NATURAL COMMUNICATION WITH COMPUTERS. VOLUME III. DISTRIBUTED COMPUTATION RESEARCH AT BBN

William R. Sutherland, et al

Bolt Beranek and Newman, Incorporated

Prepared for:

Advanced Research Projects Agency

December 1974

BBN Report 2976 December 1974

NATURAL COMMUNICATION WITH COMPUTERS

Final Report - Volume III

Distributed Computation Research at BBN

October 1970 to December 1974

Principal Investigator

Dr. William R. Sutherland
(617) 491-1850

Project Scientist

Dr. Robert H. Thomas
(617) 491-1850

This research was supported by the Advanced Research Projects Agency under ARPA Order No. 1697; Contract no. DAHC15-71-C-0088.

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY (Corporate author) Bolt Beranek and Newman Inc. 50 Moulton Street Cambridge, Mass. 02138	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3 REPORT TITLE

Distributed Computation Research at BBN

4 DESCRIPTIVE NOTES (Type of report and inclusive dates)

Scientific

5 AUTHOR(S) (First name, middle initial, last name)

Robert H. Thomas

6 REPORT DATE December 1974	7a. TOTAL NO. OF PAGES 70	7b. NO. OF REFS
8a. CONTRACT OR GRANT NO DAHC15 71 C 0088 b. PROJECT NO ARPA on 1697 c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) BBN Report N 2976, Vol. III. 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	

10 DISTRIBUTION STATEMENT

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce for sale to the general public.

11 SUPPLEMENTARY NOTES This research was sponsored by the Advanced Research Projects Agency under ARPA order No. 1697.	12 SPONSORING MILITARY ACTIVITY

13 ABSTRACT

This report describes research activities in distributed computation at BBN from July 1971 to October 1974. The objectives of this research in distributed computation are threefold: to identify and understand fundamental problems of computing in a distributed, multi-computer environment; to determine the impact of communications networks on computer systems; and, to develop techniques which enable convenient and effective use of the resources distributed throughout a computer network. Several working distributed software systems, which have been used both to explore distributed computation issues and to provide useful computational services in a computer network environment, are described. This report discusses a variety of problems unique to distributed computation as well as some techniques and approaches for addressing these problems. Among the problem areas discussed are: network transparency in distributed systems; distributed system reliability; dynamic resource selection and job relocation; security and privacy in distributed systems; management of distributed data bases. The report identifies several research areas for which results would lead to easier to use, more reliable, and more cost effective distributed computing systems. In addition, it includes an annotated bibliography of papers written as part of this distributed computation research project.

DD FORM 1 NOV 1473 REPLACES DD FORM 1473, 1 JAN 64, WHICH IS OBSOLETE FOR ARMY USE

Unclassified
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
distributed computation						
distributed systems						
resource sharing						
computer networks						
computer operating systems						
distributed data bases						

This report is one of five volumes which compose the final report of work performed over a four year period by Bolt Beranek and Newman Inc. under contract DAHC15-71-C-0088, Natural Communications with Computers. This work was supported by the Defense Advanced Research Projects Agency under ARPA order number 1697. Because of the wide spectrum of research activites performed, the final report has been structured as follows:

Title	Volume
Speech Understanding Research at BBN	I
Speech Compression at BBN	II
Distributed Computation Research at BBN	III
ARPANET TENEX	IV
INTERLISP Development and Automatic Programming	V

Distributed Computation Research at BBN

Table of Contents

1. Introduction

This volume of the final report describes work on distributed computation from July 1971 to October 1974.

The goals of our research project in distributed computation are threefold:

- To identify and understand fundamental problems of computing in a distributed, multi-computer environment.
- To determine the impact of communications networks on computer systems.

  In this area our objectives are to identify new requirements network operation places on computer systems; to develop techniques for integrating existing computer systems into a computer network to enable effective resource sharing among the systems; and, to determine how "next generation" operating systems should be structured so that they can function effectively and efficiently in a multi-machine environment.
- To develop techniques which enable convenient and effective use of the resources distributed throughout a computer network.

  Our objective here is to develop the technology base (concepts, techniques and mechanisms) necessary to support a distributed computing environment within which the network itself is transparent. Such an environment would enable users to access network resources without attention to network details or even an awareness that they are dealing with a network. In addition, our objectives here include development of techniques to support the creation, execution, and debugging of computations that require the coordinated behavior of many computers. In this respect the goal is to achieve network transparency while providing convenient access to multiple computing and storage components.

To a large extent, our work has been directed toward developing the technology base necessary to progress from situations as exemplified by the current ARPA Network, in which

machines are interconnected but operate independently, to situations in which computer networks are used as a means for coupling the operation of the machines. A user's interactions should be in terms of the services he can access and use, rather than the particular machines that happen to provide those services.

When a service is requested, the "network system" should locate a machine providing the service and connect the user to it. The collection of machines should work together throughout the service session to locate and provide access to data bases and devices as required for the successful completion of the service session. For services that are redundantly supported, coupled operation provides for both load sharing and fail soft operation. Such a service is available as long as at least one of the machines providing it is operating. Furthermore, when one of the machines fails, the users it was servicing can be redirected to the others.

In addition to this type of behavior which can be characterized as "real time", or "on demand" coupling, operation of the machines can be further coupled "in the background". By periodically exchanging and updating critical data bases, the machines can insure that the data is accessible to users even when one or more of the machines has failed. They can function together to guarantee the successful completion of tasks requiring the action of several machines, even if all of the

required machines are not available when the task is initiated.

Coupling the operation of network machines creates a unique opportunity for improved resource management by making it possible to centrally monitor, in real time and over the long term, the loading and utilization of large scale network resources. Realization of a comprehensive measurement and monitoring capability would lead to better understanding of how computer resources are really used. This understanding, together with the potential for controlling the operation of the distributed resources, could result in better management of the resources and consequent fiscal savings.

We feel that our work has served to demonstrate the soundness and the feasibility of coupling the operation of machines to create an easy to use, reliable and cost effective network computing environment. This assessment is based partly on the success of the systems we have developed and partly on the increasing number of applications using techniques developed as part of our research.

We note that achieving the degree of machine coupling necessary to support the type of distributed computing environment sketched above is a large task. Although much has been achieved to date, in many ways we have only scratched the surface of the potential capabilities which can become available.

2. Research Approach

The state-of-the-art at the time we began our distributed computation research and the environment we found ourselves in at that time were the primary factors that influenced our approach.

When we began, distributed computation was a new research area. At that time, there was little experience at BBN or elsewhere to draw upon. The area had not been explored sufficiently to identify promising concepts or approaches. As a result, we felt that "paper" studies and designs were inappropriate. We felt that we could best contribute to the state-of-the-art by acquiring "hands on" experience with distributed computation systems.

Initial development of the TENEX operating system had just been completed and TENEX had just been connected to the ARPANET when we began our research. Furthermore, at the time there were four TENEX hosts on the network with prospects for the connection of several additional TENEX hosts. The existence of the ARPANET and the growing number of TENEX systems connected to it (over which, as implementers and maintainers, we had significant software influence), as well as the existence of other network hosts, represented a unique laboratory for distributed computation research.

The approach we chose was primarily experimental. We have

designed, implemented and studied prototype multi-computer systems for the ARPANET. Our approach has been to use these multi-computer systems as vehicles for identifying and exploring problems of distributed computing and for validating solutions to those problems.

We believe this "design, implement and evaluate" approach has proven to be useful in several ways:

- The systems constructed were chosen carefully to implement important aspects of the highly coupled distributed computing environment described in Section 1. As a result, the systems produced are useful products in themselves.

- Many problems in distributed computation are, in their most general case, unsolvable. The approach of constructing working prototype systems provided a context which served to focus attention on important special cases which have practical utility and lend themselves to solution.

- At the early stages in the development of distributed computation as a research area, the experience and insight gained from constructing real systems was crucial in developing the ability to separate problems which are inherently difficult (managing distributed data bases, cross network authentication) from problems which are difficult due to inadequacies in current computer systems (providing adequate access controls for remote users) or current communications protocols (transparently passing a user from one machine to another).

- The existence of prototype systems provided a unique opportunity to assess the value of features and approaches intended to facilitate use of network resources. In this regard, there is no substitute for feedback from real users. Furthermore, this aspect of the research has benefitted from user recommendations.

It is appropriate to note two potential dangers in our approach:

- Losing sight of the larger distributed computation research issues by becoming too involved in

implementation details and the pressures of providing operational service to users; and

- Allowing the research to be limited by what is practical within existing network and current operating system environments.

We believe that we have, for the most part, successfully avoided these pitfalls.

We feel that exploratory work, such as we have done, has resulted in the identification of important distributed computation issues and the formulation of a number of promising techniques and approaches for addressing fundamental distributed computation problems. In our opinion, the state-of-the-art has advanced sufficiently that the research emphasis should shift from exploratory experimentation to in-depth, quantitative studies of the concepts and approaches suggested by the exploratory work.

3. Research Results

The results of our work in distributed computation fall in several areas.

- Working software.

  Our research activities have been largely within the context of two distributed systems, RSEXEC and McROSS. Experience with these systems has motivated the implementation of additional software designed to provide more complete integration of TENEX into the ARPANET. This software includes modifications and extensions to the TENEX operating system and the implementation of network oriented TENEX subsystems. Section 3.1 describes the RSEXEC and McROSS distributed systems as well as other TENEX software.

- Development of concepts, techniques and approaches for distributed computation.

  Our work has lead to the formulation of concepts, techniques and approaches for distributed computation and distributed systems. The software described in Section 3.1 was implemented within the context of TENEX. However, we believe that the concepts represented by those implementations are not limited to TENEX but rather are generally applicable to the problems posed by computation in a distributed, multi-computer environment. Section 3.2 describes these concepts and techniques as well as the problems they address.

- Classification of distributed system design issues.

  As a result of our work, we have been able to classify the various issues facing designers of distributed systems into five broad problem areas. Section 3.3 discusses these distributed system design issues.

- Identification of areas requiring further research.

  Our work has served to identify a number of problem areas where solutions could lead to easier to use, more reliable, and more cost effective distributed computing systems. Section 3.4 describes these problem areas.

- Documentation.

  We have shared the results of our work with the technical community by participating in meetings and writing

papers. Section 3.5 is an annotated bibliography of our technical papers, notes, and presentations.

3.1 Working Software

This section describes software for distributed computation which we have designed and implemented. The discussion focuses on the capabilities supported by the software rather than on specific implementation or usage details. More detailed description of these systems may be found in the references and in the Quarterly Progress Reports for the project (see Section 3.5).

3.1.1 RSEXEC

We have developed and are continuing to enlarge the capabilities of the Resource Sharing Executive (RSEXEC) system [1,3].* RSEXEC is an experimental, distributed, executive-like system which acts to couple the operation of ARPA network TENEX (and some non-TENEX) hosts. Its design goal is to provide an environment which allows users to access network resources without concerning themselves with network details such as communication protocols or even being aware that they are dealing with a network. RSEXEC has been successful both as an operational service facility and as a vehicle for exploring the technical problems of realizing an effective environment for resource sharing.

- - - - - - -

* Papers we have written are referenced using square brackets and are listed in Section 3.5. Other references appear as footnotes.

A major component of RSEXEC is a distributed file system which spans host computer boundaries. Through the distributed file system a user can maintain files within the network in a convenient, host independent manner. For example, to reference one of his files, a user need not remember where within the network the file is stored; rather, it is sufficient to specify the file by name. In addition, RSEXEC supports the concept of "device binding". A user can declare a "binding" between a device and a host such that, until otherwise specified, his references to that device are to be directed to the specified host. For example, by binding the line printer device to a device port at his TIP, a TIP user can insure that his line printer output appears locally at the TIP rather than at the site providing RSEXEC service.

While RSEXEC attempts to make the network transparent, it helps users take advantage of the distributed nature of the file system. One way it does this is by allowing a user to increase the "accessibility" of files he considers important by making it easy to maintain multiple copies or "images" of them at different sites. RSEXEC provides a means to create multi-image files and it recognizes existing files which are multi-image. At present, the multi-image file facility depends upon direct user intervention for many file management operations. For example, when a user attempts to delete a multi-image file, RSEXEC informs him that the file is multi-image and asks whether all or only selected images are to be deleted. We feel that

more sophisticated support for multi-image files is possible and that techniques to accomplish it can be developed.

RSEXEC will soon be extended to support the distributed file system at the executing program level in addition to the currently supported user command level. This extension will enable existing application programs, such as text editors and compilers, to operate in a context that includes the file systems of the entire collection of network (RSEXEC) machines, without requiring that the programs themselves be rewritten.

Another aspect of RSEXEC is its support of inter-site user interaction functions. These functions allow users of the individual RSEXEC sites to interact with one another as if they were members of a single network user community. RSEXEC includes commands which allow a user to determine which users are logged in at the various RSEXEC sites, the sites a specified user has access to, and where (if at all) a specified user has active jobs. In addition, it allows a user to link his terminal to that of another user on another network machine in order to engage in an on line dialogue. All of these functions are accomplished in a way in which the network itself is transparent.

The RSEXEC system is supported by a distributed collection of server processes which function together to provide service which transcends machine boundaries. The server processes regularly communicate with one another to exchange status

information; they can establish, break, and transfer communication links; and, they are driven by a machine independent, process oriented command protocol.

Coupling the TENEX hosts by RSEXEC server processes has had the side effect of making it possible to monitor system load on all ARPA network TENEXs remotely from one or more TENEX hosts. Each RSEXEC server process maintains status information obtained from the others in a dynamic data base for use by local RSEXEC users. (The data was originally -- and still is -- intended to be used for dispatching user jobs to lightly loaded hosts to accomplish load sharing and load levelling). At the request of the ARPA office, we have written a program called LDINF which periodically records the load information from this dynamic data base. This data provides an accurate picture of how heavily loaded different TENEX sites are and how their load varies with time. LDINF has been running on BBN-TENEXA since March 1973 producing daily load information files. We have written another program called MONTH which produces weekly or monthly summaries of TENEX load data from the daily LDINF files [20]. This data is available to those interested in studying network host usage patterns.

Although the RSEXEC system currently runs on a collection of relatively homogeneous processors, it is designed to permit operation on heterogeneous machines. In fact, prototype implementations have been developed at MIT (MULTICS) and NASA

AMES (IBM 360/67).

RSEXEC is one of the first working examples of a general purpose, multi-resource system. As such it exhibits significant properties of such systems. Among these are the ability to support uniform access to both local and remote resources, persistence in the presence of component failure to guarantee task completion, and the ability to manage redundant resources to achieve increased reliability.

Since its inception 18 months ago, the RSEXEC service has been in continuous operation and available to users essentially without interruption. Although it is not totally transparent for all of its currently supported uses, we believe it to be the most advanced system implementation that addresses the problems of a general purpose operating system for a computer network.

3.1.2 TIPSER-RSEXEC

One of the most important innovations in ARPA network services from both a practical and a theoretical standpoint is use of the RSEXEC by TIPs [1,3]. A service program called TIPSER, which currently runs on three ARPA network hosts, allows TIPs to make direct, transparent use of RSEXEC as a "logical front end". Development of the TIPSER-RSEXEC system has been guided by the general philosophy that the TIP should be a transparent front end component supporting only terminal device specific functions and that access control, accounting, command

language interpretation, and other "operating system-like" functions should be handled by other more capable (larger) network machines.*

At the start of a user's session, the TIP (for TIP software versions 324 and later) automatically connects to the most responsive RSEXEC available. After he correctly supplies his name and password, the user is granted access to the network and the RSEXEC as a network command language interpreter, preparatory to logging in to a particular host. (TIP login and accounting is described further in Section 3.1.3.) TIP users of RSEXEC have access to the inter-site user interaction features and to a number of information services. These information services include:

- NETNEWS and SCHEDULES services which allow the network operations staff to communicate the latest network news and service host schedules to users.
- A GRIPE service which allows users to communicate with the operations staff.
- A HOSTAT service which reports the hosts in the network that are up and available.
- A TRMINF service providing information about a user's terminal including the TIP he is using and the TIP port to which his terminal is attached.

As mentioned above, the TIPSER-RSEXEC is a redundantly implemented network service. The redundant implementation

- - - - - - -

* N.W. Mimno et al, "Terminal Access to the ARPA Network--Experience and Improvements," Proceedings of the Seventh Annual IEEE Computer Society International Conference, February 1973.

distributes the load among the multiple machines providing the service and increases the accessibility of the service by guaranteeing that it is available whenever at least one TIPSER-RSEXEC site is up.

We developed two mechanisms necessary to support the redundant implementation. The first is a "broadcast" initial connection protocol (ICP) developed jointly with the BBN TIP group. This protocol enables a TIP to connect to an available and responsive RSEXEC rather than to a particular one at a specific site. The TIP uses this mechanism to broadcast requests for service to the known TIPSER-RSEXEC sites and then selects the first site to respond as the one to provide the service.

The second mechanism is used by the TIPSER-RSEXEC system to maintain multiple images of the various information files (e.g., news and schedules). This mechanism allows additions to the distributed information files to be made from any TIPSER-RSEXEC site and guarantees that the additions are incorporated into the file images in a consistent manner.

The importance of the TIP-RSEXEC system transcends the particular functions it currently supports. It has demonstrated the feasibility of having small hosts share resources of larger hosts to provide users with features that exceed the small hosts' own limited capacities. The users obtain these services automatically in a network transparent manner.

3.1.3 TIP Access Control and Accounting System

One of the problem areas within the ARPANET is that of controlling access both to the network and within the various hosts. One problem in this area is that the TIPs themselves, because of their size, cannot provide controlled access to the network. In order to solve this problem and the related one of accounting for TIP usage, together with the TIP group at BBN, we have developed an access control and accounting system for TIPs.

This system consists of three distinct, but related, components: network login servers; data collection servers; and, data reduction software. The system itself is implemented within the context of the TIPSER-RSEXEC and the RSEXEC distributed file system.

When a user activates a TIP port, the TIP uses the broadcast ICP mechanism to connect to an RSEXEC which acts as a network login server. If the user successfully supplies a valid name and password, he is granted continued access to the TIP and the network as well as to the standard TIPSER-RSEXEC functions. In addition, the RSEXEC sends the user's ID code to the TIP (for accounting and subsequent authentication purposes) and makes a "login" entry into an "incremental" TIP accounting data file. Should the user fail to supply a valid name and password within the allowed time, he is denied further access to the TIP.

After the TIP receives the user ID code it starts "connect

time" and (outgoing) message counters to accumulate usage data for the user's session. These counters are "active" until the user terminates his TIP session. Periodically, the TIP executes an "accounting checkpoint" procedure whereby it transmits accounting data, accumulated since the last checkpoint for its active users, to a data collection server process. The data collection server stores the checkpoint data in an incremental TIP accounting file for later processing.

Like the TIPSER-RSEXEC login servers, the data collection servers are redundantly implemented to insure high availability and to achieve load sharing. The TIP uses a request mechanism similar to the broadcast ICP to select one of the servers to accept its checkpoint data [10]. The protocol used for this purpose is quite general and can be used for the collection of data other than that for TIP accounting. Furthermore, the protocol is designed to allow considerable flexibility in the choice of a server. For example, a TIP can switch from one data collection server to another after initially choosing one in the event that the chosen server can not complete the transaction (due to network or host failure).

The collection of incremental accounting files created by the data collection servers is a large, distributed and segmented data base. The reduction of data in that distributed data base to produce periodic accounting summaries is accomplished by software which executes within the environment

provided by the RSEXEC distributed file system. This software performs a complex series of data management and network access operations in response to simple commands. When the "TIP accountant" issues the proper commands, the software automatically connects to the data collection sites and selectively retrieves and processes remote (and previously unprocessed) accounting data. This software was designed to be consistent with the RSEXEC philosophy: to allow a user to deal with resources (in this case accounting data) distributed throughout the network while relieving him of the complexities of dealing with the network

3.1.4 JSYS Trap Mechanism

The JSYS trap mechanism is an extension to the TENEX operating system which provides a means whereby operating system calls made by one process can be intercepted by another process [4].

The intercepting process does this by specifying that it wishes to gain control when selected system calls (JSYSs) are executed by other (inferior) processes. When a process monitored in this way attempts to execute such a JSYS, it is suspended and the monitoring process is notified. After gaining control, the monitoring process may take whatever action it sees fit. For example, it may choose to perform the JSYS on behalf of the process, or it may choose to check and perhaps modify the

JSYS call parameters and then allow the process to perform the JSYS itself, etc. If the monitoring process chooses to handle the trap by allowing the trapped process to resume execution of the JSYS, the trap will pass up the process hierarchy to the next process (if any) monitoring execution of that JSYS. If (when) there are no further processes in the hierarchy monitoring that JSYS, the trapped process is dispatched to the standard system code for that JSYS. Should the monitoring process choose to handle the JSYS trap by executing the JSYS on behalf of the trapped processes, the monitoring process itself is subject to traps which its superior processes may have set.

The RSEXEC system will use the JSYS trap mechanism to provide user processes with an execution environment that spans machine boundaries. For example, RSEXEC will trap file operations made by application programs executing "under" it (e.g., text editors, compilers, etc.). Operations that can be handled locally will be passed directly to the local operating system by RSEXEC. Whenever a file operation is initiated that requires access to a remote file, RSEXEC will send a request across the network to a cooperating server process at the proper host to cause it to execute the operation on behalf of the application program. Because the trapping activity is transparent, the application program has uniform access to all files, both local and remote, without regard for their network location.

While the JSYS trap mechanism was strongly motivated by the RSEXEC application, it represents an important and powerful addition to TENEX which is generally useful in applications requiring a controlled process executing environment, and in implementing and testing new virtual machine concepts.

The trap mechanism has already proven to be a powerful debugging aid. For example, after being reassembled, one of the TENEX network server programs, which was believed to have been debugged, began to malfunction. It would close a critical data file for no apparent reason on the order of once a day. After unsuccessfully studying program listings and using conventional debugging techniques for several days, the programmer built a simple process to trap and examine all operations that could possibly result in closing the file. He then ran the malfunctioning service process "under" the trapping process and was able to intercept the operation that caused the malfunction the first time it occurred (approximately ten hours after the program was placed in execution). Addition of this debugging technique to the repertoire of IDDT, the invisible debugger, would enable a user to cause a program being debugged to "break" on certain system calls. For example, this technique would enable the user to gain control on all file output operations without requiring that he remember and specify the program location of each.

A somewhat different use of the trap mechanism would enable

a user to use programs written by others with the assurance that doing so would not compromise the security of his data. For example, he could encapsulate such programs in a controlled environment which selectively inhibits output operations by trapping them and allowing only those directed to "legitimate" destinations to continue.

Finally, we note that the National Software Works project requires the encapsulation of software "tools" running on "tool bearing hosts" in order to ensure that the tools adhere to the NSW discipline. The JSYS trap mechanism simplifies the NSW encapsulation for TENEX hosts.

3.1.5 Fork Groups

A wide range of network services are provided on TENEX systems by "demon" server processes which act on behalf of remote users. (The FTP, RSEXEC and TIPSER servers are examples.) These servers typically create a new process (fork) for each instance of service.

This approach has been satisfactory for providing the standard services. However, until recently, two aspects of TENEX prevented it from being used to grant remote access to a wider range of services:

. Access Control.

  To avoid compromising data (e.g., files) maintained by the system for local users, a service process must insure that users who access the system "indirectly" through it be subjected to the same access controls as if they had

accessed the system directly. In most cases a service process requires the remote user to identify himself (by name and password) before providing service. In doing so, it gains sufficient information to adjust (enlarge, reduce, or merely modify) its own access "capabilities" to match those of the particular user it is serving. Unfortunately, TENEX provided no mechanism for doing that. The access control mechanisms in TENEX were on a "per job" rather than a "per process" basis. As a result, different processes within the same job supplying service to different remote users were subject to identical access controls rather than separate controls specific to each of the different users. Furthermore, the access controls in effect were based on the job's login name (typically SYSTEM) rather than on the identity of the remote user. This required that each service process itself implement the standard TENEX access control procedures. This, in turn, required that service processes run with special privileges to enable them to access system "private" data in order to correctly implement access control for remote users.

. Terminal Interrupts.

  In TENEX, terminal interrupts (e.g., EXEC ^C) could originate only from the job "controlling" terminal. Since network service jobs ran "detached", there was no such terminal, and even if there were, a single terminal would have been insufficient for multiple instances of service. Thus, each process that wished to provide the terminal interrupt capability to remote users was forced to simulate it.

To provide a more satisfactory execution environment for service processes, the access control and terminal interrupt features of TENEX were generalized by implementing the notion of fork groups. A process and its inferiors can be designated a "fork group" for purposes of access control and terminal interrupts. All access control checks for processes in a group are based on an access control context for the group rather than one for the job as a whole. A "proxy" login capability was implemented to enable the access control context for a service

process to be based on the identity of the remote user rather than on that of user SYSTEM. In addition, the terminal interrupt concept was modified to allow more that one source of terminal interrupts per job. An "assigned" terminal can be designated as the source of terminal interrupts for (only) a particular group of processes in a job. This change represents a slight generalization of the controlling terminal concept: each process in a job still has at most one source of terminal interrupts, but different processes may now have different sources.

3.1.6 CREEPER

CREEPER is a demonstration program which can migrate from computer to computer within the ARPA network while performing its simple task. It demonstrated the possibility of dynamically relocating a running program and its execution environment (e.g., open files, etc.) from one machine to another without interfering with the task being performed.

CREEPER led to the notion that a process can have an existence independent of a particular machine. This is an important concept for applications requiring load sharing and fail soft behavior. The experience with CREEPER emphasized the need for process oriented "login" or system access protocols and for new methods of process authentication.

3.1.7 McROSS

McROSS [6] is the first distributed system we built. It is a multi-computer system for simulation and analysis of air traffic situations. McROSS demonstrated the feasibility of having a collection of host machines work together on a single application problem (the simulation, control and display of air traffic in complex airspaces). We believe that McROSS represents the first attempt to build a coherent programming system that includes consistent application oriented primitives to support the construction and execution of multi-computer programs.

The McROSS system provides two basic capabilities. One is the ability to program air traffic simulations composed of a number of parts which run distributed among many computers. These distributed parts can be thought of as forming the nodes of a "simulation network". The second is the ability of such a simulation network to permit programs running at other ARPANET sites to "attach" to particular nodes in it for the purpose of remotely monitoring and/or controlling the node's operation. Computational responsibility for performing McROSS simulations is truly distributed. For example, as an aircraft flies from one airspace (simulation node) to another, the responsibility for simulating its dynamics shifts from one computer to another.

The techniques for dynamic reconfiguration, developed in CREEPER, were applied to McROSS to enable an ongoing simulation

to redistribute its operating parts among the network hosts without interfering with the simulation itself. Experience with McROSS served to emphasize the interrelation of naming and binding issues with those of reliability. Reliability considerations require that the components of a given simulation remain unbound to specific machines until placed in execution. Consequently, cooperating components had to be able to locate each other (by name) at execution time in order to communicate.

3.1.8 Multi-TELNET

MLTNET is a program which allows a user to conveniently control a number of jobs on different computers from a single terminal [12]. This capability has proven so useful that many existing and almost all planned ARPA network user TELNET programs (e.g., those for the ANTS and ELF terminal support systems) include or will include it.

MLTNET was motivated by the McROSS experience. It represents at a primitive level the capability for single user control of multiple computer resources. Refinements to that capability, such as the ability to start, stop and debug multi-computer programs, are requisite for an effective user interface.

3.2 Concepts, Techniques, and Approaches

3.2.1 Network Transparency

An important (and straightforward) technique for achieving a degree of network transparency is to provide procedures which automate the most common interactions with the network. At the user interface level, a command interpreter can be used to transform requests into the network and remote host access commands necessary to satisfy the requests. The RSEXEC file maintenance features and the data processing components of the TIP accounting system make use of this technique. As a result, users of these systems can concentrate on the task at hand rather than on the (to them) irrelevant complexities of dealing with the network.

This technique is equally applicable at the executing program level. The issue at this level is how to accomplish the linkage between executing programs and the procedures that automate network transactions. One straightforward approach is to augment the standard collection of system calls with ones that perform the network functions. Programs may then use the new system calls to invoke the network procedures directly. While this approach serves to facilitate network interactions, it does not, by itself, result in a high degree of network transparency. If a high degree of transparency is required and if there is the additional goal of preserving (and enlarging)

the value of existing software, it is important that the linkage with the network procedures not require extensive modification to existing programs.

We believe the trapping concept, as exemplified by JSYS traps, represents an approach for transparently accomplishing this linkage which is applicable to non-TENEX as well as TENEX machines. Implementation of it for machines of, for example, IBM or Burroughs manufacture could have tremendous payoff in terms of integrating such existing systems into a network in a way that supports transparent network usage. In return for the relatively modest cost of implementing the trapping concept along with the complementary procedures which deal with the network and remote hosts to provide, for example, a distributed file system, the vast inventory of application programs written for these machines in languages such as FORTRAN and COBOL could immediately become executable in a multi-machine environment. With no modification, these programs would be able to access and operate on non-local as well as local data. We believe that achieving network transparency for existing application programs by incorporating the trapping concept into other systems (or making use of similar capabilities which they may already have) is a promising approach. This approach is currently being investigated within the context of the NSW project.

Another technique for achieving network transparency is to provide for uniform accessiblity of all network resources. That

is, there should be no logical distinction between resources which are local and those which are remote. One approach for achieving uniform accessibility is to extend the conventions for naming entities to include a "network location" field. This enables all resources to be referenced in the same, albeit cumbersome, way. The RSEXEC distributed file system uses this approach to allow a user and his programs to reference any file in the network (that resides at a host running an RSEXEC server process). We believe that all distributed systems should make use of this approach.

While extension of the name space is necessary for uniform accessibility, as suggested above, we believe that it is insufficient, by itself, to provide a satisfactory system. Use of "full path names" is cumbersome and, more importantly, requires the user to learn and remember the network location of the resources he wishes to access. We believe that distributed systems should include cataloging functions so that a user and his programs need not concern themselves with the network location of items that they manipulate. The cataloging function supports location independent access by transforming names supplied by users into the necessary access methods. The issue here is finding the most effective ways to accomplish the cataloging function in a distributed environment.

The TIPSER-RSEXEC system provides site independent access to RSEXEC processes for TIP users via the TIP "@n" command. In

this case the TIP performs the cataloging and access functions by maintaining a list of known TIPSER-RSEXEC sites which it uses for broadcasting requests for service.

The RSEXEC system uses a different approach to cataloging in order to support site independent access to user files. When a user enters the RSEXEC file environment, a locally accessible file catalog for the user is dynamically created and maintained for the duration of the session. RSEXEC creates the catalog by acquiring file directory information from sites previously specified by the user and maintains it by monitoring user initiated file operations. As a result, each RSEXEC user has a name space tailored to his own particular usage patterns. In this environment, commonly referenced files can be accessed in a site independent manner and infrequently referenced ones can be accessed uniformly via full path names. By locally maintaining the file catalog information, we insure rapid access to it at the relatively small expense of possibly maintaining out-of-date information. Because rapid access is possible, user oriented, interactive features such as, file name recognition and completion are practical.

A concept which we have not explored in detail but which we believe has a place in distributed systems is that of an "information operator". An information operator is a network service that would maintain information about other network services such as the machines on which they are available, how

to make contact with them, and perhaps other data which characterizes them. The characteristics of such information services need more detailed specification.

Site independence and uniformity of access are important characteristics. However, when access to peripheral devices is required, location is important. In this regard, we have found the notion of device binding, as exemplified in RSEXEC, to be an important one. It allows a user to define the access paths to various devices once per session (and thereafter whenever he finds it necessary to redefine them) such that subsequent site independent references to a given device are directed to the correct device. This definition can be done either explicitly via commands or implicitly via default conventions and "user profile" information. In a multi-machine environment the "device driver" function (e.g., line printer driver) should be implemented in a way that allows programs (such as the one that produces listings) to direct output to non-local as well as local devices. To support this capability in a general way requires the development of machine independent protocols for device control.

Another area where users can be relieved from attending to network details is that of establishing and breaking connections with various service machines. The TIPSER-RSEXEC experience has suggested the use of a dynamic "reconnection" mechanism [9,13] in order to transfer a user from the "logical front end" to a

service-providing machine after he logs into the network, and subsequently from one service-providing machine to another as his computing requirements change. Reconnection should be accomplished in a transparent manner that requires no manual intervention by the user. In addition, it should include the transfer of his authentication and accounting identity from machine to machine. That is, moving a user from service to service should require no explicit disconnects, connects and logins after initial connection to and authentication with the TIPSER-RSEXEC. We have designed such a reconnection mechanism and which we plan to validate soon in the TIPSER-RSEXEC context [9,13].

3.2.2 Distributed System Reliability

In a distributed system successful operation requires that multiple components function together correctly. To achieve high reliablity, the collection of components must be organized to be insensitive to individual component failure. (The individual components themselves should, of course, be made as reliable as possible.) In this section we discuss four techniques for obtaining reliable systems: redundancy; simplicity and modularity; persistence; and, active monitoring.

We have already mentioned the use of redundancy to achieve reliability in the TIPSER-RSEXEC and TIP accounting systems. For these systems the collection of redundant components (server

processes) have been organized to enable a working "system" to be configured in the presence of individual component failures. These systems illustrate how redundancy can be used to achieve high system reliability. The RSEXEC multi-image file facility demonstrates how a system can assist users in taking advantage of redundancy by allowing the users to declare the degree of file redundancy they require and then managing redundant copies of files for them.

We have found that organizing redundant components to achieve reliability requires development of:

- Mechanisms and procedures for keeping the redundant components functionally equivalent.

  These mechanisms range from ones that insure that the software versions are consistent to ones which guarantee that critical duplicated data bases are consistently maintained.

- Cataloging functions.

  The "system" must know where the redundant components are to be found.

- Access protocols and selection strategies.

  To configure a working system, one of the redundant components must be selected for use. The TIPSER-RSEXEC broadcast ICP and the more sophisticated protocol used for data collection in the TIP accounting system are examples of such mechanisms.

The presence of multiple components in a distributed system, together with the potential for redundancy, makes it possible to achieve reliability by constructing systems from modules each of which is relatively simple. By using simple

modules, component failure due to malfunction of non-essential features can be reduced. The evolution of the TIP and TIPSER-RSEXEC is a good example of this approach. Use of redundantly supported "logical" front end servers allows the network access machine to be simple and reliable without loss of function. The more complex "front end like" features can be reliably provided by network service machines rather than within the network access machine itself. The issues here are the assignment of function among the various machines, the degree of redundancy required, and the protocols used to bind the system modules together.

There are situations in which the use of a particular resource is required, and, therefore, for which the use of redundancy is not applicable. A technique for obtaining reliable behavior in these situations lies in the ability of processes to exhibit persistence. If the resources required to complete a task are not all available when the task is initiated, a persistent process can be activated with the responsibility of completing the task when the necessary resources become available. We have found persistence to be a useful technique in a number of situations.

- The various simulation components in the McROSS system exhibit persistence. If an adjacent component is not available when needed, a McROSS simulation node activates a process dedicated to establishing communication with the missing node when it becomes available.
- The TENEX MAILER is a persistent process which guarantees delivery of messages even if the site where the addressee's mailbox is maintained is inaccessible when

the message is posted.

- The RSEXEC system is persistent with respect to constructing a user's dynamic file catalog. When a user enters the distributed file environment, if a particular site is inaccessible, a background RSEXEC process will attempt to acquire file information from the site throughout the course of the session. In addition, RSEXEC exhibits persistence when a "user profile" is modified in a way that requires verification of information by a particular remote host. In this case the persistence extends across individual user sessions until the remote host becomes available or the user acts to "cancel" the modification.

Finally, critical services and facilities require constant monitoring. The RSEXEC servers, in effect, monitor one another by exchanging status information. If a particular server process misses several consecutive status reports, it is declared non-operational by the other servers until it resumes reporting on a regular basis. Each RSEXEC server also continuously monitors its own behavior. Each server is in reality implemented by a collection of cooperating processes. One process has the responsibility of monitoring the others. Each critical process, such as the one responsible for collecting status information from other remote servers, is expected to "report in" periodically. If a process fails to do so, the monitoring process assumes that it has malfunctioned and acts to restart it. In an early implementation of the RSEXEC system, whenever a particular server process whose services were required was discovered to be non-operative, an operational server would actively and persistently attempt to restart the non-operative one. We have found that active monitoring of this

type is necessary when high reliability is a goal.

3.2.3 Dynamic Resource Selection and Job Relocation

In a distributed environment it is useful to think of a user's "job" as the dynamically varying interconnection of resources used to satisfy the user's requests. One of the tasks of a distributed system is to select and configure the resources which comprise the user's job.

We have already discussed elsewhere two aspects of this task:

- Cataloging: maintenance of information where resources appropriate to satisfy user requirements are located.
- Selection: choosing a particular resource for a user's job.

With regard to the selection function, we have made use of two techniques:

- Maintain up to date status information about the various network machines and use it to select the machine best suited for a task. RSEXEC server processes exchange status information for this purpose. Although automatic job assignment has not yet been implemented, the status information is currently available to users who may use it to manually select a machine and is, in principle, available to programs for automatic resourc selection purposes.
- Dispatch "requests for service" to the appropriate machines, allowing them to respond with status information if they choose, and then make a selection on the basis of those machines which have responded as willing to accept a new task. This is the technique TIPs use when it is necessary to select a responsive RSEXEC or to select an accounting data collection server.

The first technique involves a fixed overhead - that of exchanging and maintaining the resource status information - which is independent of the frequency of resource selection. For the second technique, the overhead is incurred on a per transaction basis and is, therefore, proportional to the frequency of selection. Although the frequency of service requests is relatively high in the TIPSER-RSEXEC case, the second technique is used because it does not require the TIP to allocate limited (storage) resources for maintaining status information. Another basic difference in these two techniques is that the second allows the constituent machines to retain a higher degree of autonomy in managing their own resources. Each machine can choose to respond or not to particular requests for service.

At present it is not altogether clear what information constitutes a useful basis for job assignment. In order to be able to configure a computation in an efficient manner it is important to extract from the user as much information as possible regarding the programs he expects to use, the data he intends to access, and the manner in which the programs access the data. Furthermore, it is not clear what a-priori status information is useful for predicting how well a particular machine will perform a given task. We believe further work needs to be done in this area.

The ability to dynamically relocate a job (or parts of one,

such as an open file) is desirable for several reasons.

- Network conditions may have changed sufficiently since the job was assigned so that it would be more effective to use another host (e.g., a host has just been restarted after a crash and it is lightly loaded).
- The job has run long enough to give the system a better picture of the job's requirements and another host would be more effective.
- The host the job is running on is about to go down and the system would like to recover as much work as possible. We note here that the multiple data collection components of the TIP accounting system (running on TENEX hosts) are capable of detecting scheduled down times and can notify the TIPs to cease using a particular component shortly before its host machine is taken down.

Our experience with CREEPER and McROSS has demonstrated that the details of moving a job (an executing process and its environment) from one host to another are relatively straightforward. The question that needs to be addressed in this area is when and under what circumstances should job relocation occur. The issues here are:

- For what type of job is relocation appropriate?
- How often should the computational situation be reassessed with respect to possible relocation?
- What are sufficient criteria for relocating a job?
- What are the tradeoffs between the costs of reconfiguring a job and those of allowing it to complete with a less than optimal configuration?

3.2.4 Security and Privacy in Distributed Systems

In many cases the goals of network transparency and ease of access conflict with those of security and privacy. Each security or access check places a barrier between the user (or his program) and the desired resource. Organizational goals must ultimately dictate the level of security necessary for the various resources which require protection. Thus, it is a good strategy to develop access control techniques which allow flexibility in their application.

Our work has indicated the need for access controls above and beyond those supported by the constituent host machines. The BBN network group has recently implemented an access control mechanism within the subnetwork which allows the set of network hosts with which a particular host can communicate to be administratively set. We have implemented access control for the predominant network access machine (TIP), apart from the access control of any particular host. In this regard we have encountered one of the problems in providing security and access control unique to a network environment. Most users find it unacceptable to be required first manually to gain access to the network, and then manually to obtain access to the site they wish to use.

In this area processes can be employed to perform the necessary access procedures on behalf of users. For example, the RSEXEC acts in this way to provide the necessary access

control parameters in order to establish user access to remote files. At the service machine, "proxy login", as exemplified by the TENEX fork group facility, represents a mechanism for establishing the correct access control environment for servicing such remote processes acting on behalf of users.

We believe that implementation of a uniform user identification and authentication scheme for the network as a whole would serve to simplify the task of making multiple access control checks transparent to users.

However, such a step still leaves many issues unresolved. After a user has been identified to the satisfaction of one network machine, when he or his program needs access to another machine, his identity for access control purposes can be passed along to the new machine. The security problem is then in the hands of the new machine, which must decide whether or not it can "trust" the calling machine to provide reliable (unforged) information. That is, after the new machine knows who the user claims to be, how should it decide to use mechanisms such as proxy login to create an access control environment?

The traditional approach is for the new machine to require the user's process to supply a password before allowing further access. In a distributed environment, there are a number of problems associated with this approach. A single password which is sufficient to obtain access to all services is desirable for simplicity. However, if one of the machines providing service

is untrustworthy or non-secure, the privacy of a user's data at all machines may be compromised. Use of a separate password for each site solves this problem, at the expense of added complexity, by insuring that the password for a site is valid only for access to that site. However, there are still security problems with this approach. Unless the user is required to supply a password himself each time one is needed (which is what we are trying to avoid), processes acting on his behalf must be able to access the password. This implies that the various passwords must be stored where they are accessible to such processes but yet protected from unauthorized access by other processes. Within the RSEXEC system user passwords are protected by encryption as well as by the standard TENEX protection mechanisms. To use the RSEXEC system from any network site, a user need only specify his name and RSEXEC password. The RSEXEC password is used by RSEXEC as a key for decrypting (and encrypting prior to storage) the individual user passwords necessary to access resources at other sites. Because the RSEXEC password is used as a key for decryption it need not (and should not) be stored anywhere. To validate the RSEXEC password, RSEXEC uses it to decrypt and verify the (stored and encrypted) user password for the local system.

Because of the difficulties of insuring the privacy of passwords, we believe that the approach of supplying a password each time access to a resource is required is inadequate in a distributed system. We believe that the following alternative

approach has promise. A set of (secure) sites is designated as authentication sites. When a user initially accesses the network, an authentication process running at such a site (selected, for example, by a broadcast ICP) is assigned to him. In order to gain continued access to the network and its resources, he must supply a valid name and password. If he is successful, the authenticator process remains as part of his job for the duration of the session, and is called upon when it is necessary to gain access to various network resources.

Gaining access to a resource is a three party procedure. It involves the user's authenticator process, the process attempting to gain the access on behalf of the user (the user process) and the process responsible for controlling access to the particular resource. The trusted authenticator process supplies the user's identity (i.e., his network wide unique ID code) to the resource manager and guarantees its authenticity. This enables the resource manager to establish an access control environment within which to provide service for the user process. We note that the interaction between the authenticator process and the resource manager need not include the user's password since the authenticator is a "trusted" process.

The security of this approach depends upon:

. The security of the authentication sites.

  In this regard we note that the noninvertible password transformation scheme suggested by Purdy* can be used to increase the security of passwords which must be stored at the authentication sites. A transformation of each

password is stored rather than the password itself. When a user supplies a password, the transformation is applied to it, and the result compared with the stored transformation. The security of this scheme derives from the fact that the transformation is non-invertible. The TIP authentication system makes use of this scheme to protect user passwords.

. The ability of the various resource managing processes to be confident that the process they are communicating with is, in fact, a valid authenticator process.

Within the ARPANET environment the subnetwork provides reliably secure identification of the host location of a remote process. Thus, the resource manager can reliably determine whether the process is at an authentication site. We have designed a mechanism for reliably ascertaining the identity of a process within a host which relies upon a host's ability to provide controlled access to special host communication ports [11].

The details of the three party protocol necessary to support this authentication procedure need to be worked out. We believe that the existing TIPSER-RSEXEC authenticator is an ideal context for doing this.

3.2.5 Management of Distributed Data Bases

Our experience indicates that data tends to be distributed for a variety of reasons.

. To insure reliability.

The accessibility of critical data can be increased by redundantly maintaining it. RSEXEC multi-image files and the TIP user ID data base are examples of data bases which are redundantly distributed to achieve highly reliable access.

- - - - - - -

* Purdy, G.B., "A High Security Log-in Procedure," Communications of the ACM, Vol. 17, No. 8, August 1974.

. To insure efficiency of access.

  Data can be more quickly and efficiently accessed if it is "near" the accessing process. A copy of the TIP user ID data base is maintained at each of the TIPSER-RSEXEC sites to insure rapid, efficient access. (Reliability considerations dictate that this data base be redundantly maintained, and efficiency considerations dictate that a copy be maintained at each authentication site.)

. The data is generated or collected in a way that causes it to be naturally distributed.

  The data base represented by the collection of incremental TIP accounting files is an example of a data base generated in this way. Individual data items are stored at the data collection site best prepared to handle them at the time they were generated by some TIP. The RSEXEC distributed file system is another example of a data base of this sort. Unless otherwise specified, files tend to be stored at the site where the process (e.g., text editor) that creates them happens to be running.

We have been concerned with two fundamentally different types of distributed data bases. The first is one which is maintained "identicallY" at a number of sites. The second type consists of distributed, non-overlapping segments; that is, the data base is a collection of segments, each of which is singly maintained at a (possibly) different location. We recognize that these two types represent extremes and that applications may call for "intermediate" types - for example, a data base comprised of a collection of segments some, but not all, of which are redundantly maintained.

The emphasis of our work with the first type of data base has been to develop techniques for consistently and automatically maintaining the redundant copies. These

techniques and the situations for which they are applicable are described below.

- Each RSEXEC server process maintains a small data base which contains status information about other sites. Each such data base can be regarded as a copy of the "true" site status information. Each site takes (almost) total responsibility for maintaining its copy of this relatively simple data base by actively acquiring the new data necessary to keep its copy up to date. Occasionally the set of sites for which status information is to be maintained changes due to the addition, removal, or change of network address of a site. When these situations occur, systems personnel notify one of the server processes which takes responsibility for propagating the change to all other servers. It does this by including site modification update information with the status information it normally transmits to each server until that server acknowledges receipt of it.

- RSEXEC maintains multi-image files for users in a semi-automatic way. When a user modifies a multi-image file, the system acts to incorporate the modification into all images. However, if some images are inaccessible at the time, RSEXEC merely informs the user and the system takes no further responsibility for completing the update of those images.

- The TIPSER-RSEXEC system maintains a copy of the TIP news file at each of the TIPSER-RSEXEC sites. Updates to the news file are limited to addition of new news items. The system allows additions to the data base to be initiated at any TIPSER-RSEXEC site and guarantees that all such updates are transmitted to and are incorporated into all copies of the data base.

- The TIP login system requires that the user ID/password data base be maintained in a consistent manner at all TIPSER-RSEXEC sites. Each copy of this data base is a collection of mutually independent user entries. Allowable updates to this data base include the addition, modification, and removal of individual user entries. We have designed (but not yet fully implemented) a data base management technique which allows updates to be initiated at any site and guarantees that they are consistently incorporated into all copies of the data base. By "consistently incorporated" we mean that if all updating activity were to cease, all copies of the data base would eventually be identical.

The techniques used to maintain the NET news and the user ID data bases each consist of two independent parts.

- A reliable, data independent update transmission and distribution mechanism.

  Both techniques use the same transmission mechanism. That mechanism uses persistent processes at the update entry sites to guarantee that all updates are delivered to all data base sites (once, and only once).

- A data dependent update action procedure.

  This procedure is activated at the data base sites when update commands arrive. For the NET news, the update procedure is a relatively simple one in which updates are appended to the data base as they arrive. For the user ID data base a more sophisticated update procedure is required. The nature of the data base and the operations permitted on it are such that recent updates to an entry override (rather than interact with) older updates. For example, when a user password is changed, the old password is simply replaced with the new one; when a user's access to a particular TIP is revoked, the list of TIPS he has access to is replaced by a new list with one less element. The update procedure depends upon the ability of each site to regenerate a sufficient portion of the time sequence of update events to determine how a particular update command is to be incorporated into the data base. When updates are initiated they are time stamped. Furthermore, each entry (and modifiable subfield) in the data base retains the time stamp of the update which resulted in its current value. When an update command (with the exception of a deletion command) arrives at a data base site, the command can be incorporated or rejected simply by comparing its time stamp with that of the data base entry it refers to. Deletion and creation require slightly special treatment. For example, if create and delete commands for an entry are initiated at separate sites, the command which creates the entry could arrive at a third site after the one which deletes it, due to network partitioning or system down times. To properly handle such cases the data base update procedure must defer "final" action on a delete command until the site is certain that all update commands for the entry which were initiated prior to the delete have arrived. Only at that point is it safe to remove the entry from the data base.

This data base update technique (described more fully in [24]) depends upon time stamps to sequence data base update commands. We believe the use of time stamps is fundamental to the management of distributed data bases in the presence of distributed updating. Although the individual time-of-day clocks at sites where updates can originate are not currently synchronized, we believe they are (barely) adequate for generating time stamps in our application. However, we believe that techniques for synchronizing time-of-day clocks on network machines should be developed such that uniform time stamps can be obtained at all sites.

Our experience with segmented distributed data bases has been in the context of the TIP accounting system. We have been concerned primarily with questions of data base organization and convenient data access. For this particular application the data base issues are:

- cataloging: it is clearly important to know where the various data segments reside so that they can be accessed.
- insuring that no duplicate entries occur. Because the entries contain accounting information, it is critical that there is no redundancy. The data collection protocol insures that no duplicate data entries occur.
- insuring that each data base entry is processed exactly once when accounting summaries are produced. We note that time stamping appears to be fundamental to guaranteeing "once only" processing.

3.2.6 Network Protocols

As discussed elsewhere, we believe that achieving a satisfactory distributed computing environment requires that processes acting on behalf of users take responsibility for the many explicit human interactions currently required to deal with the network. The existing ARPANET function oriented protocols and the software that implements them are, in many ways, inadequate in this regard because they were designed with a great deal of human intervention in mind. As we have discovered concepts and functions absent from the standard protocols, we have augmented the protocols and in some cases designed new ones in order to support our experimental research work.

We have developed and implemented protocol concepts in the following areas.

- Multi-party interactions.

  Situations frequently arise which require the interaction of more than two processes. A simple example of such a situation (which occurs often in the RSEXEC context) is one in which a process at one site finds it necessary to move a file from a second to a third site. A more complex example is the three party interaction that occurs when one process acts to authenticate another (see Section 3.2.4). The RSEXEC protocol is designed to support such multi-party interactions. Furthermore, we have been responsible for extensions to the standard File Transfer Protocol which enables it to support multi-party interactions [14,17].

- Treatment of communication paths as objects.

  Communication paths are the binding matter for distributed computations. It is important that processes be able to manipulate them as any other kind of object. The reconnection protocol, in effect, allows a process to reconfigure a computation by modifying the structure of

the computation's communication paths. As an aside, we believe that reconnection is a fundamental notion which belongs at the communication oriented host-host protocol level rather than at the function oriented TELNET protocol level where it currently resides. The RSEXEC protocol allows processes to allocate and manipulate communication paths which are used for data transfer and, in addition, to terminate communication paths at devices such as terminals in order to accomplish cross network terminal linking. In addition, we have designed a mechanism for signature authentication of network mail which relies upon a host's ability to treat communication paths as protected objects.

. Exchange of Host status information.

Access to host status information is necessary to make job assignment decisions (manually or automatically by program). The RSEXEC server programs maintain up-to-date host status information which is available to both user and programs at each RSEXEC site. The status exchange function is supported by a specifically designed protocol. The protocol includes a mechanism which enables a server designated as "master" to control the sites for which other servers maintain status information. Although the protocol currently supports the exchange of identical status information among all sites, it would be straightforward to extend the protocol to support exchange of different status data among different groups of sites.

. Broadcast requests for service.

As discussed in Section 3.2.2, the use of redundancy to achieve reliability requires protocols and strategies for accessing and selecting components. The TIPSER-RSEXEC broadcast ICP and the analogous broadcast protocol used in the TIP accounting system were developed to satisfy these requirements. As discussed in Section 3.2.1, the broadcast ICP mechanism also supports site independent access to the RSEXEC service.

. Reliable Data Collection.

In order to support the TIP accounting system, a protocol was required which TIPs could use to reliably transmit accounting data they accumulate to a data collection site for storage in the TIP accounting data base for later processing. To satisfy that requirement, we developed a general purpose data collection protocol [10]. While motivated by TIP accounting requirements, the protocol is generally applicable in situations involving multiple data sources and redundantly implemented data collection

servers.

In a number of situations the existing ARPANET host-host protocol has forced difficult or clumsy implementation to support functions which are conceptually quite simple. These difficulties are largely due to the complexity of the protocol. The situations which pose such difficulties can be characterized as involving brief, transaction oriented interactions.*

The exchange of host status information by RSEXEC server processes and the TIPSER-RSEXEC broadcast ICP are good examples of such situations. Both examples require the transmission of a short message from a process (describing its status or making a request) to one or more remote processes. The standard host-host protocol requires that the process participate in an elaborate exchange of protocol commands, carefully remembering the state of each exchange, in order to transmit its simple message. For large hosts this exchange is wasteful. For small hosts it is often impossible to correctly implement. In this regard, we note that we designed the data collection protocol used in the TIP accounting system to be separate from (and exist in parallel with) the host-host protocol in order to make implementation feasible for (memory) resource limited TIPs.

- - - - - - -

* Problems associated with this protocol's lack of robustness are discussed in another volume of in this report; see also [21,23].

3.2.7 Exploiting the Server Process Concept

As noted in Section 3.1.5, "demon" server processes running on host machines provide a wide range of network services to remote users. These processes are always present to act in response to requests from remote users.

Currently, these server processes play a largely passive role in the sense that they act only when specifically requested to do so. We believe that a number of important capabilities can be realized by enlarging the role of these omni-present server processes to include more active participation in system operation. The following are examples of areas in which an extension of the server process concept could be profitably exploited.

- Network resource management.

  The use of RSEXEC server processes to actively collect host status information represents, in a rudimentary form, a very powerful tool for the management of distributed resources [20]. This basic capability could be refined to implement a comprehensive facility for real time and long term monitoring of the loading and utilization of large scale network resources. Such a facility would result in more accurate information on, and ultimately better control of, how the resources are really used with a consequent potential for dollar savings.

- Component monitoring and testing for system reliability.

  Section 3.2.2 describes how each RSEXEC server monitors its own behavior as well as that of others in order to increase system reliability. Use of this technique could be expanded by giving network server processes the additional responsibility of regularly exercising critical hardware and software components. Whenever one of these critical components is seen to fail, the server process could signal systems personnel to repair the

failed component.

- Remote system performance monitoring and tuning.

  Operating system designers are often faced with the requirement that their software run of a variety of hardware configurations under a variety of user loads. Faced with such requirements, designers often implement flexible resource management mechanisms which can be tuned to be optimal for particular configuration and load situations. Unfortunately, the designers rarely have access to more than a single system configuration. The result is often suboptimal performance for dissimilar configurations or loads. The experts simply do not have the opportunity to tune their carefully designed resource management mechanisms for each configuration. Regularly communicating server processes provide the basis for a powerful tool for remote performance monitoring and tuning. For example, the status exchanging function of the RSEXEC server processes could be extended to allow TENEX systems personnel to remotely enable and disable the transmission of selected system meters in order to monitor the performance of a particular network TENEX. A further extension could make it possible to vary remotely system resource management parameters in order to experimentally determine optimal operating points for the various configurations and loads found in the network.

- Distribution and maintenance of software.

  The network has facilitated the distribution and maintenance of software systems. However, software distribution and installation remains a largely manual procedure in which the network replaces the use of tapes and mail as data transfer media. Distributing a new release of the RSEXEC system via the network typically requires several days to complete. The procedure itself is not inherently complex. However, it requires attention to details to insure that everything is done in the correct order; all sites must get new copies of the software, and the installation itself must be coordinated to insure that the total system always consists of a compatible set of software modules. Always present, communicating server processes, such as the RSEXEC servers, represent a potential for automating much of the software distribution and installation procedure. For example, the server process at the distribution source could be given the responsibility of transmitting new software to servers at remote sites which would be responsible for implementing the installation procedure at their sites.

3.3 Distributed System Design Issues

The previous section discussed in some detail various distributed computation problems, and techniques and approaches we have found useful for addressing them. The particular problems and approaches discussed are representative of more general distributed system design issues. This section categorizes these more general issues in a way that we have found to be useful.

3.3.1 Naming and Binding.

Coupling multiple hosts results in a name space that spans many machines. By "name space" we mean the relation between the name of an entity and its location(s) within the network. Examples of entities are people, mailboxes, services, processes, data bases and communication paths. An effective computing environment requires that the name space be managed in a way that supports location independent reference to these entities. The machines must work together to locate and establish linkages with entities as necessary to satisfy user requests. The name space management problem is complicated by the fact that some entities may be redundantly available at several locations while others may not and may therefore, from time to time, be inaccessible.

3.3.2 Robustness and Reliability.

A consequence of providing computing services via coupled operation of network machines is that, in general, successful task completion requires that several machines function correctly. The essence of the reliability problem is to acknowledge that individual machines do fail and to organize the collection of machines to be insensitive to individual machine failure. To be robust, a collection of machines must be programmed to detect and recover from non-fatal errors (to restart and resynchronize), to be persistent (to insure that important tasks eventually get done), and to anticipate and prepare for possible failures (by multiplexing critical services among several machines and backing up critical data on several machines). It is also desirable to be able to move to another machine tasks that become "trapped" in a machine that has failed; the extent to which this is feasible is unclear and requires further investigation.

3.3.3 Economics and Management.

Assignment and distribution of function among a collection of machines is a question that largely involves trading off the cost of computation against that of communication. There seem to be two separate aspects to the problem. There is a static aspect concerned with assignment of classes of functions to the machines best suited to perform them. For some functions the

assignment may be clear: in the ARPA network the management of multi-billion bit data bases is the job of machines such as the Datacomputer(s); the solution of large simultaneous differential equations is the job of the ILLIAC. For other functions, such as executive control functions, the assignment is less clear. The second aspect of function assignment is a dynamic one concerned with the selection and management of resources required to satisfy user requests. For example, when a service (e.g., the network message service, the TENEX virtual machine service) is available on more than a single machine, the problem is to select the machine expected to be most effective. After the objects (e.g., files, processes) required for a particular task are located, the problem is to configure them so that the task can be performed in an efficient manner.

3.3.4 Authenticity and Validity of Information.

The performance of a system is dependent upon the accuracy, authenticity, and timeliness of information available to it. In a distributed system, decisions at all levels must be based on information from a variety of sources, not all of which are local to the decision making entity. Parts of the system responsible for such aspects as insuring robust behavior, achieving effective dynamic resource management, insuring user data security, and enforcing resource access controls must act upon information for which they have neither direct (local) access nor direct control. The problem for the distributed

system designer is to determine the extent to which sources are to be trusted with respect to the information they provide.

3.3.5 User Interface.

Attention to the areas described above will result in concepts and mechanisms which will form the technical basis for the development of a reliable, secure, and cost effective distributed computing system. How well human and process users will be able to deal with such a system will depend upon the extent and ease with which control can be exerted over such mechanisms. For example, initiation of what is believed to be a simple task could easily trigger a large amount of activity in a number of machines. To enable a user to match the expenditure of resources required to accomplish a task with the importance placed on its completion, means must be available to define the task's scope and extent, including the persistence with which the system should act to complete it.

Distributed computing systems can make feasible new capabilities supported directly by multi-computer application programs. To realize that potential, the system must include tools that facilitate the creation and debugging of multi-computer programs and it must provide users with convenient means for starting, controlling, and stopping such programs.

3.4 Areas Requiring Additional Research

3.4.1 Distributed Data Bases.

Multi-computer systems introduce a new class of data base management problems which result from the distributed nature of the data. These problems occur at all levels of system design and implementation, ranging from low level system primitives to function oriented application software. Section 3.2.5 describes some techniques we have developed for managing special types of distributed data bases.

We believe that we have made progress in this area. However, we feel that intensive research is required to develop a coherent methodology for the design, implementation, and management of distributed data bases. The methodology resulting from this research should include:

- Techniques for evaluating alternatives for data base organization in terms of cost, performance, and reliability requirements.
- Mechanisms for locating and selecting, in the case of redundancy, data items in response to data base access requests.
- Update control techniques to insure the integrity and consistency of distributed data bases.
- Procedures for authentication and data base access control.

The goal of this research effort should be to produce a viable combination of methodology, principles, and operating

procedures to enable system builders to make rational design and implementation decisions regarding the placement, management and access of data in a distributed environment.

3.4.2 Persistent Processes.

Section 3.2.2 discusses persistence as a technique for achieving reliable task completion in a distributed environment. To date, the use of persistent processes has been limited to relatively simple situations.

We believe that progress in the following areas would facilitate more widespread use of this potentially powerful approach for achieving reliability.

- Languages for persistent process task specification.

  At present, in all cases of which we are aware, persistence is accomplished by special processes that are programmed to perform a specific task. A new special purpose process must be programmed each time a situation requiring persistence arises. A more general approach would be to develop a language for defining task programs for general purpose persistent processes. Part of a task program would include the degree of persistence required (i.e., how long or often the process should try before it gives up) and the actions to be taken when errors occur. When persistence is required to complete a user's request, a "task request program" would be compiled and left for execution by a general purpose persistent process. We note that a task specification language of this sort could be the basis for more uniform system implementations. Since the language can, in principle, be used to define foreground tasks as well as persistent background tasks, a system could be structured that compiles user and program requests into task request programs which are placed into immediate execution. Whenever such a program cannot be successfully completed because of the inability to access certain network resources, the system can either abort the task or leave

the already compiled task request program for execution by a persistent process.

. Authentication procedures for persistent processes.

For many applications security considerations require that a host perform certain actions only after the requesting entity has been authenticated. While the host on which a persistent process is running may consider the process to be properly authenticated, other hosts whose cooperation is required to complete a task may not. It is unclear how to authenticate such a process in the absence of a user without compromising the security of the user's authentication information (e.g., password). We believe a fundamentally new approach to authentication is required to permit more widespread application of persistent processes. In this regard, we believe that the three party approach to authentication described in Section 3.2.4 may be an answer. When persistence is required, the authenticator process that is part of the user's job could remain after the job itself is "terminated" in order to be available for authenticating persistent processes. The precise details of how this would be accomplished need to be worked out, and then the security of the procedure needs to be analyzed.

3.4.3 Hot Switchover.

The use of organized redundancy to achieve reliability is discussed in Section 3.2.2. The techniques described there involve the selection of a particular component from a pool of functionally equivalent ones to perform a task at the time the task is initiated.

The use of redundancy can, at least in principle, be extended to allow the dynamic replacement of a component that fails while it is in use by one which is functionally equivalent. For example, it should be possible to switch transparently from using one image of a multi-image data base to

another if the image originally selected becomes inaccessible. Similarly, tasks trapped in a machine that fails can, in principle, be moved to another equivalent machine.

The desirability of such a "hot switchover" capability is clear: computational services can be made extremely robust with respect to the failure, loss, or destruction of the individual computer systems which provide the service. The extent to which the capability is practically achievable requires investigation.

One way to think of hot switchover is to think of a particular service as being provided by a single active process together with multiple redundant inactive "images" which can be activated whenever necessary. The problem is to keep the inactive images synchronized with the active process so that one of them can be activated if the active process fails. This synchronization must, of course, be accomplished without requiring that each image process duplicate the entire computation. The issues to be investigated here are:

- What constitutes synchronization and its inactive images? What state information has to be exchanged, how often does it need to be exchanged, and what is the protocol for exchanging it?
- How does the "system" decide when the active process has failed?
- How is an inactive process activated? How is one selected? How does it use the state information it has which may be out of date to "catch up" to the current computational state?

- After a switchover occurs, how are the other inactive processes, including the old active process (when and if it recovers), to be resynchronized?
- For what classes of computation is this a viable approach?

We recognize that the hot switchover concept has been used in the past to achieve reliability in very specific application areas. The thrust of the research suggested here is to develop general mechanisms for accomplishing hot switchover in order to enlarge the range of situations for which it is applicable.

We note that the techniques and protocols developed for hot switchover are potentially applicable for load levelling and sharing. At the points where synchronization of the currently active process and its images occurs, an inactive image could be activated to redistribute the computational load.

3.4.4 Efficient Job Configuration.

The techniques we have developed to achieve network transparency and uniform resource accessibility, form much of the basis for a "network operating system" within which distributed computations work correctly regardless of the precise configuration of the various data and process components.

In some situations, the computational requirements may be sufficiently constrained so that only a single configuration is

possible for a job. However, in many situations, a number of alternative configurations may be possible due to the functional equivalence of redundant components (e.g., data files, processors). For some situations any configuration may be adequate. However, for many situations performance requirements (e.g., responsiveness, cost) may make it necessary to carefully choose an optimal configuration from the alternatives.

We note that the simplistic approach of configuring a job from the most responsive individual components will not necessarily guarantee a total job configuration that is optimal because it fails to take into account the requirement for interactions among the components.

It is easy to understand in a qualitative manner, the tradeoffs between communication and computation overheads for various configuration alternatives. For example, groups of entities strongly coupled by frequent interactions (process to process, process to file) should, whenever possible, reside in the same host to minimize communication delay. However, at present, there is no quantitative methodology for evaluating alternative job configurations in terms of these tradeoffs.

Further research is required here to develop quantitative measures for configuration effectiveness. These measures would form the basis for network operating system procedures that select job configurations which are optimal along various performance dimensions.

3.5 Annotated Bibliography

This section is an annotated bibliography of papers and notes we have written as part of our research in distributed computation.

1. Johnson, P.R., R.E. Schantz and R.H. Thomas, "Interprocess Communication to Support Distributed Computing," submitted to the ACM SIGCOMM-SIGOPS Interface Meeting on Interprocess Communication, March 1975.

This paper focuses on the impact which distributed computing systems have on the interprocess communication facilities used to support them. Based on the experience of creating distributed systems, three different types of machine cooperation are described and communication facilities to couple the machine in each case are discussed. A number of other IPC issues related to distributed systems are raised.

2. Schantz, R.E., (with E. Akkoyunlu and A. Bernstein), "Interprocess Communication Facilities for Network Operating Systems," IEEE COMPUTER, Volume 7, Number 6, June 1974, pp. 46-55.

This paper describes three approaches to the problem of creating an interprocess communication facility for a network environment. The three approaches all exhibit the property that a single mechanism is used for both inter-machine and intra-machine communication. The IPC facilities are compared with respect to a number of factors relevant to network operation.

3. Thomas, R.H. "A Resource Sharing Executive for the ARPANET," AFIPS Conference Proceedings, Vol. 42, June 1973, pp. 359-367.

This paper describes the initial implementation of the RSEXEC, a distributed, executive-like system that creates an environment which facilitates the sharing of resources among TENEX hosts on the ARPANET. The first half of the paper develops the user's view of the distributed executive, which includes a distributed file system. The second half deals with basic issues in implementing a distributed operating system.

4. Thomas, R.H., "JSYS Traps - A TENEX Mechanism for Encapsulation of User Processes," submitted to the 1975 National Computer Conference, June 1975.

The JSYS Trap mechanism is an extension to the TENEX operating system which can be used to enlarge, restrict or completely redefine the standard virtual machine provided by TENEX. Although it was motivated by the distributed computation work, trapping is a generally useful operating system function. This paper describes the trapping mechanism and records design and implementation decisions that were made in adding it to the existing TENEX operating system.

5. Thomas, R.H., "ARPANET TENEX - A Step Toward a Network Operating System," reprint of presentation at 1974 National Computer Conference and Exposition Panel Session, May 1974, Chicago, Illinois.

This note describes some of the features of the ARPANET TENEX implementation which make it well suited for computer networking. It notes that network communication is through the file system, and explains the benefits of such an approach. It also shows how the process hierarchy, interprocess communication and system call trapping facilities are used to provide service for remote users and an expanded execution environment for local programs.

6. Thomas, R.H. and D.A. Henderson, "McROSS-A Multi-Computer Programming System," AFIPS Conference Proceedings, Vol. 40, June 1972, pp. 281-293.

This paper describes an experimental distributed programming system which makes it possible to create multi-computer simulation programs and to run them on computers connected by the ARPANET. It was one of the first working examples of a distributed system, and served to identify many of the problems which would be encountered in more general systems.

7. Mader, E.R., "Network Debugging Protocol," ARPA Network Working Group RFC #643, July 1974.

This document proposes a protocol to support a PDP-11 network bootstrap service and a cross-network debugger. The protocol is designed for debugging processes running under an operating system which can perform such debugging tasks as placing and removing breakpoints, and single instruction stepping.

8. Mader, E.R., W.W. Plummer and R.S. Tomlinson, "A Protocol Experiment," ARPA Network Working Group RFC #700, August 1974.

This RFC describes an experiment in which a new host-host protocol (Kahn & Cerf, INWG Note #39) was used to drive the BBN computer center line printer which is attached to an ARPANET mini-host. Protocol extensions and modifications which were needed in the implementation are discussed, and other aspects of

the protocol which still require investigati are noted. The RFC also derives equations which model the dat transfer rate for the new protocol.

9. Schantz, R.E., "A Note on Reconnection Protocol," ARPA Network Working Group RFC #671, December 1974.

This note documents the experience gained from implementing a modified, experimental version of the Telnet reconnection protocol option within the context of the RSEXEC. The first section defines a modified reconnection protocol. The second section discusses general network implementation details, while the final section describes aspects of the TENEX/RSEXEC implementation.

10. Schantz, R.E., "A Multi-Site Data Collection Facility," ARPA Netwo Working Group RFC #672, December 1974.

This RFC reproduces a document prepared during the design and implementation of the protocols for the TIP-TENEX integrated system for handling TIP accounting. The first section discusses the general problem of protocols for utilizing multiple servers with respect to reliability and data duplication. The second section details the protocol as applied to TIP accounting data collection.

11. Thomas, R.H., "On the Problem of Signature Authentication for Network Mail," ARPA Network Working Group RFC #644, July 1974.

This note describes the problem of signature authenticity in the network context. It then presents a general approach in which a problem is divided into one of local signature authentication and then network recognition of authorized mail. An implementation of the authentication procedure is given using reserved host/socket pairs.

12. Thomas, R.H., "MLTNET - A Multi-TELNET Subsystem for TENEX," ARPA Network Working Group RFC #339, May 1972.

MLTNET is a TELNET like facility for TENEX which enables a user to control a number of jobs running on different ARPANET hosts It was the prototype for most new TELNET implementations which handle multiple simultaneous transactions. This RFC describes the operation and features of the original multi-TELNET system.

13. Thomas, R.H., "Reconnection Protocol," ARPA Network Working Group RFC #426, January 1973.

This note describes several situations in which it is useful to be e to move one or both ends of a communication path from one host to another. It presents a mechanism to

achieve reconnection, sketches how the mechanism could be added to Host-Host or TELNET protocol, and recommends a place for the mechanism in the protocol hierarchy.

14. Thomas, R.H.. and R.C. Clements, "FTP Server-Server Interaction," ARPA Network Working Group RFC #438, January 1973.

This RFC suggests an extension to the File Transfer Protocol which would allow an FTP user process at one site to arrange for FTP server processes at other sites to act cooperatively on its behalf. Situations where such a facility would be useful are given, and it is shown how the protocol extension is used to handle these cases.

15. Thomas, R.H. (with R. Bressler), "Inter-Entity Communication - An Experiment," ARPA Network Working Group, RFC #441, January 1973.

This note is a status report concerned with the experiments to provide the capability for network users to converse with each other using their consoles. It indicates two such user interfaces for the inter-entity communication, and details the network protocol that was developed for these experiments. Areas for further experimentation are noted.

16. Thomas, R.H. (with R. Bressler), "Mail Retrieval via FTP," ARPA Network Working Group RFC #458, February 1973.

This RFC proposes two new FTP commands which would allow a user to read his mail at one or more sites without incurring the overhead of logging in, and without having to use several different retrieval methods. The commands provide the user with the ability to create a simple program to retrieve mail from multiple sites.

17. Thomas, R.H. (with R. Bressler), "FTP Server-Server Interaction - II," ARPA Network Working Group #478, March 1973.

This note deals with an apparent drawback of the protocol specified for FTP Server-Server interaction. By providing a new command (PASSIVE), the need for queueing RFC's for local sockets before they exist is eliminated. It is shown how this new command integrates into the FTP server-server interaction protocol exchanges.

18. Thomas, R.H. and R.S. Tomlinson (with A. McKenzie and K. Pogran), "A Note on Protocol Synch Sequences," ARPA Network Working Group RFC #529, June 1973.

This note discusses the use and misuse of the TELNET Protocol Synch Sequence. It examines the general notion of synch sequences on communication paths, and then reflects on its meaning and implementation in TELNET. Suggestions for implementing synch sequences in protocols based on TELNET are also given.

19. Thomas, R.H., "Comments on File Access Protocol," ARPA Network Working Group RFC #535, July 1973.

This RFC suggests improvements to a previously proposed file access protocol, which would permit remote access to the contents of files. The improvements are mostly additions to allow for the use of file system features which may be available locally. They include adding append access, providing for files with "holes" in them, using multiple files simultaneously, and acquiring descriptive information about a file.

20. Thomas, R.H., "TENEX Load Averages for July 1973," ARPA Network Working Group RFC #546, August 1973.

This RFC presents utilization data for the BBN and ISI TENEX systems for the month of July 1973. The data is collected as a side effect of the Resource Sharing Executive server programs regular communication with each other. The data indicates a strong "East Coast time based" user population on the ISI machine.

21. Burchfiel, J. and R. Tomlinson (with B. Cosell and D. Walden), "TIP/TENEX Reliability Improvements," ARPA Network Working Group RFC #636, June 1974.

This RFC sketches the plan that was implemented for improving the reliability of conneccions between TIPs and TENEXs and for providing the TIP user with clear messages regarding changes in the state of his connection. Reliability improvements are made possible by specifying host-host protocol additions to provide for connection resynchronization. The protocol changes apply equally well to interactions between hosts of any type.

22. Murphy, D. (with R. Bressler and D. Walden), "A Proposed Experiment with a Message Switching Protocol," ARPA Network Working Group RFC #333, May 1972.

This RFC sketches the organization of a new approach to the host-host protocol problem for the ARPANET. The approach is based on the concept of message switching, and attempts to achieve better system utilization and simpler network software than could be accommodated with the connection oriented approach. The document specifies a message switched protocol for the ARPANET, and includes the motion of an information operator as a general network utility.

23. Burchfiel, J. and R. Tomlinson, "Proposed Change to Host-Host Protocol Resynchronization of Connection Status," ARPA Network Working Group RFC #467, February 1973.

This RFC describes changes to the Host-Host protocol in order to achieve resynchronization on a network connection and handle the "half-closed connection" problem. It is shown how these changes handle the problems arising from host "allocate" messages, host system interruptions and network partitioning. These changes formed the basis of the TIP-TENEX reliability improvement procedures.

24. Johnson, P.R. and R.H. Thomas, "The Maintenance of Duplicate Databases," ARPA Network Working Group RFC #677, January 1975.

This paper describes a technique for maintaining duplicate, distributed data bases in a consistent manner in situations that require a capability for distributed initiation of data base updates. The class of data bases for which the technique is applicable is specified; the allowable update operations are carefully defined; and the "consistency" requirement is carefully specified. The technique depends upon the use of "time stamps" to properly sequence distributedly initiated update commands.

The following Quarterly Progress Reports provide a chronological description of our distributed computation research activities.

BBN Report No. 11505-14, QPR No. 3, August 1971 (Contract No. DAHC15-71-C-0088)

BBN Report No. 11505-14, QPR No. 4, November 1971 (Contract No. DAHC15-71-C-0088)

BBN Report No. 11505-14, QPR No. 5, January 1972 (Contract No. DAHC15-71-C-0088)

BBN Report No. 2356, QPR No. 6, April 1972 (Contract No. DAHC15-71-C-0088)

BBN Report No. 2404, QPR No. 7, July 1972 (Contract No. DAHC15-71-C-0088)

BBN Report No. 2465, QPR No. 8, October 1972 (Contract No. DAHC15-71-C-0088)

BBN Report No. 2501, QPR No. 9, January 1973 (Contract No. DAHC15-71-C-0088)

BBN Report No. 2544, QPR No. 10, April 1973 (Contract No. DAHC15-71-C-0088)

BBN Report No. 2607, QPR No. 11, July 1973 (Contract No. DAHC15-71-C-0088)

BBN Report No. 2670, QPR No. 12, October 1973 (Contract No. DAHC15-71-C-0088)

BBN Report No. 2721, QPR No. 13, January 1974 (Contract No. DAHC15-71-C-0088)

BBN Report No. 2822, QPR No. 14, March 1974 (Contract No. DAHC15-71-C-0088)

BBN Report No. 2869, QPR No. 15, July 1974 (Contract No. DAHC15-71-C-0088)

4. Recommendation

The ARPA/IPT program which resulted in the ARPANET has demonstrated the technical and economic feasibility of interconnecting computers via packet switched techniques. This development of intercomputer communications is analogous to the development of wireless communications in the early 1900's in that it makes previously unthinkable capabilities possible. However, just as the invention of wireless did not automatically lead to its effective use, the advent of computer data communications has not resulted in immediate realization of new capabilities. In fact, the additional complexities of computer-computer interactions compound the already serious problems we are having with software cost control on stand alone computers.

The Department of Defense has a genuine need for distributed, multi-computer systems because of the geographically dispersed and mobile nature of its operations, and its requirements for computing services which are reliably and redundantly supported. A mature distributed computation software technology could be the basis for satisfying these important DoD needs. Unfortunately, no such mature software technology currently exists.

As noted earlier, the research documented in this report can be characterized as exploratory in nature. It has provided a glimpse of what is possible, has suggested the potential for

widespread applicability of distributed computational techniques, and has uncovered specific technical problems which must be solved before such application is practical.

We believe that a coordinated, coherent and intensive research program in distributed computation is required to advance the state-of-the-art to a level capable of supporting DoD requirements.