M.I.T. Laboratory for Computer Science June 7, 1979

Computer Systems Research Division Request for Comments No. 174

# DRAFT OF ANNUAL REPORT, July, 1978 - June, 1979

Cover note by J. H. Saltzer

Attached is a rough draft of the Annual Progress Report of the activities of the Computer Systems Research Division, assembled by concatenating with only minor editing submissions from quite a number of group members. Please look it over and offer comments on

- overall organization
- omissions of significant activities
- details that are wrong
- anything else.

Check also the lists of papers, talks, committee memberships, etc., that appear at the end for mistakes and omissions.

The final version of this report will be included in the L.C.S. Annual Report, and we will also make copies for handout to visitors until the L.C.S. Annual Report is available. It is usually the case that the report gets distributed widely; many people follow our activities almost exclusively by this mechanism. Thus there is a significant payoff to making it good.

Incidentally, the final version will probably be ready by the end of June. It would be better to wait for that version to appear rather than giving copies of this one to visitors.

---

# COMPUTER SYSTEMS RESEARCH

## Academic Staff

D. D. Clark
F. J. Corbató
I. Greif
D. P. Reed
J. H. Saltzer, Group Leader
L. Svobodova

## Research Staff

J. N. Chiappa
E. A. Martin

## Undergraduate Students

R. Baldwin
D. Bollinger
H. Carter
C. Davis
D. Gorman
R. Gorman
C. Hornig
K. Khalsa
R. Lawhorn
G. Simpson
S. Szymanski
S. Toner

## Graduate Students

W. Ames
G. Arens
E. Ciccarelli
W. Gramlich
M. Herlihy
S. Kent
V. Ketelboeter
A. Luniewski
A. Marcum
A. Mendelsohn
W. Montgomery
K. Sollins
R. Wyleczuk

## Support Staff

R. Bisbee
V. Chambers
J. Jones
M. Webber

## Visitors

A. Takagi

## A. INTRODUCTION

Three LCS groups, Computer Systems Research, Programming Methodology, and Technical Services, are working closely together on a joint project to create a new kind of distributed programming environment. This environment involves software and hardware for a local ring network, internetwork interconnection, the personal desktop computer being designed by the DSSR Group, specialized service-providing computers, and finally implementation of programming language extensions that make the overall distributed environment easy to apply. Some distributed applications are also being developed, to provide additional guidance and feedback on the utility of the underlying system. The primary distinguishing feature of this research project is its rationale for distribution of function: the project assumes that the dominant force that determines where function will be distributed is administrative autonomy. Last year's progress report provides arguments for this assumption. Descriptions of the various parts of this project will be found in the individual progress reports of the three groups.

The Computer Systems Research part of this joint project this year involved four aspects:

1. Development of semantics for distributed applications;
2. Network and internetwork software design and implementation;
3. Specialized server design;
4. Experimental distributed application development.

These four aspects are discussed in the next four sections.

## B. SEMANTICS FOR DISTRIBUTED APPLICATIONS

This year, part of our group has concentrated on the problem of performing an update that involves several physical nodes. While not all distributed applications will require such rigorous control as is implied by the protocols that have emerged from this body of work, mechanisms for performing distributed updates atomically belong among the basic mechanisms of a distributed operating system. Traditional approaches to coordination and synchronization based on semaphores, locks, path expressions, or message passing do not seem to provide the guidance one might hope, because they do not include reliability and recovery mechanisms. In the distributed environment, errors and error recovery apparently must be considered explicitly as part of every mechanism and protocol including coordination of parallel activities. Since separating the problem into coordination and recovery seems not to work, other lines of separation must be sought. One line of separation that appears to have some promise is to work separately on consistency and atomicity. The idea is to on the one hand develop strategies for assuring consistency of multi-site data for single transactions run with no interference from other transactions (but with the possibility of failure) and on the other hand develop general techniques for insuring atomicity of multi-site operations in the face of possible failures, so that no transaction ever sees internal states of others.

This line of separation has been explored in depth and has provided several insights and advances. Three separate research reports by Reed [REED78], Montgomery [MONT78] and Takagi [TAKA78] present several innovative approaches and mechanisms. An analysis of the recent results and a summary of insights that have emerged are presented in a report by Svobodova [SVOB79].

Coordination of concurrent processes is difficult in a distributed system because of communication delays and modularity. In a centralized system with shared memory, coordination can be achieved inexpensively by locking the data to be accessed while the computation uses it. Locking is inexpensive, because all processes can easily access the locks, and because deadlock detection or avoidance can be centralized. In a distributed system, locking requires interactions among the users of the data and therefore imposes communications delays. Furthermore, deadlock detection is impractical because it requires global knowledge of all computations and their locks. Deadlock avoidance is impractical because a module of a distributed computation that uses modules at other nodes may not have knowledge of the data accesses or the order of access at those nodes.

Recovery from failures is made difficult in a distributed system by the peculiar nature of communication failures. In particular, when node A requires a service from node B that involves modifying data objects stored at node B, certain kinds of communication failures will leave node A in doubt as to whether node B has performed the requested action or not. The requesting computation at node A has only one option at this point, since further actions by node A are usually contingent upon successful completion of the request at B to insure consistency between various parts of the system. Node A must wait until node B's state can be ascertained, but this may take a very long time. If node A holds resources needed by other computations, then such a failure can cause deadlock.

In a monolithic distributed data base, such failures may be tolerable, since each node and communication link is maintained to a high standard of availability. In a system where nodes are autonomously managed, such failures are more likely to happen, and more likely to be of long duration. For example, after node A sends its request, but before B responds, node B (a desktop computer) may be powered off for lunch.

## 1. Atomic Actions

The goal of this research is to support the construction of atomic actions. An atomic action is an operation on data whose effects on data are completely specified by the algorithm executed by the atomic action. In particular, though the atomic action may access (read or update) many pieces of data, each many times, as part of its execution, the effect of the atomic action can be described as a relation between the initial state of all of the data items it touches and their final states when the atomic action is finished.

Atomic actions require both synchronization and recovery mechanisms in their accesses to data. Synchronization is required to ensure that no other computations within the system can observe an intermediate state of the data objects accessed. If an intermediate state of an object could be observed outside the atomic action then the behavior of the atomic action could not be specified solely in terms of a relation between initial and final states of the objects accessed. Synchronization is required to ensure also that no other computation can modify any data object used by the atomic action during its execution. That is, the atomic action's program can be written without any consideration of interference form concurrent access to the data it accesses. Recovery mechanisms are required to ensure that if a failure occurs, preventing completion of an atomic action, the intermediate state of the data resulting from partial completion of an atomic action is not exposed to observation by other computations.

Our concept of atomic actions is quite similar to that of Lomet [X14] and also similar to the sphere of control described by Davies [X6,7]. If all computations in the systems perform all their data accesses as part of atomic actions, then the observable behavior of the system will be the same as a serial schedule, as in the definition of atomic transaction developed by Eswaran, et al. [X8].

The simplest implementation of atomic actions is to delay all other computations in the system for the duration of the atomic action. This is often inefficient in a single processor system, but in a distributed system connected by a network, it may be impossible, because of communications failures.

It is sufficient, however, to guarantee that an atomic action has exclusive access to the data it actually reads or updates. Locking is often used to achieve this exclusiveness, by associating a lock (or mutual exclusion semaphore) with each data object that will be used by a computation before that computation can access the data. Locking introduces the possibility of deadlock, the detection of which may be quite difficult in a distributed system, while classic deadlock avoidance techniques cannot cope with transactions whose data accesses are unknown, due to the presence of information-hiding mechanisms that hide the representations of objects, or due to the use of pointers or accesses otherwise predicated on values obtained earlier in an atomic action's execution.

The essence of locking is to seize exclusive access to a group of objects for a period of time. Thus, the proper behavior of an atomic action is controlled indirectly, by ensuring that the timing of its steps is properly coordinated with the timing of other computations. The basis of the locking approach to implementing atomic actions is that there is one instant or interval during the atomic action at which all locks are simultaneously held. That interval must either precede or follow the corresponding interval of any potentially interfering atomic action.

In contrast, the mechanism proposed by Reed, and extended by Takagi, coordinates the access to a set of objects by a naming mechanism that gives names to a sequence of versions (virtual global states) of the system. Actions on each object specify the particular version to be affected. There are two naming mechanisms described below. Psuedo-times are a totally ordered set of names referring to successive virtual states of the system's data. Possibilities are a mechanism for referring to groups of updates to objects for the purpose of error recovery.

Atomic actions are implemented by giving the virtual processor executing the atomic action exclusive use of both a sequence of psuedo-times, derived from the real time at which the action begins, and a possibility. Access to a particular object in a particular state of the system requires that both a possibility and a psuedo-time for that state of the system be used as parameters to the access. There is a very close analogy between this approach to implementing atomic actions and the capability approach to protection of data [X5,9]. In both approaches, having a name for something is a prerequisite for its use, so exclusive use can be granted by restricting the propagation of names.

A major result of Reed's approach is that atomic actions are modularly composable operations. That is, one can implement atomic actions so that new atomic actions can be constructed out of previously existing atomic actions without either (a) modifying the preexisting implementations or (b) requiring that the new actions know what objects the

preexisting atomic actions access. Locking mechanisms for providing synchronization or recovery for atomic actions make it difficult thus to compose atomic actions because of the need to have at least one instant of time where all data touched by an atomic action is locked. Composing atomic actions in a system based on locking thus requires extending the time during which an object is locked. To ensure atomicity, updates of a distributed database are coordinated by a two-phase commit protocol [GRAY78, LAMP76]. The problem of how to schedule actions that are part of different (concurrent) operations is resolved during the first phase. In this phase, the individual participants can proceed independently. In the second phase, a careful coordination of all participants is necessary to ensure that either all of them commit or all of them abort their part of the operation. A particularly simple form of two-phase commit protocol is central to the implementation of Reed's possibilities [REED78].

### 1.1. Scheduling Actions on Distributed Objects

Our approaches differ from the traditional use of locking to schedule actions on multiple objects. Instead of locking, the key idea is that at each object the actions must take effect in the proper order, achieving the proper ordering can be done independently at each object, however. The mechanisms of Reed and Takagi use the ordered sequence of versions to record the proper order of the reads and updates applied to an object. Montgomery, on the other hand, uses an atomic broadcast protocol to ensure that object accesses always arrive in the proper order; out of order requests do not occur in his scheme [MONT78].[1]

The mechanisms of Reed and Takagi vary in how they handle out of order (outdated) updates. One approach is to discard a delayed (older) update (and consequently the operation that generated that request) if a new read (that is, a request with a higher timestamp) has already been processed [REED78]. However, this may lead to a "dynamic deadlock" where the same set of operations is aborted over and over because those operations repeatedly outdate each other - this is similar to the collision problem in contention networks such as Ethernet [METC76]. A different approach is to discard a newer request in favor of a delayed older request, given that the operation that generated this newer request has not yet been committed [TAKA78]. This solution may lead to starvation (i.e., a specific operation may never succeed since other operations will always cause it to abort), but it is free of dynamic deadlock. An extension is to allow multiple versions of objects to coexist: a delayed read request can be satisfied without having to discard any other request if the particular version still exists [TAKA78, REED78].

### 1.2. Cascading of backout

It is interesting to analyze how atomicity can be ensured without requiring that the objects updated by an operation not be available to other operations until after the final decision regarding the commitment of the first operation. Allowing early release of uncommitted information leads to the problem of cascading of backout should the final

---

1. This arrival ordering has an interesting application to the case of replicated data. Read requests are guaranteed to arrive at a copy only after any preceding write copy updates have been compelted at or on this copy, even if other copies have not yet been updated.

decision be "abort". To be able to handle such a cascaded backout, two conditions have to be satisfied:

1. During a backout of each individual operation, the operation does not need to "reacquire" (in an exclusive mode) any of the objects that it has read or modified in order to undo the changes;
2. It is possible to remember (or to reconstruct) all information flow among concurrent operations.

Since concurrent operations do not know of one another and their dependencies, it is difficult to properly backout a set of dependent operations if the recovery data is maintained by individual operations. Thus the recovery schemes ought to be object-oriented. Object-oriented recovery means that all the information needed to restore an object to some previous value if associated with the object (provided by the manager of the object) rather than with the individual operations.

To provide an object oriented recovery that allows the new value of the object to be seen before the end of the second phase of the operation that generated this new value, one can implement "multiple uncommitted versions" of the object. A new version of an object is created when the manager of the object receives the new value for the object; this action does not destroy the old value of the object (that is, the old version). A version represents the (possible) state of the object. In addition to having a value, a version has a time attribute that specifies its range of validity. The range of validity of a particular version is the time interval in the history of the object during which the object was in the state represented by that version. A version is only tentative until the operation that created it is committed. If the operation fails, the version is simply discarded. If a version is discarded, that part of the object history is erased. Now if another operation can read an uncommitted version Vx and create its own version Vy of the same or another object such that the value or even the existency of Vy depends on Vx, it is necessary to remember that Vy is dependent on Vx, since if Vx is discarded, Vy must be discarded also.

Multiple uncommitted versions were used by both Takagi and Montgomery. In Montgomery's scheme, a request to read an object that currently has several uncommitted versions will return a set of all possible values that the outstanding (not yet committed) operations could produce. This set is called a polyvalue. Thus, when an object is made visible but before the operation that modified it is completed, both the old and the new value (each of which themselves may already be a polyvalue because the outcome of some earlier operation has not yet been resolved) are presented to the next operation. If this next operation needs a precise answer, it will have to wait. If it is sufficient to know that all values possible as of that time are within an acceptable range, the operation can calculate new values for each polyvalue component, and if the answer is satisfactory, it may even commit. In this scheme, if one operation is aborted, no other operations ever have to be backed out; the only thing that has to be done is to throw away some irrelevant information, thus reducing the polyvalue set. In this sense, the scheme is symmetric for the two possible outcomes of an operation - this pruning has to be done both for the commit and the abort decision. That is, the scheme does not make any assumption about the probability of success. More important, it allows operations to be committed before the earlier operations that modified the same objects have been commited (or aborted). Takagi's scheme is more conservative. Here an operation cannot commit until all the operations on which it depends have committed. If any such earlier operation is aborted, all dependent operations must be backed out. Takagi assumes that failures are rare, that is, once a new version is created, it is very likely that it will be committed; put in different words, with high probability it is the right

value that the next operation should see. Thus, a read request returns the value of the newest version.

## 2. Summary

Although significant progress has been made towards understanding the role of atomic operations in a distributed system and the mechanisms required to implement them, more careful thought is still needed. Among the issues that need further investigation are:

1. Defining atomic operations as operations on abstract objects;
2. Atomic operation on data that has been replicated to achieve higher availability;
3. The relation between requirements for atomic transactions is distributed systems and the requirements for interruptibility in processor instruction set design.

Based on the studies of the various proposed schemes and, in particular, of the assumptions underlying those schemes, it seems that some of the approaches might be overly conservative and unbalanced in their relative emphasis on different classes of problems. An essential step towards making a significant progress in this area is to get a better understanding of the importance, frequency and severity of the specific problems that the individual schemes for atomic updates attempt to solve.

## 3. Other Work on Semantics for Distributed Applications

Sollins' thesis [Sollins, K.R., Copying Complex Structures in a Distributed System, M.S. Thesis, May 16, 1979] presents a model of a distributed system in which the universe of objects in the distributed system is divided into mutually exclusive sets, each set corresponding to a context. This model allows naming beyond the context boundaries, but limits communication across such boundaries to message passing only. A number of activities require the ability to copy objects, for example parameter passing across context boundaries and providing greater reliability through redundancy. Therefore, copying of complex data structures is investigated in this model, and semantics, algorithms, and sample implementations are presented for three candidate copy operations. Two important goals in the development of the copy operations were (1) to allow the contexts autonomy in naming the objects contained within them and (2) to reflect as much as possible the structure of an object in a copy of it. A new type of object, the message-context was developed to achieve these by recording the names of components and providing a translation into names not local to the context.

CLU provides two kinds of copy operations, copy1, which copies only the top level of a complex structure, and copy, which copies the complete structure. The copy is specified and implemented as invocations of copy1 on every component object. This mechanism does not allow for discovery of components contained more than once unless the programmer includes a procedure for detecting sharing in his own implementation of copy operations. In addition, neither of these operations reflects the model in which the context boundary plays a large part. For this reason, in addition to providing the copy1 and copy (renamed copy-one and copy-full for clarity) with modifications to achieve our goals, a new operation copy-full-local is introduced. This operation reflects the nature of the model by copying complex structures to the boundaries of the context containing the object, but not beyond. The reason this operation is different from copy-full is that containment of components can span more than one context, because naming can occur across context boundaries, although communication across such boundaries is limited to message passing. Because communication with foreign contexts may be difficult or

impossible, the copy-full-local operation may provide the most complete copying possible.

## C. NETWORK AND INTERNETWORK SOFTWARE DESIGN AND IMPLEMENTATION

A great deal of effort has been invested this year in the planning, coordination, and implementation of protocols for our local network. Much of this work is reported in detail in internal memoranda (Network Implementation Notes No. 9, 10, 11, and 12). Highlights are mentioned here.

This year saw the stabilization and implementation of an ARPA-provided internet protocol, Transmission Control Protocol, or TCP. CSR group members participated actively in the discussion that led to this newly stabilized design. In addition, we implemented TCP for the Multics System. Implementations of TCP (and its relatives) have been done elsewhere for the UNIX system and TOPS-20. These developments make use of TCP feasible as the primary protocol for our local net. The UNIX version may be transportable to the VAX, locally leaving only ITS without a minimum-effort implementation for TCP. Participation in the ARPANET TCP development and local implementation has involved a substantial amount of time for several members of our group, a cost that we hope will abate in the coming year.

Along with software for the various host machines, we have procured software and hardware for a variety of special servers to be connected to our Local Net. Perhaps most important is the gateway between the Local Net and the ARPANET. The hardware for this machine, a Digital Equipment Corporation LSI-11, has been purchased, and software for this machine is now under development. Much of the software was imported from SRI International, and the portion that must be written here is completely designed and partially written. We hope that it will be running within a month.

Another special server for the Local Net is a Terminal Interface Unit (TIU) which will allow the connection of single display terminals directly to the Local Net, rather than to a particular host machine. Again, we have procured an LSI-11 to run this server, and imported the basic TIU software from SRI. However, many local enhancements are required to this software, including the implementation of a driver for the Local Net Interface and the locally popular protocol for controlling a process from a remote terminal, SUPDUP. These enhancements are currently under way.

We have developed a new protocol, called Trivial File Transfer Protocol or TFTP, that is easier to implement than the ARPANET File Transfer Protocol. By implementing this protocol first, when adding a new machine to the Local Net, it becomes immediately possible to move files, including other protocol programs implemented elsewhere, onto the machine very early in the development of its network software. TFTP has been implemented and tested on UNIX, and files have been transferred over the net using TFTP between UNIX machines. It is being implemented for Multics, VAX, and TOPS-20. TFTP thus provides a route by which programs for VAX VMS can be transferred onto that machine, a current pressing need within the Laboratory.

In summary, the status of the Local Net is as follows. As reported elsewhere, the hardware has been operational for several months. A large quantity of software is on the verge of working, and much of it has already gone through a preliminary stage of debugging. We hope that in the next few months, perhaps by the end of the summer,

that there will be a substantial number of useful services actually available over the Local Net. The status of software for the Local Net is being reported in a series of internal memoranda, to which the interested reader is referred.

1. Specialized Servers

As discussed above, machines other than hosts are connected to the Local Net in order to provide particular network services. The services currently under development are rather low level: gateways and terminal controllers. At the same time, we are making preliminary plans for more sophisticated sorts of service, which we hope to develop during the next year. The two most interesting servers are a file server and an authentication server. The file server is very important, as more and more small machines are installed at the Laboratory. A local disk may double the cost of the small machine, so economics alone suggests that a clustered file system is a very important facility on our net. At the same time, construction of a file server allows us to explore several ideas having to do with management of remote data in a reliable consistent manner. As mentioned in section A, above, several schemes for distributed management of data have proposed by members of our group. Designing and implementing a file server with the update semantics proposed by Reed in his thesis will give us valuable experience in the area.

In support of a file server, and other servers as well, it is necessary to have some server who authenticates a particular request, so that information stored on a remote file server is not thereby made completely unprotected. We expect this to be a separate logical entity, which will take on the general task of confirming the identify of the requester of a service. Preliminary design for such a server is currently under way. The extreme importance of such a server becomes clear when one looks at the list of other services which might possibly be connected to the Local Net. For example, it has been suggested that it might be possible to connect an airline reservation service to the Local Net, so that one could obtain airline tickets from ones terminal. Clearly, we cannot allow uncontrolled purchasing of airline tickets from within the Laboratory. Thus, inclusion of such a service, which seems a very real part of an office environment, places a strong requirement for an authentication server that we can truly trust.

Other servers that we have considered for the Local Net include specialized printing machines, most particularly a laser printer, access to various institute data bases such as an online version of the telephone book, access to the Official Airline Guide, which is less dangerous than the actual ability to make reservations, access to other message sending facilities such as Telex or TWIX, a WWV accurate time source, and some sort of low cost archival storage, perhaps done by connecting the Datacomputer directly to the Local Net, or by making use of archival storage that may possibly be purchased for the M.I.T. IBM system/370. Exploring the possibilities of other servers will be done as ideas and resources permit.

## D. APPLICATIONS FOR DISTRIBUTED SYSTEMS

Research projects involving the building of applications for distributed systems can vary a great deal in character and may appear in many groups here - from office automation to artificial intelligence. We plan to coordinate the building of several applications within the distributed systems group in an effort to study the systems issues common to a variety of applications and particularly to organize the feedback into the

design of extended CLU.

Currently, the only application system being built is a calendar system. The calendar system will be built as a set of active forms. One can think of making an appointment as the filling in of a form, namely the calendar. However, filling in the form may cause actions, such as the sending of messages to other calendars. The act of filling in a form may take place during more than one interactive session, and may involve mail in cases where some calendars are either unavailable or are unable to make commitments. Emphasis is on semi-public calendars such as the schedule for the conference rooms in the building, so that we can expect the system to be used by a group of people without the overhead on putting many personal calendars online.

This approach will result in a calendar system that contrasts with others that have been built on top of shared data bases that contain information from all participating calendars. In keeping with an assumption of local autonomy and in order to mirror the existing organization of offices in which people keep their own calendars privately, we must assume that calendars (personal or for public resources) may be stored on personal computers and therefore may not always be available. This will lead to the investigation of a different set of systems issues, such as facilitating the buffering of requests to calendars which are unavailable and providing for resumptions of conversations when participants are available.

The notion of active forms should be applicable to many systems, and is particularly natural as a model for communication in office automation systems. Thus attempts will be made to use this approach in the design of applications in an effort to develop a unified approach to application design and implementation.

An immediate result of the initial design phase of the calendar has been the development of a distributed programming environment on the XX TOPS-20 time-sharing system. The environment is built on top of CLU and relies on a network interface built by David Reed. Currently, we can send and receive both CLU objects and datagrams used for internet communication. Work is in progress on implementing typed ports to facilitate compile time type checking of messages. There is a student working on implementing stable storage of CLU objects, a project that will be useful also in development of a file server. Other facilities to be implemented are guardians and a port server. We are following the design of extended CLU and are trying to simulate intended features whenever possible. However, the design of this environment is intended to be driven by application requirements and may incorporate features with which we would like to experiment even if they are not of such obvious generality as to be included in the design of extended CLU.

The design of applications is also involved with the project on development of special servers. As mentioned above, the implementation for stable storage has already appeared as a meeting point for both projects. Requirements for backup storage and processing as well as for authentication are further examples of needs that can be met by work in either or both projects. By proceeding with applications and systems development simultaneously, we will be providing immediate users of the servers' facilities.

## E. MISCELLANEOUS ACTIVITIES

An invitation-only workshop on Distributed Systems was held at the Harvard University Faculty Club on October 12 and 13, 1978. Approximately 25 leading workers in the field assembled for two days to discuss research topics and direction. A report on the workshop has been submitted for publication to Operating Systems Review.

## F. REFERENCES

To be supplied.

## G. PUBLICATIONS

1. Clark, David; Pogran, Kenneth; Reed, David. "An Introduction to Local Area Networks." Proceedings of the IEEE, Vol. 66 No. 11, (November 1978), 1497-1517.

2. Corbato', Fernando; Clingen, Charles. "A Managerial View of the Multics System Development." in Research Directions in Software Technology, P. Wegner, Ed., M.I.T. Press, Cambridge, Ma., 1979, 139-158; also reprinted in Reifer, D.J., Tutorial-Software Management, IEEE Computer Society Catalog No. EHO-146-1, 1979.
3. Greif, Irene; Meyer, Albert. "Specifying Programming Language Semantics." conference record of the Sixth Annual ACM Symposium on Principles of Programming Languages, San Antonio, Texas, January 1979, 180-189.

4. Greif, Irene; Meyer, Albert. "Specifying the Semantics of While-Programs." A Tutorial and Critique of a paper by Hoare and Lauer, M.I.T., LCS/TM130, April 1979. (Submitted for journal publication.)

5. Kent, Stephen. "Protocol Design Considerations for Network Security." Proceedings of the NATO Advanced Studies Institute on Interlinking of Computer Networks, Bonas, France, August 1978.

6. Kent, Stephen. "Privacy and Security in Networks." Protocols and Techniques for Data Communication Networks, edited by Franklin Kuo, (to be published by Prentice-Hall, April 1980).

7. Kent, Stephen. "A Comparison of Some Aspects of Public-Key and Conventional Cryptosystems." Proceedings International Conference on Communications, Boston, Ma., June 1979.

8. Reed, David. "Using Naming for Synchronizing Access to Decentralized Data." submitted to Seventh Symposium on Operating Systems Principles, April 1979.

9. Reed, David; Kanodia, Rajendra. "Synchronization with Eventcounts and Sequencers." Communications of the ACM, Vol 22 No. 2 (February 1979), 115-123.

10. Saltzer, Jerome. "Performance Analysis and Evaluation: No Connection with Reality." in Research Directions in Software Technology, P. Wegner, Ed., M.I.T. Press, Cambridge, Ma., 1979, 652-654.

11. Saltzer, Jerome; Pogran, Kenneth. "A Star-Shaped Ring Network with High Maintainability." Proceedings NBS-Mitre Local Area Communications Network Symposium, May 1979 (To be published).

12. Sollins, Karen. "Copying Complex Structures in a Distributed System." submitted to Seventh Symposium on Operating Systems Principles, April 1979.

13. Svobodova, Liba. "Performance Problems in Distributed Systems." INFOR (Canadian Journal of Operational Research and Information Processing), 1978 (To be published).

14. Svobodova, Liba. "Reliability Issues in Distributed Information Processing Systems." Proc. of IEEE FTCS-9, June 1979, (To be published).

15. Svobodova, Liba. "Building Reliable Distributed Systems: The Problem of Atomic Operations." submitted to Seventh Symposium on Operating Systems Principles, April 1979.

16. Svobodova, Liba; Liskov, Barbara; Clark, David. "Distributed Computer Systems: Structure and Semantics." M.I.T., Laboratory for Computer Science, LCS/TR-215. Cambridge, Ma., March 1979.

## H. THESES COMPLETED

1. Bollinger, Donald. "A Computer Controlled Telephone Interface." unpublished B.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1979.

2. Lamson, Richard. "An EMACS Interface to Multics Objects." unpublished B.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1979.

3. Marcum, Alan. "A Manager for Named, Permanent Objects," unpublished M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, June 1979.

4. Montgomery, Warren. "Robust Concurrency Control for a Distributed Information System". PhD Thesis, M.I.T., Department of Electrical Engineering and Computer Science, Laboratory for Computer Science, LCS/TR-207. Cambridge, Ma., November 1978.

5. Nevins, Russell. "An Efficient Logic Simulator for the Trident Guidance Computer." unpublished M.S. Thesis, M.I.T. Department of Electrical Engineering and Computer Science, December 1979.

6. Reed, David. "Naming and Synchronization in a Decentralized Computer System." PhD Thesis, M.I.T., Department of Electrical Engineering and Computer Science, Laboratory for Computer Science, LCS/TR-205. Cambridge, Ma., September 1978.

7. Simmons, Stephen. "Comparison of Microcomputers Dedicated Hardware Systems in Communications." unpublished B.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, October 1978.

8. Sollins, Karen. "Copying Complex Structures in a Distributed System." unpublished M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1979.

9. Strazdas, Richard. "A Network Traffic Generator for DECNET." M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, LCS/TM-127. Cambridge, Ma., January 1979.

10. Woltman, George. "Controlling Terminals with High-level Protocols." unpublished M.S. thesis, M.I.T., Department of Electrical Engineering and Computer Science, August 1978.

11. Wyleczuk, Rosanne. "Timestamps and Capability-Based Protection in a Distributed Data Base Environment." M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, LCS/TM-135. Cambridge, Ma., February 1979.

## I. THESES IN PROGRESS

1. Ames, Will. "A Local Area Network Simulator." M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, August 1979.

2. Hornig, Charles. "A Second Generation Network Interface for Multics." B.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, August 1979.

3. Luniewski, Alan. "The Architecture of an Object Based Personal Computer." PhD Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected data of completion, September 1979.

4. Toner, Stephen. "Dynamic Message Routing in Interconnected Local Area Data Networks." B.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, August 1979.

## J. TALKS AND PRESENTATIONS

1. Clark, David; Progran, Kenneth. "Local Networks." GTE Sylvania, Needham, Ma., December 1978.

2. Kent, Stephen. "Secure Data Communications and Storage Standards." NBS Invitational Workshop on Audit and Evaluation of Computer Security II, Miami, Florida, November 1978.

3. Kent, Stephen. "Network Security." presented at Sandia Laboratories, Albuquerque, New Mexico, March 1979.

4. Kent, Stephen. "Implementing Protected Subsystems in Decentralized Computing Environments." presented at Bell Northern Research and SRI International, Palo Alto, California, May 1979.

5. Kent, Stephen. "Comparisons of Conventional and Public-Key Cryptosystems." presented in the session on Network Security at the National Computer Conference, New York, June 1979.

6. Reed, David. "Implementing Modular Atomic Actions." IBM San Jose Research Laboratory, San Jose, California, January 1979.

7. Reed, David. "Naming and Synchronization in a Decentralized Computer System." Ford Motor Company Research Division, Dearborn, Michigan, December 1978.

8. Saltzer, Jerome. "The Impact of Changing Technology on Security." presented at Mitre Corporation, Workshop on Computer Security, Bedford, Ma., February 1979.

9. Saltzer, Jerome. "Thoughts on System Structure - The Impact on Changing Technology." Series of 7 lectures given at Tata Institute of Fundamental Research, Bombay, India, May 1979.

10. Saltzer, Jerome. IRIA 2nd International Conference on Operating Systems, Program Committee Member and Session Chairman, Rocquencourt, France, October 2-5, 1978.

10. Svobodova, Liba. "LCS Research in the Area of Distributed Computing," and Structure of Distributed Computer Systems." IBM, San Jose Research Laboratory, San Jose, California, August 1978.

11. Svobodova, Liba. "Design and Operation of Distributed Computer System." IBM, Thomas J. Watson Research Center, Yorktown Heights, New York, October 1978.

12. Svobodova, Liba. "Distributed Systems: Structure and Semantics." Bell Laboratories, Murray Hill, New Jersey, April 1979.

## K. COMMITTEE MEMBERSHIPS

Chiappa, Noel. DARPA IPTO Internet TCP Working Group.

Clark, David. DARPA IPTO Internet TCP Working Group.

Reed, David. DARPA IPTO Internet TCP Working Group.

Saltzer, Jerome. Draper Laboratory Committee on 1979 Summer Security Workshop.

Saltzer, Jerome. DoD/DDRE Security Working Group Member.