ALAN: A (Circuit-Switched) Local Area Network

BRENT HAILPERN, SENIOR MEMBER, IEEE, ANDREW HELLER, LEE W. HOEVEL, MEMBER, IEEE, AND YANNICK J. THEFAINE

Abstract—Our view is that people (that is, programmers and general computer users) tend to work together in small groups, where information and resources are shared freely within a group. Most interactions occur within the group (called a *cluster*)—sending messages, exchanging data, sharing a printer. Communication outside the cluster—sending mail to someone in another group or using a large number cruncher—is comparatively rare. Under this hypothesis, it is advantageous to optimize and simplify interactions within the group. This paper describes our attempt to design a cluster network based on a nonblocking crosspoint switch, which we call *ALAN* (*a* local *a*rea *n*et). ALAN clusters are star-connected—with intelligent workstation nodes (PC's) at the points of the star, and the ALAN switch at the center.

The Cluster Concept

We call the set of computing resources owned by a given group a *cluster*. A cluster might consist of, for example, several workstations, a print server, a file server, and one or more gateways (connecting the cluster to other networks or mainframes). One would expect elements of the cluster to be close to each other (for example, within 2000 ft).

This notion bears the same relation to general computer networking as a local intercom service bears to the worldwide telephone network. Processes in a cluster communicate like people in an office communicate. To call someone inside the office, one or two buttons are pushed; to call outside the office, a person must first obtain an outside line and then dial the standard 7 or 10 digit phone number. Similarly, communication within a cluster uses a simple (1 byte) protocol, while communication beyond a cluster requires first connecting to a gateway, and then using a more elaborate (mainframe or network) protocol. Servers and shared devices in a cluster correspond to secretaries, clerks, or librarians shared by the members of an intercom group. Local interactions do not need the full power of the telephone system; a simple point-to-point connection suffices. See Fig. 1.

Comparison to Other Networks

Most existing local area networks are packet switched. That is, the network consists of a shared medium over which processors communicate. See [1]–[3] for surveys of network protocols and architectures. Some protocol establishes how the shared medium is divided among the conversations: time-sliced, token ring, CSMA, frequency multiplexed. The conversations consist of packets of information sent from one processor to another. Because the medium is shared, source and destination information must be included with each packet to avoid confusion. On top of this packet-layer protocol, most networks implement a virtual circuit switch. This layer gives the illusion of point-to-point conversations with no interfering packets.

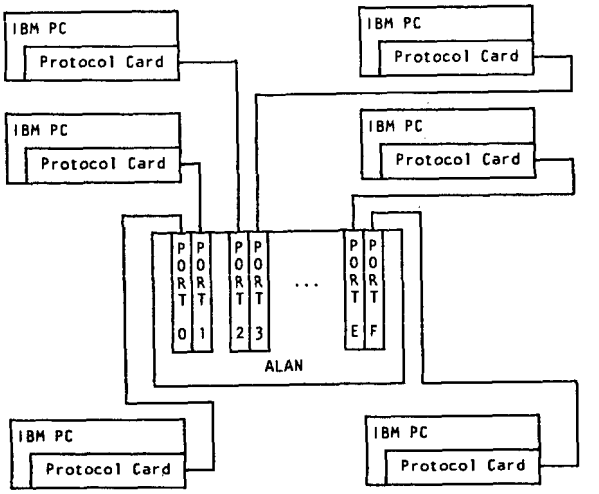


Fig. 1. ALAN's star topology.

In contrast, ALAN is a physical circuit switch. At the beginning of a conversation the initiating process *dials* the receiver. If the receiver is not busy, then a conversation is established and a point-to-point connection is made. No other processor can interfere until the conversation is ended. Attempting to dial a processor involved in a conversation results in a *busy signal.*

ALAN's protocols are simpler than most other network's protocols for two reasons. First, ALAN provides a circuit-switched environment that most other networks simulate on top of packet switching. Second, communication within the cluster does not require all the overhead of general network communication.

Most networks use hierarchical protocols; that is, each protocol layer is built on services provided by the next

Manuscript received November 20, 1984; revised January 30, 1985.
The authors are with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

0733-8716/85/0500-0427$01.00 ©1985 IEEE

lower layer. Local area networks, in particular, are conventionally built on a packet-switched foundation, in which each message between users requires source, destination, error detection, and sequencing information. Because packets can arrive from many senders at the same receiver, each computer attached to the network must be able to handle many simultaneous conversations. On top of this low level, additional protocols implement a virtual circuit-switched network, giving the illusion of point-to-point connection.

The ALAN switch provides a circuit-switched environment. This eliminates the need for those conventional layers that implement packet-switching, or use packet-switching to implement circuit-switching. If some applications require packet-switching, this environment is easily implemented on ALAN by a series of very short communications.

Most networks require fully general communication protocols. That is, the same conventions and protocols are used whether communicating with a computer in the same room, in the same building, or across the country. In particular, they require the full error detection, ACK's (acknowledgments) and NACK's (negative acknowledgments) required for TP lines even if the connection is a short cable. In the same way, fully general addresses are used even if there are only a few computers on the network.

ALAN allows communication to be optimized within a cluster. Error detection is simple because all connections are short. Addresses are simple because there are only 16 ports on the switch. When communication outside the cluster is necessary, the gateways contain the information and protocols required for more general communication. As a result, most of the computers on the switch need allocate much fewer resources to service the ALAN network than would to service a token ring or broadcast network.

Design Considerations

The cluster concept, and our available manpower, determined several key design constraints. We neither needed nor aspired to a worldwide network based on ALAN, and so were able to exploit small address widths and minimal addressing overhead. Since clusters are assumed to be physically local, we could assume a generally error-free transmission medium. Therefore, error detection and correction were not architected at each step and at each layer of the protocol. We require the end-to-end protocol to be correct, and provide mechanisms for it to detect errors and initiate recovery. The lower levels would assume that messages were transmitted correctly.

The small size of our group dictated that we could not rewrite the PC operating system to convert it into a multiprocessing system. By living within this constraint, we found that even a single-thread operating system could be used in a distributed environment by devoting an independent processor to protocol management. The underlying serial operating system, together with ALAN's "one-at-a-time" connection mechanism, forced us to simplify many aspects of server implementation. Indeed, our prototype print server was implemented in less than one person-month.

We initially planned our system with conventional hierarchical protocols using full SDLC as the transmission layer. Each connection in a conversation had its complement of error detection and acknowledgments to maintain "reliability." The error detection, it turned out, cleaned up no errors and caused many. In particular, lost ACK's would cause otherwise good connections to be closed. Why did we lose ACK's? Because each stage was required to ACK each message. Therefore, each sender had to immediately turn around and become a receiver and vice versa. If the sender to receiver turn around time was even close to the receiver to sender turn around time then errors crept in and ACK's were lost.

We began eliminating "unnecessary" ACK's, reducing the size of the protocol (figured in terms of the number of states in a finite-state machine). At each step the system became more reliable and the protocol became easier to analyze (and to prove correct). We finally realized that no intermediate ACK's were strictly necessary. If the end-to-end protocol gets all the data, then who cares what the intermediate levels said? If the intermediate levels detect an error and it is significant, then the end-to-end will also detect that error through check summing and sequence numbers. The reason for intermediate ACK's is then not correctness, but efficiency: if each level can repair an error or if the cost of completing a bad transmission is too high, then you want to detect that error as soon as possible. But if your transmission medium is usually reliable and you do not mind occasional retransmissions, then you save the considerable cost that the intermediate ACK's incur.

The resulting protocol, described in the next section, brings up some interesting issues in terms of specification and verification. What are the safety properties of a system that will tolerate some errors up to a point when no one process provides a "complete" service (for example, establishing a connection and guaranteeing that the connection exists)? What are the liveness properties of a system that (for a while) permits cooperating process to hold conflicting views of the world state (for example, A thinks the link is open and B thinks it is closed)? We are just coming to grips with these issues.

Specification and Implementation

A network conversation consists of six processors: the 2 PC 8088's (application program), the 2 Protocol Card 8086's, and the 2 Port 8749's. We provide connection establishment within the switch, connection establishment and maintenance from Protocol Card to Protocol Card, and connection establishment and message passing from application to application.

We model the protocol machine within the Port as having two states: neutral and connected. In the neutral state, the processor polls for an open request from the

attached PC or from the net. In the connected state, the processor "eavesdrops" on the PC's conversation, waiting for a close command (see Fig. 2).

The protocol in the Protocol Card has three states: closed, half-open, and open (see Fig. 3). In the closed state an open request from the application causes a open command to the Port and the state machine to become half-open. Alternatively, an open request from the Port causes the state machine to become open and an ACK to be sent. In the half-open state, an ACK from the net causes the state machine to become open; a timeout causes the state machine to become closed, with a close command sent to the port. In the open state, a close from either side causes a close to be sent to the port and the state machine to return to the closed state.

The protocol is slightly complicated to handle debugging information and line failure during a conversation. But we have found that our assumption of a highly reliable transmission medium is justified, and a minimal number of ACK's can be used.

The lowest level transmission protocol is mini-SDLC. We use only the framing bits and zero bit insertion of the SDLC chip. Each packet has a one byte header, specifying either function and address (up to 32 ports) or just function. Since we use a physical circuit switch, only open commands need to contain addresses.

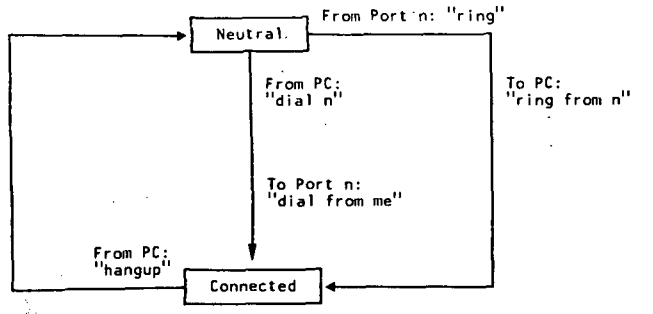


Fig. 2. Port card finite-state machine.

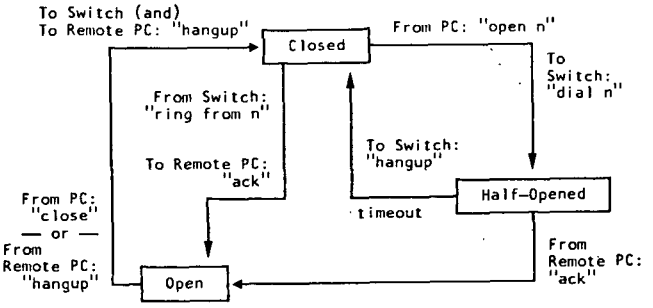


Fig. 3. Protocol card finite-state machine.

ALAN Prototype

The ALAN switch is a nonblocking, 16 port, cross-bar switch. It therefore supports up to eight concurrent data conversations. The conversations are *space-multiplexed*, in that each port has a transmit line that all other ports can receive. The receiving port selects which incoming line to listen to. Connection establishment is also space-multiplexed.

The ALAN switch is housed in a PC expansion chassis containing a power supply and eight *Switch Cards*, each of which supports two independent *Ports*. Made of off-the-shelf components, each Port contains a serializer/deserializer and a dedicated microprocessor to control connection establishment.

Each workstation (PC or PC-XT) attached to the switch contains a *Protocol Card* connected by cable to a unique Port in the switch. Communication between the PC and the Protocol Card is serial through I/O ports (for control) and DMA block transfers (for data).

A typical communication session begins with a 1 byte message from the PC to its Port, requesting a line to some other port. If the requested Port is not engaged, then the line is opened and the PC's can communicate directly. Both logically and electrically, the two PC's are in point-to-point communication. The Ports monitor, but do not interfere with, the conversation to recognize the in-band close command. The maximum aggregate bandwidth of a cluster is half the number of switch Ports times the per Port communication rate.

In token passing and broadcast networks, each Port (or network connector) must be able to function at the maximum speed of the entire network. This requires leading-edge technologies for those ports to provide speeds of 10 Mbits/s or greater. ALAN, on the other hand, uses existing, reliable, and inexpensive technology (Intel 8749 microcontrollers and 8274 protocol chips) to provide comparable service. The Intel 8274 is rated at 1 Mbit/s (full-duplex) and our prototype has 16 ports; hence the prototype has an 8 Mbit/s aggregate bandwidth. By increasing the number of Ports on the switch, ALAN's bandwidth can be increased without changing technologies.

Software Interface

One initial design decision was that we did not want to rewrite PC-DOS to be able to support the Protocol Card. We decided to interface our hardware to PC-DOS 2.0 through a device driver that makes the Protocol Card look like a virtual disk; the standard DOS "disk write" interface is used to send parameters to the Protocol Card. Among these parameters is the address of a buffer containing the actual command being issued (e.g., "open," "send message," "receive message," and "close"), together with any command-dependent data to be transferred to the Prototype Card. Information is returned to a PC application by transferring data (via DMA protocol) back into the same buffer used to define the Protocol Card command. In other words, all operations are disk writes, with the write buffer used for passing data in both directions.

An interesting alternative would be to more closely emulate a virtual disk interface. That is, opening a file would map into opening a connection, writing a record

would become sending a message, and so on. This alternative should provide a more natural programming paradigm, but may lead to a more complicated application interface.

Application programs use a procedure-call interface to the device driver that hides the underlying virtual disk implementation. The procedure is passed a pointer to a record containing an operation code, buffer segment, and buffer offset. The procedure takes care of creating the virtual file and performing the virtual output. We have used this interface from Pascal, C, and Assembler.

Conclusion

The most significant advantages of ALAN derive from our decision to limit its size. The software advantage obtained by reverting to point-to-point protocols based on a real circuit switch dramatically cut both programming effort and complexity. The star topology and distributed control have distinct advantages in reliability, availability, and maintainability. Almost all errors in code or cabling are quickly isolated to a particular PC node, cable, or Switch Port.

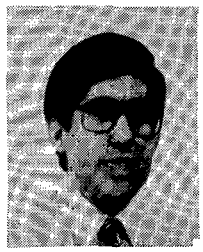
The most important result of the ALAN experiment was a renewed appreciation for the value and necessity of questioning accepted tenets in light of changing technology. Any "conventional wisdom" is developed in a particular context of cost and performance; when this environment changes, new (or old) solutions become feasible. When these new solutions are, in fact, optimal, the conventional wisdom must change. In our case, the conventional wisdom advocated shared-media LAN's with hierarchical protocol incorporating error-detection/correction at each layer. This wisdom arose in an environment where the interface to the physical medium had to be built up from discrete components, and often involved communication over noisy telephone lines or costly dedicated cables. Our design, in contrast, is based on: cheap, readily available microprocessors and multiprotocol chips; noise-free transmission over low-cost dedicated cable; and the emergence of inexpensive intelligent workstations. Thus, for the cluster environment, we developed a physical circuit-switch network out of off-the-shelf components that is competitive in cost and performance with current shared-media networks.

References

[1] P. E. Green, Jr., Ed., *Computer Network Architectures and Protocols*. New York: Plenum, 1982.

[2] J. F. Shoch, Y. K. Dalal, D. D. Redell, and R. C. Crane, "Evolution of Ethernet local computer network," *IEEE Computer*, vol. 15, pp. 10–28, Aug. 1982.

[3] A. S. Tanenbaum, "Network protocols," *ACM Comput. Surveys*, vol. 13, no. 4, pp. 453–489, Dec. 1981.



Brent Hailpern (S'80–M'80–SM'84) was born in Denver, CO, on January 11, 1955. He received the B.S. degree in mathematics from the University of Denver, Denver, CO, in 1976, and the M.S. and Ph.D. degrees in computer science from Stanford University, Stanford, CA, in 1978 and 1980, respectively.

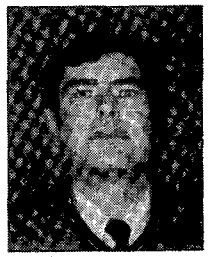
He has been employed as a Research Staff Member in the Computer Sciences Department of the IBM Thomas J. Watson Research Center, working in the area of workstation languages and protocols. His research interests include concurrent programming, program verification, network protocols, and distributed systems.

Dr. Hailpern is a member of the Association for Computing Machinery, Pi Mu Epsilon, and Sigma Xi.

Andrew Heller is responsible for advanced system technology projects across IBM's Research and Development Groups. He was appointed in 1975, and is currently Director of Advanced Technology Systems for the Information Systems and Technology Group and the Information Systems and Communication Group.

Lee W. Hoevel (S'74–M'77) was born in Kansas City, MO, on January 22, 1946. He received the B.A. degree in mathematics and economics from Rice University, Houston, TX, in June 1968, and the Ph.D. degree in electrical engineering from the Johns Hopkins University, Baltimore, MD, in 1979.

In June 1968, he joined the staff of The Johns Hopkins Applied Physics Laboratory, Silver Spring, MD. He took a leave of absence in the fall of 1971 to become a full-time graduate student of The Johns Hopkins University. He was a Research Assistant with the Department of Electrical Engineering, Stanford University, Stanford, CA, from 1975 to 1979. He has been a research staff member at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, since 1978. His current research interest include program environments, memory hierarchies, and computer architecture.



Yannick J. Thefaine was born in La Vallette-du-Var, France, on May 13, 1941. He received the B.S.E.E. degree from Union College, Schenectady, NY, and the M.S.E.E. from Syracuse University, Syracuse, NY, in 1975 and 1980, respectively.

He is currently with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, as an Engineer in Computer Science. He has been with IBM since 1978 and has been involved with designing a local area network since 1983.