MESSAGE COMMUNICATION PROTOCOL AND OPERATING SYSTEM DESIGN FOR THE DISTRIBUTED LOOP COMPUTER NETWORK (DLCN)

Ming T. Liu and Cecil C. Reames*
Computer and Information Science
The Ohio State University
Columbus, Ohio 43210

ABSTRACT

The Distributed Loop Computer Network (DLCN) is envisioned as a powerful, unified distributed computing system which interconnects midi/mini/micro-computers, terminals and other peripherals through careful integration of hardware, software and a loop communication network. Research concerning DLCN has concentrated on the loop communication network, message protocol and distributed network operating system. For the loop communication network, previous papers [2,3] reported a novel message transmission mechanism, its hardware implementation, and its superior performance verified by GPSS simulation. This paper presents an overview of the design requirements and implementation techniques for DLCN's message protocol and network operating system. Firstly, a bit-oriented distributed message communication protocol (DLMCP) which handles four message types under one common format is proposed. Besides user information transfer, this protocol supports automatic hardware-generated message acknowledgment, error detection and recovery, and network control and distributed operating system functions. Secondly, the network operating system (DLOS) is described which provides facilities for interprocess communication by process name, global process control and calling of remote programs, generalized data transfer, alterable multi-linked process control structures, distributed resource management, and logical I/O transmission in a distributed file system.

I. INTRODUCTION TO DLCN

The Distributed Loop Computer Network (DLCN) [1-5] is envisioned as a powerful distributed computing system which interconnects midi/mini/micro-computers, terminals and other peripheral devices through careful integration of hardware, software and a loop communication network (see Figure 1). The major goals of DLCN are to provide efficient, inexpensive, reliable, and flexible service to a localized community of semi-autonomous users in an environment of constantly changing user demands and requirements. The ARPANET [6] and similar national computer networks have demonstrated the importance and practicality of networking for interconnecting large-scale computer facilities; research on DLCN hopes to prove that networking is also feasible and extremely beneficial in a geographically local environment of small and medium scale machines. Such a computing environment is frequently found today in many industrial, commercial, and university areas and therefore seems to be an ideal target for research on this type of networking and distributed computing. Bringing to such groups the cost advantages and performance improvements of the computer systems architecture of the 1980's [7] will be a very important achievement.

For the purpose of this research, DLCN's stated goals can be broken down into five major design objectives:

1) to provide an efficient, low-cost loop communication network which can simultaneously and directly transmit variable-length messages between a variety of attached devices (computers, terminals, and peripherals) without the use of any centralized control;

2) to construct an inexpensive loop interface between each such attached device and the network which can automatically buffer, transmit, acknowledge, and control messages without the need for software supervision;

3) to design an appropriate message communication protocol for this network and interface which will facilitate their operations while also providing a vehicle for implementing many distributed, low-level primitives for network operating system functions;

4) to specify the design requirements and implementation techniques for a distributed network operating system which is to feature interprocess communication by name, remote program calling, alterable process control structures, distributed resource management, generalized data transfer, logical I/O transmission, and control of a distributed file system; and finally,

5) to integrate the results of the preceding four objectives into a Distributed Loop Computer Network (DLCN) which will give the network user access to a powerful, flexible and unified distributed computing system which he can use to meet his expanding processing needs without sacrificing his local autonomy.

There are many important keywords stressed in DLCN's design objectives, but possibly the most significant is integration -- integration of hardware, software and communcation technologies to produce a unified distributed computing system (see Figure 2). Many distributed functions in DLCN are made possible because of the close integration of the network's hardware, software and communication systems. The term distributed computing is used here in a rather special sense to denote the interconnection of independent computer systems, terminals and peripheral devices into a common, unified computing system whose total resources may be shared by all. Furthermore, it is hoped that the network can be constructed in such a manner that its users will see only a single, integrated computing facility with great power and many available resources. In the ideal case, the users would not even be aware of the system's actual organization and method of operation.

In order to accomplish all these design objectives for DLCN, many new techniques must be applied, and some new concepts have to be developed. Previous DLCN research concentrated on the communication network and on hardware design of a loop interface which implements a superior message transmission mechanism [1,2,4]. The functioning of the communication network was also modeled [3], and the advantages of the new transmission mechanism were verified through computer simulation using GPSS. These results are summarized in Section II.

Section III presents the message protocol to be used by DLCN (called DLMCP, for Distributed Loop Message Communication Protocol). It is a bit-oriented message protocol with a distributed control discipline and is specially designed to exploit the unique properties of a loop network. Four message types are supported by DLMCP under one common format: information, acknowledgment, control and diagnostic.

*Currently in data communications development with the Burroughs Corporation, 25725 Geronimo Rd., Mission Viejo, California, 92675.

Finally, Section IV provides a brief overview of the design requirements and implementation techniques for the network operating system, called DLOS (for Distributed Loop Operating System), which must transform a loosely coupled collection of semi-autonomous computing systems into a distributed computing system. As may be guessed, the latter task is far from an easy one.

II. SUMMARY OF PREVIOUS RESEARCH ON DLCN

As mentioned above, previous research on DLCN has been mainly in the area of communication network and hardware design. In this regard, a new message transmission mechanism has been developed for use in DLCN [1,2,4] which is faster and more efficient (multiple messages, variable length) than that used in Newhall loops (single message, variable length) [8,9,10] or Pierce loops (multiple messages, fixed length) [11,12]. The mechanism allows the simultaneous and direct transmission of variable-length messages onto the loop without the use of any centralized control. By buffering incoming messages when necessary, it guarantees nearly immediate access to the loop for infrequent users, regardless of the level of message traffic on the loop. The mechanism also provides automatic regulation of the rate of message transmission in accordance with observed system load, favoring light, infrequent users at the possible expense of heavy, frequent users. Perhaps most importantly, it yields shorter message transmission times and makes much better utilization of the loop than any existing technique.

The detailed operation of this transmission mechanism (see Figure 3) has been reported elsewhere [1-5] already, so only a brief summary will be given here. Basically, a message can be placed onto the loop whenever no other message transmission is already in progress, provided that space exists locally (of a size at least as large as the message to be inserted) to delay any incoming messages that might arrive. Once an incoming message has been delayed in this manner, it is then transmitted ahead of any more incoming messages (which are in turn delayed, etc.). During the period when no incoming messages arrive, the space occupied by the delayed messages can be gradually reduced and finally eliminated.

Inexpensive shift-register insertion hardware has been designed which allows this message transmission mechanism to be implemented totally in the loop interface [3,4]. This interface hardware has several highly significant features. It can transmit a message to a specific process executing on a particular machine, or it can broadcast the message to all active processes in one machine or in the entire network. In addition, it can automatically generate and transmit the proper acknowledgment response for any information message it receives (if so directed), can detect and remove lost messages, can prevent lockout of a node from accessing the loop, and can do all of these functions without any software supervision. A description of the hardware needed to accomplish these functions has been published elsewhere [3,4,5] and therefore will not be repeated again here.

The superior performance of DLCN's message transmission mechanism was verified by an extensive computer simulation study [3,5]. Simulation models were written in GPSS for three types of loop networks - DLCN, Pierce, and Newhall - so that relative performance could be easily judged. As just one result of this study, Figure 4 graphs mean message transmission time in each of the three networks vs. message traffic rate and clearly shows the better performance of DLCN's transmission mechanism. This fact is not surprising, since the DLCN mechanism, when constrained appropriately, can be reduced to operate like either the Newhall or the Pierce mechanism. By exploiting their weaknesses while retaining their advantages, the DLCN mechanism is able to be better than either.

III. MESSAGE COMMUNICATION PROTOCOL

The communication protocol used by DLCN for message transmission on the loop is called the Distributed Loop Message Communication Protocol (DLMCP) [4,5]. It is a bit-oriented protocol with a distributed control discipline. It is similar in many respects to the modern, bit-oriented protocols now being developed, like IBM's SDLC, ANSI's ADCCP or ISO's HDLC, which all use a centralized control discipline. One important feature of DLMCP is that messages are addressed specifically to user processes executing on particular machines in the network, thus facilitating interprocess communication by process name. Another significant feature of DLMCP is that four types of messages are provided under one common format; besides user information transfer, these messages support automatic hardware-generated acknowledgment, error detection and recovery, and network control and distributed operating system functions. Thus DLMCP is much more than just a data link control protocol, as it is designed with easy interprocess communication and network control in mind.

DLMCP's overall message format is shown in Figure 5. The flag is a special bit-sequence which denotes the start or end of a message frame. Either the bit-stuffing technique (as in SDLC) or the more efficient bipolar violation technique (as in the Newhall [8] loop) can be used to ensure that the flag sequence never occurs elsewhere in the bit stream so as to achieve data transparency. The two 12-bit address fields are each decomposed into 7-bit loop interface addresses and 5-bit process numbers for identifying the source and destination processes. The first two bits of the message control field are used to encode the message type -- information (00), acknowledgment (01), control (10) or diagnostic (11). The next bit allows for broadcast message transmission, whereby a message is copied by every interface on the loop but is not removed; this technique has considerable application in inquiry situations when the address of the desired receiver is not known. The following 3-bit subfield is interpreted differently for each message type (i.e., function code, response code, etc.). The next 3 bits are used for lost message detection and lockout prevention. The final 7-bit subfield of the control field also has different uses for different types of messages (i.e., sequence number, function modifiers, etc.). The final field of the message frame is the CRC-CCITT (Cyclic Redundancy Check) error checksum field in which a 16-bit checksum is stored by each message transmitter. Note that not only the user information field but also the address and control fields are checked by this technique, thus ensuring their integrity as well.

Information and Acknowledgment Messages

Little need be said about information messages. They exist solely to carry transparent, user-defined text between communicating processes. Their contents are completely ignored by the loop communication network.

Since information messages may not be received properly for a variety of reasons, it is necessary that each be acknowledged in some manner. In a loop network, the receiver can simply copy the message, set a field in it to indicate reception, and return the entire message to the transmitter (as in DCS [9]). However, because message frames in DLCN may be of arbitrarily great length, it is more efficient to

remove the information text at the receiver and to generate a very short acknowledgment message for return to the transmitter.

DLMCP implements two kinds of message acknowledgments. Each information message transmitted can be individually acknowledged, or only messages which require an exception response may elicit an acknowledgment, the rest being implicitly accepted through use of a single 7-bit sequence number. In either case, the interface hardware is so constructed that it can automatically generate the proper acknowledgment message (if one is required) from the information message received and can return the response to the sender, all "on the fly" and without the need of any software assistance [5].

The advantages of this method of message acknowledgment should be clear. First, acknowledgments are generated totally by the loop interface hardware, without host intervention or supervision, in the minimum possible time. Thus correct message transmission is the sole responsibility of the communication network. This method will work even when the attached host is not operational or in cases where there is no intelligent host (as when a peripheral device is attached directly to the loop). Second, the transmitter can regulate exactly how and when messages are acknowledged, either for each individual message or only when an exception is noted. By using a combination of these two methods, the transmitter can send a block of messages, only the last of which requests a definite response, thus implicitly acknowledging the entire block of messages with one return message. This method can be used as part of message flow control on the transmitting side; pacing on the receiving side is handled by another mechanism.

Control Messages

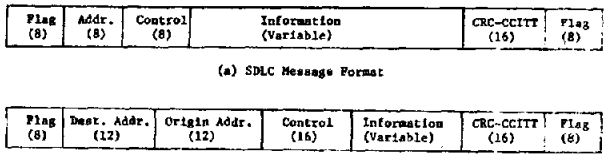
The control message, as its name implies, is used partly to control the operation of the loop communication network. Its primary purpose, however, is to implement privileged low-level primitives for accomplishing some of the basic functions of the Distributed Loop Operating System (DLOS). Table 1 lists various control message types and their control functions. Their use in the loop executive DLOS will be discussed in the next section.

Diagnostic Messages

Diagnostic messages are used for error detection and recovery during normal operations and for system initialization and bootstrap loading at start-up time. Four basic modes of diagnostic messages exist -- normal, diagnostic, recovery and initialization. Normal mode is the usual setting when no errors have been detected; periodic status checking is still performed, just to be certain that all components are working properly. Once an error is suspected, diagnostic mode is entered to pinpoint the cause of the problem. When the cause is located, recovery mode is entered to attempt to solve the problem and to continue despite it. Finally, initialization mode is the state entered when an interface is powered up for the first time or which follows reconfiguration. In this mode, not only the interface is initialized for operation, but even the host software can be bootstrap loaded from another source.

Lost Message Detection and Lockout Prevention

The lost message detection and the lockout prevention mechanisms are completely distributed and performed by all interfaces in the network on all messages. They were reported fully in [4,5] and will not be repeated here, due to lack of space.



(a) SDLC Message Format

(b) DLMCP Message Format

Address Field Specification:

Bit Positions	Specification
0 - 6	Loop interface address (0 - 127)
7 - 11	Process number (0 - 31)

Control Field Specification:

Bit Position(s)	Specification
0 - 1	Message type: 00 Information; 01 Acknowledgement; 10 Control; 11 Diagnostic
2	Broadcast message
3 - 5	Function, response or other (depending on message type)
6 - 7	Lost message detection
8	Lock-out prevention
9 - 15	Sequence number (0 - 127) or function modifier (depending on message type)

Fig. 5. Message Formats for (a) SDLC and (b) DLMCP

Group	Command	Subcommand	Function
0	PACE		Message flow control
	RESPONSE		Control Acknowledgment
1	LOCATE		Locate a process or program by name
	CALL		Call a remote program on another machine
	RETURN		Return from a remote program
2	PRIVILEGE		Establish process-to-process control structure
		Inquire	Find out which privileges are granted
		Request	Ask for a specified privilege
		Grant	Give a specified privilege
		Revoke	Take away a specified privilege
3	RESOURCE		Distributed resource management
		Test	Test availability of resource
		Obtain	Ask for a resource to be allocated
		Allocate	Allocate a resource to requester
		Deallocate	Deallocate a resource from holder
4	REGULATE		Unspecified system control/response
5	TRANSFER	Data	Generalized data/file transfer
		Parameter	Parameter passing for remote calls
6	SYNCHRONIZE		Explicit process synchronization
7	EXPANSION		Reserved for future expansion

Table 1. Control Messages

IV. DISTRIBUTED NETWORK OPERATING SYSTEM

The Distributed Loop Computer Network (DLCN) is envisioned as a loosely coupled, heterogeneous network, consisting of many different computer systems (both small and medium size), each operating for the most part independently and under local control, yet occasionally sharing data and programs with or borrowing resources from other computer systems in the network. In order to make such cooperative resource and data sharing possible in a uniform manner, it is necessary that all processors in the network execute components of a commonly structured Distributed Loop Operating System (DLOS). Each local component of DLOS is responsible for converting between local and network-wide representations for all forms of information (both user and system) that must be exchanged through the network. Thus the primary job of DLOS at the system design level is to define a standard, universal representation for all network information transfer; each local component then serves to map this universal representation to that of its particular host. By requiring only that the data structures and operating protocol be identical on all machines, each local

component of DLOS can be implemented differently, according to local requirements. It is hoped that DLOS can be added between existing local operating systems and the network with relatively minor changes.

Control of the network as a unified distributed computing system is made easier to achieve by the close cooperation of the message communication protocol, DLMCP, and the design of DLOS, since all the required low-level operating system functions are implemented using the interchange of control messages through the loop communication network. Having a loop for the communication network, which makes message routing, broadcast transmission and distributed control so easy, has had a profound effect on the structure and design of the network operating system.

Features of DLOS

To reiterate, the philosophy of DLOS is that DLCN, even though actually a collection of separate computer systems connected by communication links, should be viewed by its users as a single, unified distributed computing system. Consequently, all functions in DLOS have been implemented with this idea in mind. Interprocess communication is by process name, not address. Remote programs (procedures located in another machine) are "callable" by a global network process also by name, not by location, using a generalized parameter transfer mechanism. Similar capabilities are provided for alterable, multi-linked process control structures and for distributed resource management. Even generalized data transfer and logical I/O transmission at the file record level is possible with DLOS for a distributed file system formed from separate, heterogeneous file systems. Thus, by using the facilities specifically designed for it in the hardware and communication network, DLOS can give the distributed computing user the illusion of executing on a single, powerful machine with a great number of available resources.

Interprocess Communication

The message communication protocol used on the loop network (DLMCP) requires that physical addresses be given in every message to identify both the receiver and the sender, since messages (unless broadcast) are directed to a particular user process executing on a particular machine in the network. In order to accomplish this objective, every process address is broken down into two components: a 7-bit physical loop interface address (LIA) for locating the proper machine in the network and a 5-bit process number (PN) which identifies the process itself.

While this method of physical addressing was chosen for transmission efficiency, it is desirable that user processes be able to communicate with each other by logical process name, without having to know the actual physical locations of other processes. If interprocess communication by name is to be supported, while physical addressing of message frames is to be required for message transmission on the loop network, then DLOS must translate from process name to process address (and vice versa) for every message transmitted or received. This translation must be both dynamic (since the location of a process may change) and efficient. To see how this translation is done by DLOS, consider the following problem and its step-by-step explanation.

Suppose that process A in machine 1 wants to send a message to process B in machine 2. Process A does not know the physical location of process B (nor does it wish to know) and thus it will send its message to process B by name. The action of transmitting this message from process A to process B can be explained in four steps (see Figures 6, 7, and 8).

1) First, DLOS must set up a logical connection between the two named processes. A LOCATE control message is put on the loop in broadcast mode from machine 1, giving the address of process A and the names of processes A and B. Every machine on the network receives the LOCATE command and checks its local list of process names to see if process B is located there. If found, that machine (2 in this case) copies the physical address and logical name of process A into its Process Name-to-Address Translation Table (PNATT, see Figure 6b), then sends a LOCATE control message back to machine 1 to give it the physical address and logical name of process B. Machine 1 copies that information into its own PNATT (see Figure 6a). Should the location of either process ever change, a single LOCATE control message can cause the appropriate translation table to be updated.

2) Next, a communication (talk) path must be established between processes A and B. Thus process A must request permission to talk to process B by issuing a Request Talk Privilege command to process B (either explicitly, or implicitly by executing a Send Message command with process B as the destination). Furthermore, process B must also issue a Grant Talk Privilege command to process A (again explicitly, or implicitly by executing a Receive Message command with process A as the origin) in order to complete the talk connection. Once this communication path is established (and until it is broken by either party), process A is free to send messages to process B (see Figure 6).

3) When process A later prepares a message it wishes to send to process B, it can transmit it by executing a Send Message command. DLOS uses the parameters (see Figure 7) of the Send Message command to construct a Message Queue Element (MQE) for the message, which is then linked to the Send Queue chain pointed to by process A's PCB (Process Control Blocks, see Figure 8). If process B were in machine 1, then the Message Communication Handler for DLOS would simply transfer the MQE just formed from the Send Queue of process A to the Receive Queue of process B. However, since process B is assumed to be in machine 2, the MQE is instead removed and added to the Send Queue of the Loop Communication handler for machine 1.

The Loop Communication Handler forms an information message frame from the MQE (using the PNATT addresses for processes A and B), and transmits the frame out onto the loop. The MQE is then removed from the Send Queue and is saved on a Wait-for-Acknowledgment Queue while the Loop Communication Handler waits for a positive response to be returned from the receiving end. If a negative acknowledgment (or no response at all within a certain timeout period) is returned, then the MQE is used to form a new message frame, which is then retransmitted onto the loop. If, however, the message was received correctly, the MQE is removed and destroyed, its message buffer space is released, and the appropriate status is reported to process A (see Figure 7).

4) At machine 2, where the received message frame is removed from the loop by its own Loop Communication Handler, the message frame is dismantled, the message text is placed into a system buffer, and an MQE is formed for the message which is added to the Receive Queue chain of process B's PCB. The message sender (process A) has now been located in the PNATT, and the MQE is associated with process B. It will simply wait there on the Receive Queue until process B executes a Receive Message command specifying process A by name as the desired sender. The first (oldest) MQE associated with process A will be removed from the chain, the message text will be copied into the user's

buffer area, proper status will be reported, and the MQE will be destroyed.

The data structures and operations upon them which have been presented in the preceding paragraphs should explain how interprocess communication by name is to be achieved under DLOS. The explanation of this feature was rather detailed, since the same design philosophy and kinds of operations are found throughout DLOS. Thus, the features to be presented hereafter will not be discussed in quite so much depth.

Remote Program Calling

DLOS introduces the concept of global process control in the sense that a process can be considered an entity whose existence and scope of control is global to the entire distributed computing network, just as the latter is regarded as a single, unified computing system. The idea is that a single user process should (logically) be able to control the sequential execution of programs or procedures that are physically scattered among several heterogeneous machines (processors) in the network.

What will be explained here is essentially a generalized method of remote subroutine calling, whereby a piece of code (program X) being executed by a process A in machine 1 can call (transfer control to) another piece of code (program Y) located in machine 2, but still under the supervision of process A. The actual transfer of control (the call) from program X to program Y is effected by a Call command in program X. If programs X and Y are both in the same machine, then the call is handled in the normal manner. Otherwise, a CALL control message is sent to machine 2 (which contains program Y), specifying the name of program Y and the address of process A which is (logically) to control it. What acutally happens is that a new process A' is created in machine 2; and it is actually this new process, acting as the remote agent of process A, that controls the execution of program Y in machine 2.

Thus process A which controlled the execution of program X in machine 1 is deactivated by the remote call, for machine 1 has no work to perform for process A until the called remote program finishes and returns control to program X again. When that event occurs (signalled in DLOS by the execution of the Return command in program Y and the transmission of a corresponding RETURN control message to machine 1), several actions will be performed: process A will be reactivated, the specially created process A' will be destroyed, and execution of program X will be resumed.

Thus, the remote program call mechanism is easy to implement and totally transparent to its users. A user process need never know if a called program is local or remote, for DLOS handles all the location and linkage details. The transfer of parameters between such remote programs is considered next.

Generalized Data Transfer

Data transfer between a local calling program and a remote called program (or between two processes) is in general extremely difficult, since the two programs may reside in different machines that have completely different internal data representations. However, all machines are capable of outputing information in human-readable form as character strings in some standard code (EBCDIC, ASCII, etc.) and can accept input expressed in the same general format. Therefore, one solution to the data transfer problem is to establish a network-wide, standard character string representation for all data and to map output into this standard form on transmission and input from it on reception. Naturally, the mappings will be different for each machine.

Actual data transfer might well use two separate pieces of information: 1) the data itself, expressed as an ASCII (or perhaps EBCDIC) character string, and 2) the format description of that data, also represented as a character string (similar to a FORTRAN or PL/I format statement), extended to include all necessary device control information. By having such a generalized format capability to describe the data and its treatment, it is possible to transmit any kind of data between two machines, without having any prior agreements as to its exact format and interpretation. By having the data itself expressed as a character string, it is possible to ignore internal data representation differences and to use a standard external representation which most systems already use for communicating with the outside world. With such a powerful and generalized data transfer mechanism at its disposal, it should be possible for DLOS to support parameter transfers between remote programs (or even file transfers between remote file systems).

Unquestionably, this method is inelegant, inefficient and somewhat restrictive. Yet it is entirely workable, can be implemented fairly easily, and is quite general in nature. Every computer system that uses FORTRAN or similar languages already has elaborate data conversion and formatting routines for the input and output of data. In these cases, it should be as easy for programs to exchange data as it is to read or punch a card. The data transfer component of DLOS in each machine could even be written to perform all needed conversion automatically, so that individual programs need not be concerned with the data conversion at all.

Process Control Structures

In most systems in which processes can be created, controlled and destroyed by other processes, a tree-like hierarchical process control structure is formed. Every kind of process control function is then bound to this single, unchangeable control structure. Such a static, regulated environment makes life easy for the operating system, but it can unduely hamper and restrict user processes. Why cannot a process have control over its creator, as long as both are willing to swap roles? Why cannot two or more independent processes agree to cooperate and help each other (form coroutines)? Why cannot a process permit a group of other processes (instead of just one) to exercise joint control over it? The philosophy of DLOS is that all these forms of control (and others) are reasonable and should be possible, especially in a distributed computing system.

Such an environment as found in DLCN virtually demands that a dynamically alterable, multi-linked process control structure be permitted. Furthermore, the kind of structure suitable for one kind of process control may be entirely inappropriate for another, thus requiring a separate control structure for each kind of process control to be allowed. Therefore, DLOS provides a Privilege command which allows user processes (by mutual agreement) to change and extend their process control structures (for such things as communication with other processes, status monitoring, error recovery, etc.). With such freedom and power available to user processes, the potential for misuse is very great. But if adequate safeguards are imposed such that potential damage is limited to the cooperating processes themselves and such that ultimate control still lies within DLOS, then the resulting benefits should be tremendous. Almost any kind of complicated interaction of cooperating processes could be formed to achieve a distributed computing goal,

limited only by the imagination and talents of the problem solvers.

Distributed Resource Management

In order to accomplish distributed resource management (see Figure 9) under DLOS, it is envisioned that all resource requests in each machine will be made through the local Resource Manager. If the resource request can be satisfied locally, the Resource Manager will simply pass control over to the appropriate local resource allocator; otherwise, the Resource Manager will issue a RESOURCE control message in broadcast mode to determine if some other machine in the network has the requested resource and is willing to allocate it. No response may be obtained, in which case the resource is simply not available now, or perhaps several replied will be obtained from different machines. In the latter case, the Resource Manager can choose the one it feels is "best" in some appropriate sense and will pass the request on to that machine's Resource Manager so that it can handle the allocation of its own resources.

All usage of the allocated resource would also have to pass through the local Resource Manager. For local resources, the appropriate local routines would directly service all requests, as usual. Otherwise, a control message describing the type of resource access requested is passed through the network to the remote Resource Manager and thence to the remote access routines. The resulting reply, if any, is passed back by the same method to the original requestor.

This method of resource management is workable and attractive for a distributed computing system, since it still allows local management to work without change. As far as the local operating system is concerned, the Resource Manager is just another program to which it assigns resources -- if the Resource Manager wishes to reallocate them to remote programs, that is entirely its buisness. Moreover, it should also be possible to suballocate and exchange resources among processes, while still maintaining ultimate control over them. Obviously, there are many difficult problems to solve in an actual implementation of this approach, such as resource accounting, deadlock avoidance, or just ordinary resource access and usage. However, since resources are allocated and ultimately controlled by local Resource Managers, each under control of a local operating system, it seems likely that these problems can be overcome and that a powerful, flexible and easy to use resource management system can be obtained.

Logical I/O Transmission

The last feature to be considered is the way DLOS will go about achieving logical I/O transmission at the data record level in a distributed file system. In each machine in the network which has its own file system, with its own special data formats, device characteristics and I/O conventions, DLOS will establish a Logical I/O Handler. This local program will be capable of exchanging logical data records with user processes in response to standard read/write requests and file commands. It will also be able to handle all the intricacies of physical I/O to the local file system, including file naming and location, access protection, update interlocking, directory lookup, record searching, device dependent functions, actual I/O operations, error recovery, blocking/deblocking, code translation, formatting, and data movement. In short, the Logical I/O Handler must be able to isolate user processes from the real file system and to allow them to regard file I/O as nothing more than the exchange of data messages with a special I/O process.

The advantages of handling all I/O in this manner are clear. First, the burden of physical I/O operations and control is removed from the user process. Second, the Logical I/O Handler has complete control over the local file system, since all requests are made through it. Thus it can enforce access protection and can establish proper interlocks during file updates. Third, its functions can later be implemented in a special microprogrammed I/O controller, without any changes being necessary to user processes. Finally, and perhaps most importantly, since all data transfer is at the logical data record level, all that is needed is a standard data representation together with a set of standard file commands and conventions to make it possible to exchange data records between any two Logical I/O Handlers in the network. Since such a generalized data transfer mechanism was considered earlier in this section, it can be concluded that a user process can access any file in the entire network, so long as its request is sent to the proper Logical I/O Handler using standard file commands and conventions.

Thus DLOS is able to provide access to a network-wide distributed file system that is composed of multiple heterogeneous, local file systems. When a user process wishes to access this distributed file system, it always sends its request to its local Logical I/O Handler for initial processing. If the desired file cannot be found locally, the Logical I/O Handler puts the user's request into standard form and passes it around the loop to its proper remote counterpart. That remote Logical I/O Handler then accesses its own file system to satisfy the user's request, converts the results of the operation back into standard form, and passes them back around the loop to the local Logical I/O Handler and thence to the original requestor.

Design Philosophy of DLOS

As shown by Figure 9, the design philosophy DLOS has been to handle requests locally if possible; otherwise, they are put in a standard form and broadcast through the loop to a corresponding remote component which services the requests and returns the results to the local component. Any necessary data conversions are performed by the request handlers. Since requests are always made to the local component and all responses are obtained from it, the end user sees no difference in local and remote requests. Instead, he sees only a single, unified distributed computing system which can easily and efficiently meet all his distributed processing needs.

At the same time, this design philosophy makes implementation of DLOS as a separate component on each machine much easier, since each local component can operate almost independently of all others. All it need do is accept local requests, service them locally if possible, otherwise transform them into standard form and ask for remote help from anyone that can give it. Thus control of DLOS as a set of logically identical, cooperating components which make up a global, distributed operating system is greatly simplified.

V. CONCLUSIONS

The preceding sections have considered various facets of the message transmission protocol and network operating system design for the Distributed Loop Computer Network (DLCN), a distributed computing system envisioned as a means of investigating fundamental questions in distributed networking and computing. Research concerning DLCN is primarily directed toward geographically local communities of semi-autonomous midi/mini/micro-computer users, who occasionally have need of computing services or resources which are present elsewhere in the group, yet which are not available locally. Such a computing environment is

is typical of that frequently found today in many industrial, commercial, and unviersity settings. Bringing the cost advantages and performance improvements of distributed computer networking to such computing groups would be a very significant achievement and is, therefore, one of the major design goals for DLCN.

A university campus, which already contains dozens of mini/midi-computer facilities scattered among many academic departments, seems a logical place to test the feasibility of this goal. Thus, a proposal is now being made which may lead to the implementation of an experimental prototype version of DLCN. In support of this effort, additional research on distributed processing is now being conducted in the areas of distributed network operating systems, distributed data base management, abstraction and formalization of protocols, analytic modeling of computer networks, and fault-tolerant distributed computing. As new results are obtained from these projects, additional reports can be expected. It is to be hoped that through these efforts and with its careful integration of hardware, software and a loop communication network, the Distributed Loop Computer Network can meet its expectations and be the fore-runner for future distributed computing systems of this type.

ACKNOWLEDGMENT

The authors wish to express their appreciation to Dr. Marshall C. Yovits for his encouragement and constant support during the period of this research.

REFERENCES

1. C. C. Reames and M. T. Liu, "Variable-length message transmission for distributed loop computer networks," Tech. Rep. OSU-CISRC-74-2, Department of Computer and Information Science, The Ohio State University, June 1974.
2. _____, "A loop network for simultaneous transmission of variable-length messages," in Proc. 2nd Annual Symp. on Computer Architecture, Houston, Texas, January 1975, pp. 7-12.
3. _____, "Design and simulation of the distributed loop computer network (DLCN)," in Proc. 3rd Annual Symp. on Computer Architecture, Clearwater, Florida, January 1976, pp. 124-129.
4. M. T. Liu and C. C. Reames, "The design of the distributed loop computer network," in Proc. 1975 International Computer Symp., Vol. 1, Taipei, Taiwan, August 1975, pp. 273-282.
5. C. C. Reames, "System design of the distributed loop computer network," Ph.D. dissertation, Department of Computer and Information Science, The Ohio State University, March 1976.
6. L. G. Roberts and B. D. Wessler, "Computer network development to achieve resource sharing," in Proc. AFIPS 1970 Spring Joint Computer Conference, Vol. 36, pp. 543-549.
7. R. M. Davis, "The systems of the 1980's -- a U.S. perspective," excerpts from remarks delivered at the 35th Diebold Conference, Amsterdam, November 1975.
8. W. D. Farmer and E. E. Newhall, "An experimental distributed switching system to handle bursty computer traffic," in Proc. ACM Symposium Problems in the Optimization of Data Commun. Systems, Pine Mountain, Georgia, October 1969, pp. 1-33.
9. D. J. Farber, et al., "The distributed computing system," in Proc. 7th Annual IEEE International Conference, February 1973, pp. 31-34.
10. E. G. Manning and R. W. Peebles, "A homogeneous network for data sharing -- communications," Tech. Report, CCNG-E-12, Computer Communications Network Group, University of Waterloo, March 1974.
11. J. R. Pierce, "Network for block switching of data," Bell System Tech. J., Vol. 51, pp. 1133-1143, July/August 1972.
12. A. G. Fraser, "Spider -- a data communication experiment," Computing Science Tech. Rep. #23, Bell Laboratories, Murray Hill, New Jersey, December 1974.

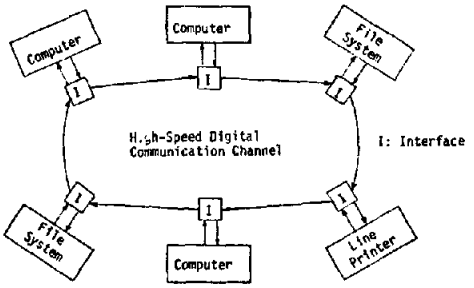


Figure 1. A Distributed Loop Computer Network

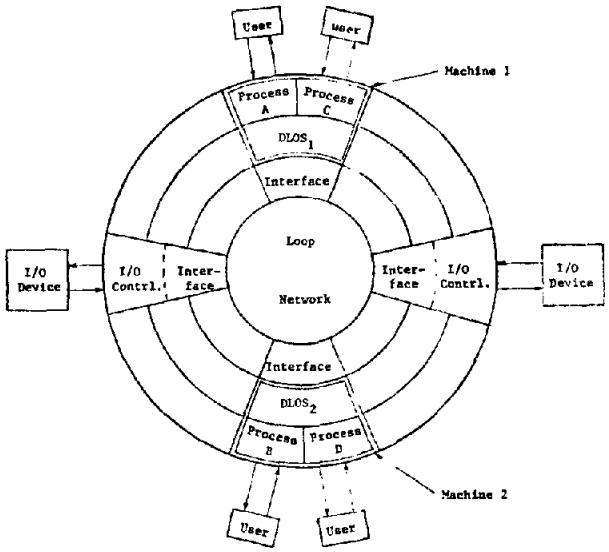


Fig. 2. A Unified Distributed Computing System

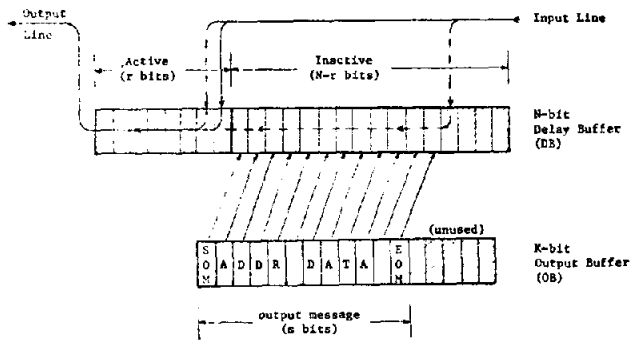


Fig. 3. Model of New Transmission Mechanism

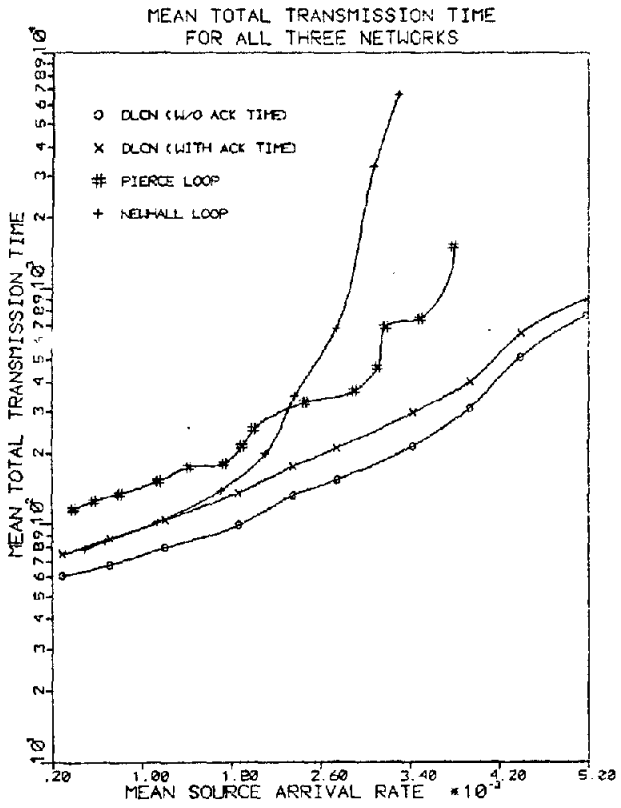


Figure 4.

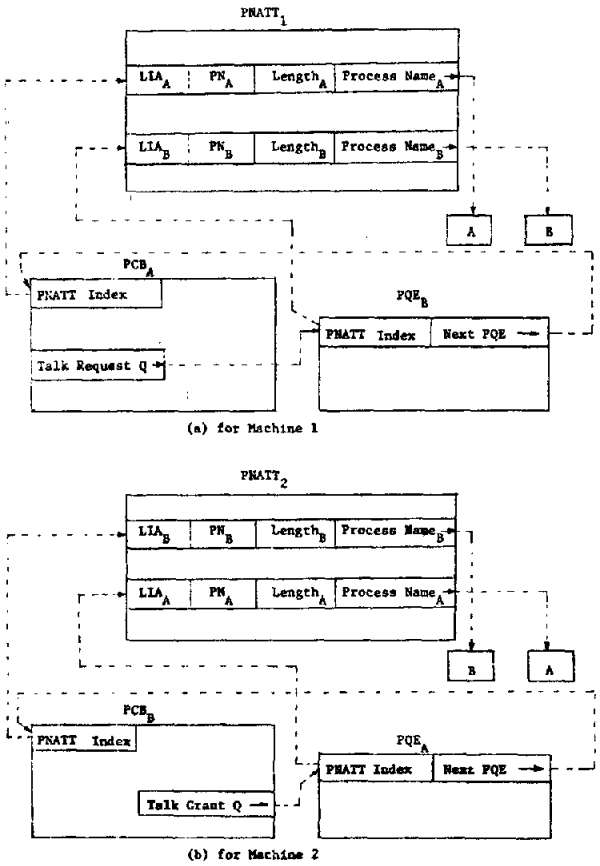


Fig. 6. Data Structure Created for Talk Privilege

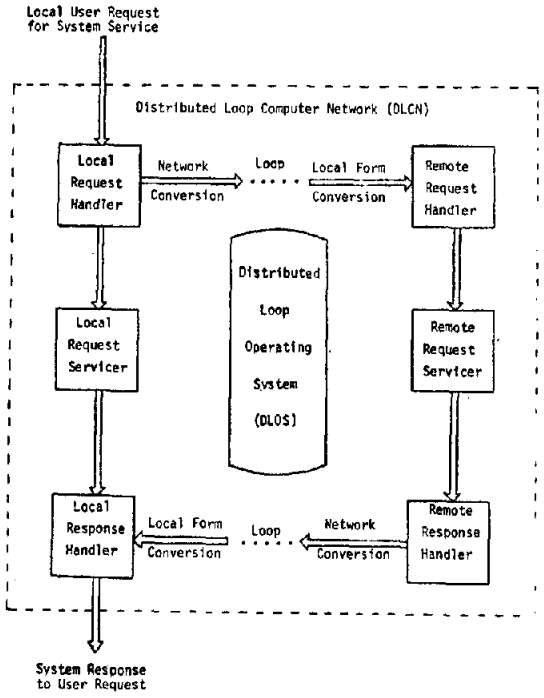


Figure 9. Request/Response Handling in DLOS

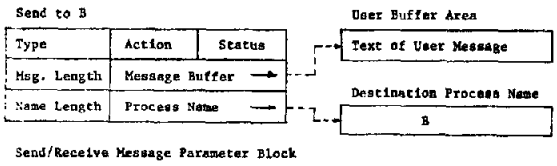


Fig. 7. Format of Send/Receive Msg. Para. Block

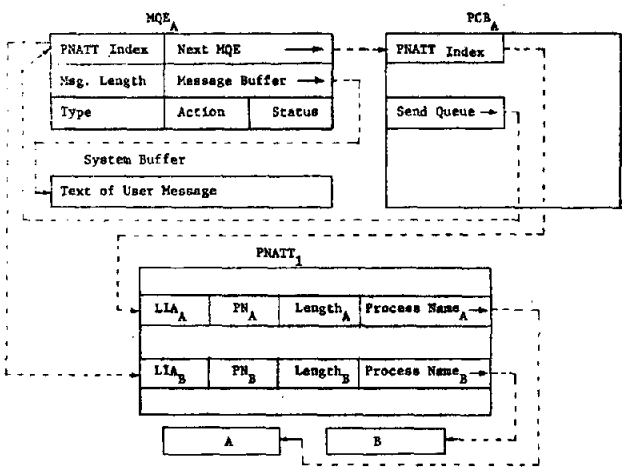


Fig. 8. Message Queue Element Format and Linkage