23. The Chaosnet

The purpose of the basic software protocol of Chaosnet is to allow high-speed communication among processes on different machines, with no undetected transmission errors.

23.1 Chaosnet Overview

The principal service provided by Chaosnet is a *connection* between two user processes. This is a full-duplex reliable packet-transmission channel. The network undertakes never to garble, lose, duplicate, or resequence the packets; in the event of a serious error it may break the connection off entirely, informing both user processes. User programs may deal explicitly in terms of packets. They may also ignore packet boundaries and treat the connection as two unidirectional streams of 8-bit or 16-bit bytes, but this really works by means of packets.

If you just want to ask a question of another process or host and receive a reply, you can use a *simple transaction*: You send only one packet to the other host, and it sends one packet back. This is more efficient than establishing a connection and using it only briefly. In a simple transaction, the server cannot tell whether the user received the answer; and if the user does not receive the answer, it cannot tell whether the server received the question. In fact, the server might receive the question more than once. If this is unacceptable, a connection must be used.

Each node (or host) on the network is identified by an *address*, which is a 16-bit number. These addresses are used in the routing of packets. There is a table (the system hosts table, SYS: CHAOS; HOSTS TXT) that relates symbolic host names to numeric host addresses. The host table can record addresses on any number of different networks, and in certain contexts a host address is meaningful only together with the name of the network it is for.

The data transmitted over connections are in units called *packets*. Each packet contains an 8-bit number, the *opcode*, which indicates what its function is. Opcodes less than 200 (octal) are special purpose. Each such opcode that is used has an assigned name and a specific function. Users need not know about all of them. Opcodes 200 through 277 are used for 8-bit user data. Opcodes 300 through 377 are used for 16-bit user data.

Each packet also contains some number of data bytes, whose meaning depends on the opcode. If the opcode is for user data, then it is up to the application user software to decide on the interpretation.

Establishing a connection:

A connection is created because one process sends a request to a host. The request is a packet containing the special-purpose opcode RFC. The data contains a *contact name* which is used to find the process to connect to. There may be a process on the target host *listening* on this contact name. If so, it decides whether to agree to the connection. Alternatively, the contact name can be the name of a standard service such as TELNET. In this case, the receiving host will create a process to respond, loaded with the program for that service.

Once a connection has been established, there is no more need for the contact name and it is discarded. The Lisp machine remembers what contact name was used to open a connection, but this is only for the user's information.

In the case where two existing processes that already know about each other want to establish a connection, they must agree on a contact name, and then one of them must send the request while the other listens. They must agree between themselves which is to do which.

Contact names are restricted to strings of upper-case letters, numbers, and ASCII punctuation. The maximum length of a contact name is limited only by the packet size, although on ITS hosts the names of automatically-started servers are limited by the file-system to six characters. The contact name is terminated by a space. If the RFC packet contains data beyond the contact name, it is just for interpretation by the listening process, which can also use it in deciding whether to accept the connection.

A simple transaction is also begun with an RFC packet. There is nothing in the RFC packet which indicates whether it is intended to start a connection or a simple transaction. The server has the option of doing either one. But normally any given server always does one or the other, and the requestor knows which one to expect.

The server accepts the request for a connection by sending an OPN packet (a packet with opcode OPN) to the requestor. It can also refuse the connection by sending a CLS packet. The data in the CLS packet is a string explaining the reason for the refusal. Another alternative is to tell the requestor to try a different host or a different contact name. This is called *forwarding* the request, and is done with a FWD packet.

The server can also respond with an answer, an ANS packet, which is the second half of a simple transaction. (Refusing and forwarding are also meaningful when a simple transaction is intended, just as when a connection is intended).

Once the connection is open:

Data transmitted through Chaosnet generally follow Lisp Machine standards. Bits and bytes are numbered from right to left, or least-significant to most-significant. The first 8-bit byte in a 16-bit word is the one in the arithmetically least-significant position. The first 16-bit word in a 32-bit double-word is the one in the arithmetically least-significant position.

The character set used is dictated by the higher-level protocol in use. Telnet and Supdup, for example, each specifies its own ASCII-based character set. The "default" character set—used for new protocols and for text that appears in the basic Chaosnet protocol, such as contact names—is the Lisp Machine character set.

Because PDP-10s need to reorder the characters in a word in order to access them conveniently, it is necessary to distinguish 8-bit data from 16-bit data. They are distinguished by the packet opcode: some opcodes imply 8-bit bytes and some imply 16-bit bytes. Packets known to contain 8-bit bytes, including opcodes 200 through 277, will be byte-swapped by the PDP-10's front end.

If one process tries to send data faster than the other can process it, the buffered packets could devour lots of memory. Preventing this is the purpose of *flow control*. Each process specifies a *window size*, which is the number of packets that are allowed to be waiting for that process to read. Attempting to send on a connection whose other side's window is full will block until the other side reads some packets. The default window size is 13, but for some applications you might wish to specify a larger value. There is little reason ever to specify a smaller value.

Breaking a connection:

Either end of a connection can break the connection abruptly by sending a CLS packet. The data in this packet is a string describing why the connection was broken.

To break a connection gently, it is necessary to verify that all the data transmitted was received properly before sending a CLS. This matters in some applications and is unnecessary in others. When it is needed, it is done by sending a special packet, an EOF packet, which is mostly like a data packet except for its significance with regard to closing the connection. The EOF packet is like the words "the end" at the end of a book: it tells the recipient that it has received all the data it is supposed to receive, that there are no missing pages that should have followed. When the sender of the EOF sees the acknowledgement for the EOF packet, indicating that the EOF was received and understood, it can break the connection with a CLS.

If a process that expects to receive an EOF gets a CLS with no EOF, it takes this to mean that the connection was broken before the transmission was finished. If the process does receive an EOF, it does not break the connection itself immediately. It waits to see the sender of the EOF break it. If this does not happen in a few seconds, the EOF recipient can break the connection.

It is illegal to put data in an EOF packet; in other words, the byte count should always be zero. Most Chaosnet implementations will simply ignore any data that is present in an EOF.

If both sides are sending data and both need to know for certain where "the end" is, they must do something a little more complicated. Arbitrarily call one party the user and the other the server. The protocol is that after sending all its data, each party sends an EOF and waits for it to be acknowledged. The server, having seen its EOF acknowledged, sends a second EOF. The user, having seen its EOF acknowledged, looks for a second EOF and *then* sends a CLS and goes away. The server goes away when it sees the user's CLS, or after a brief timeout has elapsed. This asymmetrical protocol guarantees that each side gets a chance to know that both sides agree that all the data have been transferred. The first CLS will only be sent after both sides have waited for their (first) EOF to be acknowledged.

Clearing up inconsistencies:

If a host crashes, it is supposed to forget all the connections that it had. When a packet arrives on one of the former connections, the host will report "no such connection" to the sender with a LOS packet, whose data is a string explaining what happened. The same thing happens if a CLS packet is lost; the intended recipient may keep trying to use the connection that the other side (which sent the CLS) no longer believes should exist. LOS packets are used whenever a host receives a packet that it should not be getting; the recipient of the LOS packet knows that the connection it thought it was using does not exist any more.

23.2 Conns

On the Lisp machine, your handle on a connection is a named structure of type chaos:conn. The conn may have an actual connection attached to it, or it may have a connection still being made, or record that a connection was refused, closed or broken.

chaos:inactive-state
: This conn is not really in use at all.

chaos:rfc-sent-state
: This conn was used to request a connection to another process, but no reply has been received. When the reply is received, it may change the conn's state to chaos:answered-state, chaos:cls-received-state, or chaos:open-state.

chaos:listening-state
: This conn is being used to listen with. If a RFC packet is received for the contact name you are listening on, the state will change to chaos:rfc-received-state.

chaos:rfc-received-state
: This means that your listen has "heard" an RFC packet that matches it. You can accept, reject, forward or answer the request. Accepting goes to state chaos:open-state; refusing or forwarding goes to to state chaos:inactive-state.

chaos:open-state
: This conn is one end of an open connection. You can receive any data packets that are waiting and you can transmit data.

chaos:answered-state
: This conn was used to send an RFC packet and an ANS packet was received in response (a simple transaction answer arrived). You can read the ANS packet, that is all.

chaos:cls-received-state
: This conn has received a CLS packet (the connection was closed or refused). You can read any data packets that came in before the CLS; after them you can read the CLS.

chaos:los-received-state
: This conn's connection was broken and the other end sent a LOS packet to say so. The LOS packet is the only packet available to be read.

chaos:host-down-state
: The host at the other end of this conn's connection has not responded to anything for a significant time.

chaos:foreign-state
: The connection is being used with a foreign protocol encapsulated in UNC packets (see the Chaosnet memo for more information about this).

These are the fields of a conn that you might be interested in:

chaos:conn-state *conn*
This slot holds the state of *conn*. It is one of the symbols listed above.

chaos:conn-foreign-address *conn*
Returns the address of the host at the other end of this connection. Use si:get-host-from-address to find out which host this is (see page 480).

chaos:conn-read-pkts *conn*
Internally threaded chain of incoming packets available to be read from *conn*.

Its main use for the applications programmer is to test whether there are any incoming packets.

chaos:conn-window-available *conn*
Returns the number of packets you may transmit before the network software forces you to wait for the receiver to read some. This is just a minimum. By the time you actually send this many packets, the receiver may already have said he has room for some more.

chaos:conn-plist *conn*
This slot is used to store arbitrary properties on *conn*. You can store properties yourself; use property names that are not in the chaos package to avoid conflict.

chaos:contact-name *conn*
Returns the contact name with which *conn* was created. The contact name is not significant to the functioning of the connection once an RFC and LSN have matched, but it is remembered for the sake of debugging.

chaos:wait *conn state timeout* &optional *whostate*
Waits until the state of *conn* is not the symbol *state*, or until *timeout* 60ths of a second have elapsed. If the timeout occurs, nil is returned; otherwise t is returned. *whostate* is the process state to put in the who-line; it defaults to "Chaosnet wait".

23.3 Opening and Closing Connections

23.3.1 User-Side

chaos:connect *host contact-name* &optional *window-size timeout*
Opens a stream connection; returns a conn if it succeeds or else a string giving the reason for failure. *host* may be a number or the name of a known host. *contact-name* is a string containing the contact name and any additional arguments to go in the RFC packet. If *window-size* is not specified it defaults to 13. If *timeout* is not specified it defaults to 600 (ten seconds).

chaos:simple *host contact-name* &optional *timeout*
Taking arguments similar to those of chaos:connect, this performs the user side of a simple-transaction. The returned value is either an ANS packet or a string containing a failure message. The ANS packet should be disposed of (using chaos:return-pkt, see below) when you are done with it.

chaos:remove-conn *conn*
Makes *conn* null and void. It becomes inactive, all its buffered packets are freed, and the corresponding Chaosnet connection (if any) goes away.

chaos:close *conn* &optional *reason*
Closes and removes the connection. If it is open, a CLS packet is sent containing the string *reason*. Don't use this to reject RFC's; use chaos:reject for that.

chaos:open-foreign-connection *host index* &optional *pkt-allocation distinguished-port*
Creates a conn that may be used to transmit and receive foreign protocols encapsulated in UNC packets. *host* and *index* are the destination address for packets sent with chaos:send-unc-pkt. *pkt-allocation* is the "window size", i.e. the maximum number of input packets that may be buffered. It defaults to 10. If *distinguished-port* is supplied, the local index is set to it. This is necessary for protocols that define the meanings of particular index numbers.

See the AI memo on the chaosnet for more information on using foreign protocols.

23.3.2 Server-Side

chaos:listen *contact-name* &optional *window-size wait-for-rfc*
Waits for an RFC for the specified contact name to arrive, then returns a conn that will be in chaos:rfc-received-state. If *window-size* is not specified it defaults to 13. If *wait-for-rfc* is specified as nil (it defaults to t) then the conn will be returned immediately without waiting for an RFC to arrive.

chaos:server-alist *Variable*
Contains an entry for each server that always exists. When an RFC arrives for one of these servers, the specified form is evaluated in the background process; typically it creates a process that will then do a chaos:listen. Use the add-initialization function to add entries to this list.

chaos:accept *conn*
conn must be in chaos:rfc-received-state. An OPN packet will be transmitted and *conn* will enter the *Open* state. If the RFC packet has not already been read with chaos:get-next-pkt, it is discarded. You should read it before accepting, if it contains arguments in addition to the contact name.

chaos:reject *conn reason*
conn must be in chaos:rfc-received-state. A CLS packet containing the string *reason* will be sent and *conn* will be removed.

chaos:forward-all *contact-name host*
Cause all future requests for connection to this host on *contact-name* to be forwarded to the same contact name at host *host*.

chaos:answer-string *conn string*
conn must be in chaos:rfc-received-state. An ANS packet containing the string *string* will be sent and *conn* will be removed.

chaos:answer *conn pkt*
conn must be in chaos:rfc-received-state. *pkt* is transmitted as an ANS packet and *conn* is removed. Use this function when the answer is some binary data rather than a text string.

chaos:fast-answer-string *contact-name string*
If a pending RFC exists to *contact-name*, an ANS containing *string* is sent in response to it and t is returned. Otherwise nil is returned. This function involves the minimum possible overhead. No conn is created.

23.4 Stream Input and Output

chaos:open-stream *host contact-name* &rest *options*
Opens a chaosnet connection and returns a stream that does i/o to it. *host* is the host to connect to; *contact-name* is the contact name at that host. These two arguments are passed along to chaos:connect. *options* is a list of alternating keywords and values.

:window-size
:timeout — These two arguments specify two arguments for chaos:connect.

:error — If the value is non-nil, a failure to connect causes a Lisp error. Otherwise, it causes a string describing the error to be returned.

:direction
:characters
:ascii-translation
These three arguments are passed along to chaos:make-stream.

chaos:make-stream *conn* &key &optional *direction characters ascii-translation*
Creates and returns a stream that does i/o on the connection *conn*, which should be open as a stream connection. *direction* may be :input, :output or :bidirectional.

If *characters* is non-nil (which is the default), the stream reads and writes 8-bit bytes. If *characters* is nil, the stream reads and writes 16-bit bytes.

If *ascii-translation* is non-nil, characters written to the stream are translated to standard ASCII before they are sent, and characters read are translated from ASCII to the Lisp machine character set.

:foreign-host *Operation on* chaos:basic-stream

Returns the host object for the host at the other end of this stream's connection.

:close *Operation on* chaos:basic-stream

Send a CLS packet and remove the connection.

:force-output *Operation on* chaos:basic-output-stream

Any buffered output is transmitted. Normally output is accumulated until a full packet's worth of bytes are available, so that maximum-size packets are transmitted.

:finish *Operation on* chaos:basic-output-stream

Waits until either all packets have been sent and acknowledged, or the connection ceases to be open. If successful, returns t; if the connection goes into a bad state, returns nil.

:eof *Operation on* chaos:basic-output-stream

Forces out any buffered output, sends an EOF packet, and does a :finish.

:clear-eof *Operation on* chaos:basic-input-stream

Allows you to read past an EOF packet on input. Each :tyi will return nil or signal the specified eof error until a :clear-eof is done.

23.5 Packet Input and Output

Input and output on a Chaosnet connection can be done at the whole-packet level, using the functions in this section. A packet is represented by a chaos:pkt data structure. Allocation of pkts is controlled by the system; each pkt that it gives you must be given back. There are functions to convert between pkts and strings. A pkt is an art-16b array containing the packet header and data; the leader of a pkt contains a number of fields used by the system.

chaos:first-data-word-in-pkt *Variable*

This is the index in any pkt of the element that is the first 16-bit word of user data. (Preceding elements are used to store a header used by the hardware.)

chaos:pkt-opcode *pkt*

Accessor for the opcode of the packet *pkt*. To set the opcode, do

```
(setf (chaos:pkt-opcode my-pkt) my-opcode)
```

The system provides names for all the opcodes standardly used. The names useful to the applications programmer appear at the end of this section.

chaos:pkt-nbytes *pkt*

Accessor for the number-of-data-bytes field of *pkt*'s. This field says how much of *pkt*'s contects are valid data, measured in 8-bit bytes. This field can be set with setf also.

chaos:pkt-string *pkt*

An indirect array that is the data field of *pkt* as a string of 8-bit bytes. The length of this string is equal to (chaos:pkt-nbytes *pkt*). If you wish to record the contects of *pkt* permanently, you must copy this string.

chaos:set-pkt-string *pkt* &rest *strings*
Copies the *strings* into the data field of *pkt*, concatenating them, and sets (chaos:pkt-nbytes *pkt*) accordingly.

chaos:get-pkt
Allocates a pkt for use by the user.

chaos:return-pkt *pkt*
Returns *pkt* to the system for reuse. The packets given to you by chaos:get-pkt, chaos:get-next-pkt and chaos:simple should be returned to the system in this way when you are finished with them.

chaos:send-pkt *conn pkt* &optional (*opcode* chaos:dat-op)
Transmits *pkt* on *conn*. *pkt* should have been allocated with chaos:get-pkt and then had its data field and n-bytes filled in. *opcode* must be a data opcode (200 or more) or EOF. An error is signalled, with condition chaos:not-open-state, if *conn* is not open.

Giving a pkt to chaos:send-pkt constitutes giving it back to the system. You do not need to call chaos:return-pkt.

chaos:send-string *conn* &rest *strings*
Sends a data packet containing the concatenation of *strings* as its data.

chaos:send-unc-pkt *conn pkt* &optional *pkt-number ack-number*
Transmits *pkt*, an UNC packet, on *conn*. The opcode, packet number, and acknowledge number fields in the packet header are filled in (the latter two only if the optional arguments are supplied).

See the AI memo on the chaosnet for more information on using foreign protocols.

chaos:may-transmit *conn*
A predicate that returns t if there is any space in the window for transmitting on *conn*. If the value is nil, you may have to wait if you try to transmit. If the value is t, you certainly do not have to wait.

chaos:finish *conn* &optional (*whostate* "Net Finish")
Waits until either all packets have been sent and acknowledged, or the connection ceases to be open. If successful, returns t; if the connection goes into a bad state, returns nil. *whostate* is the process state to display in the who-line while waiting.

chaos:get-next-pkt *conn* &optional (*no-hang-p* nil)
Returns the next input packet from *conn*. When you are done with the packet you must give it back to the system with chaos:return-pkt. This can return an RFC, CLS, or ANS packet, in addition to data, UNC, or EOF. If *no-hang-p* is t, nil will be returned if there are no packets available or the connection is in a bad state. Otherwise an error will be signalled if the connection is in a bad state, with condition name chaos:host-down, chaos:los-received-state, or chaos:read-on-closed-connection. If no packets are available and *no-hang-p* is nil, chaos:get-next-pkt will wait for packets to come in or the state to change. The process state in the who-line is "Chaosnet Input".

chaos:data-available *conn*
A predicate that returns t if there any input packets available from *conn*.

Here are symbolic names for the opcodes that an applications programmer needs to know about:

chaos:rfc-op *Variable*
This special-purpose opcode is used for requesting a connection. The data consists of the contact name terminated by a space character, followed optionally by additional data whose meaning is up to the server for that contact name.

chaos:lsn-op *Variable*
This special-purpose opcode is used when you ask to listen on a contact name. The data is just the contact name. This packet is never actually sent over the network, just kept in the chaosnet software and compared with the contact names in RFC packets that arrive.

chaos:opn-op *Variable*
This special-purpose opcode is used by the server process to accept the request for a connection conveyed by an RFC packet. Its data serves only internal functions.

chaos:ans-op *Variable*
This special-purpose opcode is used to send a simple reply. The simple reply is sent back in place of opening a connection.

chaos:los-op *Variable*
This special-purpose packet is what you will receive if you try to use a connection that has been broken. Its data is a message explaining the situation, which you can print for the user.

chaos:cls-op *Variable*
This special-purpose packet is used to close a connection. Its data is a message explaining the reason, and it can be printed for the user. Note that you cannot count on receiving a CLS packet because it is not retransmitted if it is lost. If that happens you will get a LOS when you try to use the connection (thinking it is still open).

CLS packets are also used for refusing to open a connection in the first place.

chaos:eof-op *Variable*
This special-purpose opcode is used to indicate the end of the data that you really want to transmit. When this packet is acknowledged by the other process, you know that all the real data was received properly. You can wait for this with **chaos:finish**. The EOF packet carries no data itself.

chaos:dat-op *Variable*
This is opcode 200, which is the normal opcode used for 8-bit user data. Some protocols use multiple data opcodes in the range 200 through 277, but simple protocols that do not need to distinguish types of packets just use opcode 200.

23.6 Connection Interrupts

chaos:interrupt-function *conn*

This attribute of a conn is a function to be called when certain events occur on this connection. Normally this is nil, which means not to call any function, but you can use setf to store a function here. Since the function is called in the Chaosnet background process, it should not do any operations that might have to wait for the network, since that could permanently hang the background process.

The function's first argument is one of the following symbols, giving the reason for the "interrupt". The function's second argument is *conn*. Additional arguments may be present depending on the reason. The possible reasons are:

:input — A packet has arrived for the connection when it had no input packets queued. It is now possible to do chaos:get-next-pkt without having to wait. There are no additional arguments.

:output — An acknowledgement has arrived for the connection and made space in the window when formerly it was full. Additional output packets may now be transmitted with chaos:send-pkt without having to wait. There are no additional arguments.

:change-of-state — The state of the connection has changed. The third argument to the function is the symbol for the new state.

chaos:read-pkts *conn*

Some interrupt functions will want to look at the queued input packets of a connection when they get a :input interrupt. chaos:read-pkts returns the first packet available for reading. Successive packets can be found by following chaos:pkt-link.

chaos:pkt-link *pkt*

Lists of packets in the NCP are threaded together by storing each packet in the chaos:pkt-link of its predecessor. The list is terminated with nil.

23.7 Chaosnet Errors

sys:network-error (error) *Condition Flavor*

All errors from the chaosnet code use flavors built on this one.

23.7.1 Local Problems

sys:local-network-error (sys:network-error error) *Condition Flavor*

This flavor is used for problems in connection with the chaosnet that have entirely to do with what is going on in this Lisp machine.

sys:network-resources-exhausted (sys:local-network-error sys:network-error error) *Condition*

Signaled when some local resource in the NCP was exhausted. Most likely, there are too many chaosnet connections and the connection table is full.

sys:unknown-address (sys:local-network-error sys:network-error error) *Condition*

The *address* argument to chaos:connect or some similar function was not recognizable. The :address operation on the condition instance returns the address that was supplied.

23.7.2 Problems Involving Other Machines' Actions

sys:remote-network-error (sys:network-error error) *Condition Flavor*

This flavor is used for network problems that involve the actions (or lack of them) of other machines. It is often useful to test for as a condition name.

The operations :connection and :foreign-host return the chaos:conn object and the host object for the foreign host.

Every instance of sys:remote-network-error is either a sys:connection-error or a sys:bad-connection-state.

sys:connection-error (sys:remote-network-error error) *Condition*

This condition name categorizes failure to complete a connection.

sys:bad-connection-state (sys:remote-network-error error) *Condition*

This condition name categorizes errors where an existing, valid connection becomes invalid. The error is not signaled until you try to use the connection.

sys:host-not-responding (sys:remote-network-error error) *Condition*

This condition name categorizes errors where no packets whatever are received from the foreign host, making it seem likely that that host or the network is down.

sys:host-not-responding-during-connection (sys:connection-error sys:host-not-responding sys:remote-network-error error) *Condition*

This condition is signaled when a host does not respond while it is being asked to make a connection.

sys:host-stopped-responding (sys:bad-connection-state sys:host-not-responding sys:remote-network-error error) *Condition*

This condition is signaled when a host does not respond even though a connection to it already exists.

sys:connection-refused (sys:connection-error sys:remote-network-error error) *Condition*

This is signaled when a connection is refused.

The :reason operation on the condition instance returns the reason specified in the CLS packet (a string) or nil if no reason was given.

sys:connection-closed (sys:bad-connection-state sys:remote-network-error error) *Condition*

This is signaled when you try to send on a connection which has been closed by the other host.

The :reason operation on the condition instance returns the reason specified in the CLS packet (a string) or nil if no reason was given.

sys:connection-lost (sys:bad-connection-state sys:remote-network-error error) *Condition*

This is signaled when you try to use a connection on which a LOS packet was received.

The :reason operation on the condition instance returns the reason specified in the CLS packet (a string) or nil if no reason was given.

sys:connection-no-more-data (sys:bad-connection-state sys:remote-network-error error) *Condition*

This is signaled when you try to read from a connection which has been closed by the other host, when there are no packets left to be read. (It is no error to read from a connection which has been closed, if you have not yet read all the packets which arrived, including the CLS packet).

The :reason operation on the condition instance returns the reason specified in the CLS packet (a string) or nil if no reason was given.

23.8 Information and Control

chaos:host-data &optional *host*

host may be a number or a known host name, and defaults to the local host. Two values are returned. The first value is the host name and the second is the host number. If the host is a number not in the table, it is asked its name using the STATUS protocol; if no response is received the name "Unknown" is returned.

chaos:print-conn *conn* &optional (*short* t)
Prints everything the system knows about the connection. If *short* is nil it also prints everything the system knows about each queued input and output packet on the connection.

chaos:print-pkt *pkt* &optional (*short* nil)
Prints everything the system knows about the packet, except its data field. If *short* is t, only the first line of the information is printed.

chaos:print-all-pkts *pkt* &optional (*short* t)
Calls chaos:print-pkt on *pkt* and all packets on the threaded list emanating from it.

chaos:status
Prints the hardware status.

chaos:reset
Resets the hardware and software and turns off the network.

chaos:assure-enabled
Turns on the network if it is not already on. It is normally always on unless you call one of these functions.

chaos:enable
Resets the hardware and turns on the network.

chaos:disable
Resets the hardware and turns off the network.

The PEEK program has a mode that displays the status of all of the Lisp machine's chaosnet connections, and various other information, in a continuously updating fashion.

23.9 Higher-Level Protocols

This section briefly documents some of the higher-level protocols of the most general interest. There are quite a few other protocols which are too specialized to mention here. All protocols other than the STATUS protocol are optional and are only implemented by those hosts that need them. All hosts are required to implement the STATUS protocol since it is used for network maintenance.

The site files tell the Lisp machine which hosts at your site implement certain higher-level protocols. See section 32.11, page 657.

23.9.1 Status

All network nodes, even bridges, are required to answer RFC's with contact name STATUS, returning an ANS packet in a simple transaction. This protocol is primarily used for network maintenance. The answer to a STATUS request should be generated by the Network Control Program, rather than by starting up a server process, in order to provide rapid response.

The STATUS protocol is used to determine whether a host is up, to determine whether an operable path through the network exists between two hosts, to monitor network error statistics, and to debug new Network Control Programs and new Chaosnet hardware. The **hostat** function on the Lisp machine uses this protocol.

The first 32 bytes of the ANS contain the name of the node, padded on the right with zero bytes. The rest of the packet contains blocks of information expressed in 16-bit and 32-bit words, low byte first (PDP-11/Lisp machine style). The low-order half of a 32-bit word comes first. Since ANS packets contain 8-bit data (not 16-bit), machines such as PDP-10s which store numbers high byte first will have to shuffle the bytes when using this protocol. The first 16-bit word in a block is its identification. The second 16-bit word is the number of 16-bit words to follow. The remaining words in the block depend on the identification.

This is the only block type currently defined. All items are optional, according to the count field, and extra items not defined here may be present and should be ignored. Note that items after the first two are 32-bit words.

word 0 — A number between 400 and 777 octal. This is 400 plus a subnet number. This block contains information on this host's direct connection to that subnet.

word 1 — The number of 16-bit words to follow, usually 16.

words 2-3 — The number of packets received from this subnet.

words 4-5 — The number of packets transmitted to this subnet.

words 6-7 — The number of transmissions to this subnet aborted by collisions or because the receiver was busy.

words 8-9 — The number of incoming packets from this subnet lost because the host had not yet read a previous packet out of the interface and consequently the interface could not capture the packet.

words 10-11 — The number of incoming packets from this subnet with CRC errors. These were either transmitted wrong or damaged in transmission.

words 12-13 — The number of incoming packets from this subnet that had no CRC error when received, but did have an error after being read out of the packet buffer. This error indicates either a hardware problem with the packet buffer or an incorrect packet length.

words 14-15 — The number of incoming packets from this subnet that were rejected due to incorrect length (typically not a multiple of 16 bits).

words 16-17 — The number of incoming packets from this subnet rejected for other reasons (e.g. too short to contain a header, garbage byte-count, forwarded too many times.)

If the identification is a number between 0 and 377 octal, this is an obsolete format of block. The identification is a subnet number and the counts are as above except that they are only 16 bits instead of 32, and consequently may overflow. This format should no longer be sent by any hosts.

Identification numbers of 1000 octal and up are reserved for future use.

23.9.2 Telnet and Supdup

The Telnet and Supdup protocols of the Arpanet exist in identical form in Chaosnet. These protocols allow access to a computer system as an interactive terminal from another network node.

The contact names are TELNET and SUPDUP. The direct borrowing of the Telnet and Supdup protocols was eased by their use of 8-bit byte streams and of only a single connection. Note that these protocols define their own character sets, which differ from each other and from the Chaosnet standard character set.

For the Telnet protocol, refer to the Arpanet Protocol Handbook. For the Supdup protocol, see AI memo 644.

Chaosnet contains no counterpart of the INR/INS attention-getting feature of the Arpanet. The Telnet protocol sends a packet with opcode 201 in place of the INS signal. This is a controlled packet and hence does not provide the "out of band" feature of the Arpanet INS, however it is satisfactory for the Telnet "interrupt process" and "discard output" operations on the kinds of hosts attached to Chaosnet.

23.9.3 File Access

The FILE protocol is primarily used by Lisp machines to access files on network file servers. ITS and TOPS-20 are equipped to act as file servers. A user end for the file protocol also exists for TOPS-20 and is used for general-purpose file transfer. For complete documentation on the file protocol, see SYS: DOC; FILE TEXT. The Arpanet file transfer protocols have not been implemented on the Chaosnet (except through the Arpanet gateway described below).

23.9.4 Mail

The MAIL protocol is used to transmit inter-user messages through the Chaosnet. The Arpanet mail protocol was not used because of its complexity and poor state of documentation. This simple protocol is by no means the last word in mail protocols; however, it is adequate for the mail systems we presently possess.

The sender of mail connects to contact name MAIL and establishes a stream connection. It then sends the names of all the recipients to which the mail is to be sent at (or via) the server host. The names are sent one to a line and terminated by a blank line (two carriage returns in a row). The Lisp Machine character set is used. A reply (see below) is immediately returned for each recipient. A recipient is typically just the name of a user, but it can be a user-atsign-host sequence or anything else acceptable to the mail system on the server machine. After sending the

recipients, the sender sends the text of the message, terminated by an EOF. After the mail has been successfully swallowed, a reply is sent. After the sender of mail has read the reply, both sides close the connection.

In the MAIL protocol, a reply is a signal from the server to the user (or sender) indicating success or failure. The first character of a reply is a plus sign for success, a minus sign for permanent failure (e.g. no such user exists), or a percent sign for temporary failure (e.g. unable to receive message because disk is full). The rest of a reply is a human-readable character string explaining the situation, followed by a carriage return.

The message text transmitted through the mail protocol normally contains a header formatted in the Arpanet standard fashion. Refer to the Arpanet Protocols Handbook.

23.9.5 Send

The SEND protocol is used to transmit an interactive message (requiring immediate attention) between users. The sender connects to contact name SEND at the machine to which the recipient is logged in. The remainder of the RFC packet contains the name of the person being sent to. A stream connection is opened and the message is transmitted, followed by an EOF. Both sides close after following the end-of-data protocol described in page 486. The fact that the RFC was responded to affirmatively indicates that the recipient is in fact present and accepting messages. The message text should begin with a suitable header, naming the user that sent the message. The standard for such headers, not currently adhered to by all hosts, is one line formatted as in the following example:

```
Moon@MIT-MC 6/15/81 02:20:17
```

Automatic reply to the sender can be implemented by searching for the first "@" and using the SEND protocol to the host following the "@" with the argument preceding it.

23.9.6 Name

The Name/Finger protocol of the Arpanet exists in identical form on the Chaosnet. Both Lisp machines and timesharing machines support this protocol and provide a display of the user(s) currently logged in to them.

The contact name is NAME, which can be followed by a space and a string of arguments like the "command line" of the Arpanet protocol. A stream connection is established and the "finger" display is output in Lisp Machine character set, followed by an EOF.

Lisp Machines also support the FINGER protocol, a simple-transaction version of the NAME protocol. An RFC with contact name FINGER is transmitted and the response is an ANS containing the following items of information separated by carriage returns: the logged-in user ID, the location of the terminal, the idle time in minutes or hours-colon-minutes, the user's full name, and the user's group affiliation.

23.9.7 Time

The Time protocol allows a host such as a Lisp machine that has no long-term timebase to ask the time of day. An RFC to contact name TIME evokes an ANS containing the number of seconds since midnight Greenwich Mean Time, Jan 1, 1900 as a 32-bit number in four 8-bit bytes, least-significant byte first.

23.9.8 Arpanet Gateway

This protocol allows a Chaosnet host to access almost any service on the Arpanet. The gateway server runs on each ITS host that is connected to both networks. It creates an Arpanet connection and a Chaosnet connection and forwards data bytes from one to the other. It also provides for a one-way auxiliary connection, used for the data connection of the Arpanet File Transfer Protocol.

The RFC packet contains a contact name of ARPA, a space, the name of the Arpanet host to be connected to, optionally followed by a space and the contact-socket number in octal, which defaults to 1 if omitted. The Arpanet Initial Connection Protocol is used to establish a bi-directional 8-bit connection.

If a data packet with opcode 201 (octal) is received, an Arpanet INS signal will be transmitted. Any data bytes in this packet are transmitted normally.

If a data packet with opcode 210 (octal) is received, an auxiliary connection on each network is opened. The first eight data bytes are the Chaosnet contact name for the auxiliary connection; the user should send an RFC with this name to the server. The next four data bytes are the Arpanet socket number to be connected to, in the wrong order, most-significant byte first. The byte-size of the auxiliary connection is 8 bits.

The normal closing of an Arpanet connection corresponds to an EOF packet. Closing due to an error, such as Host Dead, corresponds to a CLS packet.

23.9.9 Host Table

The HOSTAB protocol may be used to access tables of host addresses on other networks, such as the Arpanet. Servers for this protocol currently exist for Tenex and TOPS-20.

The user connects to contact name HOSTAB, undertakes a number of transactions, then closes the connection. Each transaction is initiated by the user transmitting a host name followed by a carriage return. The server responds with information about that host, terminated with an EOF, and is then ready for another transaction. The server's response consists of a number of attributes of the host. Each attribute consists of an identifying name, a space character, the value of the attribute, and a carriage return. Values may be strings (free of carriage returns and *not* surrounded by double-quotes) or octal numbers. Attribute names and most values are in upper case. There can be more than one attribute with the same name; for example, a host may have more than one name or more than one network address.

The standard attribute names defined now are as follows. Note that more are likely to be added in the future.

ERROR The value is an error message. The only error one might expect to get is "no such host".

NAME The value is a name of the host. There may be more than one NAME attribute; the first one is always the official name, and any additional names are nicknames.

MACHINE-TYPE
The value is the type of machine, such as LISPM, PDP10, etc.

SYSTEM-TYPE The value is the type of software running on the machine, such as LISPM, ITS, etc.

ARPA The value is an address of the host on the Arpanet, in the form *host/imp*. The two numbers are decimal.

CHAOS The value is an address of the host on Chaosnet, as an octal number.

DIAL The value is an address of the host on Dialnet, as a telephone number.

LCS The value is an address of the host on the LCSnet, as two octal numbers separated by a slash.

SU The value is an address of the host on the SUnet, in the form *net#host*. The two numbers are octal.

23.9.10 Dover

A press file may be sent to the Dover printer at MIT by connecting to contact name DOVER at host AI-CHAOS-11. This host provides a protocol translation service that translates from Chaosnet stream protocol to the EFTP protocol spoken by the Dover printer. Only one file at a time can be sent to the Dover, so an attempt to use this service may be refused by a CLS packet containing the string "BUSY". Once the connection has been established, the press file is transmitted as a sequence of 8-bit bytes in data packets (opcode 200). It is necessary to provide packets rapidly enough to keep the Dover's program (Spruce) from timing out; a packet every five seconds suffices. Of course, packets are normally transmitted much more rapidly.

Once the file has been transmitted, an EOF packet must be sent. The transmitter must wait for that EOF to be acknowledged, then send a second one, and then close the connection. The two EOF's are necessary to provide the proper connection-closing sequence for the EFTP protocol. Once the press file has been transmitted to the Dover in this way and stored on the Dover's local disk, it will be processed, prepared for printing, and printed.

If an error message is returned by the Dover while the press file is being transmitted, it will be reported back through the Chaosnet as a LOS containing the text of the error message. Such errors are fairly common; the sender of the press file should be prepared to retry the operation a few times.

Most programs that send press files to the Dover first wait for the Dover to be idle, using the Foreign Protocol mechanism of Chaosnet to check the status of the Dover. This is optional, but is courteous to other users since it prevents printing from being held up while additional files are

sent to the Dover and queued on its local disk.

It would be possible to send to a press file to the Dover using its EFTP protocol through the Foreign Protocol mechanism, rather than using the AI-CHAOS-11 gateway service. This is not usually done because EFTP, which requires a handshake for every packet, tends to be very slow on a timesharing system.

23.9.11 Remote Disk

The Remote Disk server exists on Lisp machines to allow other machines to refer to or modify the contents of the Lisp machine's disk. Primarily this is used for printing and editing the disk label.

After first establishing a connection to contact name REMOTE-DISK, the user process sends commands as packets which contain a line of text, ending with a Return character. The text consists of a command name, a space, and arguments interpreted according to the command. The server processing the command may send disk data to the user, or it may read successive packets and write them to the disk. It is up to the user to know how many packets of disk data to read or send after each command. The commands are:

READ *unit block n-blocks*
: Read *n-blocks* of data from disk *unit* starting at *block* and transmit their contents to the user process.

WRITE *unit block n-blocks*
: Read data from the net connection and store it into *n-blocks* disk blocks on disk *unit* starting at *block*.

SAY *text*
: *text*, which is simply all the rest of the line following SAY, is printed on the screen of the server host as a notification.

Each disk block is transmitted as three packets, the first two containing the data for 121 (decimal) Lisp machine words, and the third containing the data for the remaining 14 (decimal) words of the disk block. Each packet's data ends with a checksum made by adding together all the 8-bit bytes of the actual disk data stored in the packet.

23.9.12 The Eval Server

The Eval server is available on Lisp machines with contact name EVAL. It provides a read-eval-print loop which reads and prints using the chaosnet connection. The data consists of text in the ASCII character set.

Each time a complete s-expression arrives, the Eval server reads it, evaluates it and prints the list of values back onto the network connection, followed by a CRLF. There is no way for the user process to tell the end of the output for a particular s-expression; the usual application is simply to copy all the output to a user's terminal asynchronously.

The Eval server is disabled when the Lisp machine is logged in, unless the user requests to enable it.

chaos:eval-server-on *mode*
Turn the Eval server on this Lisp machine on or off. *mode* can be **t** (on), **nil** (off), or :notify (on, but notify the user when a connection is made).

23.10 Using Higher Level Protocols

qsend *user* &optional *text*
Sends a message to another user. **qsend** is different from **mail** because it sends the message immediately; it will appear within seconds on the other user's screen, rather than being saved in her mail file.

user should be a string of the form "*username@hostname*"; *host* is the name of the Lisp Machine or timesharing system the user is currently logged-in to. Multiple recipients separated by commas are also allowed. *text* is a string which is the message. If *text* is not specified, you will be prompted to type in a message.

Unlike **mail** and **bug**, **qsend** does not put up a window to allow you to compose the message; it just reads it from the input stream. Use Converse if you wish to compose sends in the editor. Converse can be invoked by typing System-C.

reply &optional *text*
Sends *text* as a message to the last user who sent a message to you. This is like **qsend** with an appropriate first argument provided.

chaos:shout &optional *message*
Sends *message* to every Lisp machine at your site. If you do not specify *message*, it is read from **standard-input**.

print-sends
Reprints any messages that have been received. This is useful if you want to see a message again.

supdup &optional *host*
host may be a string or symbol, which will be taken as a host name, or a number, which will be taken as a host number. If no *host* is given, the machine you are logged-in to is assumed. This function opens a connection to the host over the Chaosnet using the Supdup protocol, and allows the Lisp Machine to be used as a terminal for any ITS or TOPS-20 system.

To give commands to **supdup**, type the **Network** key followed by one character. Type **Network** followed by **Help** for documentation.

telnet &optional *host simulate-imlac*

telnet is similar to supdup but uses the Arpanet-standard Telnet protocol, simulating a printing terminal rather than a display terminal.

hostat &rest *hosts*

Asks each of the *hosts* for its status using the STATUS protocol, and prints the results. If no hosts are specified, all hosts on the Chaosnet are asked. Hosts can be specified either by name or by number.

For each host, a line is output that either says that the host is not responding or gives metering information for the host's network attachments. If a host is not responding, that usually means that it is down or there is no such host at that address. A Lisp Machine can fail to respond if it is looping inside without-interrupts or paging extremely heavily, such that it is simply unable to respond within a reasonable amount of time.

chaos:finger &optional *spec* (*stream* standard-output)
chaos:whois &optional *spec* (*stream* standard-output)

Print brief (finger) or verbose (whois) information about a user or users specified by *spec*, on *stream*. *spec* can be a user name, @ followed by a host name, or a user name, @, and a host name. If there is no host name, the default login host is used. If there is no user name, all users on the host are described.

chaos:finger-all-lms &optional *stream print-free return-free hosts*

Print a line of information about the user of each Lisp machine in *hosts* (the default is all Lisp machines at this site) on *stream* (default is standard-output).

If *print-free* is non-nil, information on free Lisp machines and nonresponding Lisp machines is also printed.

If *return-free* is non-nil, then this function returns two values, the first a list of host objects of free Lisp machines, the second a list of host objects of nonresponding Lisp machines.

chaos:user-logged-into-host-p *username host*

Returns t if there is a user named *username* logged in on *host* (a host name or host object).

chaos:find-hosts-or-lispms-logged-in-as-user *user hosts*

Return a list of host objects for hosts on which *user* is logged in. All Lisp machines at this site are checked, and so are *hosts* (which are presumably non-Lisp machines).

tv:close-all-servers *reason*

Close the connections of all network servers on this Lisp machine, giving *reason* (a string) as the reason in the CLS packet.

Note that PEEK has a mode that displays information on the active network servers.