This reference describes the AppleTalk application programming interface (API) specifications jointly developed by AT&T Computer Systems and Apple Computer, Inc. These API specifications define a standard interface to AppleTalk network protocols for AT&T's UNIX System V Release 4 operating system.

Software developers can create AppleTalk network application programs for UNIX systems that support both these API specifications and AppleTalk. Such applications are readily portable across various UNIX platforms.

AppleTalk network applications running in a UNIX environment can provide a standard desktop interface to Macintosh users accessing network services -- such as file sharing, printer sharing, electronic mail, distributed databases, and other client/server applications. UNIX file servers that support AppleTalk allow Macintosh and other personal computers to store and share information on the servers.

APIs implemented according to these specifications provide interfaces to the AppleTalk family of protocols. AT&T's Transport Interface (TLI) provides the interface to the AppleTalk Data Stream Protocol (ADSP) and the Datagram Delivery Protocol (DDP).

AT&T's TLI and the STREAMS input/output system provide developers with the tools needed to build distributed applications that are both protocol and media independent. The TLI's library of functions facilitates the development of network applications that are compatible with most industry-standard protocols. Such applications can easily provide support for multiple protocols on a UNIX system, migration to new protocols, and the addition of new features to existing network applications.

ADSP resides in the session layer of the International Standards Organization's (ISO) Open System Interconnection (OSI) reference model. Protocols in the session layer guarantee reliable data delivery. ADSP provides bidirectional, sequential, duplicate-free byte-stream service between any two sockets on an AppleTalk internet. ADSP services allow client processes to establish socket connections, to send and receive data -- either as a continuous stream or as logical messages intelligible by the receiving client -- and to close socket connections. ADSP provides a flow-control mechanism to ensure that transmitted data does not exceed the capacity of the receiving client's buffer.

DDP corresponds to the network layer of the OSI model and provides best-effort data delivery service. DDP is an end-to-end data flow protocol that extends socket-to-socket datagram delivery to an AppleTalk internet consisting of one or more AppleTalk networks connected by routers. A datagram is a packet of data that carries its own routing information. DDP defines logical addresses for sockets on an internet.

The AppleTalk Session Protocol (ASP) resides in the session layer of the OSI model. ASP provides the transport services needed for higher-level interactions between workstations and servers. ASP uses the services of the AppleTalk Transaction Protocol to open, manage, and close sessions; and to sequence requests and replies. ASP services include sending commands to a server and returning replies to a workstation, writing blocks of data to a

server, and retrieving status information from a server. During a session -that is, when a logical connection exists between a workstation and a server
-- ASP guarantees the delivery and execution of a sequence of transactions in
the order sent.

The AppleTalk Transaction Protocol (ATP) resides in the transport layer of the OSI model. ATP provides loss-free transaction service between sockets on an internet, using a request/response transaction model and error recovery. By binding requests to responses, ATP ensures the reliable exchange of request/response pairs. ATP provides the basis for the session-oriented services of the AppleTalk Session Protocol and the Printer Access Protocol.

The Name Binding Protocol (NBP) resides in the transport layer of the OSI model. NBP converts user-defined names for zones and devices, or entities, to internet socket addresses that AppleTalk protocols can use. Each node maintains name-to-address mapping for its sockets. NBP uses a names directory to provide services that include name registration, deletion, lookup, and confirmation.

The Printer Access Protocol (PAP) resides in the session layer of the OSI model. PAP manages transactions between workstations and servers -- including setting up, maintaining, and terminating connections; and transferring data. PAP allows multiple connections at both the workstation and the server. PAP uses NBP services to locate addresses and ATP services to transfer data. PAP can determine a server's status and filter duplicate requests.

The Routing Table Maintenance Protocol (RTMP) resides in the transport layer of the OSI model. Internet routers use RTMP to establish and maintain routing tables used in forwarding datagrams from any source socket to any destination socket on an internet. Each router on an internet periodically broadcasts RTMP data packets containing updated routing tables, allowing all other routers on the internet to update their own routing tables.

The Zone Information Protocol (ZIP) resides in the session layer of the OSI model. Routers use ZIP to maintain network-to-zone-name mapping on an internet. ZIP allows a node on an extended network to select its zone at startup. ZIP provides commands that allow nodes on a local area network to obtain zone information. NBP uses ZIP mapping to determine which networks contain nodes that belong to a zone.

What this reference contains

This reference consists of three sections, which contain the following information:

- Section 1, "ADSP TLI Specification," describes the Transport Interface to the AppleTalk Data Stream Protocol.
- Section 2, "DDP TLI Specification," describes the Transport Interface to the Datagram Delivery Protocol.
- Section 3, "AppleTalk Manual Pages," contains all AppleTalk Section 3N (Network Programming) manual pages for UNIX System V Release 4. These manual pages correspond to the following AppleTalk protocols: AppleTalk Session Protocol, AppleTalk Transaction Protocol, Routing Table Maintenance Protocol, Name Binding Protocol, Printer Access Protocol, and Zone Information Protocol.

Conventions used in this reference

This reference uses the following typographic conventions to distinguish elements of the text:

- The names of function calls, parameters, and fields in structures appear in italics in the text of the reference.
- Code samples appear in Courier type -- for example:

This is 10-point Courier type.

- The manual pages in Section 3 appear in the standard UNIX man page format.

For more information

The following documents provide information about either the AppleTalk network system or the UNIX System V Release 4 operating system:

- Sidhu, Gursharan S., Andrews, Richard F., and Oppenheimer, Alan B. Apple Computer, Inc. Inside AppleTalk, second edition. Reading, Mass.: Addison-Wesley, 1990. Explains the AppleTalk protocols in detail.
- AT&T, UNIX System V Release 4 Programmer's Guide: Networking Interfaces. Englewood Cliffs, N.J.: Prentice-Hall, 1990. Defines the TLI programming calls and describes programming with TLI.
- AT&T, UNIX System V Release 4 Programmer's Guide: STREAMS. Englewood Cliffs, N.J.: Prentice-Hall, 1990. Provides detailed information about STREAMS.
- Jacobson, Van. "Congestion Avoidance and Control." Proceedings of the ACM, SIGCOM '88, Palo Alto, Calif., August 1988. Describes algorithms developed for the TCP/IP protocol that improve performance over slow or congested data links.

SECTION 1: ADSP TLI INTERFACE SPECIFICATION

Summary of ADSP data structures

This section describes a Transport Interface (TLI) to the AppleTalk Data Stream Protocol (ADSP) and consists of several subsections:

- "General Concepts" explains ${\tt STREAMS}$ and ${\tt TLI}$ concepts relevant to this document.
- "Implementation Issues" describes several important implementation issues.
- "ADSP TLI Library Calls and Parameters" describes each ADSP function call and its parameters.
- "A Client/Server Example" presents a sample client/server program.
- "Summary of ADSP Data Structures" presents ADSP constants and data structures.

Some portions of this document are taken from The External Reference Specification for ADSP 1.5/2.0, version 0.14.

General concepts

The services of ADSP map closely to the TLI connection-mode calls. TLI's connection-mode service enables data to be transferred over an established connection in a reliable, sequenced manner. This service enables the negotiation of the parameters and options that govern the transfer of data. The following is a general scenario under TLI:

A connection end is identified by a local file descriptor (fd) returned by $t_open(3N)$. Then an address is associated with this endpoint using $t_bind(3N)$. At this point, the process can either listen for an incoming connect indication, $t_listen(3N)$, or initiate a connection request, $t_connect(3N)$. The passive user can then accept the connection on a different fd, $t_accept(3N)$. After the connection has been established, the process can send data, $t_snd(3N)$, receive data, $t_rcv(3N)$, or close the connection, $t_close(3N)$.

Error handling

Failures are indicated by a return value of -1. An external integer, t_errno, holds the specific error numbers. When a function sets t_errno to [TSYSERR] it indicates an operating system error, the specifics about which can be accessed through the external variable errno. When a function sets t_errno to [TLOOK], it indicates that an asynchronous event has happened. The user can call $t_look(3N)$ to determine what event has occured.

Synchronous and asynchronous modes

In the synchronous mode, the user has to wait for a specific event to happen before control is returned to the user. The asynchronous mode of operation provides a mechanism for notifying a user of some event without forcing the user to wait for the event. Synchronous mode is the default mode of operation. This mode can be changed through the O_NDELAY or O_NONBLOCK flag, which may be set during t_open(3N) or fcntl(2). The following is a list of asynchronous events: T_LISTEN, T_CONNECT, T_DATA, T_EXDATA, and T_DISCONNECT. In synchronous mode, a function may return -1 with t_errno set to [TLOOK] to indicate the occurrence of an asynchronous event. The t_look(3N) function is then invoked to identify the specific event. Another means to notify a process that an asynchronous event has occured is polling.

TLI functions supported for ADSP

The following TLI functions, which correspond to connection-oriented services, are supported:

t_accept(3N)

 $t_bind(3N)$

t_close(3N)

t_connect(3N)

t_listen(3N)

```
t_look(3N)
t_open(3N)
t_rcv(3N)
t_rcvconnect(3N)
t_rcvdis(3N)
t_snd(3N)
t_snddis(3N)
t_unbi nd(3N)
The following general TLI functions are also supported:
t_alloc(3N)
t_error(3N)
t_free(3N)
t_getinfo(3N)
t_getstate(3N)
t_optmgmt(3N)
t_sync(3N)
The following functions are not supported and upon calling will return -1,
with t_errno set to [TNOTSUPPORT]:
t_rcvrel(3N)
t_rcvudata(3N)
t_rcvuderr(3N)
t_sndrel(3N)
t_sndudata(3N)
Key for parameter arrays
For each function, an array is presented that summarizes the content of the
input and output parameters. The key for the parameter arrays is as follows:
The parameter value is meaningful. (The input parameter must be set before
the call and the output parameter may be read after the call.)
        The content of the object to which the x pointer points is
(x)
meaningful.
```

The parameter value is meaningful but the parameter is optional.

-- The parameter value is meaningless.

The parameter keeps the same value after the call as before the call.

Implementation issues

Options and management parameters

There are two user-selectable options: filter_addr, which supports connection-opening filters, and checksum, which is a boolean variable that can be set to TRUE or FALSE, indicating the desire to turn on or off DDP checksum. Other options or parameters are either set by the system administrator or calculated dynamically.

Connection-opening filters

The ADSP client may need to be selective about establishing connections with remote clients, because the addresses of some remote clients that make open-connection requests may not be acceptable to the local client. In order to establish a selection criterion, the client can provide ADSP with a filter of valid network addresses with which it is willing to establish connections. This filter could be as simple as specifying "open a connection only with the socket to which you are sending the open-connection request" or "open a connection only with a socket on a particular node." If ADSP receives an open-connection request from an address that does not match the filter, it sends back an open-connection denial and ignores the packet. In the case of a connection-listening socket, the end could conceivably become established with a different network address than the one to which the original open request was sent. The original requester can provide ADSP with a filter of network addresses with which it is willing to establish a connection.

Filter_addr is defined in adsp_opt structure. (See "Summary of ADSP Data Structures" later in Section 1.) It can be set when calling t_connect(3N) or before a t_listen(3N) by calling t_optmgmt(3N). A zero in the network number, node identifier, or socket number of filter_addr means that a connection can be established with any connection end on any network, node, or socket, respectively. Setting filter_addr to be the same as remote address means that a connection will be established only with a connection end on the specified remote address.

System administration parameters

There are several parameters tunable by the system administrator. These are:

- Max Receive buffer
- open connection request retries
- initial round-trip time

Dynamically set parameters

Many of the ADSP variables are dynamically determined based on algorithms that have been successful in the TCP/IP community. A researcher at the Lawrence Berkeley Labs, Van Jacobson, has done significant work to improve the performance of the TCP/IP protocol over slow or congested data links. Since ADSP is very similar in design to TCP/IP, several of the techniques developed can and will be used for ADSP.

One of the techniques used involves dynamically determining round-trip times between two connection ends, through which ADSP will dynamically determine the values to use for retry intervals.

ADSP TLI protocol address

There are two ways to view the TLI protocol address for ADSP. Primitives like t_bind(3N) and t_connect(3N) accept an NBP entity name (object:type@zone) as their protocol address. The name registration, in t_bind(3N), or name resolution, in t_connect(3N), is done by the transport provider, transparent to the applications. This eliminates the need for NBP primitives in applications. The elimination of NBP primitives and use of the Name-to-Address Mapping provided by System V Release 4 are necessary to allow the development of protocol-independent applications. Since AppleTalk nodes are not required to have network names, NBP entity names are not available in some cases. Sometimes all that is available is the internet socket address. This is the case with t_listen(3N). When a connection request arrives, t_listen(3N) returns the protocol address of the requesting user. All that is available for return is the internet socket address. An application is likely to respond with a t_accept(3N) and use the address returned by t_listen(3N) as the destination address.

The transport provider must be able to distinguish between NBP entity names and internet socket addresses, so that it knows how to process the address field. The convention adopted involves using NULL-terminated strings for NBP entity names and using a leading NULL in the first character position, followed by an at_inet_t structure to identify an internet socket address. The t-bind(3N), t-connect(3N),

t-rcvconnect(3N), and t-listen(3N) calls return an internet socket address. See "Summary of ADSP Data Structures" later in Section 1.

The object and type fields of the NBP entity names cannot contain wildcards. The zone name field may be set to an asterik (*). Using an illegal NBP entity name or a name that cannot be resolved causes t-bind(3N) or t-connect(3N) to return a -1 and set t-errno to [TBADADDR]

Names that are transparently registered with a $t_{\rm bind}(3N)$ are transparently deregistered with $t_{\rm unbind}(3N)$ or upon the closing of the corresponding file descriptor.

ADSP features not supported

The following ADSP features are outside the scope of TLI and are not supported:

- connection opening outside of ADSP
- accepting connections on alternate nodes

Non-TLI calls

In addition to the TLI calls mentioned in "TLI Functions Supported for ADSP" earlier in Section 1 the following nonstandard call is also supported:

adsp_fwdreset(3)

ADSP TLI library calls and parameters

Several of the ADSP TLI library function calls are described here. For a more detailed description of these calls refer to the UNIX System V Release 4 Programmer's Guide: Networking Interfaces.

fd = t_open (path, oflag, info)

 $T_{open}(3N)$ is called as the first step in the initialization of a transport endpoint. This function returns various default characteristics of the underlying transport protocol by setting fields in the t_{info} structure.

The following should be the values returned by the call to $t_open(3N)$ and $t_getinfo(3N)$ with ADSP as the transport provider:

Parameters	Before call	After call
path	ADSP_DEV	
oflag	x	
info->addr		99
info->options		64
info->tsdu		- 1
info->etsdu		572
info->connect		- 2
i nfo- >di scon		- 2
info->servtype		T_COTS

The argument path points to the ADSP device identifier, such as, /dev/adsp, normally extracted from /etc/netconfig (see netconfig(4) and getnetpath(3N)). Oflag may be constructed from 0_NDELAY or 0_NONBLOCK OR-ed with 0_RDWR . T_open returns a file descriptor that identifies the local ADSP endpoint. The default characteristics of ADSP are returned in info. The maximum size of the ADSP address, addr, is 99 bytes. There is no limit to the size of the Transport Service Data Unit (TSDU). The maximum size of the Expedited Transport Service Data Unit (ETSDU), the attention packet, is 572 bytes. The attention packet is composed of 2 bytes of attention code followed by up to 570 bytes of attention data. No data can be sent with connection-request or disconnect calls. The only service type supported is T_COTS, the connection-oriented mode.

t_bind (fd, req, ret)

This function associates a protocol address with the transport endpoint specified by fd and activates that transport endpoint. The transport provider can then begin accepting or requesting connections. The req and ret arguments point to a t_bind structure. Qlen is used to indicate the maximum number of outstanding connection indications.

Parameters Before call After call req->addr.maxlen -- --

req->addr.len	x>=0	
req->addr.buf	$\mathbf{x}(\mathbf{x})$	
req->qlen	x>=0	
ret->addr. maxl en	x	
ret->addr.len		x
ret->addr.buf	x	(x)
reg->glen		x>=0

req->addr.buf is a pointer to an NBP name (a NULL-terminated C string). The transport provider will allocate a dynamic socket and register that name on this socket.

If req is set to NULL or req->addr.len is zero, the transport provider will assign a dynamic socket and return the internet address of this socket in ret->addr.buf in a format described in "ADSP TLI Protocol Address," earlier in Section 1.

t_unbind (fd)

This function disables the transport endpoint specified by fd, which was previously bound by $t_bind(3N)$. On completion of this call, no further data or events destined for this transport endpoint will be accepted by the transport provider.

Parameters	Before call	After call
fd	x	

The NBP name that was transparently registered by the transport provider during t_{0} will be deregistered.

t_connect (fd, sndcall, rcvcall)

This function enables a user to request a connection to a specified destination. Fd identifies the local connection endpoint. Sndcall and rcvcall point to a t_call structure.

Parameters	Before call	After call
fd	x	
sndcall->addr.maxlen		
sndcall->addr.len	x>0	
sndcall->addr.buf	x (x)	
sndcall->opt.maxlen		
sndcall->opt.len	x>0	
sndcall->opt.buf	$\mathbf{x}(\mathbf{x})$	

sndcall->udata. maxlen		
sndcall->udata.len	0	
sndcall->udata.buf		
sndcall->sequence		
rcvcall->addr.maxlen	x	
rcvcall->addr.len		x
rcvcall->addr.buf	x	(x)
rcvcall->opt.maxlen	x	
rcvcall->opt.len		x
rcvcall->opt.buf	x	(x)
rcvcall->udata.maxlen		
rcvcall->udata.len		
rcvcall->udata.buf		
rcvcall->sequence		

In sndcall, addr specifies the protocol address of the destination transport user. The sndcall->addr.buf points to an NBP name. The transport provider will resolve the name before sending a connection request.

In sndcall, opt points to an adsp_opt structure that is used to set ADSP address filters or turn on/off DDP checksum. The user can set the filter_addr field in the adsp_opt to filter connection ends responding to this connection request. A zero in the network number, node identifier, or socket number of filter_addr means that a connection can be established with any connection end on any network, node, or socket, respectively. Setting filter_addr to be the same as remote address means that a connection will be established only with a connection end on the specified remote address.

Since data cannot be sent with a $t_connect(3N)$, sndcall->udata.len must be set to 0.

t_rcvconnect (fd, call)

This function enables a user to determine the status of a previously sent connection request and is used in conjunction with $t_connect(3N)$ to establish a connection in asynchronous mode. The connection will be established on successful completion of this function. Fd identifies the endpoint and call contains information associated with the newly established connection.

Parameters	Before call	After cal
fd	x	
call->addr.maxlen	x	

call->addr.len		X
call->addr.buf	x	(x)
call->opt.maxlen	x	
call->opt.len		x
call->opt.buf	x	(x)
call->udata.maxlen	0	
call->udata.len		
call->udata.buf		
call->sequence		

Since data cannot be sent with a connection request, call->udata.maxlen must be set to 0 before calling $t_r(3N)$. On return, the call->addr structure contains the protocol address of the responding endpoint.

t_optmgmt (fd, req, ret)

This function enables a transport user to retrieve, verify, or negotiate protocol options with the transport provider. The argument fd identifies a bound transport endpoint. The req and ret arguments point to a t_optmgmt structure.

Parameters	Before call	After call
fd	x	
req->opt.maxlen		
req->opt.len	x	
req->opt.buf	$\mathbf{x}(\mathbf{x})$	
req->flags	x	
ret->opt.maxlen	x	
ret->opt.len		x
ret->opt.buf	x	(x)
req->flags		x

The req->opt.buf points to an adsp_opt structure. The user can set the filter_addr field in the adsp_opt. A zero in the network number, node identifier, or socket number of filter_addr means that a connection can be established with any connection end on any network, node, or socket, respectively. Setting filter_addr to be the same as remote address means that a connection will be established only with a connection end on the specified remote address. The user can also change the default value of checksum.

To filter incoming connections, a user can set the desired filter by calling $t_{optmgmt}(3N)$ before doing a $t_{optmgmt}(3N)$.

t_listen (fd, call)

This function listens for a connection request from a calling ADSP user. Fd identifies the local connection endpoint where the connection indication arrived, and on return, call contains information describing the connection indication. Call points to a t_call structure. In call, addr returns the calling ADSP user's AppleTalk internet address. The value of sequence is equivalent to the notion of CID, a number that uniquely identifies the returned connection indication. The value of sequence enables the user to listen for multiple indications before responding to any of them.

1

Parameters	Before call	After call
fd	x	
call->addr.maxlen	x	
call->addr.len		x
call->addr.buf	x	(x)
call->opt.maxlen	x	
call->opt.len		x
call->opt.buf	x	(x)
call->udata.maxlen	0	
call->udata.len		
call->udata.buf		
call->sequence		x

An address filter can be set using $t_{optmgmt}(3N)$ before calling $t_{optmgmt}(3N)$ to specify the remote addresses with which the connection end is willing to establish a connection.

Since data cannot be sent with a connection request, call->udata.maxlen must be set to 0 before the call to t_listen(3N). The call->addr structure contains the remote calling AppleTalk internet address.

If the user has set qlen to a value greater than 1 (on the call to $t_bind(3N)$), the user may queue up several connection indications before responding to any of them. The user should be warned that ADSP starts a timer to be sure of obtaining a response to the connection request in a finite time. If the user queues the connection indications for too long a time before responding to them, the transport provider initiating the connection will disconnect it. This time is dynamically calculated based on packet round-trip time.

t_accept (fd, resfd, call)

This function accepts a connection request. Fd identifies the local

connection endpoint where the connection indication arrived, resfd specifies the local connection endpoint where the connection is to be established, and call contains information required by ADSP, the transport provider, to complete the connection. Call points to a t_call structure. The call->addr is the address returned by t_listen(3N), opt indicates any protocol-specific parameters associated with the connection, and sequence is the CID returned by t_listen(3N) that uniquely associates the response with a previously received connection indication.

1

Parameters	Before call	After cal
fd	x	
resfd	x	
call->addr.maxlen		
call->addr.len	x	
call->addr.buf	$\mathbf{x}(\mathbf{x})$	
call->opt.maxlen		
call->opt.len	x	
call->opt.buf	$\mathbf{x}(\mathbf{x})$	
call->udata.maxlen		
call->udata.len	0	
call->udata.buf		
call->sequence	x	

Since data cannot be sent with a connection request, call->udata.len must be set to $0. \,$

t_snd (fd, buf, nbytes, flags)

This function is used to send either normal or expedited (attention) data.

Parameters	Before call	After call
fd	x	
buf	x (x)	
nbytes	x	
flags	x	

The $T_{EXPEDITED}$ flag maps directly to the attention message concept of ADSP. Note that the data consists of 2 bytes of attention code followed by up to 570 bytes of attention data.

The T_MORE flag maps directly to the end-of-message (EOM) bit but in reverse fashion. T_MORE being set indicates more data will follow for the current

TSDU. When the EOM bit is set, it indicates the end of the current logical data.

t_rcv (fd, buf, nbytes, flags)

This function receives either normal or expedited (attention) data.

Parameters	Before call	After call
fd	x	
buf	x	(x)
nbytes	x	
flags		x

The $T_{EXPEDITED}$ flag maps directly to the attention message concept of ADSP. Note that the data consists of 2 bytes of attention code followed by up to 570 bytes of attention data.

The T_MORE flag maps directly to the EOM bit but in reverse fashion. T_MORE being set indicates more data will follow for the current TSDU. When the EOM bit is set, it indicates the end of the current logical data.

t_snddis (fd, call)

This function is used to initiate an abortive release on an already established connection or to reject a connection request. Fd identifies the local endpoint. Call points to a t_call structure.

Parameters	Before call	After call
fd	x	
call->addr.maxlen		
call->addr.len		
call->addr.buf		
call->opt.maxlen		
call->opt.len		
call->opt.buf		
call->udata. maxlen		
call->udata.len	0	
call->udata.buf		
call->sequence	x	

Since data cannot be sent with a disconnect, call may be set to NULL or call->udata.len must be set to 0.

When rejecting a connection request, call must be non-NULL and contain a valid value of sequence to uniquely identify the rejected connection indication.

t_rcvdis (fd, discon)

This function is used to identify the cause of a disconnect. Fd identifies the local endpoint. discon points to a t_discon structure.

Parameters	Before call	After call
fd	x	
di scon->udata. maxl en		
di scon->udata. l en		0
di scon->udata. buf		
di scon->reason		x
di scon->sequence		x

Since data cannot be sent with a disconnect, discon->udata.len will return 0.

Sequence may identify an outstanding connection indication with which the disconnect is associated. Sequence is only meaningful when t_rcvdis is issued by a passive ADSP user that has executed several t_listen(3N) functions and is processing the resulting connection indications. If a disconnect indication occurs, sequence can be used to identify which of the outstanding connect indications is associated with the disconnect. No protocol-specific reason codes are currently defined.

If a user does not need to know the value of reason or sequence, discon may be NULL.

t_close (fd)

The t_close function informs the transport provider that the user is finished with the transport endpoint specified by fd. This function does not check state information, so it can be called from any state to close a transport endpoint. This call is abortive and breaks any transport connection that may be associated with the endpoint.

Parameters	Before call	After call
fd	x	

The NBP name that was transparently registered by the transport provider during $t_bind(3N)$ will be deregistered.

adsp_fwdreset (fd)

Since the forward-reset feature of ADSP does not map directly to the standard TLI calls, a separate function is created to emulate that mechanism.

Parameters Before call After call

fd x --

The argument fd identifies a bound transport endpoint.

On the receiving end, the transport provider will generate a SIGUSR1 signal to notify the user that a forward reset was received and processed.

A client/server example

The following is a simple client/server example to demonstrate the calling conventions.

```
The server side
/* adsp_server.c -- this is an ADSP server */
#include <tiuser.h>
#include <stropts.h>
#include <fcntl.h>
#include <stdio.h>
#include <signal.h>
#include <netconfig.h>
#include <netdir.h>
#include "appletalk.h"
#include "adsp.h"
main ()
int s, s2, flags;
struct t_bind *bind;
struct t_call *callptr;
char buf [BUFSIZE];
voi d *handlep;
struct netconfig *netconfigp;
* Initialize the network selection mechanism.
if ((handlep = setnetpath()) == NULL) {
nc_perror ("adsp server: setnetpath");
exit (1);
```

```
* Loop through the transport providers.
while (netconfigp = getnetpath (handlep)) {
if ((netconfigp->nc_semantics == NC_TPI_COTS) &&
(strcmp (netconfigp->nc_protofmly, NC_APPLETALK) == 0) &&
(strcmp (netconfigp->nc_proto, "-") == 0))
break;
if (netconfigp == NULL) {
endnetconfig (handlep);
exit (1);
if ((s = t_{open} (netconfigp->nc_{device}, 0_RDWR, (struct t_info *)0)) < 0) {
t_error ("adsp server: t_open");
exit (1);
if ((bind = (struct t_bind *)t_alloc (s, T_BIND, T_ADDR)) == NULL) {
t_error ("adsp server: t_alloc");
exit (1);
(void) strcpy (bind->addr.buf, "Receiver: ADSP_Server@Nets_R_Us");
if (t_bind (s, bind, (struct t_bind *) 0) < 0) {
t_error ("adsp server: t_bind");
exit (1);
if ((callptr = (struct t_call *)t_alloc (s, T_CALL, T_ALL)) == NULL) {
t_error ("adsp server: t_alloc");
exit (1);
if (t_listen (fd, callptr) < 0) {
t_error ("adsp server: t_listen");
exit (1);
if ((s2 = t_open (netconfigp->nc_device, 0_RDWR, (struct t_info *)0)) < 0) {
t_error ("adsp server: t_open");
exit (1);
if (t_bind (s2, (struct t_bind *)0, (struct t_bind *)0) < 0) {
```

```
t_error ("adsp server: t_bind");
exit (1);
if (t_accept (s, s2, callptr) < 0) {
if ((t_errno == TL00K) \&\& (t_look (s) == T_DISCONNECT))
if (t_rcvdis (s, (struct t_discon *) 0) < 0) {
t_error ("adsp server: t_rcvdis");
exit (1);
t_error ("adsp server: t_accept");
exit (1);
if (t_rcv (s2, buf, BUFSIZE, &flags) < 0) {
t_error ("adsp server: t_rcv");
exit (1);
printf ("This is what we got:\n%s\n", buf);
endnetconfig (handlep);
t_close (s);
t_close (s2);
The client side
/* adsp_client.c -- this is an ADSP client */
#include <tiuser.h>
#include <stropts.h>
#include <fcntl.h>
#include <stdio.h>
#include <signal.h>
#include <netconfig.h>
#include <netdir.h>
#include "appletalk.h"
#include "adsp.h"
main ()
i nt
```

```
struct t_call
                *callptr;
voi d
       *handlep;
struct netconfig
                   *netconfigp;
#define MESSAGE "Gypsy king, play my favorite tune. \n"
* Initialize the network selection mechanism.
if ((handlep = setnetpath()) == NULL) {
nc_perror ("adsp client: setnetpath");
exit (1);
* Loop through the transport providers.
while (netconfigp = getnetpath (handlep)) {
if ((netconfigp->nc_semantics == NC_TPI_COTS) &&
(strcmp (netconfigp->nc_protofmly, NC_APPLETALK) == 0) &&
(strcmp (netconfigp->nc_proto, "-") == 0))
break;
if (netconfigp == NULL) {
endnetconfig (handlep);
exit (1);
if ((s = t\_open (netconfigp->nc\_device, 0\_RDWR, (struct t\_info *)0)) < 0) {
t_error ("adsp client: t_open");
exit (1);
if (t_bind (s, (struct t_bind *) 0, (struct t_bind *) 0) < 0) {
t_error ("adsp client: t_bind");
exit (1);
if ((callptr = (struct t_call *)t_alloc (s, T_CALL, T_ADDR)) == NULL) {
t_error ("adsp client: t_alloc");
exit (1);
(void) strcpy (callptr->addr.buf, "Receiver: ADSP_Server@Nets_R_Us");
if (t_connect (s, callptr, (struct t_call *) 0) < 0) {
t_error ("adsp client: t_connect");
```

```
exit (1);
if (t\_snd (s, MESSAGE, size of (MESSAGE), 0) < 0) {
t_error ("adsp client: t_snd");
exit (1);
endnetconfig (handlep);
t_close (s);
Summary of ADSP data structures
The following is a summary of the constants and structures used by ADSP.
Constants
The following constants are defined:
TRUE = 0
FALSE = 1
The ADSP options have the following default values:
filter_addr = 0
checksum = FALSE
Structures
The following structures are used by the ADSP interface and are defined in
the <adsp. h> header:
typedef struct adsp_opt {
at_inet_t filter_addr;
u_char checksum;
u_char
        reserved [59];
} adsp_opt_t;
The following structures can be found in the <appletalk.h> header:
typedef struct at_inet {
u_short
          net;
u_char
         node;
u_char
         socket;
} at_inet_t;
The following structures are defined in TLI:
```

```
struct netbuf {
unsigned int maxlen;
unsigned int len;
char *buf;
struct t_bind {
struct netbuf addr;
unsigned qlen;
struct t_call {
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
struct t_optmgmt {
struct netbuf opt;
long flags;
```

SECTION 2: DDP TLI INTERFACE SPECIFICATION

This section describes a TLI interface to the Datagram Delivery Protocol (DDP). DDP is one component in the suite of AppleTalk protocols.

The section consists of several subsections:

- "General Concepts" explains $\ensuremath{\mathsf{STREAMS}}$ and TLI concepts relevant to this document.
- "Implementation Issues" describes several important implementation issues.
- "DDP TLI Library Calls and Parameters" describes each DDP function call and its parameters.
- "An Example" presents a sample program.
- "Summary of DDP Data Structures" presents DDP constants and data structures.

General concepts

The services of DDP are implemented with the use of TLI's connectionless-mode calls.

Receiving data

If data is available, the $t_rcvudata(3N)$ call returns immediately, indicating the number of bytes received. If data is not immediately available, the result of the $t_rcvudata(3N)$ call depends on the chosen mode:

- Asynchronous mode:

The call returns immediately, indicating failure. The user must either retry the call repeatedly, or "poll" for incoming data using t_look(3N).

- Synchronous mode:

The call is blocked until one of the following conditions becomes true:

- a datagram is received
- an error is detected

The $t_look(3N)$ function can be called to avoid blocking in a $t_rcvudata(3N)$ call.

Sending data

- Synchronous mode:

In order to maintain some flow control, the t_sndudata(3N) function returns when sending a new datagram again becomes possible. The process that sends data in synchronous mode may be blocked for some time.

- In asynchronous mode:

The transport provider may refuse to send a new datagram for flow control restrictions. In this case, the $t_sndudata(3N)$ call fails, returning a negative value and setting t_errno to [TFLOW]. The user may only retry later.

If $t_sndudata(3N)$ is issued before the destination user has activated its transport endpoint, the data unit may be discarded.

Error handling

Failures are indicated by return value of -1. An external integer, t_errno, holds the specific error numbers. When a function sets t_errno to [TSYSERR], it indicates an operating system error, the specifics about which can be accessed through the external variable errno. When a function sets t_errno to [TL00K], it indicates that an asynchronous event has happened. The user can call $t_look(3N)$ to determine what event has occurred.

TLI functions supported for DDP

The following TLI functions, which correspond to connectionless services, are supported:

 $t_bind(3N)$

 $t_{close(3N)}$

 $t_look(3N)$

```
t_open(3N)
t_rcvudata(3N)
t_rcvuderr(3N)
t_sndudata(3N)
t_unbi nd(3N)
The following general TLI functions are also supported:
t_alloc(3N)
t_error(3N)
t_free(3N)
t_getinfo(3N)
t_getstate(3N)
t_optmgmt(3N)
t_sync(3N)
The following functions are not supported and on being called will return -1,
with t_errno set to [TNOTSUPPORT]:
t_accept(3N)
t_connect(3N)
t_listen(3N)
t_rcv(3N)
t_rcvconnect(3N)
t_rcvdis(3N)
t_snd(3N)
t_snddis(3N)
t_rcvrel(3N)
t_sndrel(3N)
Key for parameter arrays
For each function, an array is presented that summarizes the content of the input and output parameters. The key for the parameter arrays is as follows:
The parameter value is meaningful. (The input parameter must be set before
the call and output parameter can be read after the call.)
```

The content of the object to which the x pointer points is

(x)

meaningful.

The parameter value is meaningful but the parameter is optional.

-- The parameter value is meaningless.

The parameter keeps the same value after the call as before the call.

Implementation issues

DDP TLI protocol address

There are two ways to view the TLI protocol address for DDP. Primitives like t_bind(3N) and t_sndudata(3N) accept an NBP entity name (object:type@zone) as their protocol address. The name registration, in t_bind(3N), or name resolution, in t_sndudata(3N), is done by the transport provider, transparent to the applications. This eliminates the need for NBP primitives in applications. The elimination of NBP primitives and use of the Name-to-Address Mapping provided by System V Release 4 are necessary to allow the development of protocol-independent applications. Since AppleTalk nodes are not required to have network names, NBP entity names are not available in some cases. Sometimes all that is available is the internet socket address. This is the case with t_rcvudata(3N). When a datagram is received, t_rcvudata(3N) returns the protocol address of the sending user. All that is available for return is the internet socket address. An application that has received a datagram is likely to respond with a datagram by calling t_sndudata(3N) and is likely to use the address returned by t_rcvudata(3N) as the destination address for its datagram. Hence, the t_sndudata(3N) primitive may be passed either an NBP entity name if an application is originating a datagram, or an internet socket address if an application is responding to one.

The transport provider must be able to distinguish between NBP entity names and internet socket addresses, so that it knows how to process the address field. The convention adopted involves using NULL-terminated strings for NBP entity names and using a leading NULL in the first character position, followed by an at_inet_t structure to identify an internet socket address. The t-bind(3N), t-rcvudata(3N), and t-rcvuderr(3N) calls return an internet socket address. See "Summary of DDP Data Structures" later in Section 2.

The object and type fields of the NBP entity names cannot contain wildcards. The zone name field may be set to an asterik (*). Using an illegal NBP entity name or a name that cannot be resolved causes t-bind(3N) or t-sndudata(3N) to return a -1 and t-errno to [TBADADDR]. Names that are transparently registered with a t_bind(3N) are transparently deregistered with t_unbind(3N) or upon the closing of the corresponding file descriptor.

 $\ensuremath{\text{Options}}$ and management parameters

The functions $t_sndudata(3N)$, $t_rcvudata(3N)$, and $t_rcvuderr(3N)$ contain an opt argument that is of the type struct netbuf. The opt.buf argument of the netbuf structure should point to a ddp_opt structure that contains the following fields:

Field Description

checksum Turns on/off DDP checksum

Checksum is a boolean variable that can be set to TRUE or FALSE, indicating the desire to turn on or off DDP checksum. Type identifies the DDP type. If opt.len is set to zero prior to a call to t_sndudata(3N), the default values of these options will be used. These default values can be changed by a call to t_optmgmt(3N). See "Summary of DDP Data Structures" later in Section 2 for a definition of the data structures described here and the default values of the options.

DDP TLI library calls and parameters

The following section describes several of the DDP TLI library function calls. For a more detailed description of these calls refer to the UNIX System V Release 4 Programmer's Guide: Networking Interfaces.

fd = t_open (path, oflag, info)

The t_open(3N) function is called as the first step in the initialization of a transport endpoint. This function returns various default characteristics of the underlying transport protocol by setting fields in the t_info structure.

The following should be the values returned by the call to $t_{open(3N)}$ and $t_{getinfo(3N)}$ with DDP as the transport provider:

Parameters	Before call	After call
path	DDP_DEV	
oflag	x	
info->addr		99
info->options		32
info->tsdu		586
info->etsdu		- 2
info->connect		- 2
i nf o- >di scon		- 2
info->servtype		T_CLTS

The argument path points to the DDP device identifier, for example, /dev/ddp, normally extracted from /etc/netconfig (see netconfig(4) and getnetpath(3N)). Oflag may be constructed from 0_NDELAY or 0_NONBLOCK OR-ed with 0_RDWR . T_open returns a file descriptor that identifies the local DDP endpoint. The default characteristics of DDP are returned in info. The maximum size of the DDP address, addr, is 99 bytes. The maximum size of TSDU is 586 bytes. No expedited data is allowed. No data can be sent with connection-request or disconnect calls. The only service type supported is T_CLTS, the connectionless mode.

t_bind (fd, req, ret)

This function associates a protocol address with the transport endpoint specified by fd and activates that transport endpoint. The transport provider can then begin sending or receiving data units. The req and ret arguments point to a t_bind structure.

1

Parameters	Before call	After call
req->addr. maxl en		
req->addr.len	x>=0	
req->addr.buf	x (x)	
req->qlen		
ret->addr.maxlen	x	
ret->addr.len		x
ret->addr.buf	x	(x)
req->qlen		

Req->addr.buf is a pointer to an NBP name (a NULL-terminated C string). The transport provider will allocate a dynamic socket and register that name on this socket.

If req is set to NULL or req->addr.len is zero, the transport provider will assign a dynamic socket and return the internet address of this socket in ret->addr.buf in the format described in "DDP TLI Protocol Address," earlier in Section 2.

t_unbind (fd)

This function disables the transport endpoint specified by fd, which was previously bound by $t_bind(3N)$. On completion of this call, no further data or events destined for this transport endpoint will be accepted by the transport provider.

Parameters	Before call	After call
fd	X	

The NBP name that was transparently registered by the transport provider during t_{0} will be deregistered.

t_sndudata (fd, unitdata)

This function is used to send a data unit to another DDP user. Fd identifies the local DDP endpoint and unit data points to a t_unit data structure.

Parameters	Before call	After call
fd	x	
unitdata->addr.maxlen		
uni tdata->addr. l en	X	

unitdata->addr.buf	$\mathbf{x}(\mathbf{x})$	
unitdata->opt.maxlen		
unitdata->opt.len	x	
unitdata->opt.buf	x(x)	
unitdata->udata.maxlen		
unitdata->udata.len	x	
uni tdata- >udata. buf	$\mathbf{x}(\mathbf{x})$	

In unitdata, addr specifies the protocol address of the destination transport user. The unitdata->addr points to a buffer containing an NBP name or to a buffer containing the address returned by t_rcvudata. (See "DDP TLI Protocol Address," earlier in Section 2.) When using a name, the transport provider will resolve the name before sending the ddp packet.

In unitdata, opt points to a ddp_opt structure, which is used to set the DDP type and checksum. The default values of the options will be used if opt.len is set to zero.

T_sndudata will send zero-length data units.

t_rcvudata (fd, unitdata, flags)

This function is used to receive a data unit from another DDP user. Fd identifies the local DDP endpoint and unitdata point to a t_unitdata structure.

Parameters	Before call	After call	
fd	x		
uni tdata->addr. maxl en	x		
uni tdata->addr. l en		x	
uni tdata->addr. buf	x	(x)	
unitdata->opt.maxlen	x		
uni tdata->opt. l en		x	
uni tdata->opt. buf	x	(x)	
unitdata->udata.maxlen##T##x##T##			
uni tdata- >udata. l en##T## ##T##x			
unitdata->udata.buf##T##x##T##(x)			
uni tdata- >fl ags##T## ##T##x			

In unitdata, addr specifies the DDP address of the destination user and opt

points to a ddp_opt structure, which identifies the DDP type and checksum of the incoming packet.

The maxlen fields of addr, opt, and udata must be set to indicate the maximum sizes of their buffers before issuing this function.

On return from this call, addr specifies the AppleTalk internet socket address of the sending user in the format described in "DDP TLI Protocol Address," earlier in Section 2; opt returns a ddp_opt structure, which identifies the DDP type and the checksum of the received packet; and udata specifies the user data that was received.

If the buffer defined in the udata field of unitdata is not large enough to hold the current data unit, the buffer will be filled and T_MORE will be set in flags on return to indicate that another t_rcvudata should be issued to retrieve the rest of the data unit. Subsequent t_rcvudata calls will return zero for the length of the address and the options until the full data unit has been received.

t_rcvuderr (fd, uderr)

This function is used to receive information concerning an error on a previously sent data unit and should only be issued following a unit data error indication $(t_look(3N) \text{ returning a } T_UDERR)$.

Parameters##T##Before call##T##After call

fd##T##x##T##--

uderr->addr. maxl en##T##x##T##--

uderr->addr.len##T##--##T##x

uderr->addr. buf#T##x##T##(x)

uderr->opt. maxl en##T##x##T##--

uderr->opt.len##T##--##T##x

uderr->opt. buf#T##x##T##(x)

uderr->error##T##--##T##x

Fd identifies the local DDP endpoint and uderr points to a t_uderr structure. See "Summary of DDP Data Structures," later in Section 2.

The maxlen fields of addr and opt must be set to indicate the maximum sizes of their buffers before issuing the function.

On return from this call, the addr structure specifies the destination address of the erroneous data unit, the opt structure identifies the DDP type and the checksum fields, and error indicates the specific DDP error code. See "Summary of DDP Data Structures," later in Section 2. No DDP error codes are currently defined.

If the user does not care to identify the data unit that produced an error, uderr can be set to NULL, and t_rcvuderr will simply clear the error indication without reporting any information to the user.

```
t_optmgmt (fd, req, ret)
This function enables a transport
protocol options with the transpo
```

This function enables a transport user to retrieve, verify, or negotiate protocol options with the transport provider. The argument fd identifies a bound transport endpoint. The req and ret arguments point to a t_optmgmt structure.

Parameters##T##Before call##T##After call

```
fd##T##x##T##--
req->opt. maxl en##T##--##T##--
req->opt. l en##T##x##T##--
req->opt. buf##T##x(x) ##T##--
req->fl ags##T##x##T##--
ret->opt. maxl en##T##x##T##--
ret->opt. l en##T##--##T##x
ret->opt. buf##T##x##T##(x)
req->fl ags##T##--##T##x
```

The req->opt.buf points to a ddp_opt structure. The user can set the default values of checksum and type.

t_close (fd)

The t_close function informs the transport provider that the user is finished.

Parameters##T##Before call##T##After call

fd##T##x##T##--

The NBP name that was transparently registered by the transport provider during $t_bind(3N)$ will be deregistered.

An example

The following is a simple example to demonstrate the calling conventions:

The sender side

```
/* ddp_sender.c -- this is the DDP sender side */
#include <tiuser.h>
#include <stropts.h>
#include <fcntl.h>
#include <stdio.h>
```

```
#include <signal.h>
#include <netconfig.h>
#include <netdir.h>
#include "appletalk.h"
#include "ddp.h"
main ()
int##T##s;
struct t_unitdata##T##*datagram;
struct ddp_opt##T##*optbufp;
voi d##T##*handl ep;
struct netconfig##T##*netconfigp;
#define MESSAGE "Gypsy king, play my favorite tune.\n"
* Initialize the network selection mechanism.
if ((handlep = setnetcpath()) == NULL) {
nc_perror ("ddp sender: setnetpath");
exit (1);
* Loop through the transport providers.
while (netconfigp = getnetpath (handlep)) {
if ((netconfigp->nc_semantics == NC_TPI_CLTS) &&
(strcmp (netconfigp->nc_protofmly, NC_APPLETALK) == 0) &&
(strcmp (netconfigp->nc_proto, "-") == 0))
break;
if (netconfigp == NULL) {
endnetconfig (handlep);
exit (1);
if ((s = t_{open} (netconfigp->nc_{device}, 0_{RDWR}, (struct t_{info} *)0)) < 0)  {
t_error ("ddp sender: t_open");
exit (1);
if (t_bind (s, (struct t_bind *) 0, (struct t_bind *) 0) < 0) {
```

```
t_error ("ddp sender: t_bind");
exit (1);
if ((datagram = (struct t_unitdata *)t_alloc (s, T_UNITDATA, T_ADDR|T_0PT))
== NULL) {
t_error ("ddp sender: t_alloc");
exit (1);
optbufp = (struct ddp_opt *) datagram->opt.buf;
optbufp->checksum = TRUE;
optbufp->type = 2;
datagram->udata.maxlen = sizeof (MESSAGE);
datagram->udata.len = sizeof (MESSAGE);
datagram->udata.buf = MESSAGE;
(void) strcpy (datagram->addr.buf, "Receiver: DDP_Server@Nets_R_Us");
if (t_sndudata (s, &datagram) < 0) {
t_error ("ddp sender: t_sndudata");
exit (1);
t_close (s);
endnetconfig (handlep);
The receiver side
/* ddp_reciver.c -- this is the DDP receiver side */
#include <tiuser.h>
#include <stropts.h>
#include <fcntl.h>
#include <stdio.h>
#include <signal.h>
#include <netconfig.h>
#include <netdir.h>
#include "appletalk.h"
#include "ddp.h"
main ()
```

```
int##T##s, flags;
struct t_bind##T##*bind;
struct t_unitdata##T##*datagram;
voi d##T##*handl ep;
struct netconfig##T##*netconfigp;
* Initialize the network selection mechanism.
if ((handlep = setnetpath()) == NULL) {
nc_perror ("ddp receiver: setnetpath");
exit (1);
* Loop through the transport providers.
while (netconfigp = getnetpath (handlep)) {
if ((netconfigp->nc_semantics == NC_TPI_CLTS) &&
(strcmp (netconfigp->nc_protofmly, NC_APPLETALK) == 0) &&
(strcmp (netconfigp->nc_proto, "-") == 0))
break;
if (netconfigp == NULL) {
endnetconfig (handlep);
exit (1);
if ((s = t_{open} (netconfigp->nc_{device}, 0_RDWR, (struct t_info *)0)) < 0) {
t_error ("ddp receiver: t_open");
exit (1);
if ((bind = (struct t_bind *)t_alloc (s, T_BIND, T_ADDR)) == NULL) {
t_error ("ddp server: t_alloc");
exit (1);
(void) strcpy (bind->addr.buf, "Receiver: DDP_Server@Nets_R_Us");
if (t_bind (s, bind, (struct t_bind *) 0) < 0) {
t_error ("ddp receiver: t_bind");
exit (1);
if ((datagram = (struct t_unitdata *)t_alloc (s, T_UNITDATA, T_ALL)) == NULL)
```

```
t_error ("ddp receiver: t_alloc");
exit (1);
printf ("Waiting for data...\n");
if (t_rcvudata (s, &datagram, &flags) < 0) {
t_error ("ddp receiver: t_rcvudata");
exit (1);
printf ("This is what we got:\n%s\n", datagram->udata.buf);
t_close (s);
endnetconfig (handlep);
Summary of DDP data structures
The following is a summary of the constants and structures used by DDP.
Constants
The following constants are defined:
TRUE = 0
FALSE = 1
The DDP options have the following default values:
checksum = FALSE
type = 0
Structures
The following structures are used by the DDP interface and are defined in
the <ddp. h> header.
typedef struct ddp_opt {
u_char checksum;
u_char type;
u_char##T##reserved [30];
} ddp_opt_t;
The following structures can be found in the <appletalk.h> header.
typedef struct at_inet {
u_short##T##net;
u_char##T##node;
```

```
u_char##T##socket;
} at_i net_t;
The following structures are defined in TLI:
struct netbuf {
unsigned int maxlen;
unsigned int len;
char *buf;
struct t_bind {
struct netbuf addr;
unsigned qlen;
struct t_call {
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
int sequence;
struct t_unitdata {
struct netbuf addr;
struct netbuf opt;
struct netbuf udata;
struct t_uderr {
struct netbuf addr;
struct netbuf opt;
```

long error;