SHUSTEK

Experiences with a Layered Approach to Local Area Network Design

GARY M. ELLIS, MEMBER, IEEE, SAM DILLON, SKIP STRITTER, MEMBER, IEEE, AND JERRY WHITNELL

Abstract—**The Nestar Plan 4000 network was designed using the ISO Open Systems Interconnection model. The Xerox Network Systems Internet Transport Protocols (XNS) were chosen for the network and transport layers, a token-passing protocol (Arcnet) for the physical and datalink layers, and existing Nestar server software for the highest layers.**

The physical and datalink layers are supported by a VLSI chip and their implementation was straightforward. In spite of their detailed specification, implementing the network and transport layers presented some unanticipated challenges. This paper discusses details of our experience in implementing the network with emphasis on the problems encountered and how they were solved.

I. Introduction

THE International Standards Organization (ISO) has defined the "Open Systems Interconnection—Basic Reference Model" [1] as a guideline for structuring the architecture of local area networks. The model is characterized by a "layered" approach in which the architecture is divided into seven independent layers, each of which builds on the function provided by lower layers and adds function of its own. The OSI model itself is not a standard, but rather a guide to network design. The use of a conceptual framework or "model" is useful in software engineering because it helps understanding of complex systems such as networks. Our reasons for using the layered approach include, in addition, the potential for compatibility with other networks and the possibility of interchangeability of different implementations of individual layers.

Manuscript received April 15, 1983; revised August 19, 1983.
The authors are with Nestar Systems, Inc., Palo Alto, CA 94303.

This paper presents a case study of how Nestar Systems, Inc., Palo Alto, CA, developed a new network [2] using this model. This network will typically be used to provide an underlying data-sharing, service-sharing, and interstation communication environment for application systems that run on personal computers. These applications range from individualized programs such as word processors to network-wide systems such as a medical records database or a foreign exchange trading package. Our purpose is not to present new knowledge in the field of local area networks, but rather to explain the reasoning behind our choices of existing technologies and to detail our experiences in the implementation of the network. The primary design goal was the need to replace our aging Cluster/One technology [3] with a faster network in time to meet changing market needs.

II. The Strategy

A. Overall Design Goals

The user stations environment supported by the new network was to consist of relatively low cost personal computers, therefore the cost per station to connect to the network had to be low also. When the project was started in June 1982, the cost of a workstation was roughly \$2000–\$3000. It was desired that the cost per connection be about 25 percent of this. An order of magnitude increase in maximum data throughput (from 250 kbits/s to 2.5 Mbits/s) over the previous generation [3] was desired, although we were aware that other considerations would

0733-8716/83/1100-0857$01.00

prevent the user from seeing the full ten times speed improvement. In addition, the forecasted marketing window limited the amount of time available for the implementation. We had approximately five months to produce a working prototype plus two to three additional months to finish, tune, and reliably manufacture the system.

Our existing network supported a single type of personal computer. This allowed a relatively high-reliability design in which the file server was hosted on one of the same computers. If the file server processor failed, another workstation could easily take its place. Since the new network was planned to support several types of personal computers, a decision was made to move away from hosting the file server on a workstation. Our experience to date has been that almost all of our networks are sold to users who purchase the workstations at the same time. As such, individual networks usually consist of a single type of computer, and for marketing reasons, would require the file server to run on the same type of machine. This would have led to supporting a large body of software in a number of different machine environments. For this reason, and the goal of higher computing performance, we chose to design proprietary file server hardware based on a commercially available 68000 processor board.

<table>
<tr><th>Layer</th><th>Purpose</th><th colspan="2">Nestar Implementation</th></tr>
<tr><td>Application</td><td>The program that the user runs</td><td rowspan="3">client station software</td><td rowspan="3">server station software</td></tr>
<tr><td>Presentation</td><td>Library routines provided by operating system</td></tr>
<tr><td>Session</td><td>Users interface to the network</td></tr>
<tr><td>Transport</td><td>Maintains virtual connections between user processes. Divides messages into packets</td><td colspan="2" rowspan="2">Xerox Network Systems Internet Transport Protocal</td></tr>
<tr><td>Network</td><td>Maps global (Ethernet) address into local network address. Performs gateway function for packets bound for other networks</td></tr>
<tr><td>Datalink</td><td>Receives packets from other stations on local network. Makes best effort to send packets to stations on local network</td><td colspan="2" rowspan="2">ARCNET</td></tr>
<tr><td>Physical</td><td>Cable and electrical interface</td></tr>
</table>

Fig. 1. ISO model of Plan 4000 network.

B. Choices For the Levels

We decided on the layered approach for several reasons. Ease of parallel development, maintainability, and ability to easily interface to other networks are all strong points which are inherent in a modular approach. A side effect of this decision was that it allowed us to choose implementations for groups of layers that are relatively independent of each other, yet have advantages that a more homogeneous approach would not have. A major drawback that we considered was that layering the software could seriously degrade performance. The limited interfaces that are a goal in modular programming can lead to more data copying and procedure calls than a monolithic approach.

We chose to use Datapoint's Arcnet technology [4] for the physical and datalink layers (levels 1 and 2), the Xerox Network Systems Internet Transport Protocols (XNS) [5] for the network and transport layers (levels 3 and 4), and existing Nestar server software for the upper layers (levels 5–7) (see Fig. 1).

C. Motivation for Arcnet Physical and Datalink Protocols

The decision as to which technology to use at the datalink level involved a number of tradeoffs. Besides the obvious ones of speed and class of service (token-passing versus CSMA/CD, baseband versus broadband), these factors included the availability of a standard, the cost and time required for implementation, the availability of parts, the amount of circuit board space required, and the ease of installation and maintenance. We chose baseband over broadband because of our relatively simple requirements —there was no requirement for voice or video transmission in the desired system. The difficulty in installing and tuning a broadband system would have added an undesirable level of complexity. Also, the lack of VLSI implementation would have greatly increased the cost of the interface.

The choice of Arcnet was made for several reasons. The first was the overall cost of connecting a station to the network. In late 1982, an Arcnet chip set was being shipped in sample quantities, but there were no VLSI Ethernet parts available. This meant that we could produce an Arcnet interface for 30–50 percent of the cost of an Ethernet interface. Having been installed in over 4000 installations and in use for six years, Arcnet possessed a maturity of use that we felt was lacking in Ethernet. Datapoint had decided to make the formerly proprietary technology freely available and was willing to share technical information with us. This was an important factor in minimizing implementation time. Finally, there are some technical advantages (detailed in Table I) that Arcnet has over Ethernet.

We felt that these advantages were more important than the (theoretical) four times speed advantage of Ethernet (10 Mbits/s compared to 2.5 Mbits/s for Arcnet). Our experience indicates that the lower speed does not restrict the actual throughput achieved in a working network. As discussed in a later section, other issues limit end-to-end effective throughput to substantially less than the theoretical potential in this and other networks.

D. Motivation for Xerox Internet Transport Protocols

For the next two layers of the protocol (network and transport), the Xerox XNS protocols were chosen. Initially, there were three choices for levels 3 and 4. We could use XNS, DARPA TCP/IP [6], [7], or we could design our own. Other standards, such as those of the National Bureau of Standards (NBS) or X.25, were either unavailable at the time the decisions were made, or judged to be inappropriate for local area networks.

TABLE I
ARCNET VERSUS ETHERNET COMPARISON

	ARCNET	Ethernet
Physical topology	Unrooted tree of cable segments	Cable segments connected by repeaters (max 2 between stations)
Maximum cable segment length	610 meters	500 meters
Maximum distance between stations	6700 meters	1500 meters
Packet collision avoidance	No collisions—only token holder xmits	Collisions detected by transmitter listening to itself
Acknowledged packet delivery	Yes	No
Datalink flow control	Yes	No

We quickly rejected the option of designing our own protocols because of the time constraint. We had an average of 12 engineers and programmers and five months to do the job. Subtracting some time for management and other nondevelopment activities, there were 40–45 man months available for both the software and hardware development efforts. In order to meet schedules, we felt that a well-defined protocol that was already debugged and in use was required. It was also felt that some commercial acceptance of the selected protocols would help in marketing the new product.

The TCP/IP protocol, used mainly for ARPAnet, was originally designed for data transfer over low speed serial communication lines such as the telephone. As such, the protocol is not well suited to the interactive applications we intended to run on the new Nestar network. Also, there were essentially no major commercial environments supporting TCP/IP. This meant that a large amount of protocol translation software would be required to interface with other vendors' products.

Because of Xerox's early entry into the field, and the fact that their protocol is very well specified and in the public domain, we felt that XNS would become a de facto standard for internetwork communication. It has the advantage that it is independent both of the layers below it (how a packet is transmitted from one node to another) and the layers above it (how a client process wishes to use the network). This allowed us to keep most of our existing upper level server software. It also gave the option of adding support for other low level transmission protocols (e.g., Ethernet layers 1 and 2) without changing upper level software. The published specification allows other suppliers to write applications which use the protocol. A significant number of other vendors (more than 30 in mid 1982) had announced corporate support for Ethernet for their local networks and although we are using token passing rather than CSMA/CD, using the Ethernet Internet protocols would make it relatively easy to produce gateways to those networks.

E. Motivation for Nestar High Levels

It was necessary that the change to a new network technology be invisible to existing user application software because providing an upgrade path for existing customers was a primary goal. On the user station side of the connection this meant that everything from the interface of the session layer software up had to be identical to that in the current network. There were consequently no major design alternatives to choose from in the user station higher level software—it was necessary only to change the session layer software to interface to the new transport layer below it.

The situation with the file server however, was somewhat different. New higher performance server hardware based on the 68000 and 256 kbytes of memory was being developed, and the possibility existed of rewriting a large part of the existing software to take advantage of it. The file server software was originally written for a lower performance noninterrupt driven computer and operating system. A faster network and a somewhat higher cost file server (which probably meant less servers per network) made it desirable to squeeze as much power as possible out of the new file server. This would have meant changing the presently single-threaded code into a number of concurrent tasks. The major disadvantage to this alternative was the need to find (or possibly develop) a suitable real-time operating system for the new software to run under. The time constraint for implementation (five months) made this an almost impossible task.

The chosen alternative, using the present software relatively unmodified, was more attractive for several reasons. A single-tasking operating system (and Pascal compiler) already existed for the new hardware which was very similar to the language/operating system environment of the previous server. A port of the software could be made relatively easily. The only additional software needed was a new session layer to interface to the new transport layer. The existing software had been operating in the field for about three years, so it was reliable and its behavior was well understood. This allowed us to concentrate the greatest part of our development resources on producing new network protocol software.

F. Advantages to the Layered Approach

Implementations of the different layers were assigned to different programmers. This concurrent development continued until the final debugging stages. As a measure of the independence of the software development efforts, support for one user station was written at a sister company in the United Kingdom. Only after both user station and file server software were essentially complete did the two attempt to communicate over the network. Four people working less than a month managed to repair most bugs. Having two completely separate groups implement the protocols in entirely different ways (one used Pascal, one used assembly language) proved to be a good method of validating the protocols and the documents defining them.

The modular approach also allows changing any layer of the network software without modifying the others. This is a great advantage for future gateway or bridge implementations. For example, to create a Plan 4000/Ethernet gateway would only require creating Ethernet levels 1 and 2 for the gating computer. Similarly, by replacing levels 3 and 4 we could create a gate to the higher levels of Datapoint's Arcnet software. It also allows the freedom to

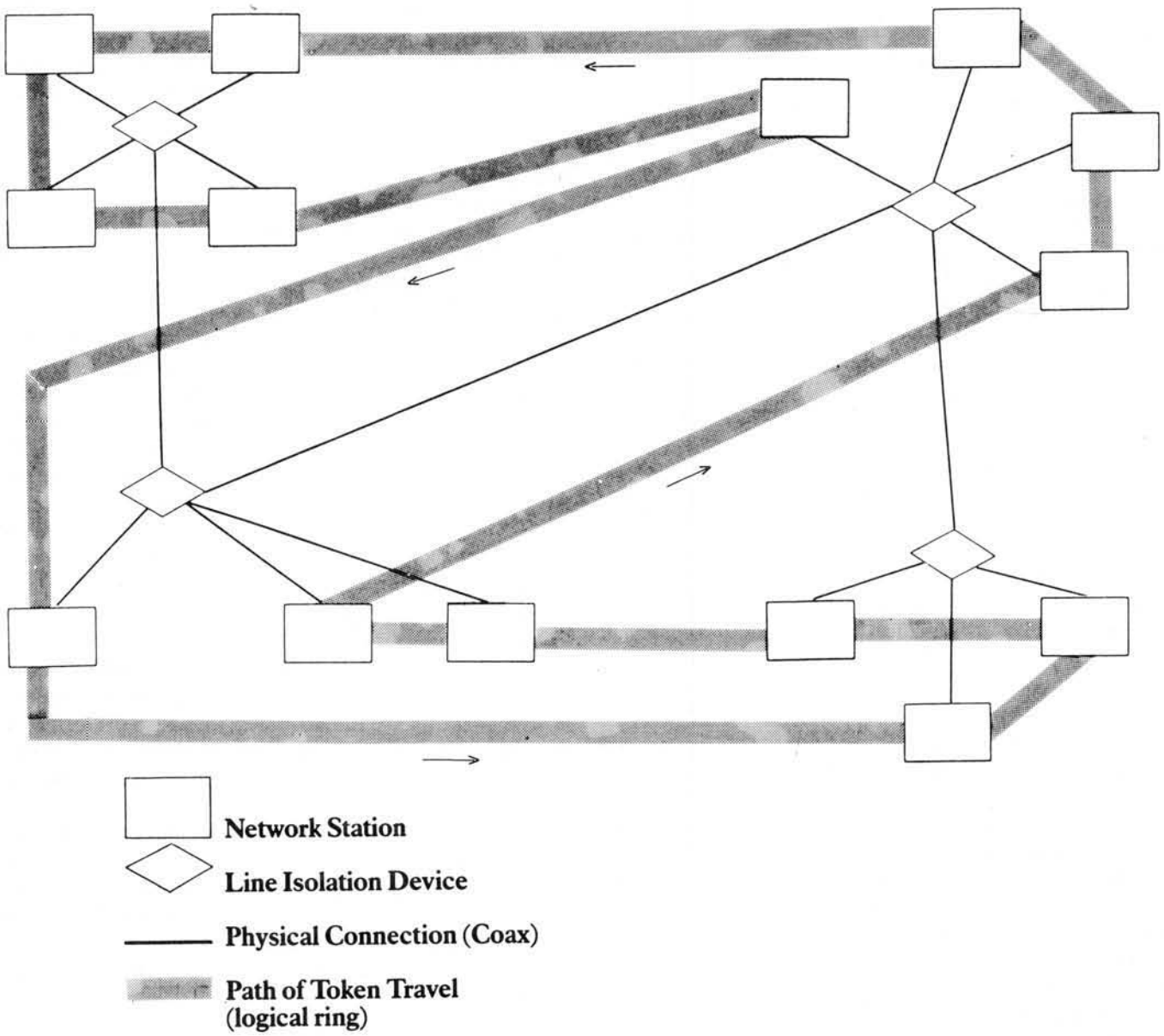


Fig. 2. Example of Plan 4000 network topology.

adopt a different physical layer for the network. We could move to a fiber optic network by only redesigning the transmission/reception hardware of level 1 and not modifying any software.

III. Summary of the Characteristics of the Chosen Layers

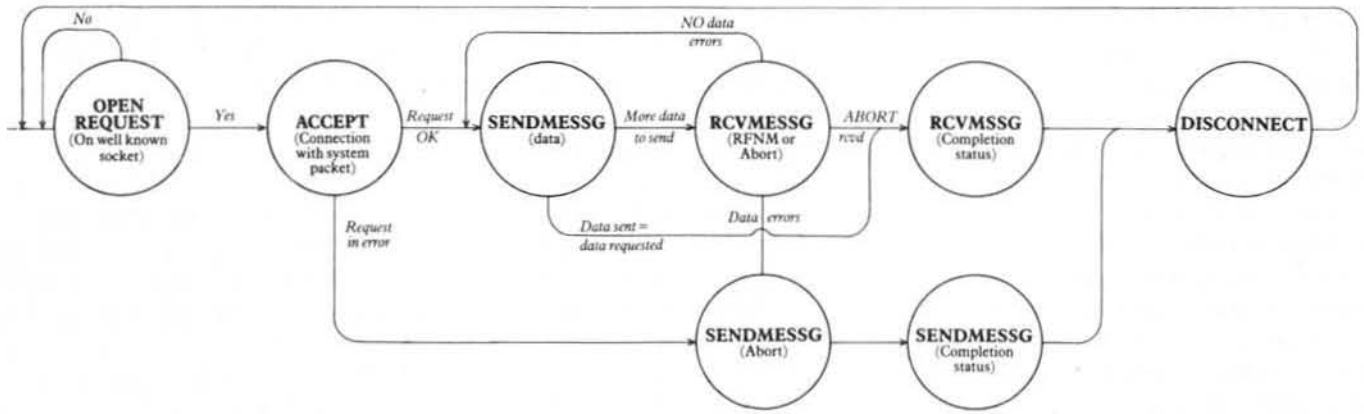
A. The Physical and Datalink Layers

The first two layers (physical and datalink) are Datapoint's Arcnet. Physically, Arcnet is an unrooted tree structure (Fig. 2) with up to 255 stations allowed. New stations may be connected to the network at any point through a length of coaxial cable. The only constraints are that there must be no loops in the structure and the distance between any two nodes must not exceed 6700 m. Details of how Arcnet works are given in Appendix A and [4].

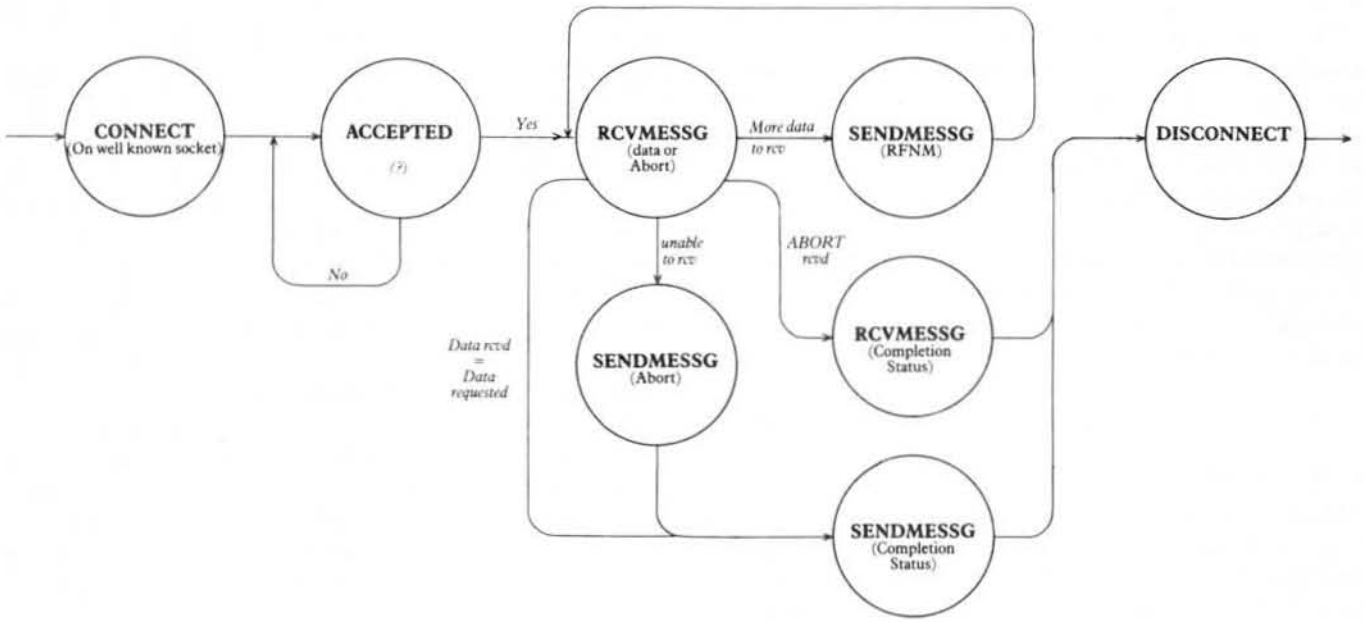
The datalink layer software provides a clean interface to the network protocol chip (called the RIM chip, Standard Microsystems 9026; see Appendix A for more detail) and other hardware. It supplies its client (the network layer) with the ability to send a packet to any station on the network, to send to all stations on the network, to determine if a packet has been received and if so, what station it was from. All details of token management and packet transmission are handled transparently by the network protocol chip.

B. The Network and Transport Layers

The network and transport layers are based on the Xerox XNS standard, which defines the internetworking protocols used by Xerox with Ethernet. (They are not, strictly speaking, part of the Ethernet standard [8], [9], which defines only the physical and datalink layers.) (See Appendix B for a description of the entry points for Nestar's implementation of levels 3 and 4.) The network layer implements the translation from Ethernet addresses (which are used at the higher levels) to Arcnet addresses



(a)



(b)

Fig. 3. (a) File server disk read transaction. (b) Client disk read transaction.

(for the low levels). It decides whether the station is on the local net, or on one which is connected by a gateway or bridge, and routes the message appropriately. Finally, it implements the concept of "sockets" (virtual ports into a machine). Our initial implementation was required to access only the attached local network, so most of the internet routing function was left out. Using the Arcnet address as one byte of the Ethernet host address allows a simple Ethernet to Arcnet address mapping in the isolated network case.

The transport layer provides reliable and safe conversations between sockets. It allows its client (the session layer) to connect to another station on the network, to send and receive arbitrarily long messages and then disconnect, knowing that all the messages will get to the destination in the order they are sent, without any losses, as long as the destination machine is available.

C. The Session, Presentation, and Application Layers

The Nestar software at the upper levels provides the ability to request services from or exchange data with other stations. The file server accepts I/O requests over the network and emulates local disks for the workstation on its hard disk. The file server also supports commands directly from the workstation to create, delete, or rename virtual volumes on the server's disk and to provide synchronization services. Other servers exist which can print files from any file server on the network, transfer files between file servers, and provide communication with mainframe computers.

The file server and its clients communicate at the session layer through transactions which are produced through the use of transport layer services. Fig. 3 shows a disk read transaction from the perspectives of both a server [Fig. 3(a)] and a client [Fig. 3(b)]. In the diagrams, the bubbles represent transport layer entry points. The conditions on control flow (arrows) can be either the value of an associated bubble (if it represents a Pascal function) or the value of a session layer variable. A "RFNM" (request for next message) is a special session layer message that a receiver uses to notify a sender of the amount of buffer space available for a data message. That buffer space is allocated by the application layer and its size is simply passed to the

sender by the session layer. An "abort" is a zero (data) length message that one side's session layer uses to notify the other of its inability to continue the transaction. Since there are no data associated with an abort, it can be sent at any time without the necessity to allocate buffer space. Whenever one end of a connection sends an abort, our protocol requires that end to send a completion status message indicating the reason for the abort. In the case of a normal termination (indicated by received-data-length = requested-data-length) the receiver sends a completion status message to the sender to indicate that the transaction has been successfully finished. Note that the transport layer acknowledgment protocol is not sufficient to mark end-of-transaction because nonnetwork related problems, such as a disk error, could require the transaction to be retried.

The session layer message types referred to above are actually defined by a field in the transport layer header. This method of definition is part of the Xerox XNS specification. One of the parameters returned by the transport layer from a RCVMESSG call is that message type. (In an analogous fashion, the session layer supplies a message-type parameter to the transport layer in a SENDMESSG call.) This is essentially a performance enhancing part of the XNS protocol that gives one layer a quick indication of the contents of the data passed to it by the next lower layer.

IV. Implementation Experience and Problems

A. Design and Coding

Despite the fact that the services provided by each of the layers in the ISO model are well documented in the Xerox XNS specification, the interfaces and entry points are not defined. This was both a help and a hinderance. We were not forced to emulate an inappropriate design or implement an interface that was not really suitable for our needs. On the other hand, we had much more design work to do. Also, because there is no published or standard interface, our code will not interface to other implementations of specific layers without being modified. It is not clear whether the lack of interface definition allows enough freedom so that separate implementations could be truly incompatible, as opposed to merely stylistically different. Appendix B details our choice of interface.

In addition to the interface, certain details of how each layer should provide its services are not defined in the specification. For instance, the usage of certain control bits in headers, such as when acknowledgments should be requested, became clear only after we had had some experience with data transfer between stations on the network. Our initial understanding of the function of the Send Acknowledgment bit in the transport layer header was to request an immediate ACK of the last received packet. The implementations in both the file server and the user station, had the transport layer setting Send Acknowledgment on every message boundary. This had the effect of causing a system packet to be sent by the receiver after every message. (A system packet is used by the transport layer to communicate protocol information to another transport layer—it carries no higher level data.) The XNS protocol allows acknowledgments to be "piggy-backed" on regular user-data packets so that system packets are not always necessary for this function. In our attempts to improve performance by limiting the number of system packets, we experimented with not setting Send Acknowledgment at the same time as End of Message. This led to deadlock situations where the sender of a message required an ACK before sending the next message, and the receiver was waiting until its next send to piggy-back the ACK. It soon became apparent that the real meaning of the Send Acknowledgment bit is "I need an acknowledgment before I will send any more data—send it in the most efficient manner possible." The solution was to change the transport layer code to record the fact of the acknowledge request, and if it is determined at a later time that a piggy-backed ACK is not possible (i.e., the session layer makes two RECVMESSG calls in a row), then send a system packet.

One of the technical advantages that Arcnet has over Ethernet is the packet acknowledgment protocol performed automatically by the RIM chip. Other physical and datalink protocols, such as Ethernet, give best-effort packet transmission only. A design decision was made to avoid reliance on this automatic acknowledgment by higher levels of the protocol—each layer uses only its own reliability checks. This way if we produce an Ethernet low level interface (or some other best-effort-only lower level), we will not have to rewrite level 3 and above to make up for nonguaranteed service from lower levels. Performance studies done after the installation showed that checksum failures occur on only 1/10 of 1 percent of all transmitted packets, so the cost of not using the acknowledgment protocol is small.

The limited interfaces that are a part of a modular design proved helpful several times. The file server transport layer code was heavily modified during the integration testing with the user stations. After the first release of the product, the datalink layer was rewritten to enhance performance. The transport layer code on some of the supported computers was completely rewritten several times, either to improve performance or to conserve memory space on very small computers. None of these modifications led to any code changes in other layers on the same computer, or in any layers in other machines.

No insurmountable problems were experienced while porting the file server application layer software to the new environment. Since memory address space was restricted to 64 kbytes in the Cluster/One file server, a fair amount of nonstandard Pascal had been added to the software during its life which had to be cleaned up for the new environment. Some of the utilities that were supplied with the new operating system had bugs that occasionally slowed progress. In spite of this, however, it took five people only three months to produce an operating file server on the new network. An additional 14 man months produced network integration for three different user station computers running a total of six different operating systems.

B. Verification

One major problem was the lack of verification tools. We found that making the protocols work was not difficult. A more subtle question is how to monitor the network to see what is really going on and be sure that it is doing what is intended in the most efficient manner. As others have reported [10], the very design of the network, with elaborate mechanisms for recovery from errors and lost packets, can interfere with finding subtle problems. For example, an error which results in generating many duplicate packets may be hidden because the receiving end (by design) discards the duplicates. We found several instances of such errors, which were not causing network failure, but were degrading network performance.

We built a special hardware network monitor from a specially modified network interface dubbed the "sniffer." It accepts every packet that goes over the net regardless of destination address, which is something not ordinarily permitted by the RIM hardware. We developed filtering software that allows us to keep only the packets we want in a large circular buffer holding about 2000 packets. This allowed us to manually verify network transactions and proved invaluable in debugging the protocols.

The interactions of various timeout parameters in the system, both intraserver and between servers and clients, have proved to be the most complex part to understand and tune. One example involves how long the datalink layer continues to attempt transmitting a packet and how long the transport layer attempts to pass a packet to the network layer for transmission. The latter operation may fail because of a lack of an available transmit buffer in the physical layer. Originally, the datalink and transport layers in the file server were implemented by different programmers and there was little coordination of timeout values between the layers. (In fact, our lack of understanding of the way the network functioned as a system would have made such coordination of little value anyway.) The datalink timeout was set to several seconds, with the idea that a less powerful user station might take a relatively long time to empty its receive RIM buffers. The transport "send-packet" timeout however, was about 1/5 as long. The overall effect of this was that the datalink was attempting, to service a packet for about five times longer than the transport layer was. Aside from being a tremendous waste of datalink resources, it was causing a great deal of transport error recovery code to be executed whenever a user station failed to complete a transaction (a common occurrence in an environment where a user can abort a program at will).

Using the "sniffer" to observe the interstation network behavior, we discovered two problems. One was that since the network code in the user stations was designed to keep the RIM buffers empty as much as possible (either through interrupts, or a tight polling loop during a network transaction), it was highly likely that an unavailable buffer condition meant that the client was in an error state, and would be unable to complete the transaction anyway. As the file server is a resource to be shared among many users, it could not afford to wait too long for a client to fulfill its protocol duties. The datalink timeout could therefore be safely set to a much shorter time—less than a second. The other change made was to adjust the transport "send packet" timeout to slightly more than the datalink timeout. This ensures that transport layer will always be able to send a packet through the lower layers.

Another open question is how to verify that our implementation of the published protocols really conforms to the specification. The proof will have to wait until we have an opportunity to interconnect our network with others which claim to conform to the same specifications. We have no automatic way (like the NBS testbed) to verify either the design or the implementation.

C. Performance

The implementation of a network following the ISO model has each layer (starting with 7, the application) passing data and some form of control information to the layer below. Each layer does whatever housekeeping is necessary before passing (possibly translated) data and control information to the next layer down. At the transport layer, the message is broken into packets and encapsulated with header information that uniquely identifies that packet to the corresponding layer in the receiving machine. This is usually done by prepending a header to the packet. The software in the receiving machine passes the packet back up through the layers. Each level strips off the header and sends up only the data that were given to the corresponding layer in the source machine.

A layered system like this, when implemented naively, can be very inefficient if the data are copied each time a layer encapsulates or decapsulates information from neighboring layers. Our software and hardware has been designed to avoid unnecessary data movement. For example, messages containing information to be written to disk by the file server are processed by levels 1–5 but the disk data are moved only once, from the network buffer to the file server's disk controller buffer.

As in most real systems, a few "cheats" were made. To prevent performance degradation, pointers to data are passed through the levels, rather than copying data several times. Also, certain levels are allowed access to data "belonging" to other levels. As an example, the transport layer looks at the length field of the network level header as a quick check of whether the incoming packet was large enough to contain a full level 4 header. This could be considered a violation of the separation of layers since truly separate layers would not share memory, except through interface parameters.

Another attempt at reducing the number of system packets transmitted led to allowing the session layer to be aware of the existence of transport layer acknowledgments, something that should be transparent to a client of the transport layer. Since the session layer knew the direction of traffic flow in a transaction, it was able to determine whether or not a piggy-backed acknowledgment was possible and could inform the transport layer with each call to SENDMESSG, whether or not the send ack bit in the transport header should be set. Our later discovery of the

correct way for the transport layer to respond to the send acknowledgment bit obviates the need for this "cheat."

We have also found that the data transfer rate between applications on the network is largely independent of the raw bit rate of the physical layer. In general, local area networks are able to transfer data at an effective bandwidth of 20–50 kbits/s (see, for instance, [11]). The Plan 4000 moves about 32 kbits/s during an average virtual disk data transfer (from a file server disk across the network into the user station application). The greatest constraints on throughput seem to be the efficiencies of the various network software layers, along with the ability of the server to respond quickly to incoming service requests. The implication is that a 10 Mbyte/s medium, though four times as fast as that of Plan 4000, would not give a factor of four improvement in throughput.

V. Summary

We have implemented a new local area network following the ISO Open Systems Interconnection model, using Datapoint's Arcnet for the data transmission levels, Xerox Internet protocols for the network and transport levels, and existing Nestar server software and application programs for the highest levels. Using a layered approach to network design, both during the design and implementation phases, and in subsequent stages of network "tuning," allowed the implementation to occur in a relatively short period of time for a project of this magnitude. It also gave us the ability to choose an optimum implementation for each layer, based on its individual merits, rather than on its membership in a particular technology. We found that although the Xerox XNS internet protocols were well specified, certain subtle aspects were only understood after its implementation.

Appendix A
Description of Arcnet Protocols

Logically, Arcnet appears to the user (the datalink layer) as a token-passing ring. One token is passed between stations and only the station that has the token may transmit. When a station receives the token, if it has nothing to transmit it passes the token on to the next station in the "ring." If it has something to transmit to a particular station on the network (rather than a "broadcast" packet which is sent to all stations on the network) a "free-buffer enquiry" handshake is used to ensure that the receiver has sufficient buffer space for the packet. If it does, or the packet is for broadcast, it is transmitted at this point. The receiver will return an acknowledgment if the packet is received correctly. The datalink layer is informed of the success or failure of the packet transmission and the token is passed on.

To connect to the network, each station has one or more network interface cards (NIC). A VLSI circuit, called a resource interface module (RIM), implements the Arcnet protocol. The part was designed by Datapoint using their experience with the discrete RIM's used in most of their existing installations. The RIM manages a 2 kbyte RAM buffer in which up to four packages may be stored (either to be sent or that have been received). The RIM is also given a station id (a number from 1 to 255), which must be unique on the net, which it uses to identify itself to other stations, to implement the logical ring, and to identify messages intended for it that come from other stations.

All the algorithms associated with token passing and packet transmission are handled by the RIM. Lost or destroyed tokens are automatically regenerated by a distributed algorithm. Reconfiguration, as workstations join the network, is done transparently and efficiently. When a new station joins the network it is not part of the logical ring and does not receive the token. After 840 ms it times out and sends a reconfigure burst which destroys the token. All other stations on the network detect the loss of the token and start a timeout based on station number. The timeout value is (146 μs) (255—station number), so that the highest station number on the net will timeout first. If any network activity is detected during the timeout period, then some other station has timed out and will generate a new token. If the timeout expires at a station, then that station creates a new token and sends it to the next higher address (mod 256). If there is no response in 74.7 μs, the station increments the target station number and tries again until a station accepts the token. The original station remembers that address as the next station number in the logical ring. The second station continues the operation, looking for the next higher station on the net. This is continued until all 255 possible id's have been tried ("0" is used for a broadcast packet, so it is not in the set of legal station numbers). The time required to reconfigure the network depends on the number of nodes, the propagation delay between nodes, and the highest station number on the net. It falls in the range of 24–61 ms. Other than the destruction of the token there are no packet collisions, even during reconfiguration.

When a station is powered off there will be no response to the token holder's attempt to pass the token. Rather than cause a reconfiguration, the token holder will timeout on passing the token and send the token to successively higher numbered stations until the next station in the ring is found. That station number is recorded internally as the logical "next" station to which the token is to be passed.

There is network flow control at the datalink level through the free-buffer enquiry and acknowledgment protocols. These prevent wasted cable bandwidth on collisions and lost packets when the target station is not listening, is not there, or does not have buffer space. Unlike Ethernet, the protocol is efficient when efficiency is important—during periods of heavy traffic [12]. Also, damaged packets are detected immediately using the CRC, and a negative acknowledgment (NAK) is sent back to the sender. The sender does not have to wait for a software timeout to determine whether retransmission is required.

Network interface cards are connected to the network through RG62 coaxial cable (the same as used for IBM 3270 display units). Rather than having all stations tap into one cable (as in common-wire networks like Ethernet), each station connects through a port on a line isolation

device (LID). The LID provides an electrically-ideal connection to the rest of the network, reducing or eliminating problems caused by a physical tap into the cable. In addition, it suppresses reflections from unterminated lines, allowing any station to be detached from the network either at the station or the LID. LID's are available with 4–16 ports and may be cascaded (up to ten LID's in the path between any two stations).

The LID also helps isolate hardware faults. Shorts or opens in a cable affect only stations that must transmit through that cable segment, leaving the rest of the network operational. There is a light for each station on the LID that is lit when the station is circulating the token normally. Failing units are easily detected by observing the lights and each can be disconnected at a few central points (the LID's). This eliminates the need for crawling around in the ceiling or visiting each office. In addition, the network is transformer isolated which limits ground-loop problems.

The raw transmission speed is 2.5 Mbits/s. Passing the token through an idle station takes 31 μs, and transmitting a packet takes 129.6 μs plus 4.4 μs/byte of data. A 256 byte packet then, takes 1256 μs, giving an effective throughput of 204 000 bytes/s (or 1.63 Mbits/s).

Appendix B
Pascal Description of Network and Transport Layer Entry Points

```
{*** NETWORK LAYER ***}
TYPE
    address     =   ↑byte;
    netid       =   fourbytes;
    hostid      =   sixbytes;
    socketno    =   twobytes;
    socketaddr  =   record
                      (network  :  netid;
                       host     :  hostid;
                       socket   :  socketno);
    end;

    connectno   =   twobytes;  {Level 4 types}
    headerptr   =   ↑headdesc;
    headdesc    =   packed record

      {*** Level 3 (network) header ***}

        chksum      :   twobytes;   {Checksum of packet (-1 = not used)}
        length3     :   integer;    {Length of packet.}
        notused     :   nibble;
        hopcount    :   nibble;     {How many stations to pass through}
        ptype       :   byte;       {Packet type (SPP, etc.)}
        destin      :   socketaddr; {Ethernet address of receiver}
        source      :   socketaddr; {Ethernet address of sender}

      {*** Level 4 (transport) header ***}

        control     :   byte;       {Control bits}
        packettype  :   byte;       {what type of level 4 packet}
        srcid       :   connectno;  {Sender's connection number}
        destid      :   connectno;  {Receiver's connection number}
        sequence    :   integer;    {Sequence number for this packet}
        ackno       :   integer;    {Seq num of last packet received}
        allocno     :   integer;    {Seq num of packet we can accept to}

  end;

FUNCTION listen (socket : socketno) : boolean;
  {Condition level 3 to accept incoming packets on this socket (May be a well known socket or may be allocated)}

FUNCTION allosocket (VAR socket : socketno; migrate : boolean) : boolean;
  {Allocate a floating socket and return the number of the socket in 'socket', with allosocket returning TRUE. If
```

allosocket returns FALSE, then there are none available. If a socket is allocated, a listen is started automatically. If migrate is TRUE, socket should be set to unmigrated socket (for consistency check) and packet socketno will be changed to the new socket.}

```
FUNCTION freesocket (socket : socketno) : boolean;
```
{Free the specified socket. If not allocated, returns false. Freeing a well known socket simply stops any listener set up for it.}

```
FUNCTION rcvready (VAR hdr : headerptr) : boolean;
```
{Return false if no packet is waiting (hdr is nil). Return TRUE if a packet is waiting, with hdr pointing to the header in the RIM buffer.}

```
PROCEDURE rcvblockmove (socket : socketno;
                        offset : integer;
                        bufadr : address;
                        length : integer);
```
{Move a block of data from the current packet for socket "socket" to the buffer pointed to by "bufadr". Start move at 'offset' bytes and move 'length' bytes.}

```
PROCEDURE rcvrelease (socket : socketno);
```
{Release the current packet and listen for another.}

```
FUNCTION sendpkt (socket : socketno;
                  hdr : headerptr;
                  data : address;
                  datalen : integer) : boolean;
```
{Send a packet through socket 'socket'. 'Hdr' is a pointer to the level 3/level 4 header with the level 4 and the level 3 destin and ptype fields filled in. Data points to the level 5 data to be sent, with datalen the number of bytes of level 5 data to send. Level 3 copies all of the data to the appropriate buffer and sends it out. If sendpkt returns false, no buffer is available}

```
{*** TRANSPORT LAYER ***}
TYPE

    transfer_status = (                {replies to send/rcv functions}
                tf_ok,                 {stable state}
                tf_unknown,            {transaction/connection ID unknown}
                tf_in_prog,            {status OK but unfinished}
                tf_nosockets,          {no free sockets}
                tf_hostfail,           {conn abend due to host loss}
                tf_overflow,           {input buffer overflow}
                tf_error               {some sort of protocol error}
               );

FUNCTION newlistener (socket : integer) : boolean;
```
{Initializes for reception on socket "socket." Return TRUE if socket was free.}

```
FUNCTION openreceived (     socket : integer;
                       VAR conid : integer;
                       VAR caller : socketaddr) : boolean;
```
{Checks for incoming connection on "socket." If there is one, return TRUE, allocate and return a conid, and the caller's socketaddr. If the connect packet was also a data packet, do not move it anywhere yet.}

```
FUNCTION connect (     destination: socketaddr;
                       msgtype:     byte;
                       msgptr:      address;
                       msglen:      integer;
                   VAR conid:       integer;
                   VAR status:      transfer_status) : boolean;
```
{Send an open-connection packet, which is also the first packet of the specified message if msglen > = 0. Return the connection id to be used for subsequent identification of this connection. Transmission of the message proceeds asynchronously as for sendmsg. The connection is not necessarily known to be established until a packet is received from the destination.}

```
FUNCTION accept (conid : integer) : transfer_status;
```
{Accept the new connection by sending a system packet which contains an ack for the opening packet and migrates the socket to a newly allocated socket. This call is optional, and should be used when the connect packet is not a complete message, or the first sendmessg for this connection will not be for awhile yet.}

```
FUNCTION accepted ( conid: integer;
                    VAR status: transfer_status): : boolean;
```
{This is an inquiry procedure which indicates whether a previous connect has yet been accepted by the destination. Another equivalent indication of an accepted connection is receipt of a message from the destination.
This function may be used to get early indication of successful connection, since it will be TRUE after the first packet has been received.}

```
FUNCTION rcvprepare (conid:   integer;
                     bufaddr: address;
                     bufsize: integer) : transfer_status;
```
{Supplies the buffer to be used for the next incoming message. Once a buffer has been supplied, calls to any level 4 routines are free to add to the message as the packets are received. If a previously received message has not yet been acknowledged, send a system packet acknowledging it.}

```
FUNCTION rcvmessg ( conid:   integer;
                    VAR msgtype: byte;
                    VAR msglen:  integer;
                    VAR status:  transfer_status
                  ) : boolean;
```
{Check if a complete message has been received in the supplied message buffer. If so, return TRUE, and return the message type and actual length. Record the number of the highest packet which must now be acknowledged.
If an outgoing message has not been completely sent, this call (as well as all others) will continue the progression of outgoing packets as data-link-level resources are available.}

```
FUNCTION acknow (conid: integer) : transfer_status;
```
{If a previously received message has not yet been acknowledged, send a system packet with an ack.}

```
PROCEDURE sendmessg ( conid:      integer;
                      msgtype:    byte;
                      msgptr:     address;
                      msglen:     integer;
                      requestack: boolean;
                  VAR status:     transfer_status);
```
{If any previous outgoing message is not completely sent and acknowledged, then complete the message and wait for an acknowledgment. This is the only condition under which these subset level 4 routines block the caller. If the caller wants to be nonblocked, he can guarantee that the previous message has been sent and acknowledged by calling l4_status.
Start sending the specified message. Send a piggy back acknowledgment for all received packets in all outgoing packets. Remember the parameter information so that the message can be resent if necessary. If ACKREQUEST is TRUE, set the ACK_REQ bit on the last packet of the message.
The first message sent after an "openreceived" is equivalent to the "accept" procedure of the full level 4, and causes the socket to change to a newly allocated socket, thus freeing the listening socket for new connections.
The buffer may not be reused by the caller until the message has been sent and acknowledged. This can be guaranteed to have happened by any of the following:

1. repeat until L4_status = tf_ok
2. sendmessg for the next message
3. disconnect}

```
FUNCTION l4_status (conid : integer) : transfer_status;
```
{Return the status of the connection and last outgoing message. This can be used as the "idling" procedure to pass periodic control to level 4 while waiting for a message to be sent. If the message has been sent but not acknowledged after a suitable delay, it will be retransmitted by level 4.}

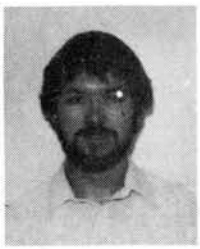
```
FUNCTION disconnect (conid : integer; abort : boolean) : transfer_status;
```
{Terminate the connection. If there is an unacknowledged outgoing message, and "abort" is FALSE, wait for the ACK and retry as necessary. If unacknowledged packets have been received, send a system packet acknowledging everything received.}

ACKNOWLEDGMENT

The authors wish to thank H. Saal, L. Shustek and the referees for their constructive criticism of drafts of this paper. Thanks also go to J. Thagard for doing the artwork.

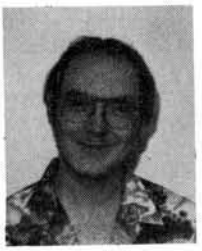
REFERENCES

[1] H. Zimmermann, "OSI reference model—The ISO model of architecture for open system interconnection," *IEEE Trans. Commun.*, vol. COM-28, pp. 425–432, Apr. 1980.
[2] W. P. Pearson *et al.*, "3-in-1 local network links personal computers," *Electronics*, Dec. 29, 1982.
[3] E. P. Stritter and L. J. Shustek, "Local network links personal computers in a multi-user, multi-function system," *Electronics*, June 16, 1981.
[4] J. A. Murphy, "Token-passing protocol boosts throughput in local networks," *Electronics*, Sept. 8, 1982.
[5] *Internet Transport Protocols*, Tech. Manual, Xerox Corp., Dec. 1981.
[6] *DOD Standard Transmission Control Protocol*, U. S. Dep. Commerce, Nat. Tech. Inform. Service, Publ. AD-A082 609.
[7] *DOD Standard Internet Control Protocol*, U. S. Dep. Commerce, Nat. Tech. Inform. Service, Publ. AD-A079 730.
[8] The Ethernet: A Local Area Network, *Data Link Layer and Physical Layer Specifications*, Version 1.0, Tech. Manual, DEC-Intel-Xerox, Sept. 1980.
[9] R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed packet switching for local computer networks," *Commun. ACM*, vol. 19, pp. 395–404, July 1976.
[10] J. F. Schoch and J. A. Hupp, "Measured performance of an Ethernet local network," Palo Alto Res. Center, Xerox Corp., Feb. 1980.
[11] J. G. Mitchell and J. Dion, "A comparison of two network-based file servers," *Commun. ACM*, vol. 25, pp. 233–245, Apr. 1982.
[12] C. K. Miller and D. M. Thompson, "Making a case for token passing in local networks," *Data Commun.*, pp. 79–88, Mar. 1983.



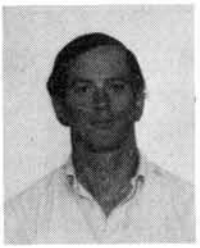
Gary M. Ellis (S'72–M'77) received the B.S. degree in electrical engineering from the University of Washington, Seattle, in 1977.

He worked on in-flight software verification at the Boeing Company and operating system implementation at Advanced Micro Computers. He has been with Nestar Systems, Inc., Palo Alto, CA, since January 1982 where he has been involved in networking protocols and file server design and implementation.



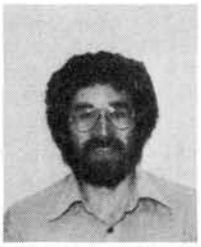
Sam Dillon was born in Ft. Hood, TX, on December 3, 1955. He received the B.S. degree in biochemistry from the University of California at Riverside, Riverside, CA, in June 1978.

In September 1978 he joined the Burroughs Corporation and worked in production of mainframe peripherals. In 1979 he moved to the Research and Development Group, Magnuson Computer Systems, to work on the design of a cache memory for a 4300 compatible CPU. In February 1980 he joined the Software Development Group, Advanced Micro Computers, first writing diagnostics and then porting the Thoth operating system to the Z8000. Currently, he is with Nestar Systems, Inc., Palo Alto, CA, where he has worked on network protocols, network device drivers for workstations, system backup software, and porting network utility software between different workstations.



Skip Stritter (M'77) received the B.A. degree in mathematics from Dartmouth College, Hanover, NH, in 1968, and the M.S. and Ph.D. degrees in computer science from Stanford University, Stanford CA, in 1969 and 1967, respectively.

He worked at Motorola on microprocessor architecture, as a Professor of Computer Science, University of Texas, Austin, TX, and as a member of the Technical Staff, Bell Laboratories. He now directs new product development at Nestar Systems, Inc., Palo Alto, CA. He has contributed to the design and managed the implementation of the local area network described in this paper.



Jerry Whitnell received the B.S. degree in computer science from the University of California at Santa Barbara, Santa Barbara, CA.

He joined Nestar Systems, Inc., Palo Alto, CA, in May 1982 to work on the Plan 4000 network. His interests include programming languages, software tools, computer architecture, and local area networks.

Mr. Whitnell is a member of the IEEE Computer Society.