS

SEE ALSO

x25(7), x25trace(1), x25-ddn(7), x25-pad(7), x25-x29(7), RFC877.

CREDITS

SunLink is a registered trademark of Sun Microsystems Inc.

COPYRIGHT INFORMATION

NAME

x25-config – configuration options for Morning Star X.25

DESCRIPTION

The configuration file, by default found in **x25.config** in the present working directory in which the daemon is started, is used to set and examine configuration options of the Morning Star X.25 communications package.

The configuration may be examined at any time by instructing the daemon to dump its internal data structures. First, edit the configuration file with your favorite text editor, such as *vi*(1) or *emacs*(1).

Commands are in ASCII text; they have a LISP-like format, using parentheses to enclose ''function'' calls and sub-lists of tokens. Tokens are separated by parentheses or white space (spaces, tabs or newlines) and come in several flavors. *Numbers* consist of a string of the digits 0 (zero) through 9. *Strings* are a series of up to 50 characters, none of which may be a double quote, enclosed by double quotes. *Comments* start with a semicolon and end with a newline, and are discarded on input. Everything else is a *symbol*, which includes function names, constant expressions and defined (and undefined) symbols.

All functions consist of a left parenthesis, followed by the function name, optionally followed by arguments (each of which might also be a function call) and ending with a closing right parenthesis. The top-level function types are:

(define *symbol-name value*)
: Assign *value* to the symbol *symbol-name*. Defined symbols can be used to make a large file of configuration commands more readable by assigning symbolic names to, for example, line numbers.

(create-group *group-name arguments*)
: Attach a symbolic name to a group of user interfaces and define attributes for the members of the group.

(activate-line *line-number arguments*)
: Enable (or re-enable, or disable) the named communications line and assign operating parameters.

(line-stat *line*)
: This causes a display of the status counters for the specified line to be written to the log file. If no argument is given, status on all activated lines will be displayed.

(ddn-stat)
: This causes a display of the status counters for the DDN interface to be written to the log file.

(print-config)

(print)

Special Symbols

There are several symbolic names that have specific meanings or side effects when assigned. They are:

(define verbosity *n*)
: Error message logging level; larger values print more messages. Default value is 2. Receipt of a SIGUSR1 increases verbosity level by 1; SIGUSR2 decreases by 1.

(define tracing *expression*)
: Enables packet tracing. Default value is **nil**, meaning packet tracing is disabled. Receipt of a SIGQUIT toggles tracing on or off.

(define buffer-size *n*)
: The size of the largest message which can be received or sent, including frame header (2 or 3 bytes), packet header (3 or 4 bytes) and data. Default value is 135.

(define verbose-line-config *value*)
: When the current configuration is displayed with the *print-config* command, only the parameters that differ from the defaults are printed in *activate-line* statements. If *verbose-line-config* is not equal to **nil**, then *all* parameters are displayed. This can also be used to get a list of all line parameters and their default values.

(define in-route-table *expression*)
: An expression to be evaluated whenever an incoming call arrives. It should have the side effect of dispatching the call to a group or outbound line, else the call will be

cleared.

(define out-route-table *expression*)
An expression to be evaluated whenever an outbound call is sent. It should have the side effect of dispatching the call to an outbound line, else the call will be cleared.

(define ddn-address *address*)
Defines the IP address of this interface.

(define ddn-network *network*)
Defines the class-A network number to be used for MILNET-style IP-to-X.121 algorithmic address translation. The default is 10.

(define ddn-netmask *netmask*)
Defines the IP address mask of the network connected to this interface, expressed as a set of four integers in the range 0-255, separated by periods. The default value for a class-A network will be 255.0.0.0, for a class-B will be 255.

(define ddn-mtu-size *n*)
Defines the maximum-size transmission unit (in octets) of the network connected to this interface. Can range between 128 and 4096. RFC-877 networks usually use 576, which is the default.

(define ddn-address-table *list-of-pairs*)
Lacking a standard Address Resolution Protocol for X.25, this defines a two-way address translation table for DDN circuits. When an IP (Internet Protocol) packet is sent from the host, this table is used to convert the Internet address to its X.121 equivalent. Likewise, when an incoming call is sent to the DDN interface, its calling-address is extracted from the call packet and the matching Internet address is looked up in the table. Each table entry must look like **(a.b.c.d "X.121 address string")**

(define disable-ddn-translation nil)
If set to a non-**nil** value, this disables the automatic translation of a DDN Internet address to its corresponding X.121 address (or vice-versa) if the translation is not already present in ddn-address-table.

(define ddn-reject-bogons *reject-flag*)
If set to **nil**, incoming calls with addresses not listed in the ddn-address-table will be accepted. The call will not carry any return traffic, however, since no Internet address can be associated with the virtual circuit. If non-**nil**, the calls will be rejected. A warning message will be printed in either case if verbosity is 2 or greater.

(define ddn-packet-size *n*)
If set to a non-zero value, then outgoing DDN calls will contain the packet size facility (0x42) requesting *n*-byte packets. *N* must be between 16 and 4096. If it is not a legal packet size, the next larger size will be used.

(define ddn-window-size *n*)
If set to a non-zero value, then outgoing DDN calls will contain the window size facility (0x43) requesting a packet-window-size of the specified size. *N* must be between 1 and 127.

(define ddn-standard-service *standard-service-flag*)
If set to a non-**nil** value, then outgoing calls will have the DDN Standard Service facility (0x0401) inserted into their facility fields following a facility marker (two octets of zero).

(define ddn-call-precedence *precedence*)
If set to a non-**nil** value, then the numeric argument will be used to set the call's precedence request. Legal values are from 0 to 3, inclusive. Outgoing calls will have the DDN Call Precedence facility (0x080**X**

where the call's requested precedence increases as **X** goes from 0 to 3) inserted into their facility fields following a facility marker (two octets of zero).

If both *ddn-standard-service* and *ddn-call-precedence* are specified, only one facility marker will be sent, followed by the Standard Service facility, followed immediately by the Call Precedence facility.

(define ddn-idle-timeout 600)
If set to a non-zero value, DDN circuits will be cleared if they are idle (no traffic in either direction) for the specified number of seconds.

(define ddn-parallel-vcs 1)
This is the maximum number of virtual circuits beyond which the X.25 package will not start extra ones to a particular destination. Additional calls are placed when existing VC queues toward the X.25 network are too full.

(define ddn-parallel-delay 5)
This is the delay, in seconds, between starting up additional virtual circuits to a particular destination. A non-zero value helps to prevent erratic behavior when a connection is starting up.

(define ddn-call-retry-time 60)
This is the minimum time, in seconds, allowed between attempts to place new DDN calls to a destination which is refusing call requests.

(define ddn-min-call-time 10)
This variable sets a minimum time, in seconds, for which any DDN virtual circuit should be kept open. In order to prevent thrashing when demand for DDN virtual circuits exceeds the configured supply, this

timer should be set to a value long enough to allow a call to be set up and a small but adequate amount of data to be transferred before other demand forces the circuit to be cleared. The speed with which TCP reacts is also a variable in this situation. Large timer values may make some sites occasionally unreachable due to resource exhaustion.

(define ddn-buffers-per-vc 10)
This is the number of large (data) buffers to reserve per DDN virtual circuit the next time a DDN create-group is performed (i.e. define this first or it will have no effect). Some TCP implementations will work poorly if the circuit does not behave as the TCP designers anticipated, and do not adapt well to low speed links with limited buffering. The minimum value for ddn-buffers-per-vc is 4, which is probably too small to be useful, and the default is 10. If ftp's or rcp's fail while dropping many of their packets (use ddn-stat), increase this value and restart X.25.

Routing Tables

The *in-route-table* and *out-route-table* symbols are expressions. These expressions can use the following functions:

(and *expressions*)
Evaluates its arguments until one returns **nil**. Returns **t** if all arguments return **t**, otherwise **nil**.

(or *expressions*)
Evaluates its arguments until one returns non-**nil**. Returns **nil** if all arguments return **nil**, otherwise **t**.

(if *expression expression* **[** *expression* **]**)
Evaluates its first argument and, if it is non-**nil**, returns its second argument, else it returns its optional third argument, or **nil** if it is not supplied. *Cond* may be used as a synonym for *if*.

(not *expression*)
: Evaluates its argument and returns **t** if it is **nil**, and **nil** otherwise.

(+ *expressions*)
: Return the sum of its arguments, all of which must evaluate to numeric (integer) values.

(- *expression1 expression2*)
: Return the difference of its two numeric arguments, *expression1 - expression2*.

(< *expression1 expression2*)
: Return **t** if *expression1* is less than *expression2*, else return **nil**. Both arguments must evaluate to numeric (integer) values.

(> *expression1 expression2*)
: Like **<**, but reversed.

(= *expression1 expression2*)
: Like **<** and **>**, but return **t** if the two arguments evaluate to the same numeric value.

(print *expressions*)
: Evaluate each of the named expressions and print their concatenated values to the **log** file, appending a newline to the end.

(n-free-outgoing-vcs *line*)
: Return the number of currently unused virtual circuits available for outgoing calls on the named communications line. If the named line does not exist, has not been configured or is not currently connected, **nil** is returned and a message is printed to the **log** file if *verbosity* is set to 4 or higher.

(route-to *destination*)
: Route a call packet to the named destination. *Destination* must be a line number for outgoing calls, and can be either a group name or a line number for incoming calls. Returns **t** if the call is successfully routed or **nil** upon failure. Reasons for failure include:

  - No such group
  - No inbound-only or two-way SVCs in a group
  - No available destinations in a group
  - No such line configured
  - The packet level is not connected
  - No outgoing VCs are available on a line

(clear *cause diagnostic*)
: Clears the current call. Zero, one or two numeric arguments can be supplied. If *cause* is omitted, 0 (“DTE originated”) is used for the cause field in the clear packet. If both arguments are omitted, 64 (“Call set up, call clearing or registration problem”) is used for the diagnostic. The return value will always be **t**.

(is-line *n*)
: If the call is incoming and arrived on line *n*, then **t** is returned, else **nil** is returned.

(called-address *pattern*)
: If the pattern given matches the called address, then the value is **t**, else **nil**. See Patterns.

(calling-address *pattern*)
: If the pattern given matches the calling address, then return **t** else return **nil**. See Patterns.

(facility *pattern*)
: If the pattern given matches any facility in the call packet, then return **t**, else return **nil**. See Patterns.

(call-user-data *pattern*)
: If the pattern given matches the call user data field, then returns **t**, else **nil**. See Patterns.

Patterns

Several functions used when routing call packets do pattern matches on various parts of the call packet. Patterns are *strings*, a series of characters enclosed by double quotes (e.g. "123"). When matching patterns against portions of call packets, the call packet data is converted to a string of hexadecimal digits, then the pattern is matched against the resulting string. Characters in patterns include hexadecimal digits (case of alphabetic “digits” is ignored), asterisks (“*”), and question marks (“?”). When matching a pattern to a data string, asterisks match any string of any length, question marks match any single digit, and hexadecimal digits must match exactly. At most one asterisk may appear in any pattern.

Here are some examples of expressions that might be found in *in-route-table* or *out-route-table*:

```
(called-address "*12")
; Match sub-address 12

(and (call-user-data "01000000*")
     (route-to Login-group))
     ; Match X.29 protocol ID field

(and (called-address "3110*")
     (route-to Telenet))
     ; Match calls from Telenet sites
     ; (Telenet's DNIC is 3110)

(and (facility "01?1")
     (clear 25 65))
     ; Reject collect calls
     ; 25 = "Reverse charging acceptance not subscribed"
     ; 65 = "Facility/registration code not allowed"
```

Groups

All X.25 network activity must be associated with functional groups. Each function must be attached to a particular user interface and assigned into groups with the *create-group* statement. All facilities within a group are basically clones of each other, sharing a common user interface type, a common virtual circuit setup and a common name.

The following commands can be specified to assign attributes within the *create-group* statement:

(type *name*)
: Group types are **X.25** (a socket programming interface), **X.28** or **PAD** (an X.3/X.28 dial-out interface, similar to an autodial modem), **X.29** (a dial-in interface, similar to an auto-answer modem), and **DDN** (a TCP/IP router with encapsulation as per RFC-877).

(max-ptys *n*)
: Maximum number of virtual terminals to be associated with **X.29** interfaces (inbound PAD).

(program *string*)
: When an incoming call is routed to this X.29 group, run the program in the supplied quoted string. If the string contains spaces, the first token is the program name and the others are command-line arguments. If one of the arguments is the string **calling-address**, it will be replaced by the calling X.121 address in hexadecimal. The default value for *program* is **"/bin/login"**.

(pty-names *names*)
: List of names of virtual terminals to be associated with **X.28** interfaces (outbound PAD).

(tcp-port *port*)
: Number of the TCP port to which clients may connect in order to talk to the X.25 server daemon. We suggest using the value 87, as per RFC-1060.

(vcs *n n n n*)
: Number of permanent virtual circuits, incoming only virtual circuits, two way virtual circuits, and outgoing only virtual circuits (in that order) associated with this group.

  Permanent virtual circuits are not allowed in (type DDN) groups.

  PVCs are only available in the board-level product, not when running x25d over SnapLink.

(pvcs *n*)
: Alternative to *vcs* specification.

  PVCs are only available in the board-level product, not when running x25d over SnapLink.

(in-only-svcs *n*)
: Alternative to *vcs* specification.

(two-way-svcs *n*)
: Alternative to *vcs* specification.

(out-only-svcs *n*)
: Alternative to *vcs* specification.

Communication Lines

Communication lines are configured with the *activate-line* statement. The first time a line is activated, its parameters are set to default values which are then modified by the arguments supplied to *activate-line*. Subsequent invocations of *activate-line* modify (rather than replace) this set parameters, so a statement like

    (activate-line 0 (dce))

which has relatively few options might be issued for a line that has already been activated in order to correct an error in configuration.

The complete configuration can be listed by defining *verbose-line-config* to a non-**nil** value, which will cause the displayed configuration to include the value every parameter rather than only those with non-default values.

Miscellaneous and level 1 parameters:

(reset-to-defaults)
: Cancel the effect of any previous initialization parameters and reload the internal parameters with default values.

(on)
: Enable this X.25 line.

(off)
: Disable this X.25 line (default is *on*).

(device *device-name-string* [*subdevice-number*])
: *device-name-string* indicates the physical device through which the daemon maintains its X.25 connections. A host's first native serial port is named **/dev/hdlc0**. The first SCSI-attached *SnapLink* unit on a Sun is named **/dev/rsd1a**, and on a NeXT is named **/dev/rsd1h**. A *SnapLink* requires the inclusion of *subdevice-number*, to specify the line number for the connection. It is an integer in the range from 0 to the number of ports on the device.

(external-clocking)
: Use the default signal clocking mode.

(internal-clocking *baud-rate*)
: Set the speed in bits per second used for internally clocked lines. The transmit clock on this line will be generated internally, and will be sent to the external device. The actual value used may be rounded (up or

down) to a value supported by a particular hardware module. Note that not all hardware modules support internal clocking, and those that do usually require special physical configuration. The default is *external-clocking*.

Frame level (level 2) parameters:

(dte)
: Act as the DTE (Data Terminal Equipment) side (the user) of a network connection. Also sets the packet level dte parameter.

(dce)
: Act as the DCE (Data Circuit-terminating Equipment) side (the network) of a network connection. Also sets the packet level dce parameter. The default is *dte*.

(frame-window-size *n*)
: The maximum number of frames which can be transmitted at the frame level without acknowledgement before blocking. Both sides of a connection must use the same value for this parameter. The default value is 7.

(max-frame-size *n*)
: Maximum size of an I-frame which can be received. The default value is the value assigned to the symbol *buffer-size*.

(t1-seconds *n*)
: The time in seconds to wait for acknowledgement before the frame level initiates error recovery (default is 3).

(t1-mseconds *n*)
: The T1 timer, but expressed in milliseconds. The default is **(t1-seconds 3)**.

(t2-seconds *n*)
: Frame level will withhold acknowledgement for this many seconds pending return I-frame traffic. The default value is zero.

(t2-mseconds *n*)
: The T2 timer, but expressed in milliseconds. The default value is zero.

(t3-seconds *n*)
: If set to a non-zero value, the T3 timer runs while the line is in the idle condition. If it expires, we declare the frame level down. If non-zero, T3 should be greater than T1. The default is zero (disabled).

(t3-mseconds *n*) The T3 timer, but expressed in milliseconds. The default is zero.

(n2-counter *n*) The number of times the T1 timer is allowed to expire before the frame level takes other recovery action. The default value is 20.

(network-type *n*) Set to 1 (one) if connecting to the DDN (Defense Data Network); set to 0 (zero) for all others (default is zero).

(rr-idling) Poll the connection every T1 seconds when connected but idle. This option allows detecting link loss when the link is otherwise idle.

(no-rr-idling) Don't send polls when idle. The default is *no-rr-idling*.

(extended) Prefer extended (modulus 128) operation. SABME (Set Asynchronous Balanced Mode, Extended) messages will be sent when actively trying to connect the frame level (if *lapb* is specified), and will be accepted properly when received in any case. The default is *no-extended*.

(no-extended) Use modulus 8 frame counters at the frame level. Never send SABME and treat it as an unrecognized frame when received.

(lap) Use LAP (Link Access Procedure) for the frame level protocol. SARM (Set Asynchronous Response Mode) messages will be sent when actively trying to make a connection; SABM (and SABME, if *extended* is specified) will be accepted (and LAPB used) if received.

(lapb) Use LAPB (Link Access Procedure, Balanced) for the frame level protocol. SABM (Set Asynchronous Balanced Mode) (or SABME, if *extended* is specified) messages will be sent when actively trying to make a connection; SABME will be accepted in addition to SABM (and the appropriate protocol used) if *extended* is specified, and

SARM will be treated as an unrecognized frame.

(disc-mseconds *n*) The number of milliseconds to remain in the disconnected state before trying to establish (or reestablish) contact. The frame level may send non-final DM frames at T1 intervals to invite the other end to set up the link. A value of zero means attempt without delay; a value of 65535 means never attempt to connect (or reconnect) after entering the disconnected state, thus making the frame level completely passive. This timer can be used to give the other side of the connection a chance to send the first link setup command and thus establish the protocol to be used. The default value is 1000 milliseconds.

(active-disc) The frame level will come up sending DISC frames.

(passive-disc) The frame level will come up sending SABM, SABME or SARM.

(srej) Allow use of the HDLC/ADCPP SREJ (Selective REJect) command during error recovery. When retransmitting with REJ, up to an entire window's worth of frames may have to be resent, which can be very costly on high delay lines such as satellite links where large windows are desirable. The SREJ command allows the single damaged frame to be resent by itself, such that all following frames do not have to be resent. The SREJ command is **NOT** part of the X.25 specification, but it may be useful on specific point to point links.

(no-srej) Do not use or recognize the SREJ frame-level command. This is the default.

Packet level (level 3) parameters:

(address *nnn*) This should be set to the X.121 address of this line. When an outbound call packet has an empty *calling-address* field, this value is inserted. If an outbound call packet has a *calling-address* field exactly two digits long, it is assumed to be a subaddress, and this digit string is inserted in front of the two digits. The default value is the empty string.

(standard-year *n*)
The year of issue of the X.25 version to adhere to. Values are 0, 1980, 1984 (the default), and 1988. If a non-zero value is selected, several of the *packet-options* and *registration-options* are forced to the appropriate settings.

(default-packet-size *n*)
The number of bytes in the data field of a data packet. Valid settings are 128, 256, 512, 1024, 2048 and 4096. This value is used for the packet size when an incoming or outgoing call does not explicitly use the packet size negotiation facility. This parameter must be less than or equal to the *max-packet-size* parameter. Note that buffer-size must be defined to an appropriately large value.

(max-packet-size *n*)
The maximum number of bytes in the data field of a data packet. Valid settings are 128, 256, 512, 1024, 2048 and 4096. The packet level will not allow packet size negotiations to specify a packet size larger than the value in this parameter. The default value is 128. Note that buffer-size must be defined to an appropriately large value.

(vc-ranges *v1 v2 v3 v4*)
Set the ranges of Logical Channel Numbers (LCNs) for each of the four virtual circuit types. The four parameters specify 1) permanent, 2) switched incoming, 3) switched two-way, and 4) switched outbound virtual circuits. Each parameter is either **nil**, meaning no LCNs of that type, or a list containing two numbers representing the minimum and maximum LCNs of that type. The default setting is

X25-CONFIG(5) X25-CONFIG(5)

(vc-ranges nil nil (1 32) nil)

PVCs are only available in the board-level product, not when running x25d over SnapLink.

(default-packet-window-size *n*)
The default value for the number of data packets that can be outstanding before acknowledgement. Valid settings are 1-7 (1-127 when using packet level extended sequence numbers). This value is used in the absence of any window size negotiation in call setup packets. The default value is 2.

(max-packet-window-size *n*)
Set the maximum window size that may be used on this line. The default value is 7.

(t20-seconds *n*)
The amount of time to wait (rounded up to the nearest multiple of 10 seconds) before retransmitting a restart request packet. Zero means never retransmit. The default value is 180 seconds.

(r20-counter *n*)
Number of times to retransmit the restart request before taking other recovery action. The default value is 255.

(t21-seconds *n*)
The amount of time to wait (rounded up to the nearest multiple of 10 seconds) before clearing a call that has not been accepted. Zero means never send the clear. The default value is 200 seconds.

(t22-seconds *n*)
The amount of time to wait (rounded up to the nearest multiple of 10 seconds) before retransmitting a reset request packet. Zero means never retransmit. The default value is 180 seconds.

(r22-counter *n*)
The number of times to retransmit a reset request before clearing the call. The default value is 1.

(t23-seconds *n*)
The amount of time to wait (rounded up to the nearest multiple of 10 seconds) before

retransmitting a clear request packet. Zero means never retransmit. The default value is 180 seconds.

(r23-counter *n*)
: The number of times to retransmit a clear request before taking retry limit action. The default value is 1.

(t28-seconds *n*)
: The amount of time to wait (rounded up to the nearest multiple of 10 seconds) before retransmitting a registration request packet. Zero means never retransmit. This parameter is only used when the packet level is acting as a *dte*. The default value is 300 seconds.

(r28-counter *n*)
: The number of times to retransmit a registration request before declaring registration a failure. See *packet-option S.4* for more information concerning registration request retries. The default value is 1.

(initial-restart)
: Send a restart request packet when first connected. The default is initial-restart.

(no-initial-restart)
: Don't send a restart request packet when first connected. This is useful for certification testing procedures such as FIPS 100 in which the automatic generation of a restart request after the link comes up may confuse the tester. Keep in mind that the packet level will never get connected if neither end of the link sends a restart packet. The default is initial-restart.

(prefer-ccitt-over-iso)
: Follow CCITT (rather than ISO) rules for timer retry expiration. The ISO convention is to leave the restart or clear request unconfirmed, thus hanging the logical channel. The CCITT convention is to act as if the expected confirmation packet had arrived.

(no-prefer-ccitt-over-iso)
: Follow the ISO rules for timer retry expiration. This is the default.

(lpe-clear)
: Enable local procedure error on repeated clear request commands. This option is used in situations in which the user clears calls and the network is slow in confirming the clears. With this option set, if the user issues the clear request a second time,, the packet level deliberately commits a procedure error to force the network to clear the call. This frees the logical channel for other use. The default is no-lpe-clear.

(no-lpe-clear)
: Sending extra clear requests before receiving a clear confirm have no effect. The default is no-lpe-clear.

(clear-diagnostic)
: Clear request packets may include a diagnostic field. The default is clear-diagnostic.

(no-clear-diagnostic)
: Clear request packets may not include a diagnostic field. The default is clear-diagnostic.

(reset-diagnostic)
: Reset request packets may include a diagnostic field. The default is reset-diagnostic.

(no-reset-diagnostic)
: Reset request packets may not include a diagnostic field. The default is reset-diagnostic.

(restart-diagnostic)
: Restart request packets may include a diagnostic field. The default is restart-diagnostic.

(no-restart-diagnostic)
: Restart request packets may not include a diagnostic field. The default is restart-diagnostic.

(min-packet-size *n*)
: The minimum size allowed in packet size negotiations. Legal values are 16, 32, 64 and 128. The default is 16.

(default-throughput-class *v1 v2*)
: The values to use in throughput class negotiation if none are provided by the user. The first value is the numeric code of the throughput class for data flowing from the packet level toward the network. The

second value is for data flowing from the network toward the packet level. The default for both values is 10 (9600 bits per second). Legal values and the throughput classes they represent are:

3	75 bps
4	150 bps
5	300 bps
6	600 bps
7	1200 bps
8	2400 bps
9	4800 bps
10	9600 bps
11	19200 bps
12	48000 bps

(registration-options *arguments*)
Arguments is a list of the 1.*n*, 2.*n* and S.*n* options that we want to allow to be negotiated by registration packets. Note that the *standard-year* setting can override some options. See the next section for descriptions of the allowable arguments.

(packet-options *arguments*)
Arguments is a list of all 1.*n*, 2.*n* and S.*n* options that should be enabled or disabled for the packet level (X.25 level 3). Note that the *standard-year* setting can override some options. See the next section for descriptions of the allowable arguments.

Facilities and Registration Options

This section describes the packet level facilities, options and registration options that can be specified in the *registration-options* and *packet-options* configuration commands (see the previous section).

Each function argument consists of either a plus (+) or a minus (-), followed by a specific 1.*n*, 2.*n* or S.*n* option. Any options preceded by a plus will be set, any preceded by a minus will be reset, and any not mentioned at all will be left unchanged. Some option settings are overridden by the setting of standard-year. A complete display of the current options settings can be gotten by defining *verbose-line-config*

and issuing the *print-config* command.

CCITT standard X.2 gives a list of packet level facilities. Symbolic names have been given to each facility or feature; the name is usually the X.2 paragraph number. For example, **1.1** refers to *Extended Packet Sequence Mode*, described in paragraph 1.1 of the X.2 specification.

Since these options all work off of subsets of the same master list of parameters, we will document the complete set in one place. The X.2 standard defines a number of facilities that may be offered by DCEs and subscribed to by DTEs. Some of those facilities are subject to negotiation via online facility registration packets. The X.2 parameters that are numbered 1.*n* amount to features that one can subscribe to. The parameters that are numbered 2.*n* represent facilities that may or may not appear in a call setup or call clearing packet. In some cases, an option may be enabled in varying degrees, more than simply on and off. For these cases, the *packet-options* setting may be either 1.*n*.1 or 1.*n*.2, depending on "how much" of the feature should be enabled.

In some cases, for example flow control negotiation, enabling a feature (1.5) implies enabling a facility code (2.5). These features (and facility codes) are so marked in the table below.

The CCITT's distinction between the 1.*n* and 2.*n* facilities is not always clear either. Also, some things, such as logical channel range, are negotiable in registration packets but not identified as a facility by the CCITT. For these we have created facility bits of our own. A supplementary option set, numbered S.*n*, represents additional options that pertain to the behavior of the packet level of X.25. Some of these options are used internally only and do not appear in this list.

In the following tables, values of entries with a usage field containing a *Y* are forced to certain values (overridden) when the *standard-year* parameter is set to one of the values 1980, 1984, or 1988. Those marked with a *U* are set by the *user* independent of the standard year.

The letter *E* means that this calling facility (2.*n*) is *enabled* by setting the corresponding feature bit (1.*n*). Similarly, the letter *F* means that this feature (1.*n*) implies a calling *facility* (2.*n*). The designations *P* or *R* mean that the option is meaningful in the *packet-options* or *registration-options* statements, respectively.

The *packet-options* are those that we have elected to enable. For example, putting *1.7* (packet retransmission) in the *packet-options* statement allows REJ packets to be considered as defined packets and allows the packet level to utilize that feature. The options currently in use can be set either by configuration through *packet-options* or via negotiation using registration packets, for those allowed with the *registration-options* statement.

X.2 Feature	*Usage*
1.1 Extended Packet Sequence Mode	U,P,R
1.2 Nonstandard Default Window Sizes	Y,P,R
1.3 Nonstandard Default Packet Sizes	Y,P,R
1.4 Default Throughput Classes Assignment	Y,P,R
1.5 Flow Control Parameter Negotiation	Y,R,F
1.5.1	Y,P,F
1.5.2	Y,P,F
1.6 Throughput Class Negotiation	Y,R,F
1.6.1	Y,P,F
1.6.2	Y,P,F
1.7 Packet Retransmission	U,P,R
1.8 Incoming Calls Barred	U,P,R
1.9 Outgoing Calls Barred	U,P,R
1.10 One-way Logical Channel Outgoing	U,P
1.11 One-way Logical Channel Incoming	U,P
1.12 Closed User Group	U,P,R
1.13 Closed User Group with Outgoing Access	U,P,R
1.14 Closed User Group with Incoming Access	U,P,R
1.15 Incoming Calls Barred within a Closed User Group	
1.16 Outgoing Calls Barred within a Closed User Group	
1.17 Bilateral Closed User Group	
1.18 Bilateral Closed User Group with Outgoing Access	
1.19 Reverse Charging Acceptance	U,P,R
1.20 Fast Select Acceptance	U,P,R
1.22 Charging Information	U,R
1.22.1	U,P
1.22.2	U,P
1.23 Direct Call	
1.24 Hunt Group	
1.25 On-line Facility Registration	Y,P
1.26 D-bit Modification	U,P,R
1.27 Local Charging Prevention	U,P,R
1.28 Call Redirection	

1.29 Network User Identification	Y,P,R,F
1.31 RPOA Selection per Interface	Y,P,R

X.2 Call Facility	*Usage*
2.1 Closed User Group Selection	Y,P,R
2.2 Bilateral Closed User Group Selection	Y,P,R
2.3 Reverse Charging	Y,P,R
2.4 RPOA Selection per Call	Y,P,R
2.5 Flow Control Parameter Negotiation	Y,E
2.6 Fast Select	Y,P,R
2.7 Throughput Class Negotiation	Y,E
2.8 Abbreviated Address Calling	
2.9 Charging Information	Y,P,R
2.10 Transit Delay Selection and Indication	Y,P,R
2.11 Call Redirection Notification	Y,P,R
2.12 Called Line Address Modified Notification	Y,P,R
2.13 Network User Identification	Y,E
2.14 Closed User Group with Outgoing Access Selection	Y,P,R

Supplementary Option	*Usage*
S.1 Logical Channel Range Negotiable	Y,P,R
S.2 D-bit Allowed	U,P
S.3 Perform Closed User Group Functions	U,P
S.4 DTE Send Registration Request at Startup	U,P
S.5 Check Facilities Field	Y,P
S.6 Allow Undefined Facilities	U,P
S.7 Generate Diagnostic Packet	Y,P
S.8 Check Q-bits in Complete Packet Sequence	Y,P
S.9 Allow Non-zero Cause from DTE	Y,P
S.10 Allow OSI Facilities (1984)	Y,P,R
S.11 1988 Call Deflection	Y,P,R
S.12 1988 A-bit	Y,P
S.12.1	Y,P
S.13 Allow Malformed Clearing Packets	U,P
S.14 Fix Invalid Facilities	U,P
S.15 Fix Undefined Facilities	U,P
S.16 Fix Call/Clear User Data	U,P
S.17 Fix Call/Clear Addresses	U,P
S.18 Allows OSI Facilities (1988)	Y,P,R
S.19 Fixes Apply to Outgoing Packets	U,P

S.20 Fixes Apply to Incoming Packets	U,P
S.32 Charging Info Money Present	U,P
S.34 Charging Info Segments Present	U,P
S.35 Charging Info Duration Present	U,P
S.36 Bit-encoded Facilities Present	Y,P,R

A description of each of the *registration-options* and *packet-options* follows.

1.1 Extended Packet Sequence Mode

This option is not controlled by the *standard-year* parameter. The setting of this option in *registration-options* will be propagated into the "all calls cleared" and "availability of facilities" registration parameters. If it is set in *packet-options* then the packet level will use modulo 128 sequencing for all packets exchanged via this access line. If not selected, modulo 8 sequencing will be used.

1.2 Nonstandard Default Window Sizes

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* will be propagated into the "availability of facilities" registration parameter if the packet level is a DCE. Its setting in *packet-options* controls the legality of the "nonstandard default window size" registration parameter.

1.3 Nonstandard Default Packet Sizes

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* will be propagated into the "availability of facilities" registration parameter if the packet level is a DCE. Its setting in *packet-options* controls the legality of the "nonstandard default packet size" registration parameter.

1.4 Default Throughput Classes Assignment

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* will be propagated into the "availability of facilities" registration parameter if the packet level is a DCE. Its setting in *packet-options* controls the legality of the "default throughput classes assignment" registration parameter.

1.5 Flow Control Parameter Negotiation

This option is controlled by the *standard-year* parameter. Setting 1.5.1 or 1.5.2 in *packet-options* will be propagated into the "negotiated at any time" registration parameter. If 1.5.2 is selected,

then the packet level will perform flow control negotiation using the window size and packet size facilities. The setting of this option is propagated to option 2.5 which allows the relevant facilities to appear in calling packets.

1.6 Throughput Class Negotiation

This option is controlled by the *standard-year* parameter. Setting 1.6.1 or 1.6.2 in *packet-options* will be propagated into the "negotiated at any time" registration parameter. If 1.6.2 is selected, then the packet level will perform throughput class negotiation. The setting of this option is propagated to option 2.7 which allows the relevant facility to appear in calling packets.

1.7 Packet Retransmission

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* will be propagated into the "all calls cleared" and "availability of facilities" registration parameters. If it is set in *packet-options* then the packet level will utilize the packet retransmission feature. When acting as DTE, it will send an REJ packet in response to a data packet out of sequence. When acting as a DCE it will accept a REJ packet and retransmit the requested data packet(s).

1.8 Incoming Calls Barred

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* will be propagated into the "negotiated at any time" registration parameter. If it is set in *packet-options* then incoming calls, that is, calls going from the DCE to the DTE, will be disallowed and cleared by the packet level using a cause of 0x0B and a diagnostic of 0x46.

1.9 Outgoing Calls Barred

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* will be propagated into the "negotiated at any time registration parameter. If it is set in *packet-options* then outgoing calls, that is, calls going from the DTE to the DCE, will be disallowed and cleared by the packet level using a cause of 0x0B and a diagnostic of 0x46.

1.10 One-way Logical Channel Outgoing

This option is not controlled by the *standard-year* parameter. Its setting in *registration-options* affects the treatment of the "logical channel types ranges'' registration parameter in received registration packets. If the option is offered then the one-way outgoing channel range can be changed by this parameter, otherwise the registration attempt will be rejected. Internally this bit is set in *packet-options* if it turns out that there are in fact one-way outgoing channels defined.

1.11 One-way Logical Channel Incoming

This option is not controlled by the *standard-year* parameter. Its setting in *registration-options* affects the treatment of the ''logical channel types ranges'' registration parameter in received registration packets. If the option is offered then the one-way incoming channel range can be changed by this parameter, otherwise the registration attempt will be rejected. Internally this bit is set in *packet-options* if it turns out that there are in fact one-way incoming channels defined.

1.12 Closed User Group

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* determines whether or not any closed user group functions are enabled in the packet level. The setting in *registration-options* currently has no function since there is no registration procedure associated with closed user groups.

1.13 Closed User Group with Outgoing Access

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* is used by the packet level in its evaluation of the rules of tables 24 and 25 of the X.25 standard. The setting in *registration-options* currently has no function since there is no registration procedure associated with closed user groups.

1.14 Closed User Group with Incoming Access

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* is used by the packet level in its evaluation of the rules of tables 24 and 25 of the X.25 standard. The setting in *registration-options* currently has no function since there is no registration procedure associated with closed user groups.

1.15 Incoming Calls Barred within a Closed User Group

There is no meaning given to this facility since it is a property of

the CUG and not an option that applies to the packet level.

1.16 Outgoing Calls Barred within a Closed User Group

There is no meaning given to this facility since it is a property of the CUG and not an option that applies to the packet level.

1.17 Bilateral Closed User Group

The packet level does not implement bilateral closed user group and so has no need of the value of this option. The bilateral closed user group facility is enabled via option 2.2. It is assumed that this facility is implemented at higher levels.

1.18 Bilateral Closed User Group with Outgoing Access

The packet level does not implement bilateral closed user group and so has no need of the value of this option. The bilateral closed user group facility is enabled via option 2.2. It is assumed that this facility is implemented at higher levels.

1.19 Reverse Charging Acceptance

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* will be propagated into the ''negotiated at any time'' and ''availability of facilities'' registration parameters. If it is set in *packet-options* then reverse charge calls will be allowed by the packet level. Only calls that have the reverse charge facility are affected by this option. If the packet level is acting as a DTE then it check incoming calls that it receives from the DCE. If it is acting as a DCE then it checks incoming calls that it is about to send to the DTE.

1.20 Fast Select Acceptance

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* will be propagated into the ''negotiated at any time'' registration parameter. If it is set in *packet-options* then fast select calls will be allowed by the packet level. Only calls that have the fast select facility are affected by this option. If the packet level is acting as a DTE then it checks incoming calls that it receives from the DCE. If it is acting as a DCE then it checks incoming calls that it is about to send to the DTE.

1.22 Charging Information

This option is not controlled by the *standard-year* parameter. Setting either 1.22.1 or 1.22.2 in *packet-options* will be propagated into the "negotiated at any time" and "availability of facilities" registration parameters. The 1.22.2 option must be used to allow the packet level to generate charging information in clearing packets. If this is the case then the options S.32, S.33, and S.34 enable charging information in units of money, segments and duration, respectively.

1.23 Direct Call

This facility is not only not implemented by the packet level, we don't even know what it means.

1.24 Hunt Group

This facility is not implemented by the packet level. Its option setting is therefore irrelevant as there is no registration procedure connected with it. It is assumed that this facility is implemented at higher levels.

1.25 On-line Facility Registration

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* enables the on-line facility registration mechanism of the packet level. If the option is reset then registration packets are considered to be invalid packets.

1.26 D-bit Modification

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* will be propagated into the "all calls cleared" and "availability of facilities" registration parameters. If it is set in *packet-options* then the packet level will perform the D-bit modification function if it is acting as a DCE.

1.27 Local Charging Prevention

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* will be propagated into the "non-negotiable" registration parameter. If it is set in *packet-options* then the packet level will disallow calls that would be charged to the local DTE. If it acts as a DTE then incoming calls are examined for the presence of the reverse charge facility and outgoing calls are examined for the absence of the reverse charge facility. If the packet level acts as a DCE then calls about to be sent to the DTE are given the same examination.

1.28 Call Redirection

The packet level does not implement this facility and there is no registration procedure that concerns it, therefore the option has no meaning. It is assumed that this facility is implemented at higher levels.

1.29 Network User Identification

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* is propagated into option 2.13, the option that enables the NUI facility code in calling packets. The packet level does not implement the NUI mechanism. It is assumed to be implemented at higher levels.

1.31 RPOA Selection per Interface

This option is controlled by the *standard-year* parameter. Its setting in both *registration-options* and *packet-options* are ignored by the packet level. The RPOA facility is assumed to be implemented at higher levels. The per call facility is enabled by option 2.4.

2.1 Closed User Group Selection

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* enables the closed user group facilities (0x03 and 0x47) to be used in calling packets.

2.2 Bilateral Closed User Group Selection

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* enables the bilateral closed user group facility (0x41) to be used in calling packets.

2.3 Reverse Charging

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* enables the reverse charge facility (0x01) to be used in calling packets. Option S.36 must be set in conjunction with this option. Its setting in *packet-options* is propagated into the "availability of facilities" registration parameter.

2.4 RPOA Selection per Call

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* enables the RPOA selection facilities (0x44 and 0xC4) to be used in calling packets. This setting is also propagated into the "availability of facilities" registration parameter if the packet level is a DCE.

2.5 Flow Control Parameter Negotiation

This option is controlled by the *standard-year* parameter. Its value in *packet-options* enables the packet size and window size facilities (0x42 and 0x43) to be used in calling packets.

2.6 Fast Select

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* enables the fast select facility (0x01) to be used in calling packets. Option S.36 must be set in conjunction with this option.

2.7 Throughput Class Negotiation

This option is controlled by the *standard-year* parameter. Its value in *packet-options* is forced to the same value as that of option 1.6. Its setting in *packet-options* enables the throughput class facility (0x02) to be used in calling packets.

2.8 Abbreviated Address Calling

This facility has no facility code associated with it and no registration procedure connected with it. It must be implemented at higher levels. It is therefore not meaningful to the packet level of X.25.

2.9 Charging Information

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* enables the charging facilities (0x04, 0xC1, 0xC2 and 0xC5) to be used in clearing packets. Its setting will be propagated into the "availability of facilities" registration parameter. Note that it is option 1.22 which governs the ability of the packet level to generate charging information in clearing packets.

2.10 Transit Delay Selection and Indication

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* enables the transit delay facility (0x48) to be used in calling packets.

2.11 Call Redirection Notification

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* enables the call redirection notification facility (0xC3) to be used in calling packets.

2.12 Called Line Address Modified Notification

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* enables the called line address modified notification facility (0x08) to be used in calling and clearing packets. Its setting will also be propagated into the "availability of facilities" registration parameter.

2.13 Network User Identification

This option is controlled by the *standard-year* parameter. Its value in *packet-options* is forced to the same value as that of option 1.29. Its setting in *packet-options* enables the network user identification facility (0xC6) to be used in calling packets.

2.14 Closed User Group with Outgoing Access Selection

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* enables the closed user group with outgoing access facilities (0x09 and 0x48) to be used in calling packets.

S.1 Logical Channel Range Negotiable

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* allows the logical channel ranges to be negotiated via registration packets. Its setting in *packet-options* is propagated into the "availability of facilities" registration parameter if the packet level is acting as a DCE.

S.2 D-bit Allowed

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* enables the use of the D-bit in calling packets and data packets. With this option reset calls containing the D-bit will be rejected by the packet level.

S.3 Perform Closed User Group Functions

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* enables the packet level to implement the closed user group facility (rather than simply allowing the facility codes). The packet level's implementation of closed user groups requires that CUG numbers have network-wide significance. Of course, the CUG facilities themselves (1.12, 1.13, 1.14, 2.1 and/or 2.14) must be enabled for this option to have any effect.

S.4 DTE Send Registration Request at Startup

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* causes the packet level, if acting as a DTE, to send a registration request packet to the DCE as soon as the link comes up. The registration parameters will take their initial settings from the rest of the packet level parameters. If the DCE does not accept our parameter settings the packet level will attempt to adapt to the settings offered by the DCE. In this evaluation, the *registration-options* settings are consulted to see which parameters the packet level is allowed to change. If, after several retries, it cannot accept the DCE's parameters, then the packet level will have failed initialization and will be nonfunctional. The packet level will send a "packet level down" notification to upper layers with an indication that registration failed.

S.5 Check Facilities Field

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* enables the packet level to check the facilities field of calling and clearing packets. Resetting this option means that you do not care what facilities appear in the facilities field of packets. None of the facility enable options (2.n) will apply unless this option is also set.

S.6 Allow Undefined Facilities

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* allows undefined facilities to appear in the facilities field of calling and clearing packets. The undefined facilities will be gathered together and placed at the end of the defined facilities (but before any national options marker).

S.7 Generate Diagnostic Packet

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* allows the packet level, when acting as a DCE, to use the X.25 diagnostic packet to report certain errors to the DTE. If this option is reset, then those errors will lead to the silent discarding of the offending packets.

S.8 Check Q-bits in Complete Packet Sequence

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* allows the packet level to make sure that M-bit sequences of data packets are uniform with respect to Q-

bits. A violation will lead to a reset of the virtual circuit.

S.9 Allow Non-zero Cause from DTE

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* allows the DTE to send non-zero cause field values in reset request and clear request packets. If the non-zero cause field is allowed, it will be forced to have the high-order bit set to one. If it is disallowed then the packet level will force all such cause fields to zero irrespective of their value from the DTE. If this option is set, and if the packet level is acting as a DTE, then it will use non-zero cause fields for internally generated reset request and clear request packets. The cause field values will be the same as for a DCE, except that the high order bit will be set. Thus, clear local procedure error will have a value of 0x93 and reset local procedure error a value of 0x85.

S.10 Allow OSI Facilities (1984)

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* allows the CCITT defined OSI facilities defined by the 1984 standard to appear in calling and clearing packets.

S.11 1988 Call Deflection

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* allows the 1988 call deflection parameter to appear in clearing packets.

S.12 1988 A-bit

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* allows the 1988 A-bit to be used in calling and clearing packets. The A-bit occupies the same position as the Q-bit and indicates that the packet contains the new long form of the address block. The new address block format has eight bit address length fields; the old format has four bit length fields. Setting S.12.1 in *packet-options* indicates that the packet level should additionally try to transform incompatible (non-A-bit) call packets into A-bit format. Setting S.12.2 disables A-bit use, but allows the packet level to transform A-bit packets into non-A-bit packets when possible (this is the default).

S.13 Allow Malformed Clearing Packets

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* causes the packet level to accept as valid clear request and clear indication packets that have errors in the address, facilities, or user data fields. If this option is not set then the packet level will treat such packets as invalid and take special recovery action. The CCITT specifies that this option should be reset, so that malformed clearing packets are considered in error. This behavior, however, could conceivably lead to infinite clearing loops if each side sends a clear request to the other that the other side deems to be invalid.

S.14 Fix Invalid Facilities

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* allows the packet level to "fix" calling and clearing packets that contain invalid facilities. By an invalid facility we mean a facility that is disallowed for the packet type or by an option to the packet level, a facility that contains an invalid value, or a facility field in a packet type that is not allowed to have facilities. The means of "fixing" the invalid facility differs depending upon the particular type of problem. Disallowed facilities are simply dropped from the facilities field. Invalid values, for example in window size negotiation, are brought into range, if possible, or the facility dropped if not.

S.15 Fix Undefined Facilities

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* allows the packet level to "fix" calling and clearing packets that contain undefined facilities. Undefined facilities are those that occur before an options marker and are not defined by the CCITT, ISO, or DDN. The facilities are "fixed" by deleting them from the facilities field of the packet.

S.16 Fix Call/Clear User Data

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* allows the packet level to "fix" calling and clearing packets that contain an invalid user data field. A user data field is invalid if it is longer than allowed or is present when no user data field would be allowed for the particular packet type. The user date field is "fixed" by deleting it from the packet.

S.17 Fix Call/Clear Addresses

This option is not controlled by the *standard-year* parameter. Its

setting in *packet-options* allows the packet level to "fix" calling and clearing packets that contain an invalid address field. An address is invalid if it appears where no address is allowed, is too long, or contains non-BCD digits. It is "fixed" by deleting it from the packet.

S.18 Allows OSI Facilities (1988)

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* allows the CCITT defined OSI facilities defined by the 1988 standard to appear in calling and clearing packets.

S.19 Fixes Apply to Outgoing Packets

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* allows the packet level to "fix" calling and clearing packets that are about to be sent to the access line. Options S.14 through S.17 determine which types of errors can be "fixed."

S.20 Fixes Apply to Incoming Packets

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* allows the packet level to "fix" calling and clearing packets that have just been received from the access line. Options S.14 through S.17 determine which types of errors can be "fixed."

S.32 Charging Info Money Present

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* allows the packet level to insert the monetary unit form of charging information in clearing packets (facility 0xC5). Note that parameter 1.22 must also be set in *packet-options* to allow the packet level to insert any charging information facilities in clearing packets.

S.34 Charging Info Segments Present

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* allows the packet level to insert the segment count form of charging information in clearing packets (facility 0xC2). Note that parameter 1.22 must also be set in *packet-options* to allow the packet level to insert any charging information facilities in clearing packets.

S.35 Charging Info Duration Present

This option is not controlled by the *standard-year* parameter. Its setting in *packet-options* allows the packet level to insert the call duration form of charging information in clearing packets (facility 0xC1). Note that parameter 1.22 must also be set in *packet-options* to allow the packet level to insert any charging information facilities in clearing packets.

S.36 Bit-encoded Facilities Present

This option is controlled by the *standard-year* parameter. Its setting in *packet-options* allows the facility 0x01 to appear in the facilities field. That is, it must be set along with options 2.3 and 2.6 to enable reverse charge and fast select.

SEE ALSO

x25(7), x25-pad(7), x25-socket(3), x25-x29(7), x25-ddn(7), RFC-950, RFC-877, RFC-1060.

COPYRIGHT INFORMATION

NAME

x25-ddn – RFC-877 compliant interface of Morning Star X.25

DESCRIPTION

This section describes the RFC-877 interface of Morning Star X.25. Internet RFC-877 describes a mechanism for providing IP (Internet Protocol) transport over X.25 packet networks, as used by the DDN (Defense Data Network, or MILNET) and by CSNET. It can be used as a router between two physically distant TCP/IP networks (typically Ethernet® LANs).

Morning Star X.25 is a daemon process used in UNIX® systems. It supports several configurable user interface options, one of which is an RFC-877 compliant interface to in-kernel IP. See *x25*(7) for configuration information and references to other X.25 user interfaces.

The DDN interface requires the Morning Star IP Tunnel device driver as well as in-kernel IP. These may not be available on all systems.

The DDN interface is enabled with the *create-group* statement (see *x25-config*(5)), but all features are controlled by the settings of defined symbols, set like this:

(define ddn-call-retry-time 10)

The available configuration options are:

ddn-address
: Defines the IP address of this interface.

ddn-netmask
: Defines the IP address mask of the network connected to this interface, expressed as a set of four integers in the range 0-255, separated by periods. The default value for a class-A network will be 255.0.0.0, for a class-B will be 255.255.0.0, and for a class-C will be 255.255.255.0. If your network uses RFC950 subnetting with eight bit subnet masks, it will use the next-larger size netmask than its class would normally indicate.

ddn-mtu-size
: Defines the maximum transmission unit (in octets) of the network connected to this interface.

ddn-call-retry-time
: The time in seconds (default 60) to wait before repeating a call request which was refused. This timer is used to keep resources from being wasted on frequent

call attempts to a destination which is down.

ddn-min-call-time
The time in seconds (default 10) before which no call will ever be forcibly terminated. During periods when demand for virtual circuits exceeds capacity, circuits must be shared between destinations. In order to prevent thrashing in such circumstances, this timer should be set to a value large enough to allow a call to be set up and a reasonable amount of data transferred before other demand forces the circuit to be cleared.

ddn-address-table
This table defines a one-to-one mapping between Internet addresses and X.121 addresses. The symbol's value is a list of lists, each one of which consists of two tokens: an Internet address (two, three or four decimal integers separated by dots (periods)) and an X.121 address (a series of zero to 15 decimal digits enclosed by double quotes).

ddn-network
Defines the class-A network number (default 10) to be used for MILNET-style IP-to-X.121 algorithmic address translation.

disable-ddn-translation
When an address (of either type) cannot be found in the ddn-address-table, it will be translated using the DDN's mapping algorithm if the address looks like a DDN address. This translation attempt can be disabled by setting the symbol disable-ddn-translation to a non-nil value.

DDN Internet addresses look like *ddn-network*.*h*.0.*i*, and correspond to X.121 addresses "00000*iiihh*00", if *i* is less than 64, or "00001*rrrrr*00" otherwise, where $r = (h * 256) + i$. All values are represented in decimal, padded on the left with zeros if necessary. *ddn-network* defaults to 10, the old

ARPAnet Class-A network number. Incoming X.121 DDN addresses may also contain a trailing two-digit "sub-address", which is not used. See the Assigned Numbers RFC, currently RFC1060, for further details on ARPAnet and MILnet X.25 address mappings and mappings between IP addresses and X.121 Public Data Network Addresses.

ddn-reject-bogons
If set to a non-nil value, then incoming calls whose calling address field cannot be mapped into an Internet address will be rejected. If ddn-reject-bogons is set to **nil** (the default), such calls will be accepted. No return traffic will ever be sent over such circuits, however, since the Internet address of the caller is not known. A message will be printed to the log file in either case when an unrecognized call arrives if the message verbosity is two or greater.

ddn-packet-size
If set to a non-zero value, then outgoing DDN calls will contain the packet size facility (0x42) requesting packets of the specified size, which must be between 16 and 4096. If a legal packet size is not specified, the next larger size will be used.

ddn-window-size
If set to a non-zero value, then outgoing DDN calls will contain the window size facility (0x43) requesting a packet-window-size of the specified size. Values must be between 1 and 127.

ddn-standard-service
If set to a non-nil value, then outgoing calls will have the DDN Standard Service facility (0x0401) inserted into their facility fields following a facility marker (two octets of zero).

ddn-call-precedence
If set to a non-**nil** value, then outgoing calls will have the DDN Call Precedence facility (0x080**X** where the call's requested precedence increases as **X** goes from 0 to 3) inserted into their facility fields following a

facility marker (two octets of zero).

If both *ddn-standard-service* and *ddn-call-precedence* are specified, only one facility marker will be sent, followed by the Standard Service facility, followed immediately by the Call Precedence facility.

ddn-idle-timeout
The time in seconds (default 600) after which idle circuits are cleared. This prevents circuits from remaining open for lengthy periods of time and consuming resources unnecessarily. A message will be printed to the log file if the message verbosity is 4 or more when an idle call is cleared.

ddn-parallel-vcs
The maximum number of virtual circuits (default 1) which may be open to a single destination. Multiple virtual circuits will be created in order to maximize throughput if demand warrants it, limited only by this ceiling.

ddn-parallel-delay
The number of seconds to wait between throughput expansion call requests to a single destination. The default value of 5 seconds is usually long enough to keep brief bursts of traffic from opening additional circuits, yet short enough to expand throughput during large file transfers.

ddn-buffers-per-vc
The number of large (data) buffers to reserve per DDN virtual circuit the next time a DDN create-group is performed (i.e. define this first or it will have no effect). Some TCP implementations will work poorly if the circuit does not behave as the TCP designers anticipated, and do not adapt well to low speed links with limited buffering. The minimum value for ddn-buffers-per-vc is 4, which is probably too small to be useful, and the default is 10. If ftp's or rcp's fail while dropping many of their packets

(use ddn-stat), increase this value before the next front-end reload.

Accumulated DDN interface statistics and the state of active DDN virtual circuits can be displayed on the log file by including the line

```
(ddn-stat)
```

in the config file (see *x25-config*(5)) and sending a SIGHUP to the x25d process. The resulting log file output might look like:

```
DDN (Internet Protocol) Interface Statistics:
      56         45  RX/TX Successful calls
       0         13  RX/TX Rejected calls
   61614      30291  RX/TX X.25 data packets
 8438823    1316537  RX/TX Data bytes
       1         25  RX/TX Dropped IP packets
       0          0  RX/TX Clears
                  0  Idle timeout clears
                  0  Forced clears
                  5  Virtual Circuits in use
VC Address      Age/State  Idle     Rx/Tx Packets
 0 192.9.201.4  5:49:08    0:00:03  25319      3
 3 192.9.201.1  0:01:11    0:00:00    492      0
 4 192.9.201.1  0:00:47    0:00:00    380      0
 6 192.9.201.2  call
 7 192.9.201.1  0:02:41    0:00:00    272    217
 8 192.9.201.4  5:49:09    0:00:03    489  18683
```

SEE ALSO

x25-config(5), RFC-877.

COPYRIGHT INFORMATION

NAME

x25-pad – PAD interface to Morning Star X.25

DESCRIPTION

This section describes the X.28 PAD (packet assembler-disassembler) interface of Morning Star X.25.

Morning Star X.25 is a daemon process used in UNIX® systems. It supports several configurable user interface options, one of which is a CCITT X.3/X.28 compliant PAD interface. See *x25*(7) for general information and references to other X.25 user interfaces.

A PAD is used to establish a terminal session with another system over an X.25 network, allowing a ''start-stop mode DTE'' (i.e. a terminal) to use a public packet network as a (mostly) transparent connection to a remote computer system.

PAD devices are known by names chosen at X.25 system configuration time by the system administrator, but */dev/cux0* would be a typical device name. Since the PAD function is implemented in the X.25 daemon *x25d*, a separate user-level communications program (such as *cu*(1) or *tip*(1)) must be used to provide a communications path between the PAD device and the user terminal. File transfer communications programs such as *uucp*(1) can also use PAD devices, and their database files (including addresses of remote machines, dialing scripts and device name information) can often be shared with session-oriented programs like *cu*.

When a connection to a PAD device is first established, a banner message is printed and the device is placed in *command mode*:

```
% cu -l cux0
Connected
Morning Star PAD v1.0:  Device 0

*
```

Commands are case-insensitive and many can be abbreviated. Commands that may be abbreviated are listed with the optional part enclosed in brackets. Available commands are:

C[ALL] [*address*] [***P** | ***D** *call-user-data*]

Generate a call request packet – this is the equivalent of the CCITT ''selection'' command. *Address* is the called address that is inserted in the call request packet. It will be inserted in subsequent call request packets generated by CALL commands with no *address* argument. A ***P** or ***D** following the

address can be used to introduce up to 12 bytes of data that will be inserted into the *call-user-data* field of the call packet following the protocol identifier field. Any data beyond 12 bytes will be ignored. It will also be inserted in subsequent call request packets generated by CALL commands that don't have a *call-user-data* argument. If ***P** is used, the following characters will not be echoed. This feature is provided because many systems expect to find the user's password in the call-user-data field.

Examples:

    CALL 22200064 DME

or

    C 22200064

CL[R] Generate a clear request packet.

F[ACILITIES] [* | *facilities*]

The hexadecimal digit string *facilities* describes a set of codes to be inserted into the facilities field of outgoing call request packets until another facilities command is issued.

The command FACILITIES with no argument displays the current facilities.

The command FACILITIES * clears the current facilities so that the default facilities will be used.

Note that the FACILITIES command simply stores all bytes from the first syntactically significant character after the command name up to the end of the line. It makes no attempt to ensure that the facilities argument consists of legitimate facilities values, or even that they are hexadecimal digits.

Sample FACILITIES commands are:

    FACILITIES 420505

set the window size to 5,

    F 430808

set the packet size to 256,

    fac 0101

ask for reverse charging,

    Facilities 0101420505430808

all of the above, and

    F *

use the default facilities.

FU[LL] Select full duplex mode. If the pad session is already in full duplex mode, FULL has no effect.

HA[LF] [*] | [[–] *ch1, ch2,...,chn*]
Select half duplex mode, and specify characters which are to be echoed. In half duplex mode, most characters are not echoed. If line feed insertion is enabled, the inserted line feeds will be sent to the terminal device. If tab expansion is enabled, the resulting spaces will be sent to the terminal. If line feed or carriage return padding is enabled, the padding will be sent to the terminal, even if the padded character is not echoed.

Also, the HALF command allows the user to select characters which should be echoed – a type of reverse echo mask. For example,

    HALF 13,10

selects carriage return and line feed to be echoed.

    HALF *

de-selects all characters so that none will be echoed. A character list preceded by a hyphen (‘‘-’’) de-selects only the characters in the list. For example:

    HALF – 10,13

The HALF command with no arguments sets half duplex mode without altering the characters which have been selected for echo via any previously entered HALF commands.

More examples:

    HA 13,10,9

selects half duplex mode, echo carriage return, line feed, and

tab.

    ha – 9

selects half duplex mode, disable previously enabled echo of tab.

    HALF *

selects half duplex mode, disable previously enabled echo for all characters.

H[ELP]
Display the list of PAD commands.

I[NTERRUPT]
Generate an interrupt packet.

PA[R?] [*ref1* [*,ref2,...,refn*]]
Display the current value of one or more X.3 parameters. For example,

    PAR?

displays all X.3 parameters.

    PA 13,16,17

displays values of X.3 parameters 13, 16, and 17.

PR[OF] [*profile* | **?**]
Enable a standard PAD parameter profile. For example,

    PROF

displays the current profile name.

    PROF ?

displays the current profile name followed by a list of available profiles.

    PR UNIX

selects the default UNIX profile.

    prof uucp

selects a profile suitable for UUCP or other applications requiring a fast feature-free data pipe.

Q[UIT] Terminate the current session and disconnect from the PAD device.

R[ESET]
Generate a reset request packet with zero cause (DTE originated) and zero diagnostic (no additional information).

SE[T] [*ref1:val1* [,*ref2:val2*,...,*refn:valn*]]
Set one or more X.3 parameters to the specified values. For example,

    SE 13:5,16:8

enables line feed insertion on echo and output and makes backspace the character delete character.

SET? [*ref1*:*val1* [,*ref2*:*val2*,...,*refn*:*valn*]]
Set one or more X.3 parameters to specified values and display the new values. For example,

    SET? 13:4

enables line feed insertion on echo, then displays the newly set value of parameter 13.

S[TATUS]
Print the current virtual call status.

T[ABS] [LC[L] *tab-num*] [REM *tab-num*] [E[XP] *exp-num*]
Set and read three non-standard parameters which control tab expansion. These parameters are not accessible by the remote host via Q-bit packet PAD commands. The arguments have the following meanings:

EXP Enable (1) or disable (0) expansion of tabs by the pad to the number of blanks specified by the LCL argument.

LCL Set the number of columns to which tabs are expanded locally, i.e. to the terminal on echo and output from the network – if the EXP parameter is zero and the LCL parameter is nonzero, then LCL means the number of columns to which the terminal is expanding tabs sent to it (this situation is for tracking absolute line position for line folding and destructive line deletion). Zero means no expansion.

REM Set the number of columns to which tabs are expanded remotely, i.e. on input from the terminal towards the

network. Zero means no expansion. For example:

    TABS LCL 8 EXP 1

expands tabs to the terminal to eight blanks, and

    T LC 4 REM 4 E 1

expands tabs to both the terminal and the remote to four blanks.

BUGS

The Morning Star X.25 *pad* implementation of parameter 3 (data forwarding mask) is somewhat nonstandard. The standard makes it impossible to specify data forwarding on every character by omitting the punctuation characters from the data forwarding ranges. However, the Morning Star *pad* accepts the 0x80 (128 decimal) bit and gives it the meaning "all characters not covered by the other ranges." Thus a parameter 3 value of 255 means "data forwarding on all characters."

The implementation of parameter 20 (echo mask, 1984 only) is somewhat nonstandard. The 0x40 (64 decimal) bit is accepted, but has no meaning. Echo mask of editing characters does not function.

Section 3.3.2 of the 1984 CCITT X.28 specification describes which function should be applied to a character typed by the user when the selection of parameter values causes more than one function to apply to that character. The Morning Star *pad* uses a slightly different priority list – priority 6 (Data forwarding character, parameter 3) and priority 7 (Line delete, parameter 17) are reversed, allowing the use of a data forwarding character as the local line delete character.

The implementation of parameter 22 (page wait, 1984 only) is nonstandard. The page wait line feed count is reset by only two events:

1) Echo of a line feed from the terminal

2) Cancellation of the page wait condition

The page wait condition is canceled by receipt of *any* character from the terminal. The received character is processed normally and data flow is restarted toward the terminal. Thus, the line feed count tracks the number of line feeds in service signals and output since the last echoed line feed. If you wish to cancel the page wait condition without having any other effect, you may send the *pad* an XON (control-Q). If you wish to type ahead without scrolling the screen when it is frozen because of page wait, enter an XOFF (control-S) and then proceed.

SEE ALSO

x25(7).
CCITT Data Communication Networks Interfaces, Recommendations X.3 and *X.28.*

COPYRIGHT INFORMATION

NAME

x25-x29 – remote login interface to Morning Star X.25

DESCRIPTION

This section describes the X.29 interface of Morning Star X.25.

Morning Star X.25 is a daemon process used in UNIX® systems. It supports several configurable user interface options, one of which is a CCITT X.3/X.29 compliant remote login interface. See *x25*(7) for general information and references to other X.25 user interfaces.

The X.29 interface is used to accept incoming calls from remote network PADs. The remote PAD places a call which is terminated in an X.29 session. This interface offers the ability to connect those X.29 sessions to a virtual terminal interface on the host UNIX system. No special configuration of the UNIX host system is required, since those virtual terminals are generally already in place for any system on which **x25d** runs.

By default, such sessions are connected to the program **/bin/login**, but this may be changed by using the *program* option to *create-group*. Information about the incoming call is saved in the following environment variables:

X25_CALLING_ADDRESS	The X.121 address of the calling X.25 node
X25_CALLED_ADDRESS	The X.121 address of the destination X.25 node
X25_FACILITIES	The facilities field from the X.25 call packet, not including the length octet
X25_CALL_USER_DATA	The call-user-data field from the X.25 call packet

All of the above are hexadecimal strings.

Be aware that most implementations of **/bin/login** clear the environment when they start up. Some implementations supply an option to disable this behavior.

See *x25-config*(5) for information about *create-group*, *program*, and *in-route-table*.

SEE ALSO

x25d(8), x25(7), x25-config(7), x25-pad(7), pty(4).
CCITT Data Communication Networks Interfaces, Recommendation X.3 and *X.29.*

COPYRIGHT INFORMATION

NAME

x25 – general information about Morning Star Technologies X.25

DESCRIPTION

Morning Star X.25 is a daemon process used in UNIX® systems. It supports several configurable user interfaces, including a PAD (Packet Assembler/Disassembler) interface, an X.29 interface for logins from remote PADs, a SunLink® compatible socket interface, an RFC-877/DDN compliant interface for transport of IP (Internet Protocol) packets, and a means of acquiring configuration and performance information from the running system.

The RFC-877/DDN, SunLink, and X.29 interfaces are accessed invisibly and transparently by the UNIX system's IP implementation, but the PAD interface is attached by configuration commands to "general purpose" tty devices. See *x25-pad*(7), *x25-x29*(7), *x25-ddn*(7), and *x25-config*(7) for detailed descriptions of each of these interfaces.

System informational and error messages can be read from the *log* file after the daemon has been started, if it is so configured. Packet trace information can be read from the *trace* file after the daemon has been started, if it is so configured. The *x25trace*(1) program can be used to convert the trace data into human-readable form.

Configuration commands may be written in the configuration file, in order to set up communications lines, call routing information, or general system parameters such as verbosity levels or global buffer sizes. The current values of configured parameters may be extracted by including the *print-config* command in the configuration file, instructing the daemon to re-read it, and then examining the log file. The *line-stat* and *ddn-stat* commands, when the daemon reads them in the configuration file, cause status information to be written to the log file.

NOTE

This product contains RSYS(tm) X.25 software distributed under license from Gcom, Inc.

SEE ALSO

x25d(8), x25-config(5), x25-pad(7), x25-x29(7), x25-ddn(7), x25trace(1), x25-socket(3), x25-auth(5).

COPYRIGHT INFORMATION

NAME

x25d – X.25 protocol daemon

SYNOPSIS

x25d [**–f** config-file] [**–l** log-file] [**–a** authorization-file] [**–t** trace-file] [**–d**]

DESCRIPTION

X25d is a daemon process used in UNIX® systems to manage a connection to a network using the X.25 protocol. It communicates with an X.25 network via either the UNIX host's serial ports, or via a Morning Star SnapLink® interface. It communicates with software on the UNIX host via a SunLink-compatible socket interface (see *x25-socket*(3)), via an RFC877-compliant interface from the kernel's own TCP/IP implementation (see *x25-ddn*(7)), or via an X.3/X.28 PAD or X.29 virtual terminal interface (see *x25-pad*(7) or *x25-x29*(7)).

OPTIONS

–f config-file
: The configuration file for this invocation of the daemon. The default is **x25.config** in the present working directory. See *x25-config*(7).

–l log-file
: File to which all logging output is to be directed. The default is **x25.log** in the present working directory. Specify **–l –** to direct logging information to the daemon's standard output.

–a authorization-file
: File describing constraints on and priviliges of users of the SunLink compatibility facilities provided by the daemon. Default is no constraints. See *x25-auth*(5).

–t trace-file
: File to which packet trace information is to be directed. Default is **x25.trace** in the present working directory, though by default no packet tracing is enabled until a SIGQUIT is received. See *x25trace*(1).

–d
: Don't detach from the controlling terminal to run as a daemon.

SIGNALS

Upon reception of the following signals, **x25d** takes the indicated actions:

SIGHUP Re-reads the configuration file, reconfiguring itself internally in accordance with any changes it finds there.

SIGQUIT Toggles line tracing. If tracing was enabled, it is disabled. If no tracing was occuring, it is begun as of the time of reception of the signal.

SIGUSR1 Increases logging verbosity. More detailed information is written to the log file.

SIGUSR2 Decreases logging verbosity. Less detailed information is written to the log file.

SIGTERM Causes the daemon to exit.

SIGINT Causes the daemon to exit.

EXAMPLE

To run as a daemon, designating appropriate configuration and log files:

x25d –l /usr/adm/x25.log –f /etc/x25.config

To remain attached and direct logging information to this terminal:

x25d –d –l – –f /etc/x25.config

RECOMMENDATIONS

On a Sun, we recommend that the configuration file be maintained in **/etc/x25.config**, the authorization file in **/etc/x25.auth**, the log file in **/usr/adm/x25.log**, and the trace file directed to **/usr/adm/x25.trace**. The daemon might best reside in **/usr/etc/x25d**.

SEE ALSO

x25(7), x25-auth(5), x25trace(1), x25-ddn(7), x25-pad(7), x25-x29(7), RFC877, x25-socket(3).

CREDITS

SunLink is a registered trademark of Sun Microsystems Inc.

COPYRIGHT INFORMATION