M.I.T. LABORATORY FOR COMPUTER SCIENCE
Computer Systems Research Division

July 28, 1977
Request for Comments No. 147

DISTRIBUTED COMPUTER SYSTEMS: Research Proposal Submitted to ARPA
From Liba Svobodova

Enclosed is the section on distributed computer systems of the continuation proposal that the laboratory will submit to ARPA for the 1978 calendar year. The version written by our group was augmented by the introduction part written by Mike Dertouzos. Notice that the proposed research on distributed computer systems is expected to involve more than just our group, though our group probably will be the one most intensely involved.

---

3. DISTRIBUTED COMPUTER SYSTEMS

3.1 Introduction and Our Approach

The exploratory research in distributed computer systems during 1977 has led to the generation of a three-year plan of attack for this very important area. It has also resulted in a first-cut model of a distributed information processing system and is the identification and assessment of problems that may arise in a distributed computer environment.

At the outset, distributed systems are important because people and organizational units within and outside the DOD are geographically distributed. As a result, data is collected, stored and accessed in geographically distributed nodes. These nodes either are already or are rapidly becoming interconnected through appropriate networks because of their need to share and axchange information. The result is currently a chaotic conflict between the desire for local autonomy exhibited by each node and the desire for coherence necessitated by the inter-node communications. At one practical extreme of system design, coherence may be minimal, e.g. by communicating sequences of bits between nodes without attaching any meaning to these sequences; this is indeed relatively easy and permits local autonomy, but makes almost impossible meaningful inter-node transactions. At the other idealistic and currently unrealizable extreme, the interconnected nodes resemble efficient models of our current inter-organizational transactions (by phone, visit or mail) and permit, for example, content oriented querries such as "who has the answer to the question 'how many ships are within one hour of harbor A'?". Between these two extremes, and much closer to the former, lie today's networks such as the ARPANET, where mail and message systems and file transfers are meaningful exchanges beyond simple sequences of bits. In our view, the distributed systems of the future must move considerably further

away from this extreme toward the more idealistic extreme implied above. This is necessary not in order to increase convenience but in order to simply make possible the orderly evolution of distributed systems that are efficient, reliable and expandable.

Our three-year plan is based on the following five points:

1) Assumptions: The basic assumptions are:

1. Local Autonomy -- each node entails data and processes that are local and need not or should not be known to others. Besides our natural tendency toward local autonomy, this assumption seems to be necessary as an isolator of complexity leading to a more complex interconnected whole.
2. Data Intensive Nodes -- we distinguish as important the distributed systems where messages concerning local data form the dominant traffic, in contrast to active processes that orginate in one node and are run into another.
3. Nodes may be Dissimilar -- although a certain basic level of coherence is necessary if they are to inter-communicate.

2) Distributed System Model and Primitives: Central to our research is the identification of semantic entities that make possible the operation of a wide clan of distributed systems. We are interested in a "minimax" model, i.e. in a relatively small set of primitives and a simple uniform structure that cover a wide gauge of distributed system applications. If we were doing this research on programming languages rather than on distributed systems, we would expect to come up with assignment, arithmetic operations, decision, repetition, input/output, and procedures -- the common characteristics of all programming languages.

3) Applications Test: This is a set of several distributed system application areas which will be case-studied in detail. Our familiarization with these specific applications serves three purposes - i) stimulus for Point 2, above; ii) test bed for candidate structures arising from Point 2, above; and iii) potential computerization of these applications through our approach. Although we have identified several such applications, we seek ARPA's help in sharpening up and making more useful our candidate list.

4) Prototype System Development: Once a model and adequate primitives have been identified we wish to embark on the design and implementation of a prototype system that realizes this model. THis system could be an underlying operating system at each node or a front end associated with each node that interconnects that node to the front ends of other nodes. Our desire to implement a prototype system is two-fold i) to test the validity and feasibility of our theories; and ii) to reshape our theories as is always the case with large-system-experimentation. The LCS net or ARPANET could be used for a feasibility demonstration, as we get underway with this effort. If we were doing research in programming languages instead of distributed systems, then this activity would correspond to the design and implementation of a specific language.

5) Defensive Programming: While it is still too early to be certain, we believe that error handling, alternative strategy selection and in general the confrontation of problems by each node and by the software modules within each node will be the rule rathern than the exception in distributed systems. Accordingly, we wish to explore the development of a systematic programming discipline that is based on this assumption. During the first year of our research we propose to concentrate on Points 2 and 3, above, since they form the backbone of subsequent developments. It is our expectation that during

the second and third year we will embark on Point 4, while refining Points 2 and 3, and on Point 5, if it proves feasible.

The importance of this research area is well known to the ARPA IPTO and need not be stressed here. Within our Laboratory, one fourth of our faculty have expressed interest, and inter-faculty seminars have already begun to further refine our proposed course of action.

3.2 State of the Art and Uniqueness of Approach

The area of "distributed systems" has become a popular source of systems research projects. This trend has been supported mainly by the rapidly falling cost of computing hardware. However, many research efforts in this area seem to miss the most important aspect of the revolution in hardware costs: the steadily decreasing entry cost of acquiring and operating a free-standing, complete computer system encourages lower-level units within a large organization to acquire their own computers that consequently will operate somewhat independently and autonomously from one another. The administrative autonomy is really the driving force that leads to acquisition of such local computers dedicated to the applications of a particular organizational unit. However, it is necessary to anticipate that these autonomous computer systems will have to be at least loosely coupled into a cooperating confederacy that serves as the information system of the organization.

As we have already stated, the basic technical problem in these emerging systems is to provide coherence in communication among the nodes in a computer network while these nodes retain their administrative autonomy. Technically, autonomy appears as a force producing incoherence: one must assume that operating schedules, loading policy, level of concern for security, availability, and reliability, update level of hardware and software, and even

choice of hardware and software systems will tend to vary from node to node with a minimum of central control. Further, individual nodes may for various reasons occasionally completely disconnect themselves from the confederacy, and operate in isolation for a while before reconnecting. Yet to the extent that agreement and cooperation are beneficial, there will be need for communication of signals, exchange of data, mutual assistance agreements, and a wide variety of other internode interaction. We hypothesize that one-at-a-time ad hoc arrangements will be inadequate, because of their potential large number and the programming cost in dealing with each node on a different basis.

The most serious problem affecting the current work in the field of distributed processing is the lack of a global concept, a model of an application class within which individual issues could be studied separately yet with the awareness of how they are related to the concept in its entirety. As a result, much of the current and earlier work in the area appears overly complex and abstract since it attempts to solve a problem for any imaginable situation, while other work appears superficial since it solves problems without considering related problems that strongly influence the feasibility of the solution. An example of the former case is some of the work on maintaining consistency in a distributed data base with multiple copies (mutual consistency problem). By insisting on immediate and continuous propagation of changes to all copies, very complex algorithms have been invented [Ellis, 1976; Thomas, 1976], yet the real world has been operating well without complete consistency and may not really justify so complex a mechanism. In the latter category, a typical example is the work on optimal distribution of a data base in a computer network. The usual considerations are the cost of the communication and the cost of storage within a computer

network knowing the intensity of requests emanating from individual network nodes [Akoka, 1977; Levine, 1975]. The problem of maintaining consistency of the data base is usually not included in such models, yet it is a crucial issue that may decide the very feasibility of distributing a data base, and, if feasible, will have definitely an effect on performance of the data base system. Mutual consistency of multiple copies is only a part of this problem, and, as discussed above, the mutual consistency requirement can probably be quite relaxed. However, it is necessary to consider the problem of internal consistency of the data base in case where a single transaction involves several pieces of the data base which are located at different nodes. The information in these separate pieces of the data base may be related in a variety of ways, e.g. subject to the constraint that the sum of all balances in a bank should always be zero. Simultaneous transactions may interfere in such a way that the internal consistency constraint may be violated. This problem arises even in centralized shared data bases [Eswaran, 1976; Gray, 1975]; in distributed shared data bases, its difficulty is magnified because of the errors in the communication subsystem and failures of the involved nodes. Most of the published work that addresses the problem of internal consistency of a distributed data base concentrates on algorithms that perform well despite unpredictable communication delays or that can be proven correct (assuming perfect operation of all components involved) [Stearns, 1976; Rothnie, 1977; Ellis, 1976]. These algorithms are not easily extended to cover more realistic situations where components are allowed to fail, or simply become unavailable as a result of exercising local autonomy.

The assumption of autonomy of the nodes that compose a distributed system is the most important ingredient that distinguishes our work from other work on distributed computer systems. As we have already alluded, though the

ARPANET is a network of autonomous computer systems, it provides a low level of coherence; specifically, it supports exchange of messages in the form of bit streams. The interpretation of the semantic content of the messages is left to the particular application, and though there exist standard protocols for special applications such as exchange of mail or remotely submitted batch jobs, these protocols had to be invented separately for each application. In our model of a distributed system, we envision autonomous nodes manipulating and exchanging objects that have unique and universal meaning. Some work related to our model of a distributed system with local autonomy is in progress at Xerox PARC, but there the requirement of autonomy is not explicitly recognized. Also, Xerox PARC is focusing on a very specific application -- office automation. We believe that the problem of autonomy has to be handled more systematically, by making it an explicit and application independent requirement that will guide the development of our model of distributed processing.

This model which is the core of our proposed research, is essential for avoiding the typical mistakes in this area, where protocols and mechanisms are devised to solve isolated or hypothetical problems. It is expected that the analyses of specific applications will produce a model where the communications and synchronization requirements are a small subset of the primitives that are needed for a general distributed system.

3.3 Relevance to the Department of Defense

Geographical distribution of computer units appears frequently and naturally in many applications within the DOD, in advanced command and control systems at one end of the scale, and in administrative data base systems at the other end. In the past, advantages of distributed systems led to a conception of computer networks that, because of the lack of understanding of

the problems involved, have failed to support coherent distributed processing. The goal of the proposed research is to further expand our understanding of the involved problems and to develop tools that will help avoid such mistakes.

We feel that it is not necessary to repeat the argument concerning decentralized vs. centralized computational resources. Distributed processing has become a pressing issue, because on one hand there is such a need for it, while on the other hand a good design methodology has not yet been developed. We believe that the specific issues that distinguish our proposed work from other work in this area are particularly relevant to the computational and information processing problems within the DOD. The degree of autonomy within DOD organizational units will have to be reflected in the computerized information systems of these units. As explained in the next section, the concept of local autonomy is particularly relevant to command and control systems. Since there are many different situations within the DOD that require or will soon require support of a distributed computer system, it is important to have a unified design base, and a methodology that is applicable to a broad class of problems, rather than resorting to the invention of a new approach for each application. The model that we plan to construct is intended to fullfil this need. The set of applications that we propose to study will include several specific DOD applications. Thus the DOD organizations immediately affected can build directly on this model.

3.4 Proposed Work

We envision a distributed system as a facility consisting of virtually autonomous computer systems, where the distribution is based on the structure and requirements of the containing system, be it a command and control system involving geographically moving objects or a DOD or civilian organization. The requirement of autonomy of individual network nodes has been presented

earlier as a consequence of the natural administrative structure of an organization. Autonomy is also important in complex control systems where it can be viewed as a means for making possible increased levels of complexity and isolating potential failures. Earlier, we also stated that distributed systems are geographically distributed as a consequence of the geographical distribution of people and organizations. We have also assumed that the transactions between machines in the distributed system are mostly data intensive. This assumption is based on the existing and historic trends in inter-organizational and intra-organizational information transactions. It is also necessary to acknowledge that geographical distribution combined with an inherently asynchronous mode of operation creates delays that may represent significant costs. Tradeoffs between delays and the timeliness of information play an important role in the engineering of distributed systems.

Autonomy, as discussed earlier, is a natural force producing incoherence. A troublesome consequence of autonomy is that potential nodes of a distributed system may actually be represented by different incompatible hardware, and, even more disturbing, a substantial amount of incompatible software. We would like to emphasize that if the individual nodes have developed in a completely uncontrolled way, there is probably little hope for achieving deeper coherence without substantially redesigning the software. Though we do not propose to solve the general problem of heterogeneity, we do not require that all nodes be fully compatible at the hardware and at the lowest software levels. However, there must exist some minimal level of coherence in the information understood and the operations supported by individual nodes, so that higher levels may be built without concern about the actual implementation of that level.

In the case where the potential nodes of a distributed system have evolved without such a level of coherence as a goal, it may be possible to manage heterogeneity by encapsulating heterogeneous nodes in a formation where all the communications with the other nodes are handled by a special front end processor. These encapsulated nodes then appear to be homogeneous. The encapsulation will still represent a major software development effort, where the part of the front end that interfaces to the encapsulated system will have to be designed especially for each individual system -- we cannot foresee emergence of a design methodology for this encapsulation step. Thus, though the model to be developed will be capable of dealing with heterogeneity, we do not consider in our proposal construction of distributed systems out of autonomous computer systems that have not been prepared for such integration. An interesting question that we intend to explore involves the critical hardware and software features for achieving a specific level of coherence and the permissible variations in the hardware and software of individual nodes in what appears to be a coherent system.

The above encapsulation provides an interesting handle on some of the problems arising from a node's local autonomy. Specifically, a part of the front end system could be made a part of the communication network rather than being encapsulated in the node. This part could serve as a local answering service that is always operational even though the computer system within the node is down for maintenance or is disconnected for security reasons. This would allow remote nodes to distinguish between the case where messages are not deliverable because of a broken connection or other problems in the communication network and the case where the node's computer system is unavailable.

3.4.1 Model of Distributed Processing

The first step in our proposed research is to develop a semantic framework within which the system engineering issues can be properly assessed and solved.

We assume that the basic information entity in a system is an object, where an object is a program or a data structure that has a unique meaning. Specifically, we do not consider an object to be just a block of bits that be interpreted in several different ways, as is the case with low-level network protocols. A node can use an object from another node only if it understands the meaning of that object. Significantly, there exists a set of primitive objects that are understood by all nodes, and that represents the minimal level of required coherence. To allow for the diversity of hardware, the representation of these objects may be different in each machine; thus we have to use the concept of an abstract object.

Our earlier research reviewed several models of computation that seem applicable to the distributed processing problem. They are as follows:

i. Actor-like message passing semantics

ii. Data flow semantics

iii. CLU-like abstraction mechanisms.

The first two models emphasize exchange of information in asynchronous systems, but they also employ abstraction mechanisms [Hewitt, 1976; Dennis, 1975]. CLU puts emphasis on use of data abstractions [Liskov, 1977]; it seems to provide a better model for dealing with the definition and implementation of objects as discussed above. None of these models in their present form is adequate as a semantic basis for distributed systems, where it is necessary to deal with multiple instances and migration of objects among nodes. Most importantly, the model must be able to deal with a variety of errors in a

physical distributed systme, i.e. it must incorporate the notion of robustness in a system composed of autonomous nodes.

3.4.1.1 Autonomy

One of the issues that we have found ourselves constantly being confronted with is the meaning of local autonomy. We understand autonomy of an intuitive level, but since it is such a key concept in our definition of distributed systems, we expect to formally define it as a part of our distributed system model.

3.4.1.2 Objects

Technical issues that are central to the problem of engineering distributed systems are naming, accessing, and protection of objects. Data abstractions provide a convenient model for dealing with situations where multiple copies of an object are provided to increase reliability. In such cases it is suitable to have just a single instance of the abstract object. The fact that there are multiple representations of that object is hidden in the implementation of the abstraction. This means that the type manager assumes the responsibility for locating and coordinating the multiple representations. In a distributed system, we feel that it will be necessary to incorporate the notion of cost into the semantics of an object. We expect that the various representations will not always be exact replicas, but possibly will represent different time instances of the object. A decision has to be made as to which of these instances must be used or is sufficient in given circumstances. This decision may be influenced by cost considerations or by the fact that some of the object instances are not accessible at the time of request because of the availability of the node at which the particular instance is located.

At the representation level, the critical issue seems to be how to use an object that is represented at another node. There are basically three possibilities:

1) send a request to operate on the requested object to the site of the object,
2) create a copy of the requested object at the requesting node (the original site of the object retains the responsibility for the object),
3) move the requested object to the requesting node (the responsibility for the object is passed on to the requesting node).

These choices raise a number of difficult questions regarding performance, integrity of the object representation, responsibility for the object, as well as naming and locating the object.

The problem of naming objects is closely connected to the issue of movability of objects. An object may contain other (abstract) objects as part of its representation. These contained objects most probably would be specified by low level names (e.g. segment numbers). In addition, an object may refer by name to objects that are not part of its representation, but that are related (perhaps temporarily) to this object in some way (e.g. procedure parameters). Such names may have to be understandable to the user, and thus could be relatively high level names (e.g. character-string file names). In any case, if an object is copied onto another node, proper interpretation of the names embedded in the object becomes necessary.

Another important consideration is whether these contained objects have to be represented at the same node as the containing object, or whether they can be remote. Specifically, if an object is copied, either it is necessary to also copy all the contained objects or it may be possible to defer action

until the particular contained object is actually needed, in which case it can be either copied or operated upon at the remote node. The design choices at this level strongly influence the low level support mechanisms that the system must provide for locating, manipulating, and protecting objects. Autonomy plays an important part here, since it is not possible to rely on later availability of the original site of the copied object to resolve the names of the contained objects.

3.4.1.3 Reliability

Physical distribution of computational tasks and information is both a means for achieving robustness and a source of new reliability problems requiring new solutions. While in a centralized system it is possible, as a last resort, to shut down the whole system to prevent further damage to work in progress, this option has to be excluded in a system of autonomous machines. Distributed systems have been used to provide fault tolerance in real time control applications; however, such systems, though they employ distributed control, are much more tightly coupled than the systems addressed in this proposal.

Earlier we introduced the notion of Defensive Programming. We feel that handling errors in a distributed system is an extremely important topic, but also a very difficult one. Similar attempts for centralized systems have been largely unsuccessful. Rather than trying to deal with evey imaginable error, a more productive approach may be to constrain the number of failure modes that the system must understand and tolerate. An important key to this problem is that individual nodes are isolated from each other except for a well defined "thin wire" communication channel. Errors caused by one node can propagate to another node only through this communication channel. By systematic testing of requests and replies from other nodes, a node may be

able to defend itself against such errors. Also, it is possible to conjecture that individual nodes will be smaller and simpler than present multi-user computer systems that serve large communities with diverse needs and interests; as a result of this scaling, it may be possible to make the nodes internally fault tolerant by using proper methodologies and mechanisms and verifying that the mechanisms operate correctly.

It can be assumed that the most common failure in a distributed system is that one of the processes involved in a communication does not respond any more. Communication protocols have been designed that are prepared to deal with this type of failure [Reed, 1976]. However, more work is necessary in the design of robust synchronization primitives that control access to distributed shared resources. Immutability of low level objects, which is one of the key concepts in data flow models, may turn out to be a crucial requirement for robust implementation of distributed data bases [Lampson, 1976]. Immutability means that once an object is created, it cannot be changed. To change some of the characteristics of an object, a new object with these characteristics must be created after which the old object is destroyed. Combining this approach with the abstraction mechanism, the abstract object can be viewed as mutable (a necessary assumption from the point of a human user), while the low level objects in the representation are immutable; the abstract object is changed by creating a new representation. The immutability aspect may have a strong impact on the low level memory management techniques; this is again an area that has to be explored.

One of the techniques for improving reliability is redundancy. One example of the application of redundancy is providing multiple copies of objects on different storage media, or, as an extension, in different nodes of a distributed system. In some fault tolerant real time systems, redundancy is

employed at a different level: results of a computation are checked by performing the same computation several times on different processors, or even using different input information -- and consequently a different procedure to process the input information. A similar approach has been suggested as a way for achieving fault tolerant software [Randell, 1975]. Specifically, this approach requires that each function be programmed in several different ways. Different versions are tried successively, until either one is found that passes an acceptance test, or an error is reported. Thus this approach creates a function abstraction with multiple representations. Multiple representations of function abstractions and multiple representations of data abstractions are very closely interrelated: different representations of a data object will require different procedures to manipulate the representation.

This approach is more natural to a distributed system than a centralized system. Because of the expected autonomy and heterogeneity, the same type of service provided on different systems will most probably have a different implementation. In addition, since different copies will run on different hardware, the potential of this approach extends from providing software fault tolerance to the level of fault tolerant service.

3.4.2 Application Test

It is essential that models and mechanisms that we develop be applicable to real distributed processing problems. Our earlier research led us to a general class of applications such as a message passing system and a distributed data base management system. It has become apparent that these classes are still too abstract to provide good guidance for our research. We feel that it is absolutely necessary to study in depth a set of specific situations that would be best supported by a distributed computer system to

determine the real needs and characteristics of these systems. Some possible examples of distributed system applications are:

- Navy payroll and personnel files
- Distributed inventory, e.g. in Defense Supply Agency
- Interconnection of airline reservation systems
- Specialized databases (e.g. medical histories, crime records).

We welcome and request ARPA IPTOS help in refining this list of applications.

Part of our early work will be directed at identifying a minimal group of specific applications that cover a broad range of present and forseeable needs. Each application will be studied in depth, using as sources available documentation, publications, and interviews with people familiar with the present form of a respective operation, and possibly with people responsible for planning a distributed system for that application. We envision that a careful study of our analysis of several different applications will reveal common denominators that can serve as a basis for our model. Once a candidate model has been developed it should be tried on at least some of the test applications individually, to determine if it is possible to separate the application specific support from the common mechanisms, and to see how these two levels interact.

Although we do believe that most of the present and future needs in distributed systems have a common core, it is conceivable that this is true only at a very low level, such as passing uninterpreted bit packets. If this is the case, then we shall divide applications into classes and look for more substantial commonalities within each class.

3.5 1977 Accomplishments

Our exploration in the area of distributed systems clarified a number of important and so far mostly neglected issues. Work on some specific problems

has been in progress and we forsee that some of it will produce applicable results before the end of 1977. The naming problem has been studied at the hardware architecture level where names are special forms of capabilities and at the human oriented level where names are character strings denoting generic services. The latter work involves development of a semantic model (based on actor semantics) that provides graceful recovery from communication failures and remote node failures [Reed, 1977]. In this model, an available instance of a service is located automatically given its generic name; if this particular instance fails in any way, the system will find another one, reconstructing in this process the necessary linkage information.

One of the important properties that decides the feasibility or preferability of different schemes for dealing with remote objects is the degree to which objects are shared by two or more nodes. This means not just how many nodes have access to an object, but how frequently the object is requested, and especially modified, simultaneously by two or more nodes. Measurements were conducted on the Multics system to determine the current extent of information sharing among Multics users. The results indicate that most of the "sharing" involves the operating system software that will have to be replicated in each node [Montgomery, 1977].

If the nodes are incompatible at the hardware level (below the minimal level of coherence), at some point it will be necessary to translate data (object representation, returned values) between the underlying machines. Several different schemes have been studied, with tagging being the most natural support in a system using data abstractions [Levine, 1977].

The work on protocols for our local network has addressed the problem of providing a robust substrate for a distributed system, specifically, initiation and maintenance of error-free connection between processes in

different nodes, recovery from a broken connection, and flow control [Reed, 1976]. Further work in this direction will include a performance study of the flow control and the error control, and design of higher level protocols, such as the protocol for naming and use of generic services that was discussed earlier in this section.

Finally, the search for and preliminary analysis of suitable applications is planned to begin before the end of 1977.

3.6 Milestone for 1978

The study of applications will occupy us throughout 1978, and will lead us into more depth as suitable models begin to emerge. The technical issues underlying these models will be illuminated and feasible solutions will be studied. By the end of 1978 we should have:

1) a set of well documented case studies on the chosen distributed system applications;
2) one or more models suitable for these applications, and
3) identification of and at least partial solutions of the associated technical problems.

We believe that our Laboratory has the necessary qualifications for this ambitious project, because of its expertise in design methodology; in the practical development of complex systems; and in computer communication networks. Our past projects have helped us develop insights into the relationship between applications and computer systems that support such applications -- that should be invaluable to the proposed research.

REFERENCES:

Akoka, J., and Chen, P., "Optimization of Distributed Database Systems and Computer Networks," M.I.T. Alfred P. Sloan School of Management, WP 916-77, March, 1977.

Dennis, J.B., "First Version of a Data Flow Procedure Language," M.I.T. Laboratory for Computer Science Technical Memo TM-61, May, 1975.

Ellis, C.A., "The Duplicate Database Problem," M.I.T. Laboratory for Computer Science, Computer Systems Research Division, Request for Comments No. 112, May, 1976.

Eswaran, K.P., et al., "The Notions of Consistency and Predicate Locks in a Database System," Comm. of ACM 19, 11, November, 1976, pp. 624-633.

Gray, J.N., et al., "Granularity of Locks and Degrees of Consistency in a Shared Data Base," IBM Research Laboratory, RJ 1654, September, 1975.

Hewitt, C., "Viewing Control Structures as Patterns of Passing Messages," M.I.T. Artificial Intelligence Laboratory, A.I. Memo 410, December, 1976. Accepted for publication in A.I. Journal.

Lampson, B., and Sturgis, H., "Crash Recovery in a Distributed Data Storage System," to be published in the Comm. of ACM.

Levin, K.D., and Morgan, H.L., "Optimizing Distributed Databases - A Framework for Research," Proc. AFIPS NCC, 1975.

Levine, P.H., "Facilitating Interprocess Communications in a Heterogeneous Network Environment," S.M. thesis, Department of Electrical Engineering and Computer Science, M.I.T., June, 1977.

Liskov, B.H., et al., "Abstraction Mechanisms in CLU," to appear in Comm. of ACM 20, 7 (July, 1977).

Montgomery, W., "Measurements of Sharing in Multics," to appear in the Sixth ACM Symposium on Operating Systems Principles, November, 1977.

Randell, D., "System Structure for Software Fault Tolerance," IEEE Trans. on Software Engineering, SE-1, June 2, 1975.

Reed, D.P., "Protocols for the LCS Network," M.I.T. Laboratory for Computer Science, Computer Systems Research Division, Local Network Note No. 3, November, 1976.

Reed, D.P., "A Protocol for Addressing Services in the Local Net," M.I.T. Laboratory for Computer Science, Computer Systems Research Division, Local Network Note No. 5, February, 1977.

Rothnie, J.B., et al., "The Redundant Update Methodology of SDD-1: A System for Distributed Databases," Computer Corporation of America, Report CCA-77-02, February, 1977.

Stearns, R.E. et al., "Concurrency Control for Database Systems," extended abstract, IEEE Symposium on Foundations of Computer Science, CH1133-8 C, October, 1976, pp. 19-32.

Thomas, R.H., "A Solution to the Update Problem for Multiple Copy Data Bases which Use Distributed Control," Bolt Beranek and Newman Report No. 3340, July, 1976.