MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo No. 628

June, 1981

Chaosnet

David A. Moon

Abstract

Chaosnet is a *local network*, that is, a system for communication among a group of computers located within about 1000 meters of each other. Originally developed by the Artificial Intelligence Laboratory as the internal communications medium of the Lisp Machine system, it has since come to be used to link a variety of machines around MIT and elsewhere.

This memo describes both the hardware and the software protocols. It is intended to be the definitive documentation for Chaosnet.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under office of naval research contract N00014-80-C-0505.

© MASSACHUSETTS INSTITUTE OF TECHNOLOGY 1981

Table of Contents

1. Introduction 1

2. Hardware Protocol 3
2.1 The Ether 3
2.2 Packets. 3
2.3 The Transceiver 4
2.4 The Interface. 4
2.5 Hardware Protocols. 5
2.6 Ether Contention. 6

3. Software Protocol—World View. 9
3.1 Connections 9
3.2 Contact Names. 9
3.3 Addresses and Indices 10
3.4 Packet Numbers 11
3.5 Packets. 12
3.6 Data Formats. 13
3.7 Routing 13
3.8 Flow and Error Control. 16

4. Software Protocol—Details 20
4.1 Connection Establishment 20
4.2 Status 23
4.3 Data 24
4.4 End-of-Data 24
4.5 Low-level 25
4.6 Connection States 25

5. Higher-Level Protocols 27
5.1 Status 27
5.2 Pulsar 28
5.3 Telnet and Supdup 28
5.4 File Access. 29
5.5 Mail 29
5.6 Send 29
5.7 Name 30
5.8 Time. 30
5.9 Arpanet Gateway. 30
5.10 Dover. 31

6. Using Foreign Protocols in Chaosnet 32

7. Hardware Programming Documentation 35

8. The ITS Implementation 37
8.1 System Calls 37
8.1.1 Opening I/O Channels 37
8.1.2 Input and Output 37
8.1.3 Interrupts 38

8.1.4 Miscellaneous Operations . . . 39
8.2 Utility Programs . . . 40
8.3 Server Programs . . . 42
8.4 Subroutine Packages . . . 42

9. The TOPS-20/TENEX Implementation . . . 43
9.1 Opening Connections . . . 43
9.2 Stream Input and Output . . . 44
9.3 Packet Input and Output . . . 44
9.4 Special Operations . . . 44
9.5 Utility Programs . . . 45
9.6 Server Programs . . . 46

10. The Lisp Machine Implementation . . . 47
10.1 Opening and Closing Connections . . . 47
10.1.1 User-Side . . . 47
10.1.2 Server-Side . . . 48
10.2 Connection States . . . 48
10.3 Stream Input and Output . . . 49
10.4 Packet Input and Output . . . 49
10.5 Connection Interrupts . . . 51
10.6 Information and Control . . . 51

11. The VAX/VMS Implementation . . . 53
11.1 Opening and Closing . . . 53
11.2 Stream Input and Output . . . 54
11.3 Packet Input and Output . . . 54
11.4 Checking the State . . . 55

12. The Unix Implementation . . . 57

References . . . 59

1. Introduction

Chaosnet is a *local network*, that is, a system for communication among a group of computers located within one or two kilometers of each other. The name Chaosnet refers to the lack of any centralized control element in this network.

Chaosnet was originally developed in 1975 by the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology as the internal communications medium of the Lisp Machine system [CHINUAL, AIM444]. It has since come to be used to link a variety of machines around the Institute. Chaosnets also exist at several other universities and research laboratories.

The Lisp Machine system is a multi-processor in which each active user is assigned a "personal" computer consisting of a medium-scale processor, a suitable amount of memory, and a swapping disk. Files are stored in a central file-system accessed through Chaosnet. This shared file-system retains the traditional advantages of a time-sharing system, namely inter-user communication, shared programs, and centralized backup and maintenance. At the same time, by giving each active user his own processor, the Lisp Machine system is much more capable than a time-sharing system at executing Lisp programs several million words in size efficiently and with rapid interactive response. Because Chaosnet is taking the place of the file disk in a conventional system, it must be fast (both in response and in throughput), it must be reliable (this is the reason why there is no centralized control), and it must allow connection of several dozen machines. However, it does not need to operate over long distances. Chaosnet is used to access other shared resources in addition to the file system; these include printers, tape drives, and one-of-a-kind specialized processors and I/O devices.

The system goals for Chaosnet were primarily simplicity and high performance. The performance is achieved by starting with a very high speed transmission medium and operating it in a simple, low-overhead fashion, rather than by using unusually clever algorithms. Of course one has to be careful to avoid algorithms which are so simple that they don't work or waste so much of the transmission medium that performance is impacted. Simplicity was important not only to improve performance, but because Chaosnet connects a diverse set of machines, and hence had to have several implementations all of which require maintenance in proportion to their complexity.

Simplicity of design also aids greatly in maintenance and management of the network itself, which has proven to be a non-trivial task in a network involving a variety of machines and used by a variety of groups, even within the single institution of MIT. It is important to be able to isolate an apparent failure of the whole network to the cable or to a particular host's hardware or software as rapidly as possible.

The design of Chaosnet was greatly simplified by ignoring problems irrelevant to local networks. Chaosnet contains no special provisions for things such as low-speed links, noisy (very high error rate) links, multiple paths, and long-distance links with significant transit time. This means that Chaosnet is not particularly suitable for use across the continent or in satellite applications. Chaosnet also makes no attempt to provide sophisticated features such as multiple levels of access or secure communication (other than by end-to-end encryption).

Chaosnet was largely inspired by the pioneering work on local networks at Xerox PARC [ETHERNET].

Chaosnet consists of two parts—the hardware and the software—which, while logically separable, were designed for each other. The hardware provides a carrier-sense multiple-access structure very similar to the PARC Ethernet. Network nodes contend for access to a cable, or ether, over which they may transmit packets addressed to other network nodes. The software defines higher-level protocols in terms of packets. These protocols can be (and are) used with media other than the Chaosnet cable, and with multiple interconnected cables. The software contains ideas borrowed from Ethernet [ETHERNET], TCP [TCP], and Arpanet, with some original ideas and modifications.

2. Hardware Protocol

2.1 The Ether

The transmission medium of Chaosnet is called the *ether*. Physically it is a coaxial cable, of the semi-rigid 1/2 inch low-loss type used for cable TV, with 75-ohm termination at both ends. At each network node a *cable transceiver* is attached to the cable. A 10-meter flat cable connects the transceiver to an *interface* which is attached to a computer's I/O bus. A network node consists of this transceiver and interface and a computer which executes a certain piece of software called the Network Control Program (NCP), which manages and controls Chaosnet, in addition to whatever application software justifies its existence in the first place.

One network node at a time may seize the ether and transmit a packet, which arrives at all other nodes; each node decides in hardware whether to ignore the packet or to receive it. A single ether must be a linear cable; it may not contain branches nor stubs, and the ends may not be joined into a circle. The maximum length of an ether cable is about 1 kilometer. This is determined by dispersion and by DC attenuation. The maximum number of nodes on a single ether is probably a few dozen. This is determined by degradation of the electrical properties of the cable by the connectors used to attach the transceivers.

The maximum length of an ether could be increased by using repeaters (bidirectional digital amplifiers joining two pieces of cable). In practice this is not done. Instead, the protocol provides for multiple ethers, joined together by nodes called *bridges* which relay packets from one ether to another. A bridge is typically a pdp11 computer with two or more network interfaces attached to it. A bridge node usually performs other tasks as well, such as interfacing terminals. Bridges attach other network media as well as ethers; some computers connect to the network through their manufacturer's high speed computer-to-computer interface to a nearby bridge, rather than being interfaced directly to an ether. Asynchronous lines have also been used as Chaosnet media.

The form of information on the ether, the transceiver and interface hardware, and the protocols which control who may seize the ether are described in the following sections.

2.2 Packets

The basic unit of transmission is called a packet. This is a sequence of up to 4032 data bits, plus 48 bits of *header* information used by the hardware. Packets' bits are normally grouped into 16-bit words. The division of a transmitted bit stream into packets provides a conveniently-sized unit for resource allocation and error control. The job of the hardware is to deliver a packet from one node to another. Software protocols define the meaning of the data bits in a packet, manage the hardware, compensate for imperfections of the hardware, and provide more useful services than simple transmission of packets from one computer to another.

The hardware header consists of three 16 bit words called *destination*, *source*, and *check*. The source identifies the node which transmitted the packet onto the ether. The destination identifies the node which is to receive the packet. [illegible] The check word contains a checksum of the packet's contents, used to detect transmission errors. [illegible]

packet to another ether, whence it will eventually reach its final destination. The check word is a cyclic redundancy checksum, generated and checked by hardware, which detects errors in transmission through the ether, entirely-spurious packets created by noise on the cable, and memory errors in the transmitting and receiving packet buffers.

The software protocol is also based on packets, taking 128 of the data bits as a software header. This is described in section 3.5, page 12.

2.3 The Transceiver

Everyone who connects to the ether does so through a transceiver, which is a small box which mounts directly on the cable via a UHF connector and a T-joint. All nodes use identical transceivers (the interface varies depending on what computer it is designed to interface to). The transceiver contains the analog portion of the interface logic, provides ground isolation between the ether cable and the computer, and contains some protective circuitry designed to prevent a malfunctioning program or interface from continuously jamming the ether. (If it were to be redesigned, it ought to contain even more protective circuitry, since there are some possible interface failures that can get past the protection and render the ether unusable.)

The transceiver receives a differential digital signal from the computer interface and impresses it onto the cable as a level of about 8 volts for a 1, or 0 volts (open circuit) for a 0, through a very fast VMOS power FET. When the cable is idle it is held at 0 volts by the terminations. This simple-minded unipolar scheme is adequate for the medium cable lengths and transmission speeds we are using. The transceiver monitors the cable by comparing it against a reference voltage, and returns a differential signal to the interface. In addition, it detects interference (another transceiver transmitting at the same time as this one) and informs the interface.

The transceiver includes indicators (light-emitting diodes) for power OK, transmitted data, received data, and interface attempting to jam the ether. A test button simulates an input of continuous 1's from the interface, which should light all the lights (dimly) if the transceiver is working. These indicators and the test button are useful for rapidly tracking down network problems. The transceiver requires its own power supply mounted nearby; one power supply can service three transceivers if they are all adjacent. High-voltage isolation between the cable and the computer is provided by optical isolators within the transceiver.

2.4 The Interface

The interface is typically a wire-wrap board containing about 120 TTL logic chips, which plugs into the I/O bus of a computer and connects it to the ether (through a transceiver.) The interface implements the hardware protocols described in the next section, buffers incoming and outgoing packets, generates and checks checksums, and interrupts the host computer when a packet is to be read out of the receive packet buffer or stored into the transmit packet buffer. These packet buffers shield the host computer from the high speed of data transmission on the cable. Instead of having to produce bits at a high rate, the host can produce them at a lower rate, collect them into a packet, and then tell the interface to transmit the packet in a single burst of high-speed transmission.

Interfaces currently exist for Lisp machines, for DEC LSI-11 micro-computers, and for the DEC Unibus [UNIBUS], which allows Chaosnet to be attached to pdp11's, VAX's, and the peripheral processors of most pdp10's. Programming documentation for this compatible family of interfaces appears later in this paper.

Interfaces for other computers are likely to exist in the future. The existing interface design does not make any unusual special demands of its host computer and should be easily adaptable to other architectures.

2.5 Hardware Protocols

The purpose of these protocols is to deliver packets intact from one node to another node on the same ether, with fairly high probability of success, and to guarantee to give an error indication or lose the packet entirely if it is not delivered intact. An additional purpose is to provide high performance and not to bog down when subjected to a heavy load.

Bits are represented on the ether using a technique which is called Upright Biphase NRZI. Each bit cell, which is approximately 250 nanoseconds long, begins with a transition in state, from high to low or from low to high. This transition marks the beginning of a bit cell and provides self-clocking. 3/4 of the way through the bit cell, the state of the cable is sampled; high represents a 1 and low represents a 0. If the bit being represented is the same as the previous bit, there will be one transition at the beginning of the bit cell and a second in the middle of the bit cell. If the bit being represented is the opposite of the previous bit, there will be no transition in the middle of the bit cell since the clock transition will have set the cable to the desired state. The AC frequency of the signal on the cable varies between 1/2 the bit rate and the full bit rate. The information bit-rate is 4 million bits per second.

The self-clocking feature allows for slight variations in transmission and cable propagation speed. However, since the 3/4 of a bit cell delay is a fixed delay, only modest variations in speed can be tolerated. A crystal clock is used as the source of the transmit timing in the interface.

Since there is always at least one state-transition per bit cell, the states where the ether remains high or low for an appreciable time are available for non-data uses. If the ether remains low for more than about two bit cells, it is considered to be not-busy. This condition marks the end of a packet and allows someone else to transmit. Note that if no transceivers are active, the terminations will hold the ether low.

If the ether remains high for about two bit cells, this is an "abort signal". Abort signals are used for two purposes. If the transceiver detects a collision (two nodes trying to transmit at the same time), each transmitting interface ceases to transmit and sends an abort signal (four bit cells long), which tells all receivers to ignore the aborted packet and ensures that the other transmitter also aborts. Thus when a collision occurs, the ether is cleared as soon as possible to help prevent long tie-ups under conditions of heavy load. The other use for the abort signal is in hardware flow-control. When a receiving interface determines that an incoming packet is addressed to it, but its receive buffer already contains a packet, it sends an abort signal which causes the transmitter to stop. This serves the dual purpose of immediately informing the transmitter that its message did not get through, and preventing the ether from being tied up while a long packet is transmitted which the receiver cannot receive.

Packets are transmitted over the ether in reverse bit-order, for hardware convenience. The three header words, which to the software appear to be at the end of the packet, are transmitted first, in the order check, source, destination. The data words, in reverse order, follow. Words are transmitted least-significant bit first. Of course, the software need not be aware of this reversal; packets arrive at the destination in the same form as they were created by the source. At the end of the packet, an extra zero bit is appended to bring the ether to the low state so that an extra spurious clock-transition will not be generated when it goes idle. This bit is stripped off by the interface and is never seen by software.

The check word is used for error detection, as described above. The source word is made available to the software, which ignores it in most cases, and also serves to synchronize the clocks in the collision-avoidance mechanism which is described below. The destination word is compared by each receiver against its own address. If they match, or if the destination is zero, or if the software selects the "match any destination" mode, the packet is placed in the receive packet buffer and the host computer is interrupted. The zero destination feature is used for "broadcast" messages. Note that a receiver whose packet buffer is full will only generate an abort signal if the packet was specifically addressed to it.

2.6 Ether Contention

Chaosnet has no centralized control element; when a network node has a message to transmit, its interface seizes the ether and transmits a packet. The time when it seizes the ether is determined only by state inside that particular interface and by the local state of the cable at the point where that interface's transceiver is attached.

If two interfaces should decide to seize the ether and transmit at the same time, their transmissions will interfere and no useful information will be transmitted. This is called a *collision*. Collisions are the principal limitation on the bandwidth of a heavily-loaded ether-type network, and should be avoided. (However, neither PARC's network nor MIT's network has yet been operated with a heavy enough load to make collisions really significant.)

Chaosnet uses a novel collision avoidance technique. First of all, an interface will never initiate transmission unless the ether is seen to be not busy, i.e. it has been in the low state for some time. This ensures that collisions can only occur near the beginning of a packet. Once transmission of a packet has gotten well started, the ether is effectively "seized" (all interfaces realize that it is busy) and transmission will continue successfully through to the end of the packet. The amount of ether transmission time wasted by a collided packet is therefore limited to the round-trip cable propagation delay. This technique is called *carrier sense*.

Secondly, the hardware uses a time-division technique to attempt to prevent two interfaces from initiating transmission at the same time. This technique should prevent essentially all collisions while imposing only a modest delay in the initiation of transmission. It is designed so that it works better as the load on the ether increases; the wasted time between packets and the relative rate of collisions both decrease.

The basic idea is that each interface is assigned a time slot, or *turn*, according to its address. It may only initiate transmission during its turn. The turns are spaced far enough apart that if one interface initiates transmission, every other interface will perceive that the ether is busy by the time its own turn arrives, and will not initiate an interfering transmission. Each interface contains a time-slot counter which counts while the ether is not busy, keeping track of whose turn it is.

Each packet synchronizes the counters in all of the interfaces by setting them from the source address of that packet; at the time the packet was transmitted, it must have been the turn of the interface that transmitted it.

Another way to think of this is to make an analogy with ring networks. One can imagine a *virtual token* which passes down the cable until it gets to the end, then jumps to the beginning of the cable and repeats. An interface may only initiate transmission at the instant the token passes by it. When an interface transmits, the token stops moving and remains at that interface until the end of the packet, whereupon it continues down the cable, passing every other interface, giving them each a chance to transmit before letting the first interface transmit a second packet.

The token is not represented by any physical transmission on the cable. That would constitute a form of centralized control, and would lead to reliability problems if the token was lost or duplicated. Instead, every interface contains a time-slot counter which keeps track of where the token is thought to be. Every time a packet is transmitted these counters are brought up-to-date. The token cannot be lost because a counter by its nature eventually returns to all previous states. It does not matter if the token is duplicated (i.e. the counters lose synchronization) occasionally, since this will only cause collisions, which we know how to detect and deal with, and since the first successful transmission will resynchronize all counters. The basic mechanism of the ether network with contention and collisions is known to work, and the collision-avoidance scheme is an added-on optimization which improves performance without changing the basic mechanism.

There is a finite propagation delay time between interfaces, and this time is not small compared with the bit-rate of Chaosnet, nor when compared with the desirable length of a time slot. This time consists of the delay in the cable, about 5 nanoseconds per meter, and the delay through the two transceivers, about 220 nanoseconds. This propagation delay means that the time-slot counters in two different interfaces cannot be exactly synchronized, and that when interface A initiates transmission interface B will not instantaneously see that the ether is busy. Special relativity tells us that in fact the concept "exactly synchronized" is meaningless. Since the two time-slot counters are not in the same place, the only way we can compare them is to send a message from one to the other, through the ether, containing the reading of the counter. But this message takes non-zero time to get there, so the counter-reading it contains is wrong by the time it is compared against the other counter! We in fact do send messages containing counter readings; the source address in a packet equals the reading of the time-slot counter in the interface that sent it—at the time it was sent. Since the network nodes are not in relative motion, we can measure the distance between them and use that information to improve their synchronization.

What we are trying to do is to prevent collisions. This means that if interface A starts transmitting a packet in its turn, then by the time interface B thinks that its own turn has arrived, it must perceive the ether as busy. We will assign addresses (and hence time slots) and set the length of a time slot in such a way that this will happen. Suppose the maximum delay through the ether between A and B is t. This would be the delay for one of them sending a packet to the other; the delay between A's receipt of a third party's packet and B's receipt of that packet is less, especially if the third party is between A and B on the cable. Then the maximum perceived difference between a clock at A and a clock at B is $2t$; if a message is sent from B to A synchronizing the clocks, and then a message is sent from A to B containing A's clock reading, at B this clock reading will be slow by $2t$.

When a packet transmitted by A arrives at B, B's clock may read as much as $2t$ earlier or later than A's turn, depending on the transmission direction of the last synchronizing message. In order to guarantee that B's turn has not yet happened, the time between any of A's turns and any of B's turns must always be at least $2t$, twice the maximum propagation delay through the ether between A and B. This is the important idea! We cause this to be true by assigning addresses starting at one end of the cable; each node's address is the previous node's address plus twice the propagation delay between them, divided by the length of a turn. It is easy to see that if this is done for all adjacent pairs, the condition will automatically be true for non-adjacent pairs as well. When we get to the end of the cable, we must assign a number of empty slots equal to twice the propagation delay of the full length of cable, to provide the necessary separation between the turns of the two nodes at the ends of the cable.

The virtual token travels through the network at a substantially slower speed than a real signal such as a packet; in the fastest case, when nodes are very far apart, it travels at just half the speed of a real signal. Since a Chaosnet ether has the geometry of a line, as compared to the ring net's circle, the virtual token is also slowed down by the need to return from the end of the cable to the beginning. This slower speed of the token is the price one pays for the increased robustness of Chaosnet as compared with a ring network. In a real system, we slow the token down even more to provide a margin of safety. The speed of the network is not significantly affected by the slow token, since the interval between packet transmissions by a single node is much longer than the round-trip time of the token. Indeed, if the network is being used primarily for file transfer, and hence the packets are large, the transmission time alone for a typical packet is several times the round-trip time of the token. A typical value for the token's round-trip time is 64 microseconds.

In spite of all this, sometimes collisions will occur anyway. If the cable has been idle for a long time, the various clocks will have lost synchronization. If a source address is corrupted by a transmission error, any interface that sees that source address will set its clock to an incorrect value. Sometimes a packet will collide with random noise rather than another legitimate packet. In addition, the transmitter does not distinguish receiver-busy aborts from real collisions.

When a collision does occur, we recover from it (in software) by retransmitting the packet again a couple of times, hoping that we will be lucky enough not to have another collision, or that the receiver will soon clear its packet buffer. This retransmission is done by the software, not the hardware, since the hardware destroys the packet in its packet buffer in the process of transmitting it. But if collisions continue to occur, we give up and let somebody else have the ether. The packet is lost. A higher level of protocol will soon realize that it has been lost and retransmit it. We assume that there is enough randomness in the higher-level software that the two nodes which originally collided will not collide again on the retransmission by deciding to retransmit at precisely the same instant.

3. Software Protocol--World View

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. The speed for file transfers in real-life circumstances was to be comparable with an inexpensive magnetic tape drive (30000 characters per second, or about 10 times the speed of the Arpanet). We actually get about double this in some favorable cases. To achieve this speed it was important to design out bottlenecks such as are found in the Arpanet, for instance the control-link which is shared between multiple connections and the need to acknowledge each message before the next message can be sent. The protocol must be simple, for the sake of reliability and to allow its use by modest computer systems. A full Chaosnet Network Control Program is just about half the size of an Arpanet NCP on the same machine, and the protocol allows low-performance implementations to omit some features. A minimal implementation exists for a single-chip microcomputer.

3.1 Connections

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 either deal in terms of packets, or ignore packet boundaries and treat the connection as two uni-directional streams of 8-bit or 16-bit bytes.

On top of the connection facility "user" programs build other facilities, such as file access, interactive terminal connections, and data in other byte sizes, such as 36 bits. The meaning of the packets or bytes transmitted through a connection is defined by the particular higher-level protocol in use.

In addition to reliable communication, the protocol provides flow control, includes a way by which prospective communicants may get in touch with each other (called *contacting* or *rendezvous*), and provides various network maintenance and housekeeping facilities. These are discussed later.

3.2 Contact Names

When first establishing a connection, it is necessary for the two communicating processes to contact each other. In addition, in the usual user/server situation, the server process does not exist beforehand and needs to be created and made to execute the appropriate program.

We chose to implement contacting in an asymmetric way. (Once the connection has been established everything is completely symmetric.) One process is designated the *user*, and the other is designated the *server*. The server has some *contact name* to which it *listens*. The user process requests its local operating system to connect it to the server, specifying the network node and contact name of the server. The local operating system sends a message (a *Request for Connection*) to the remote operating system, which examines the contact name and creates a connection to a listening process, creates a new server process and connects to it, or rejects the request.

Automatically discovering to which host to connect in order to obtain a particular service is a subject for higher-level protocols and for further research. It is not dealt with by Chaosnet.

Once a connection has been established, there is no more need for the contact name and it is discarded. Indeed, often the contact name is simply the name of a service (such as "TELNET") and several users should be able to have simultaneous connections to separate instances of that service, so contact names must be reusable.

In the case where two existing processes that already know about each other want to establish a connection, we arbitrarily designate one as the listener (server) and the other as the requester (user). The listener somehow generates a "unique" contact name, somehow communicates it to the requester, and listens for it. The requester requests to connect to that contact name and the connection is established. In the most common case of establishing a second connection between two processes which are already connected, the index number (see below) of the first connection can serve as a unique contact name.

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.

See page 21 for complete details of how to establish a connection.

3.3 Addresses and Indices

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, `SYSBIN;HOSTS2`, in the case of ITS) which relates symbolic host names to numeric host addresses.

An address consists of two fields. The most-significant 8 bits identify a *subnet*, and the least-significant 8 bits identify a host within that subnet. Both fields must be non-zero. A subnet corresponds to a single transmission path. Some subnets are physical Chaosnet cables (*ethers*), while others are other media, for instance an interface between a pdp10 and a pdp11. The significance of subnets will become clear when routing is discussed (see section 3.7, page 13).

When a host is connected to an ether, the host's hardware address on that ether is the same as its software address, including the subnet field.

A connection is specified by the names of its two ends. Such a name consists of a 16-bit host address and a 16-bit connection index, which is assigned by that host, as the name of the entity inside the host which owns the connection. The only requirements placed by the protocol on indices are that they be non-zero and that they be unique within a particular host; that is, a host may not assign the same index number to two different connections unless enough time has elapsed between the closing of the first connection and the opening of the second connection that confusion between the two is unlikely.

Typically the least-significant n bits of an index are used as a subscript into the operating system's tables, and the most-significant $16-n$ bits are incremented each time a table slot is reused, to provide uniqueness. The number of uniquizing bits must be sufficiently large, compared to the rate at which connection-table slots are reused, that if two connections have the same index, a

packet from the old connection cannot sit around in the network (e.g. in buffers inside hosts or bridges) long enough to be seen as belonging to the new connection.

It is important to note that packets are *not* sent between hosts (physical computers). They are sent between user processes; more exactly, between channels attached to user processes. Each channel has a 32-bit identification, which is divided into subnet, host, index, and uniquization fields. From the point of a view of a user process using the network, the Network Control Program section of his host's operating system is part of the network, and the multiplexing and demultiplexing it performs is no different from the routing performed by other parts of the network. It makes no difference whether two communicating processes run in the same host or in different hosts.

Certain control packets, however, are sent between hosts rather than users. This is visible to users when opening a connection; a contact name is only valid with respect to a particular host. This is a compromise in the design of Chaosnet, which was made so that an operational system could be built without first solving the research and engineering problems associated with making a diverse set of hosts into a uniform, one-level name space.

3.4 Packet Numbers

There are two kinds of packets, controlled and uncontrolled. Controlled packets are subject to error-control and flow-control protocols, described below (see section 3.8, page 16), which guarantee that each controlled packet is delivered to its destination exactly once, that the controlled packets belonging to a single connection are delivered in the same order they were sent, and that a slow receiver is not overwhelmed with packets from a fast sender. Uncontrolled packets are simply transmitted; they will usually but not always arrive at their destination exactly once. The protocol for using them must take this into account.

Each controlled packet is identified by an unsigned 16-bit *packet number*. Successive packets are identified by sequential numbers, with wrap-around from all 1's to all 0's. When a connection is first opened, each end numbers its first controlled packet (RFC or OPN) however it likes, and that sets the numbering for all following packets.

Packet numbers should be compared modulo 65536 (2 to the 16th), to ensure correct handling of wrap-around cases. On a pdp11, use the instructions

```
CMP A,B
BMI A_is_less
```

Do not use the BLT or BLO instruction. On a pdp10, use the instructions

```
SUB A,B
TRNE A,100000
 JRST A_is_less
```

Do not use the CAMGE instruction. On a Lisp machine, use the code

```
(IF (BIT-TEST #o100000 (- A B))
    <A is less>)
```

Do not use the LESSP (or <) function.

3.5 Packets

A packet consists of a header, which is 8 16-bit words, and zero or more 8-bit or 16-bit bytes of accompanying data. In addition there are three words put on by the hardware, described earlier in this paper.

The following are the 8 header words:

Operation The most-significant 8 bits of this word are the *Opcode* of the packet, a number which tells what the packet means. The 128 opcodes with high-order bit 0 are for the use of the network itself. The 128 opcodes with high-order bit 1 are for use by users. The various opcodes are described in chapter 4, page 20.

The least-significant 8 bits of this word are reserved for future use, and must be zero.

Count The most-significant 4 bits of this word are the forwarding count, which tells how many times this packet has been forwarded by bridges. Its use is explained in the Routing section.

The least-significant 12 bits of this word are the data byte count, which tells the number of 8-bit bytes of data in the packet. The minimum value is 0 and the maximum value is 488. Note that the count is in 8-bit bytes even if the data are regarded as 16-bit bytes.

Destination Address
This word contains the network address of the destination host to which this packet should be sent.

Destination Index This word contains the connection index at the destination host of the connection to which this packet belongs, or 0 if this packet does not belong to any connection.

Source Address This word contains the network address of the source host which originated this packet.

Source Index This word contains the connection index at the source host of the connection to which this packet belongs, or 0 if this packet does not belong to any connection.

Packet Number If this is a controlled packet, this word contains its identifying number.

Acknowledgement The use of this word is described in section 3.8, page 16.

3.6 Data Formats

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 [CHINUAL]. This is basically ASCII, augmented with additional printing characters and a different set of format-effector (or "control") characters.

Because the rules for bit numbering conflict with the native byte-ordering in pdp10s, and because it is quite expensive to rearrange the bytes using the pdp10 instruction set, pdp11s which act as front-ends for pdp10s must reformat packets passing through them, and pdp10s interfaced directly to the network must have interfaces capable of rearranging the bytes. This requires that the network protocols explicitly specify which portions of each type of packet are 8-bit bytes and which are 16-bit bytes. In general the header is 16-bit bytes and the data field is 8-bit bytes, but certain packet types (OPN, STS, RUT, and opcodes 300 through 377) have 16-bit bytes in the data field. Use of 32-bit data is rare, so no provision is made for putting 32-bit data into the standard format for pdp10s. On our current network pdp10s are the only hosts which require this packet reformatting assistance, because most modern computers number their bits and bytes from least-significant to most-significant.

The effect of this is that user programs that use the Chaosnet always see the data in a packet and its header in the native form of the machine they are running on, and the necessary conversions are automatically applied by the network. This statement applies to the order of bits and bytes within a word, but not to the character set (when packets contain textual data) which is dictated by protocols.

Unlike some other network protocols, Chaosnet does not use any software checksumming. Because of the diversity of hosts with different architectures attached to the Chaosnet, it is impossible to devise a checksumming algorithm which can be executed compatibly and efficiently on all hosts. Instead, Chaosnet relies on error-checking hardware in the network interfaces, and assumes that other sources of packet damage that checksums could detect, such as software bugs in a Network Control Program, either do not occur or will produce symptoms so obvious that they will be detected and fixed immediately.

3.7 Routing

Routing consists of deciding how to deliver a packet to the network node specified by the destination address field of the packet. Having reached that node, the packet can trivially be delivered to the destination user process via the destination index. In general routing may be a multi-step process involving transmission through several subnets, since there may not be a direct hardware connection between the source and the destination. Note that the routing decision is made separately for each packet, with no reference to the concept of connections.

Any host that is connected to more than one subnet acts as a *bridge* and *forwards* packets from one subnet to another when necessary. There could also be hardware bridges which are not hosts, although we have not yet designed any such device. Since routing does not depend on connections, a bridge is a very simple device (or program) which does not need much state. This makes the bridge function inexpensive to piggyback onto a computer which is also performing other functions, and makes reliable bridge software easy to implement.

The difference between a *bridge* and a *gateway*, in our terminology, is that a bridge forwards packets from one sub-Chaosnet to another, without modifying the packets or understanding them other than to look at the destination address and increment the forwarding count, and does not deal with connections nor with flow control, while a gateway interconnects two networks with differing protocols and must understand and translate the information passing through it. Gateways may also have to deal with flow and error control because they connect networks with slow or differing speeds. Bridges are suitable for local networks while gateways are suitable for long-distance networks and for connecting networks not produced by the same organization.

To prevent routing loops, each packet contains a forwarding-count field. Each bridge that forwards the packet increments this count; if the count reaches its maximum value the packet is discarded. The error-control protocol will recover discarded packets, or decide that no viable connection can be established between the two hosts.

The implementation of routing in an operating system is as follows, given a packet to be routed, which may have come in from the network or may have been originated by the local host. First, check the packet's destination address. If it is this host, receive the packet. Otherwise, increment the forwarding count and discard the packet if it has been forwarded too many times. If the destination is some other host on a subnet to which this host is directly connected, transmit the packet on that subnet; the destination host should receive it. If the destination is a host on a subnet of which this host has no knowledge, look up the subnet in the host's *routing table* to find the best bridge to that subnet, and transmit the packet to that bridge.

Each host has a routing table, indexed by subnet number, which tells how to get packets to hosts on that subnet. Each entry contains: (exact details may vary depending on implementation)

Type The type of connection between the host and this subnet. This can be one of *Direct*, *Bridge*, or *Fixed Bridge*. *Direct* means a physical connection such as a Chaosnet interface. *Bridge* means an indirect connection, via a packet-forwarding bridge. Which bridge is best to use is to be discovered by this routing mechanism. *Fixed Bridge* is the same except that the automatic mechanism is not to change which bridge is used. This is useful to set up explicit routing for purposes such as network debugging.

Address Identifies the connection to this subnet in a way which depends on the type. For a direct connection, this identifies the piece of hardware which implements the connection. (It might be a Unibus address.) For a bridge or a fixed bridge, this is the network address of the bridge.

Cost A measure of the cost of sending a packet through this route. Costs are used to select the best route from among alternatives in a way described below. For a direct connection, the cost is 10 for a direct interface between two computers (e.g. between a pdp10 and its front end pdp11), 11 for a Chaosnet ether cable, 20 for a slow medium such as an asynchronous line, etc. For a bridge or a fixed bridge, the cost is specified by the bridge in a RUT packet (described below).

The routing table is initialized with the number of a more or less arbitrary existent host and a high cost, for each subnet to which the host is not directly connected. Until the correct bridge is discovered (which normally happens within a minute of coming up), packets for that subnet will be bounced off of that arbitrary host, which probably knows the right bridge to forward them to.

The cost for subnets accessed via bridges is increased by 1 every 4 seconds, thus typically doubling after a minute. When the cost reaches a "high" value, it sticks there, preventing problems with arithmetic overflow. The purpose of the increasing cost is to discount the value of old information. The cost for subnets accessed via direct connections and fixed bridges does not increase.

Every 15 seconds, a bridge advertises its presence by broadcasting a routing (RUT) packet on each subnet to which it is directly connected. Each host on that subnet receives the RUT packet and uses it to update its routing table. If the host's routing table says to access a certain subnet via bridges, and the RUT packet says that this is the best bridge to that subnet, the routing table is updated to say that this bridge should be used.

Note that it is important that the rate at which the costs increase with time be slow enough that it takes more than twice the broadcast interval to increase the cost of one hop to be more than the cost of two hops. Otherwise the routing algorithm is not well-behaved. Suppose subnet A has two bridges (α and β) on it, and bridge α is connected to subnet B but bridge β is not (it goes to some other irrelevant subnet). Then if the costs increase too fast and bridges α and β do not broadcast their RUT packets exactly simultaneously, sometimes packets for subnet B may be sent to bridge β because its cost appears lower. Bridge β will then send them to bridge α, where they should have gone directly. In more complicated situations packets can go around in a circle some of the time.

The source address of a RUT packet must be the hardware address of the bridge on the particular subnet on which the packet is broadcast. The destination address of a RUT packet must be zero; RUT packets are not forwarded onto other subnets. The byte count of a RUT packet is a multiple of 4 and the packet contains up to 122 pairs of 16-bit words:

word 1 — The subnet number of a subnet which this bridge can get to, directly or indirectly, right-adjusted.

word 2 — The cost of sending to that subnet via this bridge. This is the current cost from the bridge's routing table, plus the cost for the subnet on which the routing packet is being broadcast. Adding the subnet cost eliminates loops, and prefers one-hop paths over two-hop paths.

When a host receives a RUT packet, it processes each 2-word entry by comparing the cost for that subnet against its current cost; if it is less or equal the cost and the address of the bridge are entered into the routing table, provided that that subnet's routing table entry is not of the Direct or Fixed Bridge type.

When there are multiple equivalent bridges, the traffic is spread among them only by virtue of their RUT packets being sent at different times, so that sometimes one bridge has the lower cost, and sometimes the other. If this isn't adequate, hosts could have fancier routing tables which remember more than one possible route and use them according to their relative costs, but so far this has not been necessary since the network traffic is not so high as to saturate any one bridge.

The design of this routing scheme is predicated on the assumption that the network geometry is simple, there are few multiple paths, and the length of any path is quite short. This makes more sophisticated schemes unnecessary.

An important feature of this routing scheme is that the size of the table is proportional to the number of subnets, not to the number of hosts. Thus it does not take up an inordinate amount of memory in a small computer, and no complicated dynamic allocation schemes are required.

In the case of a pdp10 which accesses the Chaosnet through a front-end pdp11, we define the interface between the two computers to be a subnet, and regard the pdp11 as a bridge which forwards packets between the network and the pdp10. This gives the pdp10 and the pdp11 separate addresses so that we can choose to talk to either one, even though they are part of the same computer system. This is occasionally useful for maintenance purposes. It becomes more useful when the front-end pdp11 has peripherals which are to be accessed through the Chaosnet, since they can simply look like hosts on that pdp11's private subnet.

In the case of a host which is attached to more than one subnet, it is undesirable for the host to have more than one address, since this would complicate user programs which use addresses. Instead, one of the host's network attachments is designated as primary, and that address is used as the host's single address. The other attachments are regarded as bridges which can forward to that host. Sometimes, we simplify the routing by inventing a new subnet which contains only that host and has no physical realization. The host's address is an address on that fake subnet. All of the host's network attachments are regarded as bridges which know how to forward packets to that subnet.

The ITS host table allows a host to have multiple addresses on multiple networks, but when you ask for *the* address of a certain host on a certain network you only get back the primary address. All packets coming from that host have that as their source address.

3.8 Flow and Error Control

The Network Control Programs (NCPs) conspire to ensure that data packets are sent from user to user with no garbling, duplications, omissions, or changes of order. Secondarily, the NCPs attempt to achieve a maximum rate of flow of data, and a minimum of overhead and retransmission.

The fundamental basis of flow-control and error-control in Chaosnet is *retransmission*. Packets which are damaged in transmission, which won't fit in buffers, which are duplicated or out-of-sequence, or which otherwise are embarrassing are simply discarded. Packets are periodically retransmitted until an indication that they have been successfully received is returned. This retransmission is end-to-end; any intermediate bridges do not participate in flow-control and error-control, and hence are free to discard any packets they wish.

There are actually two kinds of packets, *controlled* and *uncontrolled*. Controlled packets are retransmitted and delivered reliably; most packets, including all packets used by the user (except for UNC packets), are of this type. Uncontrolled packets are not retransmitted; these are used for certain lower-level functions of the protocol such as the implementation of flow and error control. The usage of these packets is designed so that they need not be delivered reliably.

Retransmission of a packet continues until stopped by a signal from the receiver to the sender called a *receipt*. A receipt contains a *packet number*, and indicates that all controlled packets with a packet number less than or equal (modulo 65536) to that number have been successfully received, and therefore need not be retransmitted any more. A receipt does not indicate that these packets have been processed by the destination user process; it simply indicates that they have successfully arrived in the destination host, and are guaranteed to be there when the user process asks for them.

There is another signal from the receiver to the sender, called an *acknowledgement*. An acknowledgement also contains a packet number, and indicates that all controlled packets with a packet number less than or equal (modulo 65536) to that number have been read by the destination user process. This is used to implement flow-control. Note that acknowledgement of a packet implies receipt of that packet. In fact, if the receiving process does not fall behind, explicit receipts need not be sent, because the receiving host will not have to buffer any packets, but will acknowledge them as soon as they arrive.

The purpose of flow-control is to match the speeds of the sending and receiving processes. The extremes to be avoided are, on the one hand, too small a "buffer size" causing the data transmission rate to be slower than it could be, and on the other hand, large numbers of packets piling up in the network because the sender is sending faster than the receiver is receiving. It is also necessary to be aware that receipts and acknowledgements must be transmitted through the network, and hence have an associated cost.

Chaosnet flow-control operates by controlling the number of packets "in the network". These are packets which have been emitted by the sending user process, but have not been acknowledged. We define a *window* into the set of packet numbers. The beginning of this window is the first packet number which has not been acknowledged, and the width of the window is a fixed number established when the connection is opened. The sending process is only allowed to emit packets whose packet numbers lie within the window. Once it has emitted all of the packets in the window, the window is said to be full. Thus, the size of the window is the "buffer size" for the connection, and is the maximum number of packets that may need to be buffered inside an NCP (sending or receiving). Acknowledgements move the window, making it not full, and allowing the sending process to emit additional packets.

We do not receipt and acknowledge every single controlled packet that is transmitted through a connection, since that would double or triple the number of packets sent through the network to move a given amount of data. Instead we batch the receipts and acknowledgements. But if acknowledgements are not sent sufficiently often, the data will not flow smoothly, because the window will often appear full to the sender when it is not. If receipts are not sent sufficiently often, there will be unnecessary retransmissions.

Whenever a packet is sent through a connection, an acknowledgement for the reverse direction of that connection is "piggy-backed" onto it, using the Acknowledgement field in the packet header. For interactive applications, where there is much traffic in both directions, this provides all the necessary acknowledgement and receipting with no need to send any extra packets through the network.

When this does not suffice, STS (status) packets are generated to carry receipts and acknowledgements. STS packets are uncontrolled, since they are part of the mechanism that implements controlled packets. If an STS packet is duplicated, it does no harm. If an STS packet is lost, mechanisms exist which will cause a replacement to be generated later. An STS

packet carries separate receipt and acknowledgement packet numbers.

When a user process reads a packet from the network, if the number of packets which should have been acknowledged but have not been is more than 1/3 the window size, an STS is generated to acknowledge them. Thus the preferred batch size for acknowledgement is 1/3 the window size. The advantage of this size is that if one STS is lost, another will be generated before the window fills up (at the 2/3 point).

When a packet is received with the same packet number as one which has already been successfully received, this is evidence of unnecessary retransmission, and an STS is generated to carry a receipt back to the sender. If this STS is lost, the next retransmission will stimulate another one. Thus receipts are normally implied by acknowledgements, and only sent separately when there is evidence of unnecessary retransmission.

Retransmission consists of sending all unreceipted controlled packets, except those that were last sent very recently (within 1/30'th of a second in ITS.) Retransmission occurs every 1/2 second. This interval is somewhat arbitrary, but should be close to the response time of the systems involved. Retransmission also occurs in response to an STS packet, so that a receiver may cause a faster retranmission rate than twice a second if it so desires. This should never cause useless retransmission, since STS carries a receipt, and very-recently-transmitted packets, which might still be in transit through the network, are not retransmitted.

Another operation is *probing*, which consists of sending a SNS packet, in the hope of eliciting either an STS or a LOS, depending on whether the other side believes the connection exists. Probing is used periodically as a way of testing that the connection is still open, and also serves as a way to get STS packets retransmitted as a hedge against the loss of an acknowledgement, which could otherwise stymie the connection. SNS packets are uncontrolled.

We probe every five seconds on connections which have unacknowledged packets outstanding (a non-empty window) and on connections which have not received any packets (neither data nor control) for one minute. If a connection receives no packets for 1 1/2 minutes, this means that at least 5 probes have been ignored, and the connection is declared to be broken; either the remote host is down or there is no viable path through the network between the two hosts.

The receiver can generate "spontaneous" STS's, to stimulate retransmission and keep things moving on fast devices with insufficient buffering for 1/2 second worth of packets. This provides a way for the receiver to speed up the retransmission timeout in the sender, and to make sure that acknowledges are happening often enough.

Note that the network still functions if either or both parties to a connection ignore the window. The window is simply an improver of efficiency. Receipts have the same property. This allows very small implementations to be compatible with the same protocol, which is useful for applications such as bootstrapping through the network.

It would be possible to have dynamic adjustment of the window size in response to observed behavior. The STS packet includes the window size so that changes to it can be communicated. However, this has not been found necessary in practice. Each higher-level protocol has a standard pre-determined window size, which it establishes when it first opens a connection, and this seems to be close enough to optimum that careful dynamic adjustment of it wouldn't make a big difference.

This scheme for flow-control and error-control is based on several assumptions. It is assumed that the underlying transmission media have their own checking, so that they discard all damaged packets, making packet checksums unnecessary at the protocol level. The transit time through the network is assumed to be fast, so that a fairly-small retransmission interval is practical, and negative acknowledgements are not necessary. The error rate is assumed to be low so that overall efficiency is not affected by the simple-minded error recovery scheme of simply retransmitting all outstanding packets. It is assumed that no reformatting of packets occurs inside the network, so that flow-control and error-control can operate on a packet basis rather than a byte basis.

4. Software Protocol--Details

In the following sections, each of the packet *Opcodes* and the use of that packet type in the protocol is described. Opcodes are given as an octal number, a three-letter code, and a name.

Unless otherwise specified, the use of the fields in the packet header is as follows. The source and destination address and index denote the two ends of the connection; when an end does not exist, as during initial connection establishment, that index is zero. The opcode, byte count, and forwarding count fields have no variations. The packet number field contains sequential numbers in controlled packets; in uncontrolled packets it contains the same number as the next controlled packet will contain. The acknowledgement field contains the packet number of the last packet seen by the user.

4.1 Connection Establishment

The following packet types are associated with creating and destroying connections. First the packets are described and then the details of the various connection-establishment protocols are given.

1 RFC Request for connection

All connections are initiated by the transmission of an RFC from the user to the server. The data field of the packet contains the contact name. The contact name can be followed by arbitrary arguments to the server, delimited by a space character. The destination index field of an RFC contains 0 since the destination index is not known yet.

RFC is a controlled packet;. it is retransmitted until some sort of response is received. Because RFC's are not sent over normal, error-controlled connections, a special way of detecting and discarding duplicates is required. When an NCP receives an RFC packet, it checks all pending RFC's and all connections which are in the Open or RFC-received state (see section 4.6, page 25), to see if the source address and index match; if so, the RFC is a duplicate and is discarded.

12 LSN Listen

A server process informs the local NCP of the contact name to which it is listening by sending a LSN packet, with the contact name in the data field. This packet is never transmitted anywhere through the network. It simply serves as a convenient buffer to hold the server's contact name. When an RFC and a LSN containing the same contact name meet, the LSN is discarded and the RFC is given to the server, putting its connection into the RFC-received state (see section 4.6, page 25). The server reads the RFC and decides whether or not to open the connection.

2 OPN Open connection

OPN is the usual positive response to RFC. The source index field conveys the server's index number to the user; the user's index number was conveyed in the RFC. The data field of OPN is the same as that of STS (see below); it serves mainly to convey the server's window size to the

user. The Acknowledgement field of the OPN acknowledges the RFC so that it will no longer be retransmitted.

OPN is a controlled packet; it is retransmitted until it is acknowledged. Duplicate OPN packets are detected in a special way; if an OPN is received for a connection which is not in the RFC-sent state (see section 4.6, page 25), it is simply discarded and an STS is sent. This can happen if the connection is opened while a retransmitted OPN packet is in transit through the network, or if the STS which acknowledges an OPN is lost in the network.

3 CLS Close connection

CLS is the negative response to RFC. It indicates that no server was listening to the contact name, and one couldn't be created, or for some reason the server didn't feel like accepting this request for a connection, or the destination NCP was unable to complete the connection (e.g. connection table full.)

CLS is also used to close a connection after it has been open for a while. Any data packets in transit may be lost. Protocols which require a reliable end-of-data indication should use the mechanism for that (see section 4.4, page 24) before sending CLS.

The data field of a CLS contains a character-string explanation of the reason for closing, intended to be returned to a user as an error message.

CLS is an uncontrolled packet, so that the program which sends it may go away immediately afterwards, leaving nothing to retransmit the CLS. Since there is no error recovery or retransmission mechanism for CLS, the use of CLS is necessarily optional; a process could simply stop responding to its connection. However, it is desirable to send a CLS when possible to provide an error message for the user.

4 FWD Forward a request for connection

This is a response to RFC which indicates that the desired service is not available from the process contacted, but may be available at a possibly-different contact name at a possibly-different host. The data field contains the new contact name and the Acknowledgement field—exceptionally—contains the new host number. The issuer of the RFC should issue another RFC to that address. FWD is an uncontrolled packet; if it is lost in the network, the retransmission of the RFC will presumably stimulate an identical FWD.

5 ANS Answer to a simple transaction

This is another kind of response to RFC. The data field contains the entirety of the response, and no connection is established. ANS is an uncontrolled packet; if it is lost in the network, the retransmission of the RFC will presumably stimulate an identical ANS.

There are several connection-initiation protocols implemented with these packet types.

When an RFC arrives at a host, the NCP finds a user process that is listening for this RFC's contact name, or creates a server process to provide the desired service, or responds to the RFC itself if it knows how to provide the requested service, or refuses the request for connection. The process that serves the RFC chooses which connection initiation protocol to follow. This process is

given the RFC as data, so that it can look at the contact name and any arguments that may be present.

A *stream connection* is initiated by an RFC, transmitted from user to server. The server returns an OPN to the user, which responds with an STS. These three packets convey the source and destination addresses, indices, initial packet numbers, and window sizes between the two NCP's. In addition a character-string argument can be conveyed from the user to the server in the RFC.

The OPN serves to acknowledge the RFC and extinguish its retransmission. It also carries the server's index, initial packet number, and window size. The STS serves to acknowledge the OPN and extinguish its retransmission. It also carries the user's window size; the user's index and initial packet number were carried by the RFC. Retransmission of the RFC and the OPN provides reliability in the face of lost packets. If the RFC is lost, it will be retransmitted. If the STS is lost, the OPN will be retransmitted. If the OPN is lost, the RFC will be retransmitted superfluously and the OPN will be retransmitted since no STS will be sent.

The exchange of an OPN and an STS tells each side of the connection that the other side believes the connection is open; once this has happened data may begin to flow through the connection. The user process may begin transmitting data when it sees the OPN. The server process may begin transmitting data when it sees the STS. These rules ensure that data packets cannot arrive at a receiver before it knows and agrees that the connection is open. If data packets did arrive before then, the receiver would reject them with a LOS (see below), believing them to be a violation of protocol, and this would destroy the connection before it was ever fully established.

Once data packets begin to flow, they are subject to the flow and error control protocol described in section 3.8, page 16. Thus a stream connection provides the desired reliable, bidirectional data stream.

A *refusal* is initiated by an RFC in the same way, but the server returns CLS rather than OPN. The data field of the CLS contains the reason for refusal to connect.

A *forwarded connection* is initiated by an RFC in the same way, but the server returns a FWD, telling the user another place to look for the desired service.

A *simple transaction* is initiated by an RFC from user to server, and completed by an ANS from server to user. Since a full connection is not established and the reliable-transmission mechanism of connections is not used, the user process cannot be sure how many copies of the RFC the server saw, and the server process cannot be sure that its answer got back to the user. This means that simple transactions should not be used for applications where it is important to know whether the transaction was really completed, nor for applications in which repeating the same query might produce a different answer. Simple transactions are a simple efficient mechanism for applications such as extracting a small piece of information (e.g. the time of day) from a central data-base.

4.2 Status

7 STS Status

STS is an uncontrolled packet which is used to convey status information between NCPs. The Acknowledgement field in the packet header contains an acknowledgement, that is, the packet number of the last packet given to the receiving user process. The first 16-bit byte in the data field contains a receipt, that is, a packet number such that all controlled packets up to and including that one have been successfully received by the NCP. The second 16-bit byte in the data field contains the window size for packets sent in the opposite direction (to the end of the connection which sent the STS). The byte count is presently always 4. This will change if the protocol is revised to add additional items to the STS packet.

6 SNS Sense status

SNS is an uncontrolled packet whose sole purpose is to cause the other end of the connection to send back an STS. This is used by the *probing* mechanism described above (see page 18).

11 LOS Lossage

LOS is an uncontrolled packet which is used by one NCP to inform another of an error. The data field contains a character-string explanation of the problem. The source and destination addresses and indices are simply the destination and source addresses and indices, respectively, of the erroneous packet, and do not necessarily correspond to a connection. When an NCP receives a LOS whose destination corresponds to an existent connection and whose source corresponds to the supposed other end of that connection, it *breaks* the connection and makes the data field of the LOS available to the user as an error message. Other LOS's, that don't correspond to connections, are simply ignored.

LOS is sent in response to situations such as: arrival of a data packet or an STS for a connection that does not exist or is not open, arrival of a packet from the wrong source for its destination, arrival of a packet containing an undefined opcode or too large a byte count, etc.

LOS's are given to the user process so that it may read the error message.

No LOS is given in response to an OPN to a connection not in the RFC-Sent state, nor in response to a SNS to a connection not in the Open state, nor in response to a LOS to a non-existent or broken connection. These rules are important to make the protocols work without timing errors. An OPN or a SNS to a non-existent connection elicits a LOS.

4.3 Data

200-277 DAT 8-bit Data

Opcodes 200 through 277 (octal) are controlled packets with user data in 8-bit bytes in the data field. The NCP treats all 64 of these opcodes identically; some higher-level protocols use the opcodes for their own purposes. The standard default opcode is 200.

300-377 DAT 16-bit Data

Opcodes 300 through 377 (octal) are controlled packets with user data in 16-bit bytes in the data field. The NCP treats all 64 of these opcodes identically; some higher-level protocols use the opcodes for their own purposes. The standard default opcode for 16-bit data is 300.

15 UNC Uncontrolled Data

This is an uncontrolled packet with user data in 8-bit bytes in the data field. It exists so that user-level programs may bypass the flow-control mechanism of Chaosnet protocol. Note that the NCP is free to discard these packets at any time, since they are uncontrolled. Since UNC's are not subject to flow control, discarding may be necessary to avoid running out of buffers. A connection may not have more input packets queued awaiting the attention of the user program than the window size of the connection, except that you are always allowed to have one UNC packet queued. If no normal data packets are in use, up to one more UNC packet than the window size may be queued.

UNC packets are also used by the standard protocol for encapsulating packets of foreign protocols for transmission through Chaosnet (see chapter 6, page 32).

4.4 End-of-Data

14 EOF End of File

EOF is a controlled packet which serves as a "logical end of data" mark in the packet stream. When the user program is ignoring packets and treating a Chaosnet connection as a conventional byte-stream I/O device, the NCP uses the EOF packet to convey the notion of conventional end-of-file from one end of the connection to the other. When the user program is working at the packet level, it may transmit and receive EOF's.

EOF packets are used in the following protocol which is the recommended way to reliably determine that all data have been transferred before closing a connection (in applications where that is an important consideration).

The important issue is that neither side may send a CLS until both sides are sure that all the data have been transmitted. After sending all the data it is going to send, including an EOF packet to mark the end, the sending process waits for all packets to be acknowledged. This ensures that the receiver has seen all the data and knows that no more data are to come. The sending process then closes the connection. When the receiving process sees an EOF, it knows that there are no more data. It does *not* close the connection until it sees the sender close it, or

until a brief timeout elapses. The timeout is to provide for the case where the sender's CLS gets lost in the network (CLS cannot be retransmitted). The timeout is long enough (a few seconds) to make it unlikely that the sender will not have seen the acknowledgement of the EOF by the time the timeout is over.

To use this protocol in a bidirectional fashion, where both parties to the connection are sending data simultaneously, it is necessary to use an asymmetrical protocol. 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.

4.5 Low-level

`13` `MNT` Maintenance

MNT is a special packet type reserved for the use of network maintenance programs. Normal NCPs should simply discard any MNT packets they receive. MNT packets are an escape mechanism to allow special programs to send packets that are guaranteed not to get confused with normal packets. MNT packets are forwarded by bridges although usually one would not be depending on this.

`10` `RUT` Routing Information

RUT is a special packet type broadcast by bridges to inform other nodes of the bridge's ability to forward packets between subnets. The source address is the network address of the bridge on the subnet on which the RUT was broadcast. The destination address is zero. The byte count is a multiple of 4, and the data field contains a series of pairs of 16-bit bytes: a subnet number and the "cost" of getting to that subnet via this bridge. The packet number and acknowledgement fields are not used and should contain zero. See section 3.7, page 13 for the details.

4.6 Connection States

A user process gets to Chaosnet by means of a capability or channel (dependent on the host operating system) which corresponds to one end of a connection. Associated with this channel are a number of buffers containing controlled packets output by the user and not yet reccipted, and data packets received from the network but not yet read by the user; some of these incoming packets are in-order by packet number and hence may be read by the user, while others are out of order and cannot be read until packets earlier in the stream have been received. Certain control packets are also given to the user as if they were data packets. These are RFC, ANS, CLS, LOS, EOF, and UNC. EOF is the only type that can ever be out-of-order.

Also associated with the channel is a state, usually called the *connection state*. Full understanding of these states depends on the descriptions of packet-types above. The state can be one of:

Open The connection exists and data may be transferred.

Closed The channel does not have an associated connection. Either it never had one or it has received or transmitted a CLS packet, which destroyed the connection.

Listening The channel does not have an associated connection, but it has a contact name (usually contained in a LSN packet) for which it is listening.

RFC Received A *Listening* channel enters this state when an RFC arrives. It can become *Open* if the user process *accepts* the request.

RFC Sent The user has transmitted an RFC. The state will change to *Open* or *Closed* when the reply to the RFC comes back.

Lost The connection has been broken by receiving a LOS packet.

Incomplete Transmission
The connection has been broken because the other end has ceased to transmit and to respond to SNS. Either the network or the foreign host is down. (This can also happen if the local host goes down for a while and then is revived, if its clock runs in the meantime.)

Foreign The channel is talking some foreign protocol, whose packets are encapsulated in UNC packets. As far as Chaosnet is concerned there is no connection. See chapter 6, page 32 for the details.

5. Higher-Level Protocols

This chapter 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.

5.1 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, and the Hostat command of the CHATST program on ITS are user ends for 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 (pdp11/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 pdp10s 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 which 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 which 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.

5.2 Pulsar

For network maintenance purposes, certain network nodes support a simple transaction with contact name PULSAR, which controls a "pulsar" feature. This feature periodically transmits a short packet which can be used to test and adjust cable transceivers. The packet consists of the three header words, a zero word, and a word of alternating ones and zeros. It is addressed to host 177777 which is guaranteed not to exist.

The returned ANS contains a single character, which is a digit. A 0 means that the pulsar is turned off. Any other digit indicates the number of sixtieths of a second between pulses. Sending an RFC with a digit as an argument sets the state of the pulsar to that digit, and returns an ANS containing the new state.

5.3 Telnet and Supdup

The Telnet and Supdup protocols of the Arpanet [TELNET] [SUPDUP] 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 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.

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.

5.4 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 [FILE]. The Arpanet file transfer protocols have not been implemented on the Chaosnet (except through the Arpanet gateway described below).

5.5 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. [RFC733]

5.6 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 section 4.4, page 24. 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.

5.7 Name

The Name/Finger protocol of the Arpanet [FINGER] 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.

5.8 Time

The Time protocol of the Arpanet [TIME] exists on Chaosnet as a simple transaction. 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. Some computers—Lisp machines, for example—which don't have hardware calendar-clocks use this protocol to find out the date and time when they first come up.

5.9 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.

5.10 Dover

A press file may be sent to the Dover printer by connecting to contact name DOVER at host AI-CHAOS-11. This host provides a protocol translation service which 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, send a second one, 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 and prepared for printing, and then 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.

6. Using Foreign Protocols in Chaosnet

As seen above, foreign protocols which are based on the idea of a bidirectional (or unidirectional) stream of 8-bit bytes can simply be adopted wholesale into Chaosnet, using a Chaosnet stream connection instead of whatever stream protocol the protocol was originally designed for. This was done with the Arpanet Telnet protocol, for example.

When using such protocols between a Chaosnet process and a process on a foreign network, a protocol-translating gateway stands at the boundary between the two networks and has a connection on both networks. Bytes received from one connection are transmitted out the other. If the protocol uses any features besides a simple stream of bytes, for instance special out-of-band signals, these are translated appropriately by the gateway. The connection is initially set up by the user end connecting explicitly to the protocol-translating gateway and demanding of it a certain service from a certain host on the other network; the gateway then opens the appropriate pair of connections. For an example of this, refer to the Arpanet gateway (see section 5.9, page 30).

However, there are many packet-oriented protocols in the world and sometimes it is desirable to access these protocols at the packet level rather than the connection level, and to transport the packets of these protocols through Chaosnet links without using a Chaosnet connection. For example, there are gateways attached to Chaosnet which provide connections to other networks that use PUP and Internet as their packet protocols. User processes in Chaosnet hosts may talk to these other networks in those networks' own protocols by using the foreign-protocol protocol of Chaosnet.

A foreign packet is transmitted through Chaosnet by storing it in the data field of an UNC packet. The foreign packet is regarded as being composed of 8-bit bytes. The source and destination addresses of the UNC packet are used in the usual fashion to control the delivery of the packet within Chaosnet. The packet number and acknowledgement fields of the packet header are not used for their normal purposes, since this packet is not associated with a Chaosnet stream connection. By convention, the acknowledgement field of the packet contains a protocol number. The number 100000 octal means Internet and the number 100001 octal means PUP. Other numbers will be assigned as needed. The packet number field of the packet can be used for any purpose.

If a user process transmits an UNC packet through a Chaosnet channel which is in the *Closed* state (see section 4.6, page 25), the channel goes into the *Foreign* state and the NCP assumes that the user is not talking normal Chaosnet protocol, but is using Chaosnet to transport packets of some other protocol. The NCP fills in the source address and index in these packets, but believes whatever destination address and index are placed in the packet by the user. The packet number and acknowledgement fields of the UNC packets are not touched by the NCP. Any incoming UNC packets addressed to the user's index on this host will be given to the user, regardless of their source address/index; it is up to the user program to filter out any unwanted packets. The NCP should also provide a way for one user to receive any unclaimed incoming UNC packets, so that rendezvous subprotocols of foreign protocols may be simulated.

When a packet-translating gateway to a foreign network receives an UNC packet with the appropriate protocol number, it extracts the foreign packet from the data field and fires it into the foreign network. When it receives packets from the foreign network, it maps the destination

address of the packet into a Chaosnet address and index in some suitable fashion, encapsulates the packet in an UNC, and launches it into Chaosnet.

For PUP the address mapping is straightforward, since PUP and Chaosnet use similar addressing techniques [ETHERNET]. The host address spaces are the same. The Chaosnet index maps directly into the low-order 16 bits of the PUP port number, and the high-order 16 bits are zero. When a PUP is encapsulated in a Chaosnet packet, its PUP header duplicates the addresses in the Chaosnet header. When a PUP is received by a PUP/Chaosnet gateway, a Chaosnet header can easily be constructed from the PUP header. The AI-CHAOS-11 is attached to the MIT Chaosnet and the MIT Ethernet and provides a PUP/Chaosnet gateway. It advertises to each network its ability to route packets to host addresses in the other network, using that network's routing protocols. When it receives a packet from one network that is destined for the other, it does the appropriate encapsulation or de-encapsulation and sends the packet on its way. AI-CHAOS-11 also acts as a bridge between several Chaosnet subnets and provides a protocol-translating gateway for sending Press files to the Dover printer (a protocol-translating gateway is necessary for this application because the printer's native protocol, which could be used through the foreign-protocol protocol, cannot be implemented efficiently under a timesharing system).

In the case of Internet, only protocols built on the idea of ports can be straightforwardly supported without a table of connections in the gateway. The Internet address space includes the Chaosnet host address space as a subset but does not provide any address breakdown within a host unless ports are used. However, it appears that most protocols are built on a protocol that uses ports, such as the User Datagram Protocol [UDP] or the Transmission Control Protocol [TCP].

In the case of foreign protocols other than PUP, where the addressing structure is not identical to Chaosnet, a program must somehow know the Chaosnet address of a packet-translating gateway to the foreign network. By sending UNC packets to this gateway, a user program can initiate connections to processes on that other network without requiring his local NCP (nor any bridges involved in routing the packets) to know anything about the protocol he is using. If the inter-network gateway translates rendezvous protocols appropriately, connections may be initiated in the reverse direction also—from a user process on the foreign network to a server for the foreign protocol that resides on a Chaosnet host.

The foreign-protocol protocol may also be used between two user processes on Chaosnet, with no foreign network involved, if they simply wish to speak a different protocol from Chaosnet. They are on their own for a rendezvous mechanism, however, unless they use a Chaosnet simple transaction for rendezvous or otherwise have some way of conveying their addresses and index numbers to each other.

When foreign packets are too large to fit in the data field of a Chaosnet packet (more than 488 bytes), the user program and the packet-translating gateway must agree on a technique for dividing packets into fragments and reassembling them, unless the foreign protocol itself provides for this, as Internet does. The packet-number field in an UNC packet is available for use by such a technique, since UNC packets are not normally numbered. This is not a problem with PUP, since it provides a protocol by which parties to a connection and gateways may complain about overly-large packets and specify the maximum packet size to be used.

UNC packets not associated with a connection are useful for other things besides encapsulating foreign protocols. Any application which wants to use Chaosnet as simply a packet transmission medium, essentially the raw hardware, should use UNC packets so that its packets do not

interfere with standard packets and so that the standard routing mechanisms may be used. For example, the MIT Architecture Machine uses UNC packets to communicate with non-stream-oriented I/O devices such as graphic tablets. Here Chaosnet is being used as an I/O bus which may be attached to more than one computer. Numbers between 140000 and 177777 octal in the acknowledgement field of an UNC packet are reserved for such applications. Note that this number is not part of the protocol; it is simply a hint about what a packet is being used for. Normally no program that is not specifically supposed to deal with such packets would ever receive one.

7. Hardware Programming Documentation

This section describes the Unibus version of the Chaosnet interface, which attaches to pdp11s and Lisp Machines. The interface contains one buffer which holds a received packet and a second buffer which holds a packet to be transmitted. Packets are moved between these buffers and the computer under program control. Direct memory access (DMA) is not used; the small gain in performance was not thought to be worth the extra hardware complexity. The usual performance penalty of programmed I/O is not incurred since the packet buffers can transfer data at the full speed of the computer and neither busy waiting nor multiple interrupts are required.

To transmit a packet, successive 16-bit words of the packet are written into the outgoing packet buffer. The last word to be written is the cable address of the destination of the packet, or 0 to broadcast it. The hardware is then told to initiate transmission. It waits until the cable is not busy and this node's turn to transmit arrives, then shifts the packet out onto the cable. At the completion of transmission transmit-done is set and the computer is interrupted. If transmission is aborted by a collision, transmit-done and transmit-abort are set and the computer is interrupted. As the packet is written into the outgoing packet buffer, a 16-bit cyclic redundancy checksum is computed by the hardware. This checksum is transmitted with the packet and checked by the receiver.

To receive a packet, the clear-receiver bit is asserted by the program. The next packet on the cable which is addressed to this node, or is broadcast, will be stored into the incoming packet buffer. After the packet has been stored, the computer is interrupted. The packet buffer will then not be changed until the next clear-receiver operation is performed, giving the computer a chance to read out the packet. If a packet appears on the cable addressed to this node while the incoming packet buffer is busy, a collision is simulated so as to abort the transmission. As a packet is stored into the incoming packet buffer, the 16-bit cyclic redundancy checksum is checked, and it is checked again as the packet is read out of the packet buffer. This provides full checking for errors in the network and in the packet buffers.

The standard interrupt-vector address for the Chaosnet interface is 270. The standard interrupt priority level is 5. The standard Unibus address is 764140. These are the device registers:

764140 *Command/Status Register*

This register contains a number of bits, in the usual pdp11 style. All read/write bits are initialized to zero on power-up. Identified by their masks, these are:

1 Timer Interrupt Enable (read/write). Enables interrupts from the interval timer present in some versions of the interface (not described here).

2 Loop Back (read/write). If this bit is 1, the cable and transceiver are not used and the interface is looped back to itself. This is for maintenance.

4 Spy (read/write). If this bit is 1, the interface will receive all packets regardless of their destination. This is for maintenance and network monitoring.

10 Clear Receiver (write only). Writing a 1 into this bit clears Receive Done and enables the receiver to receive another packet.

20 Receive Interrupt Enable (read/write). If Receive Done and Receive Interrupt Enable are both 1, the computer is interrupted.

40 Transmit Interrupt Enable (read/write). If Transmit Done and Transmit Interrupt Enable are both 1, the computer is interrupted.

100 Transmit Abort (read only). This bit is 1 if the last transmission was aborted, by a collision or because the receiver was busy.

200 Transmit Done (read only). This bit is set to 1 when a transmission is completed or aborted, and cleared to 0 when a word is written into the outgoing packet buffer.

400 Clear Transmitter (write only). Writing a 1 into this bit stops the transmitter and sets Transmit Done. This is for maintenance.

17000 Lost Count (read only). These 4 bits contain a count of the number of packets which would have been received if the incoming packet buffer had not been busy. Setting Clear Receiver resets the lost count to 0.

20000 Reset (write only). Writing a 1 into this bit completely resets the interface, just as at power up and Unibus Initialize.

40000 CRC Error (read only). If this bit is 1 the receiver's cyclic redundancy checksum indicates an error. This bit is only valid at two times: when the incoming packet buffer contains a fresh packet, and when the packet has been completely read out of the packet buffer.

100000 Receive Done (read only). A 1 in this bit indicates that the incoming packet buffer contains a packet.

764142 *My Address* (read)
Reading this location returns the network address of this interface (which is contained in a set of DIP switches on the board).

764142 *Write Buffer* (write)
Writing this location writes a word into the outgoing packet buffer. The last word written is the destination address.

764144 *Read Buffer* (read only)
Reading this location reads a word from the incoming packet buffer. The last three words read are the destination address, the source address, and the checksum.

764146 *Bit Count* (read only)
This location contains the number of bits in the incoming packet buffer, minus one. After the whole packet has been read out, it will contain 7777 (a 12-bit minus-one).

764152 *Start Transmission* (read only)
Reading this location initiates transmission of the packet in the outgoing packet buffer. The value read is the network address of this interface. This method for starting transmission may seem strange, but it makes it easier for the hardware to get the source address into the packet.

8. The ITS Implementation

8.1 System Calls

Note that the NETWRK subroutine package, page 42, provides a convenient interface to most of these system calls for the assembly-language programmer.

8.1.1 Opening I/O Channels

Since ITS I/O Channels are unidirectional, a Chaosnet connection is represented by a pair of channels, one for input and one for output. Operations that are not inherently directional, such as finding out the state of the connection, may be done on either channel (it makes no difference).

Unlike every other device, you do not obtain these channels with the OPEN system call. Instead a special system call, CHAOSO, is provided. This does not open a connection; it simply gives you a pair of channels and a potential connection, in the *Closed* state, which can be opened by transmitting a packet (an RFC for instance) through the output channel.

CHAOSO takes three arguments: the input channel number, the output channel number, and the receive window size. If either channel is currently open it is closed first (just as with OPEN). CHAOSO returns no values. Error 6 (device full) is signalled if the system's connection table is full.

8.1.2 Input and Output

Input and output can be done on a Chaosnet connection in terms of either packets or 8-bit bytes. 8-bit byte I/O is usually done with the SIOT system call; the IOT system call or the .IOT uuo may also be used—the channel behaves as if it had been opened in unit mode. 8-bit byte output is collected by the system into packets containing the maximum allowed number of bytes; when a packet is full it is transmitted with the standard data opcode (200). Until a full packet's worth of bytes have been output nothing will be transmitted unless the FORCE system call is used. 8-bit input comes from packets with the data opcode (200). If an EOF packet is received, the standard end-of-file behavior will occur—IOT will return the EOF character (-1,,3) and SIOT will return with a non-zero residual byte count. If some other kind of packet is received, an IOC error will be signalled (see below). If PKTIOT (see below) is used to read out a non-data packet, stream input may be continued past it.

Bit 1.4 in the control bits is "don't hang" mode for both input and output. When SIOT is done with this bit specified, and no input is available or the output window is full, it will simply return without transferring the full number of bytes specified. The byte-pointer and byte-count arguments to SIOT are updated past the bytes that were transferred. When input IOT is done in "don't hang" mode, and no input is available, the EOF character (-1,,3) is returned. Output IOT should not be done in "don't hang" mode since it has no way to indicate that it did not transmit anything. If a non-data packet is received, "don't hang" mode will behave the same as if there was no input available.

Before doing 8-bit byte I/O, the user program must open a Chaosnet stream connection by transmitting and receiving the appropriate RFC, LSN, or OPN packets and following the protocol described on page 21. The NETWRK subroutine package can be useful for this.

Input and output can be done in terms of packets by using the PKTIOT system call. The first argument is the I/O channel number and the second is the address of the packet to be transmitted or of a 126.-word buffer to contain the packet to be received. No values are returned.

Input PKTIOT will return data packets and the following types of control packets: RFC, OPN, FWD, ANS, CLS, LOS, EOF, and UNC. The other types of control packets are for the NCP's internal use only. If no input packets are available, PKTIOT will wait. If the connection is in a bad state, an IOC error will be signalled. This is discussed in the next section.

Output PKTIOT will accept data packets and CLS, EOF, and UNC packets if the connection is open. Transmitting an UNC packet when the connection is closed puts it into the *Foreign Protocol* state. RFC, LSN, OPN, FWD, and ANS packets may be transmitted if the connection is in the appropriate state. Transmitting a bad packet type or transmitting when the connection is in the wrong state signals an IOC error (see the next section). Normally the user sets only the opcode and number of bytes fields in the packet header; PKTIOT fills in the rest. When sending an RFC, the user specifies the destination-host field and the system remembers it. When sending an UNC, the user specifies everything but the source host and index.

Note that PKTIOT does not synchronize with 8-bit byte I/O. If a partial packet's worth of bytes have been output, and then a packet is output with PKTIOT, that packet will get ahead of those bytes. The FORCE system call can be used to change this. If a partial packet's worth of bytes have been input, and then a PKTIOT is done, those bytes will remain available to the stream and PKTIOT will read the next packet after them.

8.1.3 Interrupts

I/O (second-word) interrupts occur if enabled when an I/O operation formerly would have hung but now will not. In other words, an interrupt occurs on the input channel when a packet arrives while there are no input packets available, if the packet is of a type that input PKTIOT would return. An interrupt occurs on the output channel if the window is full and an acknowledgement arrives which makes some space in the window. Completely interrupt-driven I/O can easily be programmed for the Chaosnet, using the WHYINT system call (see below).

An interrupt also occurs on the input channel when the connection state changes.

A %PIIOC interrupt (also called an "IOC error") occurs if any of a variety of illegal operations is performed. IOC error 3 (non-recoverable data error) is signalled if a packet is transmitted with PKTIOT and it has an illegal size or opcode in the header. IOC error 12 octal (channel in illegal mode) is signalled if byte or packet input or output is done with the connection in the wrong state, or if byte input is done and a packet other than a normal data packet or EOF is found.

8.1.4 Miscellaneous Operations

The CLOSE system call, or the .CLOSE uuo, may be used to close the I/O channels and the connection. When both channels are closed, all buffers and other information associated with the connection are immediately discarded. If the connection is open a CLS packet is transmitted. You can also transmit a CLS yourself, with an explanatory message in it, before doing the CLOSE.

The FORCE system call, or the .NETS uuo, can be used on an output channel. If there is buffer containing a partial packet's worth of 8-bit byte output, it is transmitted as a packet of less than the maximum size.

The FINISH system call first does a FORCE and then waits for all packets that have been transmitted to be acknowledged.

The RESET system call, and the .RESET uuo, are ignored, as on most devices. The RCHST and RFNAME system calls, and the .RCHST uuo, return the standard information for a non-file device, with device-name CHAOS:. No device-dependent information is returned.

The WHYINT system call, given either the input channel or the output channel of a Chaosnet connection, returns a lot of useful device-dependent information about the connection:

val 1 The device code, which is the value of the symbol %WYCHA.

val 2 The state of the connection. Symbols for the connection states start with the prefix %CS:

- %CSCLS Closed (or never opened).
- %CSLSN Listening for an incoming RFC.
- %CSRFC RFC received while listening; neither accepted nor rejected yet.
- %CSRFS RFC sent, awaiting acceptance, rejection, or answer.
- %CSOPN Open.
- %CSLOS Broken by receipt of a LOS packet.
- %CSINC Broken by "incomplete transmission" (no response from foreign host for a long time).
- %CSFRN Using a foreign protocol, encapsulated in UNC packets.

val 3 The left half is the number of available input packets. This does not count out-of-order packets. This number is increased by one if there is buffered input available for 8-bit byte input. This number is zero if the Chaosnet connection is directly connected to a STY (see STYNET).

The right half is the number of output packets which have not yet been acknowledged and hence are occupying space in the window.

val 4 The left half is the receive window size and the right half is the transmit window size.

val 5 The left half is the input channel number and the right half is the output channel number. These numbers are -1 if the channel has been CLOSEd or RELEASEd.

The NETBLK system call works similarly on the Chaosnet as on the Arpanet. The first argument is a channel number (input or output). The second argument is a connection state code. The third argument, which is optional, is a timeout in 30ths of a second. If it is not immediate it is modified. NETBLK hangs until the connection state is different from the specified state or the timeout (if specified) elapses. Two values are returned: the current state of the connection and the amount of time left.

The STYNET system call works similarly on the Chaosnet as on the Arpanet. It allows a Chaosnet connection and a STY (pseudo terminal) to be connected together so that a Telnet/Supdup server program can provide efficient service. The arguments are:

arg 1 The STY channel number.

arg 2 The Chaosnet input channel number to connect it to, or -1 to disconnect it.

arg 3 The Chaosnet output channel number. This is not actually used on the Chaosnet.

arg 4 Up to three 8-bit characters, left-justified, to be transmitted when an output-reset is done on the terminal. These characters are protocol-dependent. If any unusual condition occurs, including input of a byte with the high-order bit on from the network, the STY will be disconnected from the Chaosnet channel and the user will be interrupted. It is illegal to do I/O on any of the involved channels without disconnecting them from each other first.

The CHAOSQ system call allows the ATSIGN CHAOS program, described below, to peek at the queue of pending RFC's. It takes one argument, which is the address of a 126-word buffer. The first RFC packet on the queue is copied into this buffer and moved to the end of the queue. If the queue is empty, the system call takes the error return.

8.2 Utility Programs

The K mode in the :PEEK command gives one line of information for each Chaosnet connection in existence on the local machine, lists the number of packet buffers in existence and free, and lists any queued RFC's (see the following section). Packet buffers are dynamically allocated in groups of 8 from system memory. Packet buffers in existence and not free may be in use to contain packets being transmitted to or received from the hardware, unreceipted output packets, or input packets not yet read by the user program.

The information about a Chaosnet connection printed by PEEK consists of:

IDX The connection index number (not including the uniquization bits).

USR The user index or ITS job number of the user process which owns the connection.

UNAME
JNAME The name of the user process which owns the connection.

STATE The state of the connection. This is one of the states listed in section 4.6, page 25, abbreviated to six characters. LOWLVL means the foreign-protocol state.

IBF The number of input packets buffered and available to be read by the user process.

PBF The number of input packets buffered but not yet available to the user process because they are out of order. Some earlier packets are missing; when they arrive

these packets will become available.

NOS — The number of output "slots" available in the window. The user process may send this many packets before it will have to wait for an acknowledgement.

ACK — The number of received packets which should be acknowledged but which have not yet been. This will not stay non-zero for more than a half second, since if it is non-zero an STS will be transmitted.

R WIN — The window size for the incoming half of this connection.

WIN T — The window size for the outgoing half of this connection.

FOREIGN ADDR — The host name and index number of the other end of this connection. The = command switches between host names and host numbers.

FLAG — One or more single-letter flags indicating special things about the connection. The following flags exist:

- C — The connection is half-closed (one of its I/O channels has been closed and the other remains open).
- F — The connection is turned "off" as far as the interrupt side of the NCP is concerned. No packets will be received or transmitted.
- I — The connection has an input buffer for stream (non-packet) input.
- O — The connection has an output buffer for stream (non-packet) output.
- S — An STS packet needs to be sent.
- T — The connection is connected to a STY. In other words it is an incoming Telnet or Supdup connection and the system is providing the data transfer between the connection and the pseudo terminal.

The :CHASTA command is a long-winded version of the above. It prints several lines of information about each connection that exists, including the number of the next packet to be given to the user, the number of the next packet to be output by the user, and the number of the last packet acknowledged in each direction.

The :CHARFC command, given a host name and a contact name in the command line, will open a connection and print whatever comes back (refusal, simple transaction, or any data that emerge from a stream connection). The contact name may of course be followed by a space and arguments. This command can be useful in connection with the STATUS protocol, to see if a host is up (it will print its name followed by some garbage characters), and in connection with the PULSAR protocol, to turn pulsars on and off.

The :CHATST command provides a variety of simple Chaosnet manipulating commands. Run it and type ? for a list of the commands. The H (hostat) command may be the most useful; it uses the STATUS protocol to get metering information from a host and prints it nicely.

The :HOST command, given the name of a host in the command line, looks up that host in the system host table and prints what is known about it, including its numeric address. This works for hosts on all networks. The :CHATAB command prints a table of all the hosts on Chaosnet.

8.3 Server Programs

When an RFC is received for a contact name for which there is no outstanding LSN, the RFC packet is saved on a pending-RFC queue and a new process is created and made to run the program in the file SYS:ATSIGN CHAOS. This program uses the CHAOSQ system call to find out the contact name of the RFC. The contact name *name* is truncated to six characters if it is longer. If a file named DSK:DEVICE;CHAOS *name* exists, that file contains the desired server program; it is loaded into the server process. When the server process is started, register 0 contains the contact name in sixbit and channel 1 is still open to the program file. The server program must do a listen for its contact name, open the connection, log in, and so forth.

Thus a server for a new protocol can be added simply by putting a link on the DEVICE directory. This directory is also used for Arpanet servers and for ITS I/O device servers.

If no file is found, a file named DSK:DEVICE;CHAOS DFAULT is loaded if it exists. If it does not exist either (which is normally the case) the RFC is ignored and the server process kills itself. After a while the system will refuse the RFC by sending back a CLS unless someone comes along and listens for it.

8.4 Subroutine Packages

The file SYSTEM;CHSDEF > can be .INSRT'ed into a Midas program to define symbols for the format of a packet, the packet opcodes, and the states of a connection. The symbol prefixes are %CO for opcodes, %CS for states, $CPK for packet-format byte-pointers, and %CP for packet-format values.

The file SYSENG;NETWRK > can be .INSRT'ed into a Midas program to provide a library of subroutines for opening connections, listening for requests for connection, analyzing network errors and printing useful messages to the user, and accessing the host-name table (SYSBIN;HOSTS2). All system programs that use the Chaosnet do so with the aid of these subroutines. NETWRK supports both Chaosnet and Arpanet. Documentation is provided in comments at the beginning of the file.

9. The TOPS-20/TENEX Implementation

A Chaosnet connection is represented by a JFN obtained from the CHA: device. The standard I/O operations can be performed on such a JFN, in which case the system will open a Chaosnet stream connection and transfer 8-bit bytes in both directions. When a Chaosnet connection is used in this way, it is compatible with the rest of the TOPS-20 file and I/O system.

Alternatively, special operations can be used to send and receive packets and do other Chaosnet-specific operations on a Chaosnet JFN. These are described below.

For more information, see [CPR].

9.1 Opening Connections

A (potential) Chaosnet connection is represented by a JFN. When the JFN is opened, an actual Chaosnet connection is created. The GTJFN syntax is as follows:

The device name is CHA:. The filename is the symbolic name or octal number of the host at the other end of the connection, or a null string if it is desired to listen for an incoming Request For Connection rather than initiating a connection. The extension (filetype) is normally the contact name; some special cases are described below. Use of * names and JFN stepping is not permitted. The directory, generation (version) number, and semicolon attributes are ignored if present.

When the JFN is opened (with OPENF), normally the system will wait for the connection to open up; a user connection (nonblank filename) will wait for a response to be returned to its RFC, and a server connection (blank filename) will wait for an RFC to come in to its contact name. If an RFC is refused or the foreign host is not up, OPENF will return an error. If data mode 6 or 7 is used with OPENF, it will return immediately, without waiting for the connection to open. This is useful if you want to open several Chaosnet connections simultaneously, or if you want to determine the reason for failure if the connection does not open; if the normal data mode 0 OPENF fails, the operating system will not let you read the CLS packet nor do the .MOERR operation (described below).

There are a number of special cases in the GTJFN syntax. If the extension is a null string, then the contact name is specified by OPENF rather than by GTJFN; AC3 contains the number of characters in the contact name in the left half, and the address of the contact name in the right half. In listening mode (the filename is a null string), then if the extension is also null, this JFN will listen to any RFC that is not otherwise serviced. Privileges are required and only one job at a time can do this. This mechanism is used by a system server process. If the filename is null and the extension is a hyphen, the JFN is put in a special mode for simple transactions; packet-level I/O may be used to transmit any number of RFCs and receive any response packets (ANS, FWD, LOS, or CLS).

When a JFN is closed with CLOSF, if it has an open connection the end-of-data protocol is used (an EOF packet is transmitted and its acknowledgement is awaited), and then a CLS packet is transmitted. (This is not completely implemented yet: currently no EOF is sent, and .MOEOF (see below) must be used) If the JFN is in the RFCReceived state, the RFC is refused by sending a CLS.

9.2 Stream Input and Output

The normal I/O JSYSes (BIN, BOUT, SIN, SOUT) work on Chaosnet JFNs. When the connection was created by listening, doing I/O to it automatically accepts it first (sending an OPN). The input and output data are transmitted as 8-bit bytes in standard data packets (opcode 200). On input, if an EOF packet is encountered the standard end-of-file action occurs. If a CLS or LOS is encountered, or the connection is in a bad state, an error is signalled. The message from the CLS or LOS may be picked up with .MOERR (see below).

On TOPS-20, but not Tenex, the SIBE, SOBE, DIBE, DOBE, SINR, and SOUTR JSYSes may be used. The latter two treat each packet as a separate record.

The OPENF byte size may be either 7 or 8. With a byte size of 8, the raw Chaosnet data bytes are transmitted. With a byte size of 7, the system converts between the ASCII code it uses normally and the Lisp Machine character set, which is standard on Chaosnet. (This is not yet implemented; currently a byte size of 7 will be accepted but will behave the same as 8.)

9.3 Packet Input and Output

It is possible to do packet-level input and output and to deal directly with the details of the Chaosnet protocol by using the special operations described in the following section. Note that stream I/O and packet I/O should not be mixed in the same connection, unless you know exactly what you are doing, since you can get your data out of order.

9.4 Special Operations

GDSTS returns device-dependent status. AC2 returns the state of the connection, and AC3 returns the number of packet slots available in the output window in the left half, and the number of available input packets in the right half. The symbolic names for the connection states are as follows:

.CSCLS The connection is closed (or was never opened).

.CSLSN The connection is listening for an RFC.

.CSRFC An RFC has been received by a listening connection.

.CSRFS An RFC has been sent.

.CSOPN The connection is open.

.CSLOS The connection has been broken by a LOS packet.

.CSINC The connection has been broken by Incomplete Transmission (no response from the other end for a long time).

.CSPRF This is the "permanent RFC" state which is entered by GTJFN with a null filename and an extension of just a hyphen.

MTOPR performs a variety of special operations, with the JFN in AC1, one of the following function codes in AC2, and an argument and/or return value in AC3.

.MOPKS Send a packet. AC3 contains the address of the first word of the packet. An error return is taken if the connection is in a bad state for the kind of packet

being transmitted. This will wait for space to be available in the window.

.MOPKR Receive a packet. AC3 contains the address of a 126-word buffer in which the packet is to be stored. This will wait until an input packet arrives.

.MOOPN Accept a Request for Connection. Error if the connection is not in the RFC-received state.

.MOSND Force out any buffered stream output.

.MOEOF Force out any buffered stream output, then send an EOF packet.

.MONOP Force out any buffered stream output, then wait for it to be transmitted and acknowledged. (This is not a "no op", but .MONOP is the system standard name for this operation.)

.MOERR Returns the error message from a received CLS or LOS packet. An error is signalled if no error message is available. AC3 is a string pointer to where to put the error message; it is updated to point at the terminating null character which makes the message an ASCIZ string.

.MOACN Assigns PSI (interrupt) channels. The left half of AC3 is the channel number for output interrupts, and the right half is the channel number for input and state-change interrupts. Specifying -1 as a channel number disables interrupts. Output interrupts are signalled when the window is full and then an acknowledgement is received which makes some space so that more packets may be output. Input interrupts are signalled when the state changes, and when there are no input packets available and then a packet is received.

.MOSWS Sets the receive window size from AC3.

.MORWS Returns the receive window size in the left half of AC3 and the transmit window size in the right half.

.MOAWS Returns the available space in the transmit window in AC3.

.MOUAC Returns the number of unacknowledged output packets in AC3.

.MOFHS Returns the foreign host number in AC3.

.MOSIZ Returns the maximum packet size in bytes in AC3. This can be smaller than the Chaosnet standard (488) on machines encumbered with an RSX20F front end.

.MOSRT Sets the RFC timeout period in milliseconds from AC3. The maximum is 262 seconds.

9.5 Utility Programs

There are two Chaosnet utility programs, both named CHASTA. One prints one line for each connection that exists, giving its state, number of input and output packets, who it is connected to, etc. The other prints the STATUS protocol information for every host on the network, including the host name, when it was last up, and its packet throughput and error counts. This information is maintained by a system daemon process.

9.6 Server Programs

When an RFC is received for *contact-name*, if no process is listening for *contact-name* and the file SYSTEM:CHAOS.*contact-name* (or DSK:<SYSTEM>CHAOS.*contact-name* on Tenex) exists, the server program contained in that file is run. The server program should open CHA:.*contact-name*. This is implemented by the CHARFC program which runs as a daemon job and opens CHA:., the magic name which gets a copy of all unclaimed RFCs. Normally the server program is run in a freshly-created job, and may log in if it wishes, but if the file is marked as ephemeral (the ";E" attribute), it is run in a subfork of the CHARFC job. Ephemeral servers should be used for protocols that don't involve a long-term connection.

The TELNET and SUPDUP servers attach their Chaosnet connection directly to an NVT, just as the corresponding Arpanet servers do.

When the system starts up, the file SYSTEM:HOSTS2.BIN (or DSK:<SYSTEM>HOSTS2.BIN on Tenex) is read in and used to initialize the host name table inside the system used by GTJFN. This is the ITS/TOPS-20/WAITS standard multi-network host table.

10. The Lisp Machine Implementation

Lisp Machine Chaosnet support consists of a set of Lisp functions and data-structure definitions in the chaos: package. There are three important data structures. A conn represents a connection. A pkt represents a packet. A stream is a standard I/O stream which transmits to and receives from a connection. The details of these data structures are described later.

There are two processes which belong to the Chaosnet NCP. The receiver process looks at packets as they arrive from the network. Control packets are processed immediately. Data packets are put on the input packet queue of the connection to which they are directed. The background process wakes up periodically to do retransmission, probing, and certain "background tasks" such as starting up a server when an RFC arrives and processing "connection interrupts" (described below).

10.1 Opening and Closing Connections

10.1.1 User-Side

chaos:connect *host contact-name* &optional *window-size timeout*

Opens a stream connection, and returns a conn if it succeeds or 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 which 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 which may be buffered. It defaults to 10. If *distinguished port* is supplied, the local index is set to it. This is necessary for protocols which define the meanings of particular index numbers.

10.1.2 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 which will be in the *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 which 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 which will then do a chaos:listen. Use the add-initialization function to add entries to this list.

chaos:accept *conn*
conn must be in the *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 the *RFC Received* state. A CLS packet containing the string *reason* will be sent and *conn* will be removed.

chaos:answer-string *conn string*
conn must be in the *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 the *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.

10.2 Connection States

chaos:state *conn*
Returns the current state of the connection, as one of the following symbols:

chaos:inactive-state	A conn which does not correspond to any Chaosnet connection.
chaos:open-state	An open connection.
chaos:rfc-sent-state	An RFC has been transmitted and no response has yet been received.

chaos:answered-state	An ANS has been received.
chaos:cls-received-state	A CLS has been received.
chaos:los-received-state	A LOS has been received.
chaos:host-down-state	The connection is in the *Incomplete Transmission* state; communications with the foreign host have broken down.
chaos:listening-state	A LSN has been "transmitted" and the connection is awaiting an RFC.
chaos:rfc-received-state	An RFC has been received while listening and has not yet been responded to.
chaos:foreign-state	The connection is being used with a foreign protocol encapsulated in UNC packets.

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 "net wait".

10.3 Stream Input and Output

chaos:stream *conn*

Creates a bidirectional stream which accesses *conn*, which should be open as a stream connection, as 8-bit bytes. In addition to the usual I/O operations, the following special operations are supported:

:force-output	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	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	Forces out any buffered output, sends an EOF packet, and does a :finish.
:clear-eof	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.
:close	Send a CLS packet and remove the connection.

10.4 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 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 chaos:first-data-word-in-pkt'th element of the array is the first 16-bit data word. The header of a pkt contains a number of fields used by the system.

chaos:pkt-opcode *pkt*
Accessor for the opcode field of *pkt*'s header. For each standard opcode a symbol exists in the chaos: package, consisting of the standard 3-letter code and a suffix of "-op", chaos:rfc-op for example. The value of the symbol is the numeric opcode.

chaos:pkt-nbytes *pkt*
Accessor for the number-of-data-bytes field of *pkt*'s header.

chaos:pkt-string *pkt*
An indirect array which 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*).

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*
Deallocates a pkt.

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.

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).

chaos:may-transmit *conn*
A predicate which returns t if there is any space in the window.

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 "NETI".

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

10.5 Connection Interrupts

chaos:interrupt-function *conn*
This attribute of a conn is a function to be called in the background process 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.

10.6 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.

hostat &rest *hosts*

Interrogates the specified hosts, or all known hosts if none are specified, with the STATUS protocol and prints the results in columns as a table.

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.

11. The VAX/VMS Implementation

This describes the interface to Chaosnet through the routines in the "CHAOS.B32" BLISS-32 subroutine package. Definitions of standard values are in "NCPDEFS.R32". Though it is possible to interface to the NCP at the VMS I/O level, it is not recommended practice. All references to Chaosnet in this text are with respect to the subroutine package, and not VMS QIO's.

A Chaosnet connection is represented by a one longword "channel number", which has no direct relationship to a VMS channel number. However, for every Chaosnet channel currently allocated, there is an associated VMS channel maintained by the subroutine package.

All of the routines described below are declared "global".

11.1 Opening and Closing

parse_host (*host, ret-host-num*)
Parses the string pointed to by *host* (which points to a standard VMS string descriptor), and stores the resulting host number in the word pointed to by *ret-host-num*. Returns a status code.

chaos_rfc (*ret-chan, host, contact-name, wait-time*)
Opens a new Chaosnet channel and sends an RFC. *ret-chan* is a longword to receive the channel number. *host* is a string acceptable to parse_host. *contact-name* is a pointer to a string descriptor. *wait-time* is either zero, which means to wait indefinitely for a response to the RFC, or a pointer to a quadword block acceptable to the $SETIMR system service. A status code is returned, which will be SS$_TIMEOUT if the routine times out.

chaos_lsn (*ret-chan, contact-name, wait-time*)
Like chaos_rfc, but "sends" a LSN instead of an RFC. No host is specified.

chaos_accept (*chan, window, rfc-arg, ret-rfc-arg-size*)
Accepts an incoming RFC. The connection must be in RFC-received state. *window* is the window size. *rfc-arg* is an optional string descriptor which receives the argument to the RFC. *ret-rfc-arg-size* is also optional, and gets the argument's length.

chaos_ans (*chan, data, wait-time*)
Sends an ANS packet to the Chaosnet channel. *data* points to a string descriptor, *wait-time* is ignored. A status code is returned, and if an error occurs, the channel is deassigned.

chaos_close (*chan, reason*)
Closes the connection, and deassigns the channel. *reason* is a pointer to a string descriptor of a string to be included in the CLS packet.

chaos_assign (*ret-chan*)
Assigns a Chaosnet channel, and stores it in the longword pointed to by *ret-chan*. This routine allocates a VMS channel. A status code is returned.

chaos_deassign (*chan*)
Given a Chaosnet channel previously assigned by chaos_assign, deassigns it and the associated VMS channel.

11.2 Stream Input and Output

chaos_in_char (*chan, ret-char, timeout*)
Returns the next character from the channel in the longword pointed to by *ret-char*. Waits until a character is available or until *timeout*, whichever comes first. A status code is returned.

chaos_out_char (*chan, char*)
Outputs one character. Characters are buffered until a packet fills up or until the output is forced out by chaos_force_out. A status code is returned.

chaos_sout (*chan, string*)
Like repeated calls to chaos_out_char: sends string from string descriptor pointed to by *string*.

chaos_force_out (*chan*)
If doing serial output, and a partial packet is buffered, force it to be sent.

chaos_finish (*chan*)
Does a chaos_force_out, then waits for all packets to be acknowledged by the foreign end.

chaos_eof (*chan*)
Sends an EOF packet after forcing out any buffered output.

11.3 Packet Input and Output

allocate_pkt (*size, chan, ret-pkt*)
Allocates a packet suitable for chaos_in_pkt and chaos_out_pkt. The packet can hold up to *size* bytes of data; the number of bytes field in the packet's header is filled in from *size*. *ret-pkt* points to a longword to receive a pointer to the packet. A status code is returned.

deallocate_pkt (*pkt*)
Returns a previously allocated packet to the free pool. A packet may be reused, since the I/O routines do not deallocate them, as long as the I/O is being done synchronously. Returns a status code.

chaos_out_pkt (*chan, pkt, efn, astadr, astprm*)
Outputs *pkt* to *chan*, waiting if there is no window room available. *efn* is the event channel to use for waiting. *astadr* and *astprm* are as for VMS system services: an AST address and parameter, respectively, that get signalled when the packet is read by the NCP. chaos_out_pkt returns as soon as there is space in the window, without waiting for the NCP to finish transmitting the packet.

chaos_in_pkt (*chan, efn, pkt, astadr, astprm*)
Reads the next input packet, whatever opcode it may be, from the connection, waiting indefinitely if there are no input packets. *efn* is the event channel to wait on, and *astadr* and *astprm* are for an AST to be delivered when the read completes. chaos_in_pkt does not return until the read completes. A status code is returned.

11.4 Checking the State

chaos_xmit_room (*chan, wait*)
Returns SS$_NORMAL if there is room left in the transmit window. Returns an error if the connection went into a bad state. If *wait* is true, and there is no room left, then chaos_xmit_room waits until room is available. If there is no room left and *wait* is false, it returns SS$_EXQUOTA.

chaos_state (*chan*)
Updates the state of the Chaosnet channel via a request to the NCP. Returns a status code. To check the state of the connection, first call this routine then look at chan_state in the channel block described below.

chaos_wait (*chan, old-state, timeout*)
Waits until the channel goes out of the specified state or until timeout occurs. Timeout is either zero (no timeout) or a pointer to a quadword block acceptable to $SETIMR. A status code is returned.

chaos_wait_til (*chan, state, timeout*)
Waits until the channel goes into the specified state or until timeout occurs. Timeout is either zero (no timeout) or a pointer to a quadword block acceptable to $SETIMR. A status code is returned.

The channel number is used as an index into the global blockvector channel, defined in the "CHAOS.B32" file. Since BLISS-32 does not allow the field definitions to be global, they should be copied into any program that needs to look inside the channel blockvector. The most useful fields are

chan_state	One of the state codes defined below.
chan_sta_txw	The window size in the transmit direction.
chan_sta_rxw	The window size in the receive direction.
chan_sta_txwa	The number of packet slots available in the transmit window.
chan_sta_rxav	The number of input packets available.

The states are as follows:

conn_st_closed (0)	Connection closed by a CLS packet.
conn_st_rfcrcv (1)	RFC received by listening connection.
conn_st_rfcsnt (2)	RFC sent, no response yet.
conn_st_open (3)	Connection open.
conn_st_los (4)	Connection broken by a LOS packet.
conn_st_incom (5)	Incomplete transmission (no response from foreign host).
conn_st_new (6)	Connection newly allocated.
conn_st_lsn (7)	Listening for an incoming RFC.
conn_st_full (%O'400')	This bit is set when the transmit window is full. Usually, the remainder of the state will be conn_st_open.

12. The Unix Implementation

The Unix implementation of Chaosnet works on both pdp11 Unix and VAX Unix.

The network may be accessed in four different ways. A Chaosnet stream connection may be opened to a host and a contact name, and then standard Unix I/O may be done over the connection. A Chaosnet connection may be accessed at the packet level (however, this is not yet implemented). A Chaosnet stream connection may be made into a Unix tty; this is how the TELNET server works. A subtree of one Unix host's file system may be mapped into another Unix host's file system, with the intermediate Chaosnet invisible to programs operating on the remote files.

You get access to Chaosnet by opening a file pathname which includes a *special file* whose major device is Chaosnet. The minor device number of that special file, together with its name and any additional pathname components after it, are used to determine the host, RFC contact name, and RFC arguments. Remote file access is implemented by special files which map into the desired host with contact name ftpread or ftpwrite. The remaining pathname after the special file is passed along and used as a pathname on the remote host.

The 8-bit Unix minor-device number is divided up into several fields which control details of what happens when you open a Chaosnet special file. The current division is as follows:

bits 6-3 Host specifier. Small numbers are indices in a site-dependent table of hosts which are particularly useful to connect to. This is used primarily for the remote file-system facility. Otherwise this is one of:

- 015 The host number (in decimal) is read from the name of the special file.
- 016 The host number (in decimal) is read from the next pathname component after the special file.
- 017 Activates one of a number of special features selected by bits 2-0. See below.

bits 2-0 If the host specifier is not 017, this is a contact name specifier. Small numbers are indices in a site-dependent table of useful contact names. This is used primarily for the remote file-system facility. Otherwise this is one of:

- 6 Read the contact name from the name of the special file.
- 7 Use the rest of the pathname, after the special file, as the contact name and RFC arguments.

When both the host number and the contact name are read from the pathname, the host number is read first. Any character other than the digits 0 through 9 serves to separate them.

If the host specifier is 017, this is instead one of the following:

- 0 Pick up unmatched incoming RFCs. Only one process at a time, per system, may open this device. The ioctl operation chiocnext (see below) is used with this device.
- 1 Listen mode. The rest of the pathname, after the special file, is the contact name listened to. This will wait until an RFC comes in to that pathname. A

typical pathname would be "/dev/chlisten/myprotocol".

2 Listen-error mode. Same as above, but if there is no pending RFC to the contact name, this will return an error rather than waiting for one.

3 Packet mode. This is not implemented yet.

bit 7 Specifies that this Chaosnet connection should be wired to a tty.

When a Chaosnet connection is wired to a Unix tty, input data from Chaosnet appear as keyboard input characters on that tty. Output typed on the tty goes out on Chaosnet as data. When the tty is hung up (logged out), the Chaosnet connection will be closed. I/O operations done to the Chaosnet channel will be done to the tty instead, except for the special ioctl operations given below.

When a connection has been opened, the normal Unix I/O operations may be done on it. 8-bit bytes in standard data packets (opcode 200) are used. If the Chaosnet connection is not open, the I/O operations will return an error. When the close operation is performed, if the Chaosnet connection is open the standard EOF protocol will be followed (see section 4.4, page 24).

The following operations are supported for the ioctl system call:

fionread Returns the total number of data bytes in the available input packets. This is the number of bytes which can be read without waiting.

chiocflush Forces out buffered output. Normally stream output is held until a packet is filled up or 2 seconds have elapsed.

chiocrread Returns data from the RFC packet. This is only valid on a connection which was created by listening and which has not yet been read from. The source host (2 bytes) is returned, followed by a null-terminated character string which is the contact name and any following arguments.

chiocrnext This is only valid on the unmatched-RFC connection, and is the only ioctl operation valid on it. This waits until an unmatched RFC is present, and then returns its contact-name and arguments as a null-terminated string. The system unmatched-RFC process uses this to start up server processes as RFCs come in. (Currently this mechanism is used only for the remote file-system servers.)

chioctty If the Chaosnet connection is open, and is not already wired to a tty, this creates a new Unix tty and wires it to the connection.

References

The following documents are of some related interest. AIM is an AI Memo of the MIT Artificial Intelligence Laboratory. RFC is a Request for Comments of the Arpanet Network Working Group. IEN is an Internet Experiment Note of the Arpanet Network Working Group.

[AIM444] A. Bawden, R. Greenblatt, et al., Lisp Machine Progress Report, AIM-444.

[CHINUAL] D. Weinreb, D. Moon, Lisp Machine Manual, MIT AI Lab.

[CPR] C. Ryland, TOPS-20 Chaosnet Manual, unpublished.

[ETHERNET] R. Metcalfe, D. Boggs, Ethernet: Distributed Packet Switching for Local Computer Networks, CACM Vol. 19, No. 7, July 1976, p. 395.

[FILE] Documented online on the file AI:LMDOC;CHFILE >.

[FINGER] K. Harrenstien, Name/Finger, RFC-742.

[RFC733] D. Crocker et al., Standard for the Format of Arpa Network Text Messages, RFC-733.

[SUPDUP] M. Crispin, Supdup Protocol, RFC-747, RFC-734.

[TCP] DOD Standard Transmission Control Protocol, IEN-129.

[TELNET] Telnet Protocol Specification, RFC-542.

[TIME] K. Harrenstien, Time Server, RFC-738.

[UDP] J. Postel, User Datagram Protocol, IEN-88.

[UNIBUS] PDP11 Peripherals Handbook, Digital Equipment Corporation.