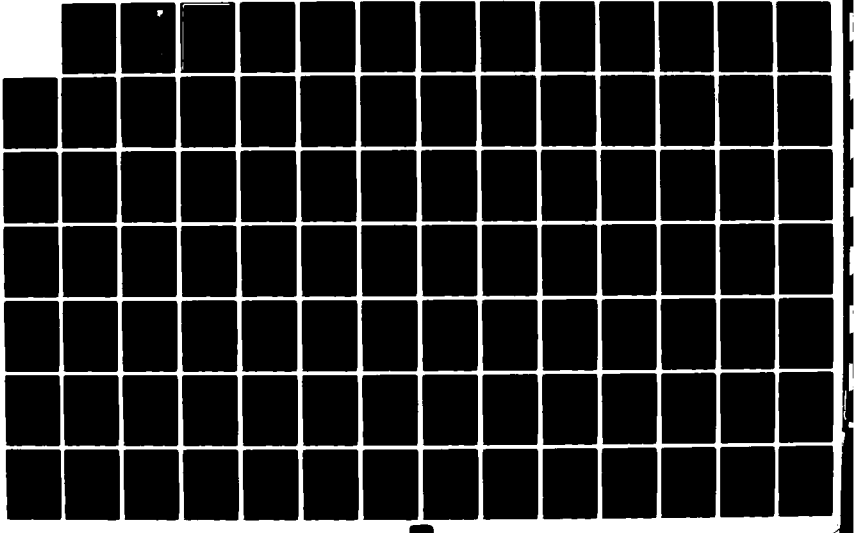
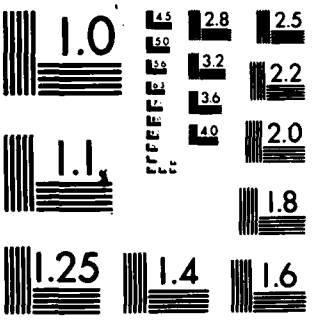
AD-A139 588 CRONUS: A DISTRIBUTED OPERATING SYSTEM(U) BOLT BERANEK AND NEWMAN INC CAMBRIDGE MA R SCHANTZ ET AL. NOV 83 BBN-5086 RADC-TR-83-236 F30602-81-C-0132 1/3

UNCLASSIFIED F/G 9/2 NL





(12)

RADC-TR-83-236
Interim Technical Report #1
November 1983

CONUS, A DISTRIBUTED OPERATING SYSTEM

Bolt Beranek and Newman, Inc.

R. Schantz, E. Burke, S. Geyer, M. Hoffman, A. Lake, K. Pogran,
D. Tappan, R. Thomas, S. Toner and W. MacGregor

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-83-236 has been reviewed and is approved for publication.

APPROVED:

THOMAS F. LAWRENCE
Project Engineer

APPROVED:

JOHN J. MARCINIAK, Colonel, USAF
Chief, Command and Control Division

FOR THE COMMANDER:

DONALD A. BRANTINGHAM
Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COTD) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-83-236	2. GOVT ACCESSION NO.: A139588
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (and Subtitle): CRONUS, A DISTRIBUTED OPERATING SYSTEM	5. TYPE OF REPORT & PERIOD COVERED: Interim Technical Report #1 8 Jun 81 - 30 Jun 82
	6. PERFORMING ORG. REPORT NUMBER: 5086
7. AUTHOR(s): R. Schantz, E. Burke, S. Geyer, M. Hoffman A. Lake, K. Pogran, D. Tappan, R. Thomas S. Toner, W. MacGregor	8. CONTRACT OR GRANT NUMBER(s): F30602-81-C-0132
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Bolt Beranek and Newman, Inc. 10 Moulton Street Cambridge MA 02238	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 63728F 25300107
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (COTD) Griffiss AFB NY 13441	12. REPORT DATE: November 1983 13. NUMBER OF PAGES: 236
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Thomas F. Lawrence (COTD)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Distributed Operating System
Local Area Network
Virtual Local Network
Distributed System

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes the primary functional capabilities which will be designed into the Distributed Operating System to be designed and implemented. The goals include coherence and uniformity in the user model of the system, survivability and integrity of the system function and data, scalability of the architecture, resource management, component substitutability, and operation and maintenance procedures. The concept of a virtual Local Network, making it possible to substitute a variety of

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

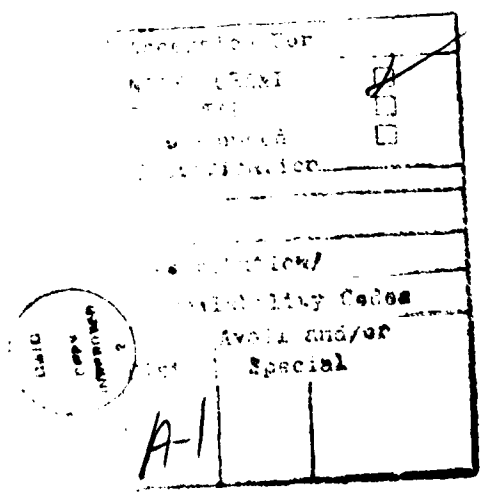
UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

local area network vendor products in the architecture, is described.

Table of Contents

FIGURES



TABLES

1 Introduction

This is the first interim technical report for contract F30602-81-C-0132, entitled "DOS Design and Implementation." The system being developed under this effort has been given the name Cronus.

1.1 Project Overview

The objective of this project is to define, design, implement and test an Advanced Development Model for a distributed operating system. The DOS controls the interactions among collections of computers interconnected via high-speed local area network technology. The overall function of the DOS is to integrate the various data processing subsystems into a coherent, responsive and reliable system. The system is to include the following functions: system monitoring, reliability and survivability, access control and authentication, and a uniform command language. In addition, the system is to provide support for the following system services: uniform file system, electronic mail message distribution, data translation, and interactive access to remote programs.

The project activity can be subdivided into five major categories:

1. Definition of the distributed operating system concept and its functions as they apply to this effort.

2. Selection of predominantly off-the-shelf hardware and software components to represent the foundation of a demonstration DOS system.

3. Design of the DOS conceptual structure and its functional elements.

4. Implementation of the design, culminating with the integration of implementation units into a complete Advanced Development Model for a distributed operating system.

5. Evaluation of the concepts and realization of the DOS in the environment of the ADM by means of test procedures and practical demonstrations.

1.2 Summary of Recent Project Activity

Some of our major accomplishments during the first year of the contract include the following:

Completion of the Cronus Functional Definition and System Concept Report.

- Selection of the local area network technology to be used in this project, including requirements and analysis of selected and non-selected products.

- Selection of the various host components which comprise the processing elements of the DOS, and which are each selected to serve one or more roles in establishing a credible and relevant demonstration environment.

- Procurement, installation and checkout of all system components.

- Start of the first phase design for the distributed operating system. This activity is continuing and is expected to be completed during the next few months.

- Establishing and stabilizing the programming environment for each of the system components.
- Initiating the complete local network interconnect of all system components using 10MB Ethernet through the purchase of network products, customized hardware, and customized software as appropriate.
- Integrating the local network into DoD standard IP and TCP protocol services on each system component.
- Adapting ARPANET internet gateway software to be used as a Cronus access path to the internet and to interconnect multiple Cronus clusters.

In addition to the above work, we participated in and gave presentations to the RADC Technology Exchange meetings held in October 1981 and May 1982. We also gave a presentation to a special local area network meeting convened at RADC for various government agencies. This presentation covered the general subject of local area base band networks, with particular focus on the process of selecting a network for the Cronus project.

1.3 Organization of This Report

Our project has instituted an informal DOS technical note series, as a means of improving intra-project communication, recording key ideas, and making available intermediate results to interested individuals outside the project. The rest of this report describes our work in many of the above mentioned areas in more detail, using the vehicle of a selected collection of DOS notes. Section 2 is the Cronus Functional Definition and System

Concept report. Section 3 is comprised of a number of notes concerning the selection phase of our effort. The notes discuss the selection of the local area network and the concept of a microcomputer based, multipurpose generic computing element. Section 4 includes a description of certain low-level aspects of the system design. One note describes a definition for a virtual local network to be used as a base for substituting other local networks in different instances of Cronus. Another note defines a facility for generating system-wide unique identifiers to be used as global ientifiers for system objects. The final note in this section discusses the interconnection of our cluster with the ARPA Internet using gateway technology. Section 5 includes a number of notes discussing work in progress toward completing the Ethernet/IP/TCP interconnect of the system components. This section includes a note describing the design of a C/70 UNIX Ethernet interface using an off-the-shelf Ethernet component as a base. This same base may be used as a starting point for other custom implementations.

2 Cronus Functional Definition and System Concepts

This section of the interim technical report reproduces the Cronus system functional definition, also available as BBN Report No. 5041.

2.1 Introduction

This report details the functional definition and system concepts for Cronus, the local area network distributed operating system being developed as part of the DOS Design and Implementation project sponsored by Rome Air Development Center. The report is the first project deliverable document, and is intended as an overview of the system which we will be developing under this effort. The functions and system concepts discussed in this report are the results of a consideration of the current state of distributed system technology and potential uses of the system in a wide variety of command and control environments.

This report is not a design document. The design of a system meeting the objectives described in this report will be covered in later reports. However, the nature of the project dictates that many design, implementation and even test and evaluation approaches be made in a coordinated manner with the system definition. Accordingly, these issues are also addressed where appropriate in the present document.

2.1.1 Project Objectives

The purpose of the Distributed Operating System (DOS) project is to develop a distributed operating system for use in command and control environments. The DOS development activity can be subdivided into five major categories:

1. Select the off-the-shelf hardware and software components that represent the foundation of the DOS system.
2. Design the DOS conceptual structure, by defining: a) the functions available to users of the system; b) models for pervasive issues such as reliability, security, and system control; and c) the top-level decomposition of the DOS software components into implementation units.
3. Develop the implementation units defined in (2), until they become complete, functioning programs in the DOS Advanced Development Model (ADM).
4. Integrate the implementation units into a coherent and useful system, both by adjustments to the functional definitions and by any optimizations necessary for acceptable performance.
5. Evaluate the concepts and realization of the DOS in the environment of the ADM, by means of formalized test procedures and practical demonstrations.

The DOS will be designed as a general purpose system to support interactive information processing. Thus, emphasis is placed on adaptability of the DOS structures along several dimensions, for example:

- Reliability: essential services can be provided with high reliability using redundant equipment, or with lower reliability at lower cost;
- Accommodation: there are well-defined paths for integrating any host under any operating system, and any special-purpose device, into the DOS;

- **Scalability:** a DOS cluster can be scaled from a few to several hundred hosts, and adjust to a similar scaling of the user population;

- **Primary use:** appropriately configured, a cluster may be efficiently utilized as a program development system, an office automation system, a base for dedicated applications, or a mixture of all three;

- **Access paths:** the DOS services and applications can be accessed from terminals and workstations attached to a cluster directly, or through the internetwork;

- **Buy-in cost:** hosts and applications can be integrated into the DOS environment in a variety of ways, that offer a range of cost/performance points to the integrator.

The DOS concepts and the software modules that implement the basic system services can be utilized in a wide variety of possible hardware configurations, and in many different operating regimes, to support the requirements of different applications. This *polymorphous* aspect of the DOS makes it difficult to describe concisely; a complete description must examine each of the dimensions of DOS adaptability. This document presents a top-level view of the project objectives and DOS design goals; further detail will be provided in system design documents.

With regard to DOS adaptability, we distinguish between accommodation, the ability of the DOS to incorporate new host types, new constituent operating systems, and new application subsystems (services), and substitution, the replacement of a hardware or software module critical to the provision of DOS essential services. It is a project goal to achieve the widest possible range of accommodation, i.e., to be able to integrate

many types of existing or future host, operating system, or application subsystems within the DOS concepts. Substitution, in contrast, will be much more tightly constrained, because the new hardware or software module must correctly implement the external interface of the old module in order for the DOS to continue to provide essential services. Certain critical interfaces (e.g., the interface to the local network, GCE operating system support) will be carefully defined to make substitution possible. Both forms of adaptability, accommodation and substitution, are important, but we expect accommodation to occur much more frequently.

In general, the DOS design is influenced more by available and projected technology than by the specific requirements of any application, since to do otherwise would violate the general-purpose nature of the DOS.

The temper of the DOS design is pragmatic. The project aims to design, build, and evaluate a complete, useful system over a period of about 3 years. The use of the DOS as a tool for its own implementation is an important incentive to the developers to be on time and down-to-earth.

The following problem areas are not considered to be important project objectives:

1. Development of high reliability or fault tolerant hardware;

2. Development of minimal-cost solutions to distributed processing problems;

3. Research into low-level communications hardware and protocols;

4. Development of support for distributed, real-time applications;

By stating (1) as a non-goal we emphasize the project orientation towards software, rather than hardware, reliability techniques. We note the mention of specific, non-fault-tolerant commercial processors as DOS constituent hosts in the Statement of Work; the implication that non-fault-tolerant machines will often be included in DOS configurations is evidence in support of (1) as a non-goal.

By stating (2) as a non-goal we express a bias towards general-purpose operating system facilities. For some applications, high-volume production (hundreds or thousands of units) may be anticipated; economic pressures will then encourage tailoring the systems to provide the required function at minimal cost per unit. General-purpose systems, on the other hand, tend to provide more functionality than is necessary for any particular application. They are thus more cost effective for small production volumes of application systems (their generality makes programming less costly), and less cost effective for large production volumes (since each replicated system contains unused general-purpose mechanisms). Because simply achieving the required distributed operating system <u>function</u> is to a large

degree still a research problem, we do not believe a major emphasis on cost effectiveness is desirable or even possible at this time.

By stating (3) as a non-goal we recognize the large investments in low-level communication protocols and hardware already made by the Department of Defense and the private sector. In the interests of interoperability and a rapid rate of progress on the other, higher-level issues of distributed operating system design, we will directly utilize the DoD IP and TCP protocol standards and commercial local network technology.

By stating (4) as a non-goal we identify a conflict between the distributed operating system structures required for high performance in real-time systems, and the structures which support a modern, general-purpose computing utility. Again, the project orientation is towards the more general-purpose concepts; however, the presence of individual hosts in a DOS cluster performing real-time processing is entirely within the DOS concept of operation, and is readily supported.

2.1.2 System Environment

To define the focus of the DOS project it is useful to classify distributed systems along architectural lines according

to the physical extent of distribution the systems exhibit. We can identify three major architectural areas of interest:

1. Node Architecture
2. Cluster Architecture (1)
3. Inter-Cluster Architecture

Each of these is related to the emerging technology of distributed systems, but the technology of distribution tends to be different in the three areas, as explained below.

Node Architecture

The development of a processor architecture, configuration, and operating system for a single host or processing node is a large-scale undertaking, usually accomplished by computer manufacturers. A host is typically physically small (can be contained in one room), is designed by computer hardware architects as a single logical unit, and is concerned with maximum event rates of approximately 1 to 1000 million events per second. Although dual-processor nodes have been common for some time, nodes with many-fold internal distribution are just now becoming commercially available. The structure and efficient utilization of such hosts is at the forefront of computer architecture research.

Cluster Architecture

A cluster is a collection of nodes attached to a high-speed local network. At present, technology limits the speed of local networks to approximately 1 to 100

(1) The term "cluster" is used here with the same meaning as in BBN Report No. 4455, "Distributed Operating System Design Study: Phase 1". The term "cluster architecture" is synonymous with the term "MINIDOS" in the OSI Report No. R79-045, "TAC C3 Distributed Operating System Study Final Report"; similarly, "inter-cluster architecture" is synonymous with "MAXIDOS" in the OSI report.

megabits aggregate throughput, and the physical size of the network to a maximum diameter of about 4 kilometers. The host systems are made to work together through the agency of the distributed operating system, which provides unifying services and concepts which are utilized by application software. The maximum event rate at the DOS level is related to the minimum message transmission time between hosts, and is on the order of 10 to 1000 messages per second. The cluster configuration and applications supported by it are typically under the administrative control of one organizational entity.

Inter-Cluster Architecture

An inter-cluster architecture typically deals with geographically distributed clusters (or in the degenerate case, hosts) which are not under unified administrative control. Because of administrative issues and the greater lifespan of inter-cluster architectures, they tend to be composed of parts from many different hardware and software technologies. The communication paths between widely separated clusters have much lower bandwidth and higher error rates than local networks; the maximum event rate for cluster-to-cluster interactions is on the order of 0.01 to 10 events per second. In the inter-cluster case, emphasis is on defining protocols for interactions between clusters, and on the appropriate rules for the exchange of authority (for access to information and consumption of resources) between clusters.

The boundaries between these areas are often indistinct, and sometimes simply the result of unrelated design efforts. Nonetheless each area has a unique set of requirements and solutions for system design. For a given area, these aspects combine to form an outlook that encompasses not just the functional properties of a system, but also many "system level" issues relating to development, administration, training, operations, documentation, and maintenance.

The principle concern for the DOS project will be the development of a system for a cluster architecture. Because a cluster is composed of nodes and connected to other clusters the the relationships between node, cluster, and inter-cluster architectures must be considered in order to produce the DOS cluster architecture. In certain specific but limited regards, problems concerning node or inter-cluster architecture will be important. For example, it must be possible to integrate a wide variety of nodes into the cluster system, and the cluster system must be able to interact with other clusters. Where feasible, the design will accommodate existing standards in the areas of node and inter-cluster architecture. Standardized node components and standardized connections to the internetwork environment will both contribute to the applicability and longevity of the DOS design. However, it is outside the scope of this project to attempt the development of unified approaches to problems of distribution in all three areas, which would involve addressing three different sets of issues.

It is important that the DOS project take full advantage of the best available off-the-shelf component technology. A "component" in this sense may be hardware (e.g., processors and storage devices) or software (e.g., the commercial UNIX or VMS operating systems, and the ARPA-sponsored internet gateway software). The current technological trends should also favor continued development of the components in applications apart

from the DOS project, so that the parallel evolution of node and inter-cluster architectures can potentially benefit the DOS cluster architecture. The DOS project can be expected, of course, to provide useful concepts and services for the other areas; synergism results from a blend of diversity and commonality among the three architectural levels.

2.1.3 System Goals

The overall objective of developing a cluster operating system can be broken down into a number of system design goals along the lines of the characteristics the system should exhibit. The resulting design goals can then be prioritized and used during the design process as a means for focusing the design effort and as a basis for making various design choices.

The system design goals for the DOS, in order of decreasing priority are:

Primary Goals.

1. Coherence and Uniformity.

   To be usable as a system the DOS must implement a coherent and uniform user model.

2. Survivability and Integrity.

   The operation of the system and the integrity of the data it manages must be resilient to outages of system

components.

3. Scalability.

   It should be possible to configure the system with varying amounts of equipment to accommodate a wide range of user population sizes and application requirements.

Secondary Goals.

4. Resource Management.

   The system should provide means for system administrators to establish policies that govern how resources are allocated to user tasks, and it should work to enforce those policies.

5. Component Substitutability.

   The ADM DOS will be built on a specific equipment base. The system should be structured to permit system components to be replaced by functionally equivalent equipment to the largest extent feasible.

6. Operation and Maintenance Procedures.

   The system should provide features which facilitate routine operations and maintenance activity by system operations personnel.

Each of these design goals is discussed in more detail in the sections that follow.

The distinction between primary and secondary goals is methodological, and principally related to test and evaluation. Each primary goal will be the subject of a well-defined evaluation procedure, specified to the greatest extent practical in advance of system implementation. For example, the tests for survivability will include a prescribed set of failure modes to

be artificially induced in the Advanced Development Model, and the behavior of the system recorded. The success of the DOS design in meeting secondary goals will not be so carefully scrutinized; written evaluations will be prepared, but less effort will be spent on planning the evaluations and producing comprehensive tests.

2.2 Coherence and Uniformity

The DOS project aims to develop a coherence and uniformity among otherwise independent application systems and computer services attached to a cluster, in such a manner that the effort required to develop composite applications from existing applications, or to develop new, integrated applications, is as small as practical.

This section discusses "coherence and uniformity" as the phrase applies to the DOS. First, an important dichotomy in the domain of anticipated DOS applications is explained, and the tensions that this dichotomy places on the design process are described. Second, the cluster architecture is described in more detail. Third, several design principles which are the basis of the design process are presented and discussed as they apply to the goal of coherence and uniformity. Finally, specific approaches to some of the issues which are believed to be well

understood at the current time are given.

2.2.1 The Outer System and Inner System Views

The interpretation of the phrase "coherence and uniformity" is ultimately subjective, and should reflect the end-users' opinions of the system concepts and realization. Thus it is fitting that this section begin with a discussion of how the DOS concepts might be used in different applications. Rather than attempt a thorough treatment of the (very large) domain of applications, two important classes of applications are considered in the abstract.

The first class of application views the DOS as an external entity, a supplier of services and communication facilities. This orientation is referred to as the outer system view of the DOS, since the applications already exist or are built outside the context of the DOS concepts of operation. The second class of application is built to run in the DOS context, with full knowledge of the DOS environment and a bias towards its full exploitation. This orientation is referred to as the inner system view of the DOS. The outer system view is most closely related to the problem of achieving connections among existing functional components built on heterogeneous hosts and operating systems; the inner system view should prevail in the design of

new, distributed applications, whether they are built on a homogeneous or heterogeneous base.

We presume that applications satisfying an organization's needs will often be developed independently of each other. During their development, these applications will frequently come to depend upon particular hardware and software objects in their environment, e.g., the host instruction set, the host operating system, and one-of-a-kind peripherals attached to a particular host. The applications may reach operational status with no explicit use of the DOS concepts, and they could be built either on conventional, stand-alone hosts or on a host attached to a DOS cluster.

At some point in time it may be necessary to form a logical connection between application programs which have been developed independently--that is, to achieve interoperability among the functional components. There may be many obstacles to interoperability; a few of the more prevalent and difficult obstacles are:

1. Incompatible data structures;
2. Application interfaces designed for program-to-human rather than program-to-program communication:
3. The absence of a suitable program-to-program communication facility in the host operating system(s);
4. An inadequate structure for the transfer of authority (for access to information and resources) between programs;

5. Poor reliability as the system becomes more and more vulnerable to single-point failures;

6. Poor reliability due to high error rates on communication channels between components;

7. The high cost of performance optimizations involving several complex software components;

8. Disparate software development environments--both automated tools and manual procedures.

In the outer system view, the primary role of the DOS is to reduce these and other obstacles to interoperability, by providing a core of common concepts and functions that become the focus of component interactions.

As an example of the outer system view, suppose there is a need to link a graphics display function executing on a personal workstation to a database management system running on a standard mainframe operating system. Initially, the database management system and the graphics support may have no relationship to the DOS whatsoever, relying entirely upon the hardware and software resources of their own hosts. In order to accomplish the logical link, the hosts must be physically attached to a DOS cluster, communication software must exist on each host, and the applications must be prepared to properly utilize the host-to-host communication path. The DOS can assist this integration by defining the common concepts required for the logical connection to be formed. In this simple example the only requirement is for communication, but in more complex situations the DOS may supply other services (e.g., user authentication, data storage and

encryption, terminal multiplexing).

The inner system view, in contrast, assumes that new applications are constructed within the framework of the DOS and use DOS mechanisms in preference to local host mechanisms whenever practical. A new application designed from the inner system perspective may or may not be distributed, and may be built on homogeneous or heterogeneous machines and operating systems. Whichever the case, by adopting the DOS conventions for process control, file storage and cataloging, and process-to-process communication (among others) such applications avoid many of the interoperability problems listed above. In fact, the process of building an application on the DOS inner system is akin to program construction on a single conventional host, in that the concepts of "process" and "file" and "directory", to name a few, are generally understood by all of the components to mean the same thing. The new application not only achieves uniform connections among its constituent pieces, but also inherits the ability to interact with other inner system tools which also conform to the common concepts. Thus inner system applications enrich the DOS environment in an incremental way, and form the basis for the continued orderly evolution of the DOS environment.

The DOS inner system is unlike a conventional operating system, however, because it addresses issues of distribution--the

development of distributed programs, the possibility of survivable operation through host redundancy, and the potential for configuration scaling beyond the limits of shared memory architectures. These system aspects motivate the development of a powerful and coherent inner system architecture.

Brief examples will reinforce the distinction.

An example of an outer system view might involve two components: a commercial DBMS running on a standard mainframe operating system, and a workstation generating color graphics displays. The objective of the application is to provide a facility for online, color graphics displays of data stored in the DBMS. This is an outer system problem, because the DBMS and mainframe operating system (and quite possibly the workstation operating system) are large and complex objects, maintained by independent organizations, difficult to modify, and were constructed with no awareness of the DOS. The most conservative (least implementation effort) approach might use the DOS only as a communication path, and achieve only minimal integration of the mainframe host into the DOS.

An example of the inner system view might involve the construction of a programming environment for a new microprocessor. The DOS already contains many program development tools--editors, compiler-compilers, linkers, debuggers, etc. By adopting the DOS concepts for process

interaction, many or most of these may be reused. (2)

A fundamental assumption of the DOS project is that both the outer and inner system views are important and must be considered in the design. Because the two views imply different system requirements this represents a burden to the design process.

2.2.2 DOS Cluster Physical Model

Before discussing the major system design principles, the equipment configuration for the DOS cluster is briefly reviewed.

The DOS cluster is composed of three types of equipment:

1. **A communication subsystem.** The subsystem consists of a high-bandwidth, low-latency local network, hardware interfaces between hosts and the local network, device driver software in the host operating systems, and low-level protocol software (the data link layer) in the hosts.

2. **DOS service hosts.** These machines are dedicated entirely to DOS functions, and exist only to provide services to DOS users and applications. In general, they represent modules with specific, system-oriented functions (e.g., archival file storage) and are not user programmable. Requirements for the DOS service host types and operating systems will be specified in the DOS design documents (3)

(2) The UNIX operating system is widely regarded as a good example of the inner system view; shell programming, the "makefile" facility, and other system facilities contribute to the growth of UNIX systems by accretion.
(3) The DOS design will permit the substitution of different service host types for the hosts actually used in the Advanced Development Model; however, any substitution must meet minimal requirements specified in the concept of operation.

3. **Application hosts.** These may be general-purpose hosts (e.g., timesharing machines) providing services to many DOS users, or workstations providing services to one user at a time, or special-purpose hosts (e.g., signal processing computers) required by just one DOS application. Application hosts are often user programmable. In general, they have many characteristics which are not under the control of the DOS; the DOS must be sufficiently flexible to incorporate application hosts of almost any kind.

Application hosts will be connected to the communication subsystem in one of two ways: 1) directly, by means of a host-to-local-network device interface; or 2) indirectly, through an intermediary DOS service host called an access machine. The intent is that standardized access machine software and hardware can reduce the integration cost for a new application host. The electrical interface between the application host and the access machine, for instance, need not be as complex as a local network interface; it need only be mutually acceptable to the two machines. (4) Access machines may have other functions as well; they could play a role in the DOS security model, for example, by isolating untrusted hosts from the (presumed secure) local network. The tradeoffs arising in direct and indirect host integration are not presently well understood; an exploration of this topic is a DOS project goal.

General-purpose application hosts will usually operate with

(4) As a concrete example, the access machine planned for the Advanced Development Model will utilize the HDLC protocol over an RS-422 or RS-423 machine interface.

standard operating systems (e.g., a Digital Equipment Corporation VAX computer running the VMS operating system) which are enhanced and/or modified to integrate the host into the DOS. Thus application hosts will support some DOS software components, at a minimum those required for communication with DOS service hosts. Some DOS services may also be partially or completely implemented on application host to realize performance advantages (by locating applications and required DOS services together) or cost advantages (through resource sharing).

2.2.3 Design Principles

2.2.3.1 Provide Essential Services System-Wide

At the heart of the DOS concept is the availability of selected, essential services to all of the applications in the DOS. The coherence and uniformity of the DOS is directly enhanced when applications and application host operating systems embrace the DOS-supplied services as the single source of these services. To the extent that applications and application host operating systems choose to utilize parallel but incompatible services, coherence and uniformity is lost.

At this time the essential services are believed to be:

- User access points (terminal ports, workstations) providing a uniform path to all DOS services and applications;

- Object management (cataloging and object manipulation) for many types of DOS objects;
- Uniform facilities for process invocation, control, and interprocess communication for application builders;
- Cluster-wide user identifiers and user authentication as the basis for uniform access control to DOS resources;
- Cluster-wide symbolic name space for all types of DOS objects;
- A standard interprocess communication (IPC) facility supporting datagrams and virtual circuits;
- A well-designed user interface that provides access to all DOS and application services;
- Input/output services for the exchange of data with people and systems apart from the DOS;
- Host monitoring and control services, and additional mechanisms needed for cluster operation.

2.2.3.2 Utilize Recognized and Emerging Standards

The DOS design will incorporate recognized and emerging standards whenever practical at many levels of the system. The adoption of standards both enhances the uniformity of the system and contributes to the likelihood of pre-existing, compatible interfaces. The longevity of the DOS concept of operation is extended by attention to standards that are the foundation of contemporary research and development activities; the possibility of interaction with other projects to the mutual benefit of both is maximized.

2.2.3.3 Preserve Choices

The DOS design will preserve choices for the application host integrator and the application builder.

There is a complex tradeoff between the cost of host and application integration into the DOS, and the uniformity and power achieved as a result of the integration. Although many issues involved in the tradeoff have been identified, the problem is not sufficiently well understood to make prescriptions confidently. Investigation of this problem is an important objective of the DOS project.

Part of the project's approach is embodied in Principle 3. This principle requires that the DOS concept of operation accommodate not just one, but a range of possible cost/uniformity points.

Similar tradeoffs exist among the DOS services supplied to application programs. For example, this principle applied to interprocess communication implies that neither datagram nor virtual circuit service is sufficient for all applications; the DOS should provide both types of communication service.

In general, this principle requires that the DOS design address the problem of how DOS installations will adapt to very different configuration and application requirements.

2.2.4 Specific Approaches

2.2.4.1 The Communication Subsystem

A high-bandwidth, low-latency local network (5) is the backbone of the DOS. The DOS concept of operation will specify the interface to the local network, so that alternate local network technologies can be substituted for the particular local network chosen for the Advanced Development Model, if they meet the interface specification. The interface specification will be as unrestrictive as possible, so that substitution is a real possibility.

The local network will permit every host to communicate with every other host in the DOS cluster, and will provide an efficient broadcast service from any host to all hosts. The local network interface specification may further restrict the minimum packet size, addressing mechanism, and other local network properties.

2.2.4.2 Generic Computing Elements

The concept of a Generic Computing Element (GCE) is important to the DOS design (6) . A GCE is an inexpensive DOS

(5) See DOS-Note 21, "DOS Local Network Selection".
(6) See DOS-Note 17, "A Generic Computing Element for the DOS Advanced Development Model".

host that can be flexibly configured, with small or large memory, and with or without disks and other peripherals, as shown in Figure 3. GCE's will be configured for, and dedicated to, specific DOS service roles, such as terminal multiplexing, file storage, access machines, and DOS catalog maintenance (7) .

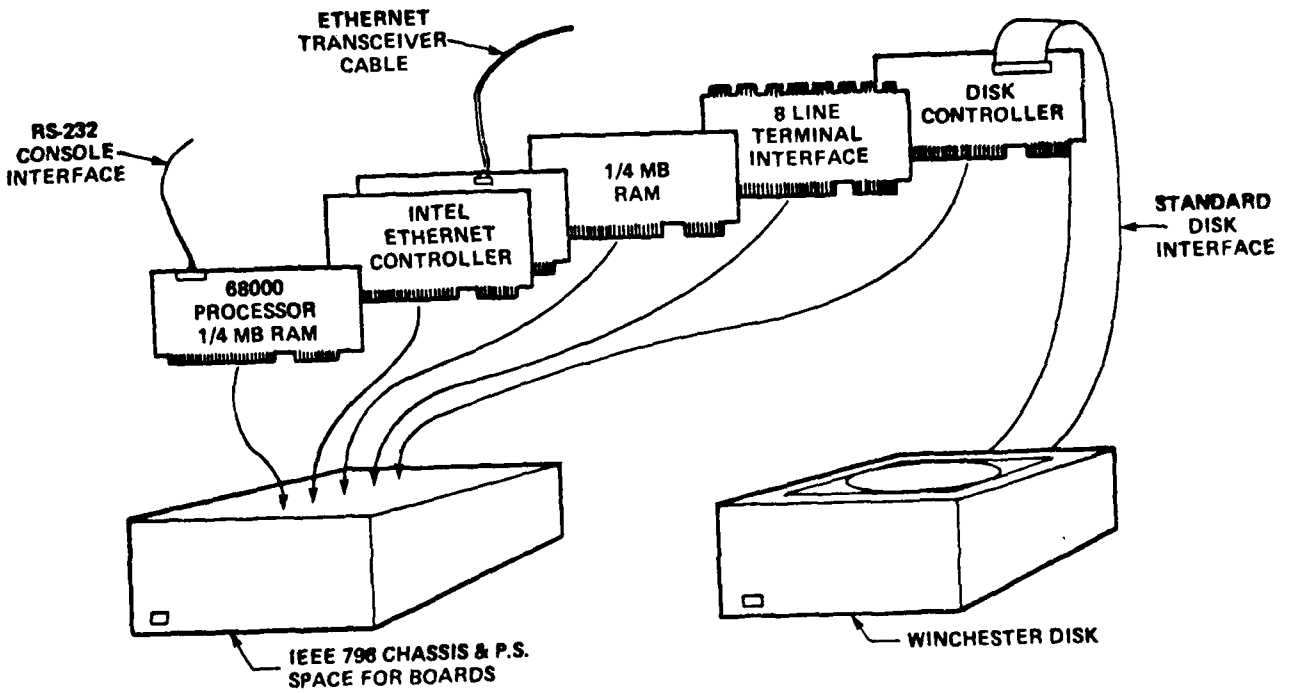
The GCE's are the basis for implementing the essential DOS services in a uniform, application-host-independent manner. Because the DOS design will specify the properties of GCE's and also the software components (8) running on them, it is possible to control the performance and reliability characteristics of the essential DOS services. A configuration consisting of the local network, some number of GCE's, and supporting the essential services represents the minimum useful DOS instance.

Application programs can be constructed above the GCE hardware and operating system; a single GCE host may support DOS services or user applications, but not both.

2.2.4.3 Standards Applicable to DOS Components

The DOS design will utilize recognized standards in several key areas; these directly contribute to both the coherence of the

(7) A single GCE may support several DOS services simultaneously.
(8) Perhaps the most important software component is the GCE operating system, CMOS.



The Generic Computing Element
Figure 1

DOS and interoperability with other computer systems. The standards which have been identified as pertinent as of this time are:

1. **IP and TCP internet protocol standards.** To the maximum extent possible, IP and TCP will be used for host-to-host communication within the DOS cluster.

2. **ARPA standard gateway.** The gateway between the ADM cluster and the ARPANET will be an LSI-11 based, ARPA standard gateway, developed and supported by BBN.

3. **Ethernet.** From the hosts' point-of-view, the local network in the Advanced Development Model will match the Ethernet transceiver cable compatibility interface (9) .

4. **IEEE 796 bus.** The GCE hardware selected for use in the Advanced Development Model is based on the IEEE 796 bus standard for circuit board interconnection.

5. **HDLC and RS-232C.** These communication standards will be used to connect hosts and terminals, respectively, to GCE's within the cluster.

6. **The programming language Ada.** The military standard language Ada will be exploited to the greatest extent practical. (10) It's use will be determined by timely completion of activities not under the control of the DOS project.

Other standards may be applicable to DOS components and are being considered for adoption by the project. Two areas in which existing standards will probably be adopted, rather than developed by the project, are the format of electronic mail messages and the interface between GCE's and mass storage

(9) As noted above, the DOS concepts will not depend upon any local network properties which are peculiar to the Ethernet; Ethernet-compatible devices will, however, be easily added to the Advanced Development Model.
(10) See DOS-Note 16, "Some Thoughts on the Selection of a DOS Implementation Language".

modules.

2.2.4.4 Flexible Application Host Integration

When a new host is integrated into a DOS cluster, it will assume one of several possible host roles. The host roles will occupy different points along the spectrum of integration cost versus degree of adherence to the DOS unifying concepts. System administrators are thus presented with a choice of integration paths, and can tailor host roles to the needs of specific applications.

When a host is integrated with minimum effort, little more than a communication path between the host and other entities in the DOS cluster will be present. This host will be able to obtain many DOS essential services through the communication path, but its resources may be unavailable to other DOS processes. Further effort must be devoted to assimilate the host partially or fully into the DOS object catalog, process model, and reliability mechanisms.

As defined above, the access machine concept is closely related to the effort required for host integration. Minimal effort integration will most likely be achieved through the use of access machines. This host integration path will probably result in lower throughput between the host and the network due

to the presence of the access machine, but may be a desirable approach on balance. For special purpose devices with limited programmability, access machines may play the dual role of device controller and DOS interface.

The host role is decided anew for each host in a cluster. It is possible, for example, for two hosts which are physically the same type of machine and which run the same operating system to be integrated to assume different roles.

2.2.4.5 Comprehensive DOS Object Model

The DOS concepts will revolve around a group of basic object types: files, processes, hosts, users, and messages, to name a few of the more important. The DOS design will attempt to treat all of these types (and others) uniformly, in accord with an abstract object model. The abstract object model recognizes that an object may be designated by one of three varieties of name:

1. **Universal Identifier (UID).** A UID is a fixed-length bitstring. Every object in the abstract object model has a unique UID, over the set of all objects in the cluster and the entire lifetime of the system. A UID is always an acceptable designator for an object within the DOS.

2. **Address.** An address is a bitstring composed of a sequence of address portions. Each successive portion serves to narrow the set of objects designated by the address; the complete address refers to a single object.

3. **Symbolic Names.** People use symbolic object names to designate DOS objects. Symbolic names can be context

> dependent (for example, relative to a directory) or context independent. The symbolic name space is hierarchically structured so that the logical grouping of related objects is reflected in a similarity among their context independent symbolic names. An object need not have a symbolic name.

Normally, people will refer to objects using symbolic name's, and programs will refer to objects using UID's, addresses, and symbolic names. The system will provide translation services, the most important supported by the object catalog, to translate among the three representations of object names.

UID's, addresses, and symbolic names will be used in different ways within the DOS. A UID is always a sufficient object name, even for objects which can move from host to host (11) , because it is completely context independent. An address will usually represent the fastest access path to an object, because its representation explicitly contains the routing information needed to reach the designated object. Symbolic names are most suitable for the user interface, but because the other object designators are available programs need not deal, in general, with variable-length symbolic object names.

A mechanism will be developed for constructing new, composite abstract types from previously defined types. This will allow objects with rich semantics to be built from simpler

---

(11) The DOS does not, in general, support movement of arbitrary objects from one host to another; some specific object types will give rise to mobile objects, however.

objects; for example, a "reliable" file could be assembled from several primitive files on different hosts, containing redundant copies of the same information.

2.2.5 A Summary of the DOS Architecture

The commitment of the DOS design to support a wide range of equipment configurations makes it difficult to give a concise description of "the DOS". The system will have widely varying characteristics for different DOS equipment configurations. It is possible, however, to identify three levels of "DOS product" which may help to clarify the boundaries of the design.

2.2.5.1 Level 1: The Minimal System

The minimal DOS system consists of the local network, a small number of GCE's supplying essential services, and a host integration guide which explains how the owning agency can integrate their own hosts into the DOS environment.

The minimal system supports the user registration and authentication functions, and the essential services pertaining to the user interface, the object model, the cluster gateway(s). It also supports the basic system monitoring and control functions present in any DOS instance. By itself, it does not

provide a user programming environment, or the utilities (electronic mail, text preparation, etc.) found in most timesharing environments.

2.2.5.2 Level 2: The Utility System

The utility system consists of the minimal system, plus one or more fully-integrated, general-purpose timesharing hosts called utility hosts (the C/70 computers will play this role in the Advanced Development Model). The utility system will be suitable for developing new applications in the framework of the DOS, and will support the utilities typical of a modern timesharing system. The utility system will also support the maintenance of its own software, and the software of the minimal system.

2.2.5.3 Level 3: The Application System

The application system consists of the minimal system and some number of application hosts, workstations, and special-purpose devices. An application system may simultaneously be a utility system, if utility hosts are present in the cluster. Applications are generally developed in a utility system and operate in an application system. Application systems,

therefore, need not be capable of supporting their own software development. Application systems are sometimes configured with redundant components and operated in a high reliability mode. Note that GCE's can be used for application programming; thus a particularly simple application system consists of just the network, the GCE's required to provide essential services, and some number of application GCE's.

Figures two and three illustrate the components and the context of the initial system configuration for the Advanced Development Model being assembled at BBN.

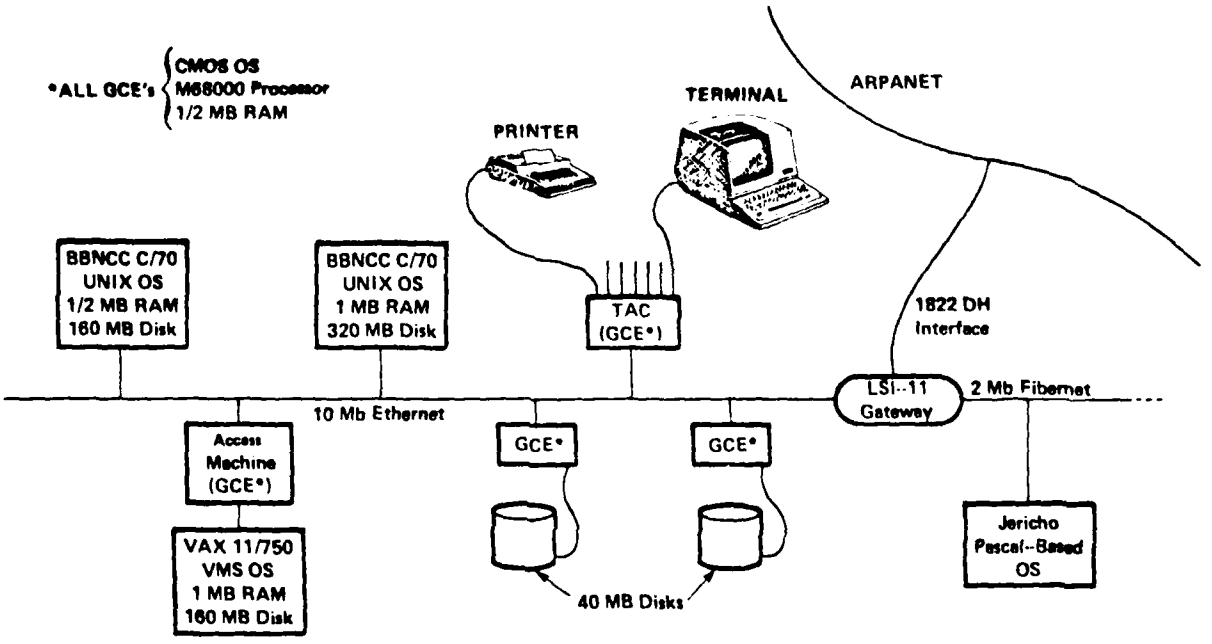
2.3 The DOS Functions and Underlying Concepts

2.3.1 Introduction

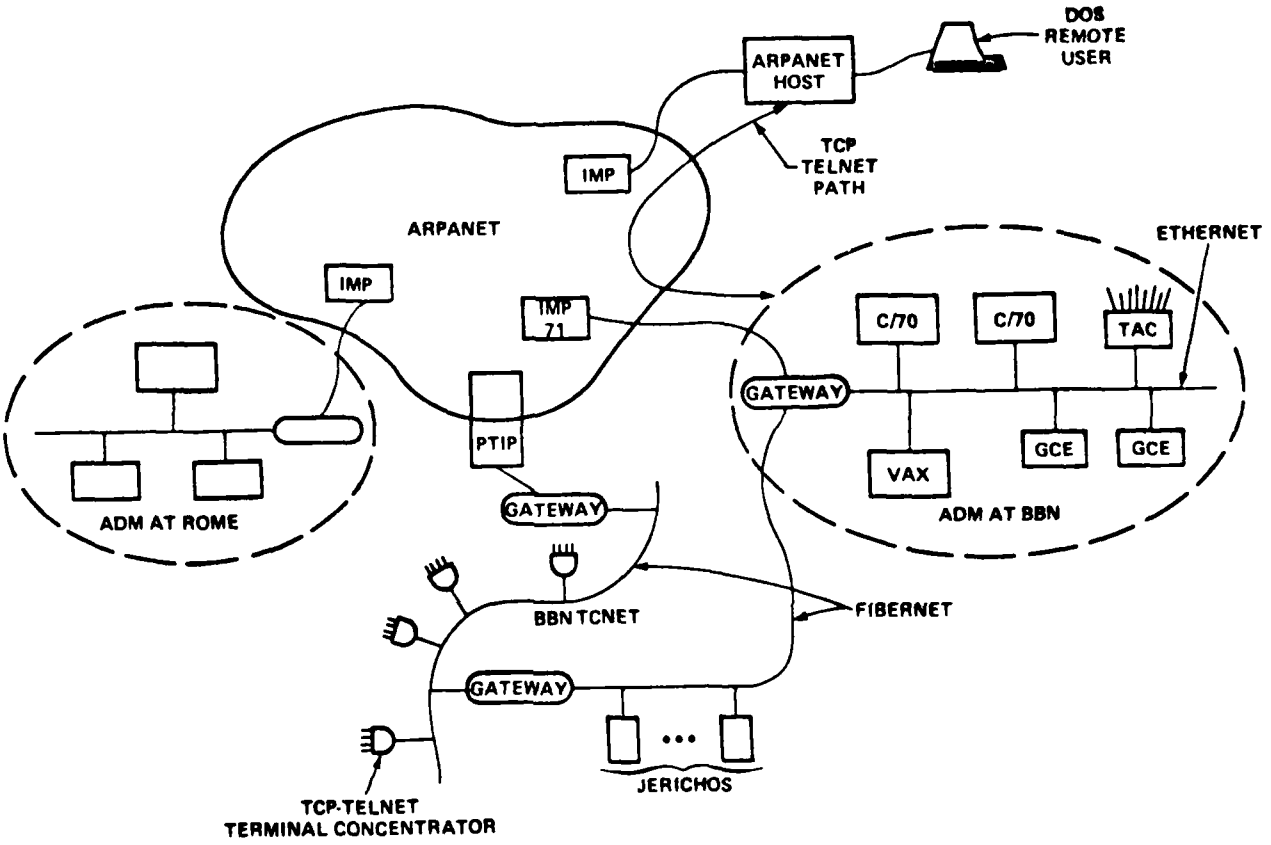
Expected usage of the DOS can be divided into five categories:

1. Applications;
2. Application development and maintenance;
3. System administration;
4. System operation;
5. System development and maintenance.

The system is intended primarily to support end application usage (1). However, to adequately support end applications it must also support the other categories of use.



The Local Cluster Configuration
Figure 2



The InterCluster Environment
Figure 3

Therefore, it should be possible for users working in each of these cases to perform their responsibilites by means of the DOS. The goal of supporting these usage categories places requirements on the functions the DOS must implement, and on the tools it must be able to accommodate. This section discusses the DOS functions.

The DOS system will provide functions in the following areas:

- **System access.** The objective is to support flexible, convenient access to the system from a variety of user access points.
- **Object management.** The notion of a "DOS object" is central to the user model for the DOS. The DOS treats resources, such as files, programs and devices, as "objects" which it manages, and which users and application programs may access. The objective of the object management mechanisms is to provide users and application programs uniform means for accessing DOS objects.
- **Process management.** Like the object abstraction, the "process" abstraction is central to the user model of the DOS. In addition, it is useful as an organizing paradigm for the internal structure of the DOS. The objective of the DOS process management mechanisms is to implement the "process" notion in a way that enables processes to be used both to support the execution of application programs for users and internally to implement DOS functions.
- **Authentication, access control, protection, and security.** The objective is to provide controlled access to DOS objects.
- **Symbolic naming.** DOS users will generally reference objects and services symbolically. Symbolic access to DOS objects will be supported by means of a global symbolic name space for objects.
- **Interprocess communication.** The objective of the interprocess communication (IPC) facility is to support

communication among processes internal to the DOS, and among user and application level processes.

- **User interface.** The user interface functions provide human users with uniform, convenient access to the features and services supported by the DOS resources.

- **Input and Output.** The objective here is to provide flexible and convenient means for users and programs that act on the behalf of users to make use of devices such as printers, tape drives, etc.

- **System monitoring and control.** The purpose of the system monitoring and control functions is to provide a uniform basis for operating and manually controlling the system.

The principal goal for the DOS in each of these functional areas is to support features that are comparable to those found in modern, conventional, centralized operating systems, such as Unix, Multics, VMS, and TOPS-20.

The development of radically new types of operating system functions and concepts, except for those required to deal with the distributed nature of the system, is not a major goal of the DOS effort. This position is motivated by two considerations:

1. It is important to avoid innovation in too many areas when building a system. The important innovations embodied by the DOS will result from addressing problems posed by distribution. These problems span the functional areas identified above. Therefore, most of the effort must be directed toward making the system operate in a coherent, survivable and efficient manner in a distributed environment rather than toward developing new operating system concepts.

2. However, unless the functions provided are comparable in power and convenience to those found in centralized systems, users will not choose to use the DOS. Thus, it is important for the success of the DOS as a system that it provide state-of-the-art capabilities.

The rest of this section discusses the functional areas identified above in terms of our objectives in each area, and sketches some of the concepts and principles that underlie our approaches for achieving the objectives.

Each functional area is discussed in a separate section. However, it will become clear from the discussion that these functions are not independent of one another. These interrelationships occur across functional areas as well as within them. For example, objects and processes are intimately interrelated. A process is a type of DOS object, and access to DOS objects is supported by interactions among processes. Furthermore, internally the system is structured to combine lower level functions and capabilities in one or more areas into higher level functions and capabilities. For example, the relatively higher level notion of reliable (multiple copy) file objects is implemented by more basic (single copy) file objects.

This internal "involuted" structure of the system is important. If the structure and interrelationships are designed well, implementation can proceed in orderly and efficient stages from the lower levels to the higher ones. Furthermore, the resulting system implementation will exhibit internal order, making it easier to maintain and to modify for adapting to new requirements.

2.3.2 System Access

The objective in this area is to provide users with flexible, convenient access paths to the system.

The system will support a number of different types of access points including:

1. **Terminal access computers (TACs).** A TAC is a terminal multiplexer connected directly to the DOS local area network. It acts to interface a number of user terminals to the DOS. The software that runs on a TAC is entirely under the control of the DOS. User programs are not premitted to run on a TAC computer.

2. **Dedicated workstation computers.** A workstation is a computer that is, at any given time, dedicated to a single user. Workstations will be connected to the DOS local network. Workstation hosts have sufficient processing and storage resources to support non-trivial application programs, such as editors and compilers, and to operate autonomously for long periods of time. A workstation may serve as its user's access point to the DOS. User programs may run on a workstation.

3. **The internetwork.** The DOS local network is connected to the internetwork by means of a gateway computer which is a host on the DOS local area network. Users remote from the DOS cluster may access the DOS through the internetwork. Remote terminal access is accomplished by means of a standard terminal handling protocol (TELNET) which operates upon a lower level, reliable transport protocol (TCP).

Because of the distributed nature of the system, user interaction with the DOS is supported by software that runs on one or more computers. This software includes two principal modules. One module is responsible for handling the user's terminal. Since this module will often run at or very "near" the user's access point, we shall call it as the "access point

agent". The other principal user interface module interacts with the user at a higher level to provide access to DOS resources in response to various user commands. We shall call this module the "user agent". It is useful to think of the access point agent and the user agent as processes. These agent processes interact with other components of the DOS and with each other by means of well defined interfaces and protocols. In addition, they play an important role in insuring the reliability of user sessions.

The access point for a user session, in part, determines where the access point agent and user agent processes run. For a user whose access point is a TAC the access point agent runs on the TAC, and the user agent runs on a shared host. The access point agent for a user with a dedicated workstation runs on the user's workstation computer, and the user agent may run on the workstation or it may run on a shared host. Users who access the DOS through the internetwork are allocated user agents that run on shared hosts, and their access point agents may run either on the (non-DOS) host used to access the DOS or on a host within the DOS cluster.

Some DOS hosts may provide support for terminals directly connected to them. It will be possible for users to access the DOS through such directly connected terminals. These users will be treated much like users who access the DOS through the internetwork in the sense that the DOS will allocate user agents

for them that run on shared hosts.

The standard user interface software (for users accessing the DOS through TACs and the internetwork) will be written to operate with CRT terminals that have cursor positioning capabilities; in particular, this includes terminals that meet a subset of ANSI standards X3.41-1974 and X3.64-1977, providing cursor positioning and various other functions such as clear to end of line, delete line, insert line, etc. More capable terminal devices (e.g., workstations with graphics displays) can emulate the standard terminal device to obtain a compatible user interface. In addition, a means will exist for users with other less capable terminal devices (e.g., printing terminals) to access the system (e.g., by using the TELNET Network Vitual Terminal or NVT as a lowest- common denominator terminal device). In the latter case, some sacrifice in the quality, uniformity, and power of the user interface is unavoidable. The user interface is discussed further in Section 3.8.

2.3.3 Object Managememt

The DOS will support a wide variety of objects. The objective of the DOS object management mechanisms is to provide access to DOS objects.

DOS object management will be based on the following

principles:

- Every DOS object has a unique identifier. At the lowest level within the system, access to a DOS object can be accomplished by specifying its unique identifier and the desired access to an "object manager" process for the object.

- The DOS will support a collection of transaction-based object access protocols. These protocols will be type dependent in the sense that there will be different access protocols for different object types.

- Access to objects will be accomplished by engaging in the appropriate access protocol with an object manager process for the object. The interactions between the accessing agent and the object manager will be accomplished by means of interprocess communication (See Section 3.7).

- Input/output devices will be treated as DOS objects. Consequently, input/output devices will have object managers, and access to the devices will be accomplished by means of interprocess communication.

- The DOS catalog (See Section 3.6) provides a means of binding symbolic names to DOS objects. The catalog supports a lookup function (a symbolic name-to-unique id mapping) which enables objects to be accessed symbolically.

- The DOS will support a fixed set of basic object types (such as "primitive" file, "primitive" process, etc.). In addition, it will support more complex object types (such as "multiple copy" file, "migratable" file, etc.) which will be built upon the properties of the basic object types. Our design objective at this time is to develop the framework for supporting more complex object types, rather than to try to specify the semantics of those object types.

Files are a particularly important type of DOS object. The storage resources of dedicated DOS hosts as well as certain constituent hosts will be used to store DOS files. Symbolic naming for DOS files will be implemented by the DOS catalog.

Each host that provides storage for DOS primitive files will

also support the object manager which implements the DOS access protocol for primitive files.

2.3.4 Process Management

As suggested above, the DOS will support the notion of a process. Processes will be used both by the implementation of the DOS and to directly support user applications. For example, there will be processes responsible for implementing the DOS object catalog and for implementing the DOS file system. In order to support user processing activity, there will be processes that execute standard tools, such as text editors and language processors, as well as specific command and control applications.

The objective of the DOS process structure mechanisms is twofold:

1. To support the process concepts required to implement DOS functions; for example, object management.
2. To provide a basis upon which to develop means for users to initiate and control processing activity within the DOS.

DOS process management will be based on the following principles:

- A basic type of process ("primitive" processes) will be implemented at a fairly low level; it will be bound to a particular host, and it will bear no special relationships

or capabilities with respect to other primitive processes.

- Primitive processes are DOS objects. As such, they have unique identifiers, and may be catalogued in the DOS catalog (See Section 3.6). So called "server" processes that provide services useful to a wide range of clients are examples of processes which are useful to catalog. Cataloging such a process enables it to be referenced symbolically by the general population of client processes.

- More sophisticated process notions will be built upon the primitive process notion. For example, the notion of hierarchical process structures, where processes are related to one another according to the manner in which they were created, and where the relationship between processes determines the types of operations a process can perform on other processes, will be built upon the primitive process notion. Similarly, "migratable" processes (processes that can move from one machine to another) will be built upon primitive processes.

- The system will support the notion of "long lived" processes. A long lived process is one which the system will take steps to ensure exists over shut downs and restarts of the system and of individual hosts. Server processes will frequently be long lived.

- Process i/o and interprocess communication will be handled in an integrated fashion. The notion of "primary" input and output streams for a process will be supported, and it will be possible to "link" processes together by connecting the input stream of one process to the output stream of another. Among other things, this will make it possible for one process to act as a filter or translator for the stream of data passing between two other processes.

2.3.5 Authentication, Access Control, and Security

The objective of the DOS in this area is to provide for controlled access to DOS objects. The purpose of the DOS access control mechanisms is:

1. To prevent the unauthorized use of DOS objects. For

example, it is important to ensure the privacy of sensitive data by preventing unauthorized users from accessing it.

2. To ensure the integrity of DOS objects. The objective here is to control the ways in which various objects may be used.

Convenient and flexible means should be available to users for specifying the types of access other users may have to their objects.

The access control mechanisms will be designed to be strong enough to protect the privacy and integrity of DOS objects against accidental disclosure or misuse, and against attacks by malicious, but inexpert users. It is extremely difficult to protect against attacks by dedicated expert users, and it is not a primary goal for the DOS to be invulnerable to such attacks.

There are two capabilities related to protection and security that are not goals for the DOS:

- **Prevention of denial of service.** Denial of service occurs when a user prevents or interferes with someone else's use of the system or parts of it. A simple example would be a user who seizes all the "job slots" on a timesharing system by logging in many times, thereby preventing others from accessing the system. Another example would be the situation that might occur if a user could run a program that floods the local area network with packets. This would prevent other users from using the network. Although the DOS will be able to prevent certain types of denial of service, including those just described, it is very difficult, in general, to comprehensively prevent denial of service.

- **Implementation of the military security model.** The DOS will not implement multi-level security. The DOS would run in a "system high" mode if it were used to process

classified data. The DOS access control mechanisms could be used, however, as a support for the Need-To-Know security model, just as access control in commercial single-host operating systems is used for this purpose.

Internally, the DOS will be organized so that much of its operation is accomplished by means of processes. Many of these internal DOS processes may be thought of as agents which act to carry out user requests. The principal DOS access control mechanism will be based on the identity of the agent attempting to access an object. An important part of access control procedures within the DOS will be to determine the identity of the accessing agent and the identity of the user on whose authority the agent is acting. Consequently, reliable authentication of users and processes will be an important element of the DOS access control mechanisms.

The DOS protection and security mechanisms will be based on the following principles:

- Each DOS user will have his own unique identity which is understood across the entire DOS system.
- Users of the DOS will be required to login once per user session. In most cases access to DOS resources during a session will not require additional "logins" that involve explicit user participation.
- User login will be accomplished in the conventional manner by supplying a valid user login name and password.
- User passwords that are stored within the system will be protected by means of a one-way (i.e., non-invertible) transformation. A password check will be performed by first applying the transformation to the password supplied by a user and then comparing the result with the transformed password for the user that is stored by the

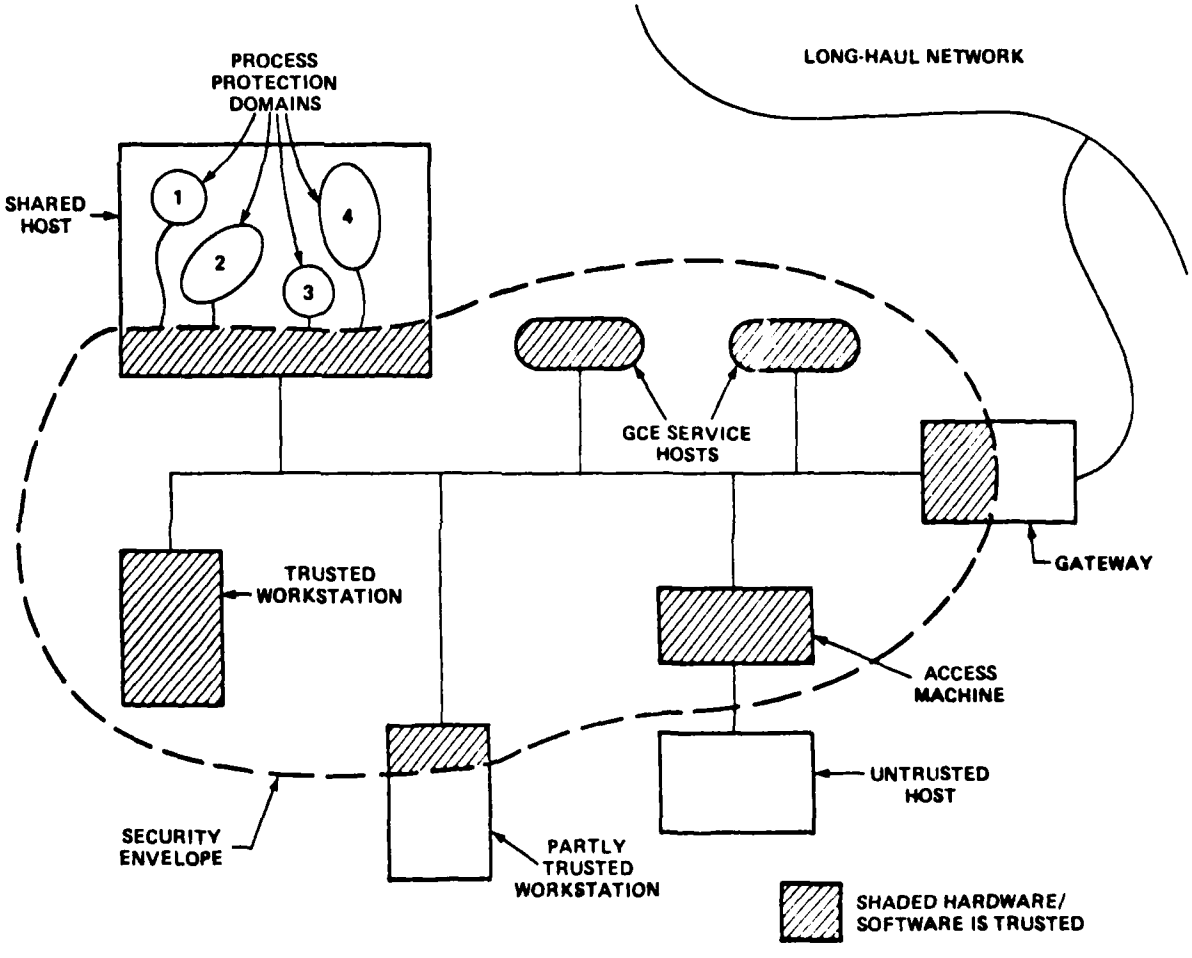
system.

- All attempts to access DOS resources will be subject to access control checks prior to access.

- All attempts to access DOS objects, including those initiated from access points which are external to the DOS, will be treated by the system as being made on behalf of some registered system user. In order to enforce the appropriate access controls the object managers for DOS resources must be able to obtain the identity of the registered user from the accessing agent or to determine it from information supplied by the accessing agent.

- We assume the existence of a "security envelope" which surrounds the DOS local area network and some of the key DOS components (see Figure 4). The security envelope protects the network in the sense that access to the network is controlled. This control is accomplished by means of physical security, system hardware, and DOS software. Unauthorized users (or hosts) are not free to listen to communication on the network, and are not free to send arbitrary packets. DOS components which are within the security envelope may trust each other, and processes outside of the security envelope are not able to masquerade as trusted processes.

Figure 4 shows the possible relationships between hosts and the security envelope. A shared host (typically a timesharing, application host) will participate in the DOS access control mechanisms by means of augmentation to its trusted "monitor" or "supervisor" processes. Generic Computing Elements which supply DOS essential services will be wholly contained within the security envelope, i.e., untrusted applications are not permitted to directly alter the programs resident in system GCE's. Gateways attached to the cluster must "protrude" through the security envelope, because they connect the trusted local network to the untrusted internet; at a minimum, gateways must explicitly mark all traffic entering the cluster as "foreign", in a



The DOS Security Envelope
Figure 4

trustworthy manner. Access machines may be used to connect completely untrusted hosts to the cluster. In this case the access machine would validate all interactions between the untrusted host and the DOS components inside the security envelope. Workstations attached to the DOS may either be fully trusted, and hence inside the boundary of the security envelope, or partially trusted. A partially trusted workstation is presumed to contain some tamper-proof hardware and software components that protect the DOS from anti-social behavior on the part of the workstation.

It is desirable to provide means for a user of the DOS who has the ability to access a particular object to pass (perhaps limited) rights to access that object outside of the DOS cluster. This would enable a user of the DOS to permit others who are not registered DOS users to access specific DOS objects in a controlled fashion. The absence of this feature on ARPANET hosts is a considerable impediment to sharing across host boundaries.

This will be accomplished by a mechanism which will enable a DOS user to create a "capability" for a particular object (e.g., the ability to read a certain file) and then pass the capability on to someone else. When a request to access an object is accompanied with the capability for the object, the DOS may grant access to the object after checking the validity of the capability. To ensure that this feature does not compromise the

privacy and integrity of DOS object, capabilities must be such that they cannot be forged. To help ensure that registered DOS users can be held accountable for their actions, it is desirable that the DOS be able to deduce the identity of the user who created a given capability.

2.3.6 Symbolic Naming

Naming is an important unifying concept for the DOS. The means provided for naming objects is one of the most important factors determining how easy and convenient a system is to use.

The DOS will implement a global symbolic name space for DOS objects. This name space will have the following properties:

- The symbolic name for an object will be independent of the object's location within the DOS.
- The symbolic name used to refer to an object will be the same regardless of the location within the DOS that the name is used.
- Common syntactic conventions will apply to symbolic names for different types of objects (including files, devices, server processes, etc.).

The symbolic name space will be implemented by means of a DOS catalog data base (or simply "catalog"). The catalog will implement a symbolic name-to-object mapping for the DOS objects it catalogs. The catalog will not usually store the objects themselves, but rather will store information about the objects.

Information about an object will be stored in a catalog entry for the object. This information will be sufficient to allow access to the object. In particular, the catalog will store the global unique identifier for each object it catalogs along with any additional information required to locate the object within the DOS. In addition, it will also maintain certain attributes of objects it catalogs.

While in some sense the catalog can be thought of as a logically centralized data base, it will be implemented in a distributed fashion. In particular, the catalog will be dispersed among a number of DOS hosts and some parts of it may be replicated. It will be dispersed to ensure that the system is scalable and that the catalog is reliable. While all of the information in the catalog, even for very large configurations, might fit on a single DOS host, it seems unwise to store it on a single host. In large configurations the load placed on that host would likely represent a performance bottleneck. Furthermore, the cataloging functions would be vulnerable to a failure of that single host. Parts of the catalog will be replicated to ensure high availability of critical catalog data.

The symbolic name space and its supporting catalog will be based on the following principles:

- The name space will be hierarchical. The name space hierarchy can be thought of as a tree with labeled branches.

  - o The leaves (terminal nodes) of the tree represent cataloged objects.

  - o The symbolic name for an object is the name of the path from the root node of the tree to the node that represents the object.

  - o Non-terminal nodes of the tree represent collections of catalog entries and are called "directories".

  - o Directories are DOS objects, and they have names. The name of a directory is the name of the path from the root to the node that represents the directory. Thus, the non-terminal nodes of the tree also represent cataloged (directory) objects.

- A set of general operations for manipulating the catalog, directories and catalog entries, independent of the types of objects, will be provided.

- The catalog can be used to obtain information about an object. However, issues associated with accessing the object, such as access protocols and object representation, are separate from the naming issues that are addressed by the catalog.

- The catalog data base will be organized to efficiently implement two types of lookup operations: symbolic name-to-catalog entry, and unique id-to catalog entry. The symbolic name lookup operation is supported for human users. "Wildcard" designators will be supported. The unique id lookup operation is supported for programs.

- Operations which modify the catalog will be implemented as atomic transactions in order to maintain the integrity of the catalog in the presence of concurrent activity and possible failures of system components.

- The catalog will have the ability to maintain "linkages" to other name spaces. This is supported to permit name spaces of constituent hosts to be (weakly) integrated into the DOS symbolic name space. This will be accomplished by an "external name space" object which can be cataloged like any other object. For example, it will be possible to catalog the directory /usr/rjones/memos on some Unix DOS host as a DOS external name space object. Coupled with appropriate file access software on the Unix system, this would permit a user to refer directly to files in the cataloged directory from the DOS name space.

- The catalog can be thought of as a (complex) DOS object. As mentioned above, directories within the catalog are DOS objects. Therefore, access to the catalog can be controlled by the same mechanisms that control access to other DOS objects. This access control will help ensure the privacy and integrity of information in the catalog. Access to the objects themselves are, of course, also subject to access controls.

- Components of the DOS may choose to cache catalog entries, or the contents of entire directories, in order to support lookup operations locally. This would be done to avoid the potential overhead associated with interacting with remote catalog data bases.

The catalog is an important component of the DOS. It will be used not only to support the cataloging requirements of DOS users, but also to support the implementation of parts of the DOS. For example, as noted above in Section 3.3 the symbolic naming requirements of the DOS file system will be supported by the DOS catalog.

Not all DOS objects will be cataloged in the catalog. It will be possible to access uncataloged objects "directly" by means of their unique ids.

2.3.7 Interprocess Communication

The objective of the DOS interprocess communication (IPC) facility is to support the communication requirements of the DOS. Requirements can be identified at two levels:

1. **The system implementation level.** The collection of software modules that implement the DOS execute as

processes on various DOS hosts. These processes must interact to implement the DOS. These interactions are supported by the interprocess communication facility.

2. **The user application level.** Some of the application programs that execute in the DOS environment may be structured as distributed programs. A distributed program is one whose components may run as cooperating processes on different hosts. The components of such a distributed application program will need to communicate.

The IPC facilities that are available at the application level will be built upon the system level IPC facility.

The DOS interprocess communication facility will be based on the following principles:

- The IPC mechanism will support a variety of communication modes including: datagrams and connections (i.e., reliable, sequenced, flow controlled data streams).

- It will be built upon the standard DoD IP (internet), and TCP (transmission control) protocols. This assumes that the implementations of the DoD protocols that are used will provide adequate performance (low delay, high throughput). If they do not, it may be necessary to build the IPC directly on the local network (Ethernet) protocol.

- Interhost and intrahost communication will be treated in a uniform fashion at the interface to the IPC facility. That is, the same IPC operations used for communicating with processes on different hosts will be used for communicating with ones on the same host. Of course, to achieve the efficiencies that are possible for local communication the IPC implementation will treat interhost communication differently from local communication.

- Several levels of addressing will be supported by the IPC facility. The details of IPC addressing within the DOS have not yet been finalized. The fundamental issue which is unresolved is what the addressable entity for the IPC facility shall be; that is, to what will datagrams be addressed and what will connections connect? One reasonable choice would be for the process itself to be the addressable entity. Alternatively, another abstraction, the "port", could be introduced for this purpose. Ports

would be objects, and like other objects such as processes, they would have unique ids and, if catalogued, could be referenced symbolically by name. Regardless of the choice for addressable entity, the IPC facility will permit addressing by means of unique id and by means of symbolic name. Other levels of addressing may also be supported. At the interface to the IPC facility wherever an IPC address is expected, any of the supported levels of addressing (unique id, symbolic name) may be used.

- The ability of the IPC facility to deal with symbolic addresses will permit it to support "generic" addressing. This will permit processes to specify interactions with other processes in functional terms.

- The IPC mechanism will provide means to directly utilize some of the capabilities of the local network. For example, the Ethernet supports efficient broadcast and multicast. The IPC will provide relatively direct access to these capabilities by supporting broadcast and multicast addressing. To achieve the design goal of component substitution it is important for the DOS system to be as independent as possible of the specific characteristics of the particular local network chosen for the ADM. Therefore, care must be taken to avoid building dependencies on the particular ADM network technology into lower level DOS mechanisms, such as the IPC. If such dependencies cannot be avoided, care must be taken to minimize their impact on the DOS. In our opinion, this is not an issue in the case of the broadcast and multicast facilities, since many state-of-the-art local network technologies support similar capabilities.

2.3.8 User Interface

The purpose of a user interface to the DOS is to provide human users with uniform, convenient access to the functions and services performed by the DOS resources.

The user interface is software that acts to accept input from a human user which it interprets as commands to perform

various tasks and to direct output to the user which the user interprets as the results of commands previously requested or as unsolicited information from the system (or possibly other users). As discussed in Section 3.2, it is sometimes useful to think of the user interface functions as being provided by access point agent and user agent processes.

"Uniform" and "convenient" are subjective characteristics which are hard to quantify. However, we can say in general terms what we mean by these characteristics in the context of a DOS user interface. By uniform, we mean that the manner in which a user requests access to various functions and resources should be similar regardless of the particular DOS components that implement them. For example, the way a user instructs the DOS to run a program should be the same (except for the name of the program) regardless of where within the DOS the program will execute. By convenient, we mean that a user should not have to pay undue attention to the details of the mechanics of establishing access to DOS functions and resources. For example, in order to run an interactive program, a user should not have to explicitly establish a communication path with the host that will run the program. Similarly, to delete a file a user should not have to explicitly establish communication with a file manager on the host that stores the file and instruct it to delete the file.

To be uniform and convenient does not mean that a user

interface must make the network or the distribution of the system invisible to users. In many situations users may want the distribution to be transparent, and the user interface should operate in a way that provides transparency. However, there will be situations where it will be important for the distribution to be visible to users, and for users to be able to exert control over how the system deals with aspects of the distribution. For example, to use the system to do their jobs, system operators and maintainers will need to deal relatively directly with the system's distributed nature. Furthermore, "normal" users, from time to time, may want to control where programs run or files are stored.

One of the ways the DOS will differ from most conventional single host operating systems is that truly parallel execution of user tasks will be possible. It is important that that a user interface for the DOS provide means for to initiate, monitor and control multiple concurrent tasks.

The development of DOS user interface functions will be based on the following principles, many of which are particularly well suited to interactive command and control environments:

- Since many user requests cannot be performed directly by the user interface, the user interface acts on the user's behalf to initiate activity by other DOS modules. The nature of the interactions with other DOS modules is governed by internal DOS "protocols" and interface conventions, and is accomplished by means of interprocess communication.

- An important type of activity a user can initiate is the execution of a program. In this case, the user interface acts to initiate execution of the program and to establish a communication path between the user and the program. In addition, means are provided to permit a user to switch his attention back and forth between the executing program and the user interface.

- The user interface will enable a user to initiate and control multiple simultaneous tasks. In particular, a user may have several application programs executing concurrently.

- Although the user interface bears a unique relationship to the rest of the DOS system, the underlying DOS system will be organized so that much, if not all, of the user interface functions can be written as application level software.

- The part of the user interface that interacts directly with the user to accept commands will be modularized in a way that allows it to be replaced on a per user basis. At login time, after the user is identified, the particular user interaction module appropriate for the user will be used. This will make it possible to accommodate users with strong preferences for radically different styles of interaction, simply by running different user interaction modules.

- The user interface functions developed for the ADM DOS will be designed to operate best with a high speed CRT display terminal, with cursor positioning capability (See Section 3.2). It will make use of multiple "windows" on the display surface. Separate windows will be used to display user interactions with the separate activities being controlled by the user. In addition, windows will be used as necessary to display system status and user help information. The ADM user interface will be tailorable to accommodate a relatively broad range of individual user preferences. This will be accomplished by means of a number of internal "style" and "mode" parameters whose settings control the way the user interface performs. The settings for these parameters will be initialized from values stored in individual user profiles and will be able to be modified at the user's request during a user session.

2.3.9 Input / Output

The term "input/output" is used here in a rather limited sense to mean the process of getting data into and out of the DOS cluster. The objective of the DOS in this area is to provide flexible and convenient means for users and application programs to make use of devices such as printers, tape drives, etc.

To support i/o adequately in its distributed environment the DOS should provide:

1. **The ability to refer to devices symbolically.** For example, users should be able to obtain listings of files by means of "print" or "list" commands which explicitly or implicitly refer to a printer symbolically. Similarly, programs should be able to direct output to a printer by referring to it symbolically.

2. **The ability to distinguish among and to refer to physical devices.** In moderate and large configurations there will be more than one printer (or tape drive, etc). These devices are likely to be located in different areas. It is critically important that the tape drive from which a program reads is the one that holds the right tape. Similarly, when a user requests a listing it is important for him to be able to control which printer will print it so that the output is near his office rather than 1/2 mile away. Thus, one user's "printer" will not necessarily be the same as another's. Furthermore, when a user accesses the DOS from a different location then normal, he should be able to rebind his "printer" to one of the printers that are near him.

The object paradigm developed above, which involves objects, object managers, and object access protocols, is almost sufficient to support DOS device i/o. In addition, the system will provide means for a user to "bind" a particular symbolic device name to a particular physical device.

In summary, DOS support for i/o will be built upon the following principles.

- Input/output devices will be treated as DOS objects. As such, they will have unique ids and may have symbolic names.
- Access to devices will be supported in the same way access to other DOS objects is supported. Access will be accomplished by interacting with an object (device) manager in accordance with an appropriate object (device) access protocols. The interactions will be supported by means of interprocess communication.
- The notion of device binding will be supported by means of the DOS catalog. This will permit users to bind symbolic names to particular physical devices.
- Some types of i/o operations when suitably abstracted are meaningful for files and for devices. Sequential i/o is a good example. File-like interfaces for device i/o have been shown to be useful in a number of systems. The DOS will support file-like interfaces for certain i/o devices.

2.3.10 System Monitoring and Control

The purpose of the DOS system monitoring and control functions is to provide a basis for system operations personnel to operate and control the system.

The system monitoring and control functions will be built upon the following notions:

- Two types of information will be gathered: system status information; and information about the occurrence of exceptional events. Status information will be collected on a periodic basis as a normal part of system operation. Information about exceptional events will be collected as the events are detected.

- Status information and information about exceptional events will be routed to an on-line display which system operations personnel can monitor.

- The detection of certain exceptional events will trigger an "alerting" mechanism to call the events to the attention of operations personnel.

- It will be possible to (selectively) log the occurrence of exceptional events in a event log data base.

- The DOS will support a system control protocol which will make it possible for operations personnel to control the system operation from a single point (e.g., operator's console) as a DOS user. This protocol will provide means to reinitialize the system ("warm" restart), to halt the system, and to set parameters within various DOS components which control aspects of the DOS operation.

- The status gathering facilities will be flexible and comprehensive enough to support performance monitoring experiments.

2.4 System Integrity and Survivability

Users of modern day computing facilities have come to expect the integrity of their computing system and the data it stores and manipulates for them, despite occasional system component failures. The command and control environment in particular requires the continuous availability of key applications despite these failures. To the extent that applications and access to applications come to depend on DOS system functions to achieve goals of system uniformity, those functions must be reliable and continuously available. Further, the role of the DOS as the common software base extending throughout the cluster, makes it a convenient and cost-effective

place (from a programming standpoint) to support generalized, system wide mechanisms for building survivable applications.

By availability we mean the fraction of scheduled up-time during which a system is, in fact, able to deliver normal services to its users. Continuous availability, then, refers to the ability of the system to supply services without pause over some relatively long period of time. The period is sufficiently long to present a significant chance of component failure. Thus a system design which achieves continuous availability must employ some elements of fault-tolerant system design. By integrity we mean the operation of the system in accordance with its specifications while it is available, despite failures from time to time which may render the system temporarily unavailable. Maintaining system integrity is basically a matter of maintaining the consistency of system and user state information ("stored data"). The term survivability is virtually synonymous with "continuous availability", although the emphasis is perhaps different, "survivability" suggesting the possibility of violent failure modes.

A goal of high (but not continuous) availability implies attention to mechanisms for orderly system restarts, that will preserve system integrity across system outages. The restart process may be partially manual, and may involve relatively lengthy integrity checks and system reconfiguration procedures

(e.g., replacing a disk pack, restoring files from backup tapes). Continuous availability, in our terminology, refers to the ability of the system to automatically reconfigure itself or to retry failed operations, in order to maintain the normal semantics of a given function in spite of failures. In a continuously available (i.e., survivable) system, a failure manifests itself only as a tolerable performance degradation and/or insignificant loss of data or function.

Our distinction between high and continuous availability can be illustrated by the following examples. Operator invoked reversal to a backup copy of a damaged file would constitute a recovery measure suitable for a goal of high availability. In contrast, designing a function (e.g. authentication service) so that the system can automatically detect a host failure and subsquently route requests to an alternate source of the function, would be a mechanism for continuous availability. In either case, the integrity of the system must be maintained whenever system services are available.

At a minimum, key system functions and applications must be highly available, and in many cases also continuously available. Ideally, all system services would be continuously available in the command and control environment. However, cost and performance criteria may dictate that high availability is acceptable for some functions, especially if the expected failure

rate is low. Functions such as authentication, initiation of user sessions, and access control must be continuously available for the system to operate at all. Other functions (e.g., access to selected application data) may satisfactorily be provided on a highly available basis, whereas still other functions (e.g., data collection for experimentation) need not be provided at all unless all system resources are operating normally.

All three aspects, integrity, high availability, and continuous availability, play important roles in the overall effectiveness of the system for command and control environments, and will be collectively referred to as system reliability.

2.4.1 Reliability Objectives

The reliability objective of an automated command and control cluster is to provide reliable command and control applications. The role of the "system" with respect to the reliability of these applications is threefold:

- Ensure the "correct" operation of the system in the presence of expected patterns of component failure and subsequent restorations of service. Included in this is that the system does not, under a broad range of failures, lose or corrupt data that is essential to either its own "correct" behavior or to the "correct" behavior of its supported applications
- Provide key DOS system functions and access to those functions in a manner which can survive a limited set of system failures, and which is designed to support high

availability.

- Provide DOS based mechanisms accessible at the user programming interface which are useful for constructing reliable applications.

2.4.2 General Approach

Our approach to failure handling in the DOS is based on first identifying the set of failure modes over which the system is expected to maintain integrity and be continuously available. The definition of each major DOS system function includes the integrity and survivability characteristics to be supported should the expected failures occur. Based on the reliability properties of the specific system functions, other functions using them can then be built which are immune to the outages handled by the abstract function.

The integrity and consistency of system functions are derived from the careful ordering and synchronization of the parts of the individual and parallel operations, and the grouping of related parts into atomic operations that have coordinated outcomes. DOS functional survivability always derives from redundancy of one form or another, either in processing elements and executable programs, or in data, or in time (operation retries). Making the data accepted for storage by the system resilient to component and storage media failures, in the sense that data is not lost despite these failures, is one special case

of the general redundancy concept.

The DOS architecture calls for hardware redundancy to support all survivable functions. The approach is to provide a homogeneous processing base for each particular survivable function, as a means of simplifying the issues of fidelity and coordination between the redundant elements. The role of the DOS software is to support the replication of critical code and data, to control the detection of failures, and to induce recovery procedures. In some cases, such as transaction processing, multiple redundant servers will be supported to share the processing load in the absence of failures, as well as to provide continued service during failures. In other cases, such as data processing application survivability, restart from a prior consistent checkpointed state represents a powerful base on which to build. In all of these cases, the presence of a homogeneous processing base is essential in limiting the complexity of implementation.

2.4.3 Specific Approach

We expect the key functions of the the DOS to be able to recover from the following types of failures.

- Single host outage at arbitrary time without loss of non-volatile memory. This comes in two forms, transient, in which the host is restarted within minutes, and long term (hours at minimum) during which the host is effectively no

longer available. Transient failures of this sort are expected frequently (a few times per day for large configurations) while long term failure is relatively infrequent (a few times per month).

- Single host outage at arbitrary time with additional loss of long term non-volatile memory (e.g. disk crash). These failures are always long term, and occur infrequently (a few times per year).

- Operator controlled forced host shutdown, with ample warning for proper shutdown preparation (e.g. down for emergency or preventive maintenance). This occurs relatively frequently (a few times per week).

- Transient pair wise communication failures. This is predominantly a temporary failure, with the expectation that subsequent retries over a sufficiently long interval will succeed. This condition frequently occurs due to temporary congestion, random noise, hardware and software interfaces not designed for worst case timing conditions, etc.

- Single host temporarily loses communication with the rest of the system but continues to operate. This is the long term version of the pair wise transient communication failure pattern, across all pairs for this host. It occurs relatively infrequently and can be the result of a malfunctioning network interface. This single host isolation represents the most likely pattern of network partitioning which can be anticipated using current local communication bus architectures.

- Any failures that can be made to look like one of the above.

In general, handling failures involves techniques for failure detection, reconstitution of remaining components into a working system, and subsequent reintegration of temporarily failed components back into the operational system after they are repaired. The techniques selected to detect and recover from these failures will vary depending on the expected duration and relative frequency of the failure. Mechanisms selected to handle

infrequent events can usually be of limited performance, and include manual procedures. Mechanisms for frequently occurring events must also take into account the performance characteristics the solutions adopted.

The following techniques have been well studied and are suitable for supporting various aspects of system reliability in the DOS. (12)

- Redundancy of program, file, and processing elements as sources of alternate site service;
- Atomic operations and isolation of partial results to ensure the consistency of function and data;
- Stable storage and guaranteed permanence of effect to ensure that data and decisions, once accepted by the system, will not be lost;
- Checkpoint and restart to support backward error recovery;
- Timeouts to recoginze failure conditions and initiate recovery activities;
- Status probes and status reporting to ensure current operability.

In addition, the GCE concept of interchangeable parts is viewed as a manual approach toward easily reconfiguring components for continued support of important system functions by using parts from less important functions utilizing a common hardware base. It also serves to reduce the inventory of spare parts necessary to achieve a satisfactory level of backup reliability.

(12) "Distributed Operating System Design Study: Final Report" BBN Report No. 4671, May 1981.

The following problems are not being addressed at this time, except as a secondary consideration:

- Complete, extended communication outage within cluster;
- Arbitrary and general partitioning within the local cluster;
- Loss of global (internetwork) communication services.

Handling these problems may be important to the command and control environment. However, we believe that their solution is beyond the scope of the current effort.

2.5 Scalability

The objective in this area is system architecture and design that is cost-effectively scalable over user population sizes ranging from small configurations (e.g., tens of users) to large configurations (e.g., hundreds of users). The aim is to attain uniform functional and performance characteristics over reasonably scaled versions of the system by adding additional hardware and software capacity without introducing excessive escalation of per user cost and performance or requiring redesign of the system structure.

2.5.1 General Approach

The scalability of a computer system is dependent on many capacity and performance factors ranging from hardware component interconnect structures to high level software resources fabricated through systems programming. Due to the off-the-shelf nature of many of the primitive system components being used and the generalized nature of the eventual applications, efforts to achieve system scalability must necessarily be focussed on the scalability of the system functions supported by the DOS.

In general, system scalability and support for system growth can be somewhat different things. Scalability is often achievable by procuring "larger" units for larger configurations, whereas growth is often associated with "additional" units over a period of time. Clearly, addressing the growth issues can, in many ways, subsume the scalability issues. One of the major attractions of a distributed architecture is that it can potentially support growth beyond the limits of conventional systems and hence can attack large scale system scalability from a growth standpoint. Additionally we believe it is operationally and logistically more attractive to support scalability needs from an incremental growth viewpoint in order to limit the number of distinct parts and limit the effects of losing a single unit.

Our system concept for meeting scalability objectives relies on five main points supporting system growth:

1. Adoption of an inexpensive communication architecture which makes it simple to include additional processing elements.

2. Selection of modular, inexpensive DOS hardware so that DOS processing elements can be added in small increments as needed without grossly impacting total cost of the system.

3. Careful attention to the potential size estimates for a maximum configuration to ensure that software structures can be made large enough (e.g. address fields) and that, where appropriate, their implementation is partitionable across multiple instances of the function which share the processing and data load.

4. Avoidance of so-called N squared solutions which require each element to interact with every other element. While these approaches are usually acceptable for smaller configurations, they often break down for larger ones.

5. Select application systems for inclusion in the demonstration configuration which themselves scale through a range of sizes.

2.5.2 Specific Approach

The selection of a bus communication architecture and Ethernet in particular is in large measure based on providing a simple, underlying basis for system scalability. The bus architecture provides a simplified means for supporting a hardware base in which every processing unit can a priori communicate equally well with every other processing unit without regard for routing, processor placement, and other such issues. In addition, Ethernet can physically support large numbers of processing units which can be added or removed at will, and can also inexpensively support small configurations. An important

non-goal at this stage of the project is the scalability of the network communication medium itself. Any future work in this area will be based on adding an additional Ethernet link to each processing element (also a reliability measure) or on complete network substitution.

Low cost incremental expansion also motivates the selection of the M68000-based GCE, which will be used as a building block for many DOS functions. While it is too early to tell the precise number of GCE's required for a minimum configuration, our approach here is to support some degree of GCE functional multiplexing to be used in small configurations, and to make use of dedicated function units in larger or higher performance configurations. The ability to scale up or down also played a role in selecting application hosts for the initial demonstration environment. Both the UNIX and VMS subsystems are or are expected shortly to be supported on a range of hardware bases both larger and smaller than those for the current configuration.

The VAX which was chosen as one of the major mainframes of the system is a good example of a system which can scale over a wide range. For the initial system, configuration will include a VAX 11/750. Without any significant software or peripheral changes, we could substitute any processor from the VAX family, with presently includes the VAX 11/780 and VAX 11/782, with an

increase in capacity of about two and four respectively. In addition, a VAX 11/730 was recently announced which allows substitution of a smaller and less expensive machine.

The choice of a C70 host represents another kind of provision for scalability. In this case, the desire was to include a computer running an operating system which ports to a variety of machine architectures of varying sizes. The leading candidate for operating system portability over medium to large computers is UNIX, so we chose the C70, one of the most cost-effective computers supporting UNIX. We expect that the substitution of another UNIX system would require only a modest effort.

Supporting system software scalability implies ensuring adequate or adequately expandable address fields, table sizes, etc. to meet anticipated needs of target configuration sizes. It also implies including growth as a factor during the design of the implementation of DOS system functions. There are two distinct aspects of a distributed implementation of a given function. One aspect is concerned with redundancy, as described in the previous section. The other is concerned with partitioning responsibility for a function to provide support for a larger client base. It is generally easier and hence more desirable to build a self-contained implementation of a function, than it is to develop a partitioned implementation, since there are fewer

error recovery considerations, and fewer resource management considerations. However, to meet our scalability objectives, some functions may require a partitioned design for supporting large configurations, although they may also be run unpartitioned for small configurations. The analysis of the need for a partitioned implementation will be done at design time, on a function by function basis. We expect that the exercise of the expansion capabilities of the system will be sufficiently infrequent to allow off-line, system reconfiguration time approaches to many scalability problems.

2.6 Global Resource Management

In many computing environments, and most especially a command and control environment, the administering organization needs some degree of control over the ways in which system resources are allocated to tasks to meet their processing demands. This control is frequently provided by the ability to designate some tasks as more important than other competing tasks, and in the ability to effect automated resource management decisions in an attempt to improve some measure of system performance. These functions are often referred to as "priority service" and "performance tuning" respectively. Most computer systems provide some facilities in these areas and many provide rather elaborate facilities which more than adequately address

command and control needs within a single processing node. The objective of this project is to provide support for sustaining these elements of system control in areas that transcend a single processing node.

2.6.1 Objective

The objective in the area of global resource management is to augment the resource management facilities already present on single node systems with simple, additional mechanisms for supporting various policies of administrative control of automated distributed resource management decisions. The emphasis is on methods for ensuring the prompt completion of important processing tasks and on the distribution of processing load across redundant resources.

2.6.2 General Approach

Global resource management in a communications oriented environment is an area where the system wide ramifications of employing such techniques are yet not completely understood. As a consequence, and because of the desire to achieve an operational prototype in a short time frame, we are following a simple, low risk approach. The focus of our effort is on those aspects of global system control directly related to the

distributed nature of the processing environment. In particular, the DOS will focus on the coordination of the priority handling of all parts of any single distributed computation, and on the selection procedures for choosing among replicated, redundant resources present in the DOS cluster. DOS global resource management control will be applied only on large grain decisions (e.g. initiation of a session, opening a file, initiating a program) in an effort to simplify the system and limit the communication and processing overhead that would be required for finer-grained global decision making. We do not anticipate the necessity for reevaluating these resource management decisions at finer grains as a potential source of further optimization. The system concept is that adequate administrative control will be achievable by controlling the set of tasks which may be competing for resources (load limitation), and by controlling the pattern of use of specific instances of the resource which they will be competing for. This is to be accomplished by providing means for administratively limiting the offered load and influencing both the resource selection procedures (where a selection is possible) and the sequencing of the use of the resource after selection using priority. The insertion of DOS control points for limiting load, effecting global binding decisions, and controlling order of service are a sufficient set to carry out administratine policy. The low risk nature of our effort comes in emphasizing simple mechanisms at

these points of control, which in some cases might prove to be suboptimal.

2.6.3 Specific Approach

The DOS system model is based on active user agents (processes) which access a wide variety of abstract resource types, some of which are directly associated with physical resources (e.g., a VAX processor), and others of which may have distributed implementations built out of composite non-distributed objects. All of the resource types have some form of type dependent resource management software associated with them. The following three points are important to our global resource management concepts.

1. Every resource request has a "priority" attribute associated with it which is derived from the initiating agent. Although the resource management discipline will be different for different types of objects, the intent of the priority attribute is to provide an object type dependent form of preferential access relative to the use of the resource. Users will have a range of administratively set priorities available for their use. To ensure access to the system for potential high priority tasks, system login is a "prioritized" request and may result in preemption of a lower priority user, should there be no additional slots. This is accomplished by ensuring that the system "reserves" enough capacity to always accept another login request. If the priority of the potential new user exceeds the priority of one of the current users, and if the login would otherwise fail due to lack of available resources, a lower priority user will be preempted in favor of the initiation of a new job for a high priority user. Once a job is initiated, the current priority of the initiator

will determine how the task competes with other active tasks. Other forms of load limitation will be added as necessary as a means of administratively controlling system responsiveness on available resources.

2. Automated DOS global resource management decisions will be made predominantly when an agent accesses an object which has multiple instances (e.g., multiple processors able to execute the same code, multiple instances of a file, etc.). The algorithms for making the selection will be controllable by the "owner" of the composite object. Control will be in the form of choosing from a standard set of algorithms supported by the system, making use of relevant available data which could include object attributes, collected load data, previous selection, first to respond to broadcast, etc.

3. We are assuming adequate network transmission capacity when smoothed over reasonably short time frames (i.e. no continual network overload). This assumption, which seems to be substantiated by early available local network operational experience, (albeit not in a command and control environment) makes resource management of the network bandwidth generally unnecessary at this time. If scaled load projections indicate potential long term overload situations, our approach for the Ethernet will be to attempt to develop techniques for detecting and limiting the effects of this situation. While it is premature to discuss details of such techniques, a promising approach is to attempt to establish a dynamic network transmission priority level, forcing temporary deferral of data transfers below this priority level, and providing a means for raising the current level until the overload subsides.

Using these mechanisms, controlling the processing activities of the DOS cluster becomes a policy issue of selecting appropriate priorities and parameters to maximize the ability of the system to meet specific organization objectives.

2.7 Substitutability of System Components

Over the course of time and especially when deployed in non-laboratory operating environments, we anticipate the need to substitute alternative hardware and operating system components which are more appropriate for their environment than those selected for the ADM configuration. It is desirable to be able to alter components in order to match the system characteristics to the needs of operational command and control environments and also to reflect changing availability and cost-effectiveness of components. The ability to perform appropriate substitutions of components in the DOS system is expected to expand the applicability of the DOS system and to lengthen it's useful lifetime.

2.7.1 Objective

The objective in this area is to design the system so as to maximize the potential for component substitution in the system hardware architecture at a later time. System components which are candidates for substitution are the local area network, the GCE configurations, the application hosts, and the gateway.

2.7.2 Approach: Use of Abstract Interfaces

The intent of component substitution is to replace a functioning unit with another one capable of performing basically similar operations, but with other properties which make it more attractive or appropriate than the original. For example, substituting a fiber optic communication network for a coaxial cable network might make sense for a command and control environment concerned with portability or electromagnetic radiation. While the basic communication properties of the two systems are equivalent as far as the DOS is concerned, environmental considerations might motivate the substitution. Similarly, most computer systems can be made to perform a wide range of tasks. However, some are judged better than others for certain applications, and hence would motivate the selection of different application hosts to suit the needs of particular command and control applications.

Our approach for supporting component substitutablity is to define and use appropriate abstractions of the substitutable components as the entity incorporated into the DOS. The abstract interfaces are based on common properties of a class of interchangeable components, not on specific capabilities of a single component. Except under special cirumstances, unique properties and peculiarities of the hardware selected for the ADM will be avoided in the definition of abstract interfaces, and

where used will be isolated in the code supporting the abstraction to facilitate emulations within other components.

Two additional implications fall out of this policy. We must expect to lose some efficiency of implementation, since we may need to avoid features that have been built into some components explicitly to solve problems which we may encounter. We expect this effect to be small. The second side effect of the abstract interface should be increased productivity during the development of the DOS, since an abstract interface is easier to understand and work with. This is, in effect, the argument used for higher-level programming languages and standards of all kinds. The adoption of standards of various kinds, as mentioned earlier, also enhances component substitutablity by providing abstractions which are already incorporated into many product interfaces.

2.7.3 Approach: Specific Interface Plans

This section presents a number of interfaces or models which we plan to employ. While this list is not exhaustive, we believe it captures the major interfaces on which the success of substitutability will most depend.

The initial version of the DOS is using the Ethernet standard as a communication subsystem. We expect to be able to

switch between optical fiber and coaxial cable implementations of the Ethernet as may prove desirable based on a cost and availability basis. More importantly, our abstract network interface will avoid using features of the Ethernet protocol which are not common to local network technology. We expect to use only packet transfer, broadcast, and possibly multicast in developing the network abstraction. In addition, we expect to use IP datagram service as the lowest level IPC abstraction. This enhances our independence of the underlying network, and makes it easier to later substitute alternate communication subsystems which can support the abstraction such as the Flexible Intraconnect.

The GCE's represent the implementation base for a number of important DOS functions. It is therefore critical that we address the issue of substitutability for the GCE's. GCE substitution has two aspects: one is the ability to substitute another machine for the present GCE; the second is the ability to substitute for parts of the GCE.

We plan to address the first problem, the ability to switch GCE's at some future date, by programming in common high level languages to the greatest extent possible. We are focussing on two languages: C and Ada. C is a language developed as part of the UNIX system with the goal of being portable to a variety of machines. It has largely met that goal,

although it requires careful attention to coding style to assure the portability of programs written in C (13) . However, there is the possibility of a better choice, Ada, being available in the near future. Since Ada is a DOD standard language, its availability on a variety of processors relevant to command and control environments is assured. In addition, Ada is a more modern and capable language, which should enhance our ability to write code with minimal machine dependency.

Substitutability within the GCE is also a matter of concern and attention. We are building the GCE strictly out of off-the-shelf components using published and emerging standards to minimize our commitment to any particular part of the GCE. For instance, the GCE uses a Multibus bus and backplane, which is supplied by a variety of vendors in a wide range of capacities. The processor board is a design developed by Stanford and licensed to at least four manufacturers, who are producing compatible boards. In addition, with only software changes, the type of processor board can easily be changed, since there are probably more different processor boards available for the Multibus than for any other computer bus. The use of the Multibus also assures easy substitution of memory, Ethernet Controller, I/O ports, etc. It also assures that any as yet

(13) The choice of C was dictated by its immediate availability and the software support already available for C on the GCE processor, a Motorola 68000.

unidentified needs for hardware interfacing can likely be met with off-the-shelf components, due to the popularity of the bus.

Our ability to do general substitutions for application hosts is based on our attempts to use portable languages, a network (Ethernet) which will soon have interfaces available for a wide range of computer systems, and the concept of a DOS access machine. Use of portable languages in the DOS means that we may be able to move software from one DOS host to another. The use of an access machine as a means of connecting an application host to the DOS is intended specifically to minimize the effort of host substitution by maximizing the retained software in the access machine GCE. Precisely which DOS functions can be handled within the access machine GCE without incurring a similarly complex interaction with the host is yet to be determined.

Finally, the most likely substitution to be made during the course of our effort is a substitute for the ARPANET gateway. We have adopted the use of an LSI-11 as the gateway to be able to use standard, off-the-shelf ARPA internet gateways. A successor to the LSI-11 gateway is currently being developed as part of another BBN project. One aspect of our attempt to keep in step with Internet community activities is an anticipated changeover to a new gateway when that development completes. One of the candidate architecures being considered for the future gateway is the equivalent of the DOS GCE.

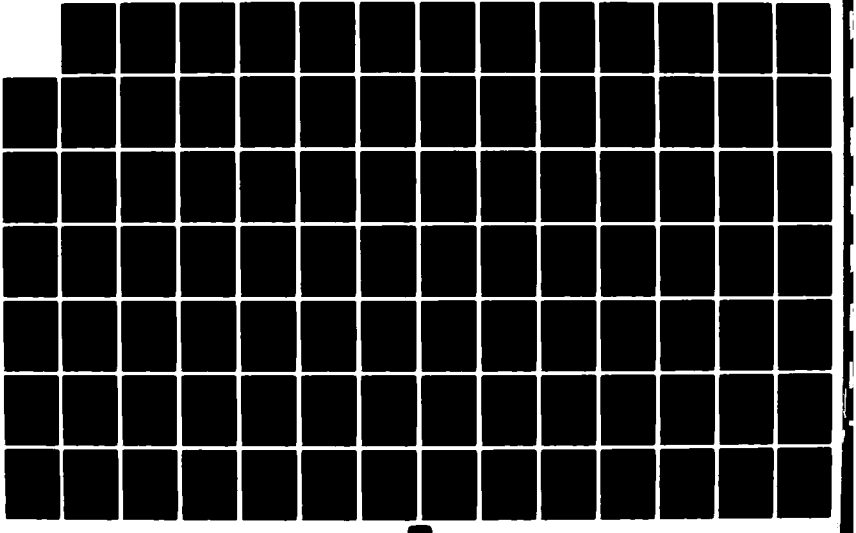
2.8 Operation and Maintenance

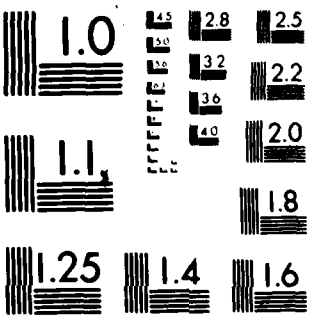
It is desirable for the design of any computer system to facilitate the operation and maintenance of the system. In our opinion, this is one of the areas that has not yet received adequate attention, predominatly because few extensively distributed systems have reached operational status. Distributed systems, and especially systems incorporating many heterogeneous parts, are far more complex than their centralized, homogeneous counterparts. Routine chores, such as adding new components to the configuration, coordinating new releases of system software, and initiating diagnostic routines, become much more complex in a distributed system environment. The natural tendency to handle each component separately has shortcomings in the effort required and the sophistication needed to correctly complete simple maintenance activities. The reason for citing operation and maintenance as a goal is our belief that the success of the distributed system concept in Air Force command and control environments will to some extent be dependent on the management of the routine housekeeping chores associated with any computer system.

The objective in this area is to simplify the operation and maintenance procedures for the system so that these tasks are manageable by personnel other than system programmers. Simplified procedures do not necessarily mean automated

AD-A139 588 CRONUS: A DISTRIBUTED OPERATING SYSTEM(U) BOLT BERANEK AND NEWMAN INC CAMBRIDGE MA R SCHANTZ ET AL. NOV 83 BBN-5086 RADC-TR-83-236 F30602-81-C-0132

UNCLASSIFIED F/G 9/2 NL





procedures (although many such functions, including those mentioned earlier while discussing system control and monitoring, will be automated), nor will they necessarily be as simple as in current computer systems (the environment is quite a bit more complex).

At this time, our approach to operations and maintenance issues includes the following elements:

- The monitoring and control functions designated as part of the system coherence objective address a number of automated operations issues, and serve as a base of operations support.
- The DOS will provide a number of other mechanisms (e.g., distributed file system, software tools) which can serve as a useful foundation for developing simplified maintenance and operations procedures throughout the system;
- As part of the test and evaluation phase, we will operate and maintain the system, and are ourselves self-motivated toward simplified operating procedures.

2.9 Test and Evaluation

One of the important aspects of introducing new system concepts or approaches is the need to answer the question of how successful they have been in meeting their objectives. The test and evaluation phase of our project is intended to provide these answers. We include a discussion of test and evaluation in this "early" project documentation to emphasize our approach of applying considerations in this area throughout the project.

Test and evaluation should be more than an after-the-fact activity and can be a positive factor in driving the design and the implementation.

We can identifiy four distinct stages, spanning the project lifetime, that are relevant to test and evaluation:

1. **Setting goals.** Section 3 outlined the approach and named the three primary goals for which prior test and evaluation methodologies will be developed, namely, coherence and uniformity, survivability and integrity, and scalability.

2. **Defining test and evaluation methodologies.** In parallel with the system design, test and evaluation procedures will be developed for the three primary goals. Insofar as practical, these procedures will each define a "figure of merit" for their respective aspects of the design and implementation, and an effective means for determining the figure of merit. In some cases, the need to carry out these tests may influence the system implementation to more effectively support evaluation.

3. **Extended system test.** During the last few months of the contract period of performance, the system will be subjected to an extended test phase. Operational testing will be done by monitoring the DOS ADM as it is used by the system developers and other groups which may be solicited to build example application systems; synthetic testing will be done through the use of synthetic workload generators for reliability and scalability testing.

4. **Reporting.** The results of the extended system test will be analyzed and judged by means of the yardsticks defined in the second stage. Documentation will be prepared which reflects the results of the test and evaluation phase.

The following sections discuss our current view of the test and evaluation issue as it relates to each of the primary DOS goals.

2.9.1 Coherence and Uniformity

A system is coherent if the system concepts "play together"; coherence makes a system easier to understand and use. A system is uniform if different components perform the same or similar functions in the same or similar ways. Both coherence and uniformity are largely subjective measures of a system, and thus our test and evaluation procedures for this goal will be to gather and analyze the subjective reactions of the user population at the end of the extended test period. Users will be asked to evaluate the system both on absolute terms (what they liked and didn't like) and on a relative basis (comparing the file system, for example, to the UNIX file system). Users will be asked to respond to questions in specific areas, and will also be given an opportunity for open-ended comment. The user statements will be collected, digested, annotated and presented in an organized format (14) .

We anticipate that the user population available for system evaluation will consist of two, probably overlapping, groups: the system developers, and one or more groups selected to develop exemplary application and demonstration programs. There is, of course, a special motivation in requiring the system developers

(14) The paper "Reflections in a pool of processors--an experience report on C.mmp/Hydra", W. A. Wulf and S. P. Harbison, AFIPS Proceeding of the National Computer Conference V47, 1978, is an interesting example of an evaluation of this type, and will serve as a model.

to use the system in the normal course of their work--the feedback path from user to developer is minimized. Design decisions which cause great difficulties will be rapidly exposed and revised. The system developers are also likely to be more tolerant than other users of small "rough edges", which means that they can begin to use the system earlier, before the polishing is finished. This practice generally encourages the developers to be prompt, careful, and down-to-earth, because their own productivity is at stake. A consequence of this is that the initial services developed for the system will be oriented toward the needs of the system developers. In many cases (e.g., text editing) these services have utility in other environments. In those cases where utility is limited to system devlopers, they do form the foundation of supporting the enhancement of the DOS system through it own facilities.

The system developers will further test the system design through the implementation of some system services, such as file archiving and command language interpreters, as application level programs. The implementation of these services will test the ability of the DOS to support such system functions without resorting to modifications of the software within the DOS security envelope. Minimizing the amount of software within the security envelope is a problem analogous to minimizing the size of a security kernel in a conventional, single-host operating system; thus experience gained relating to this aspect of system

extensibility is especially important.

The experiences of the system developers, however, are no substitute for those of application programmers. Application programmers can be expected to make demands upon the completeness and accuracy of the documentation, for example, and to exercise the system in ways that were not anticipated, or not often used, by the developers. Because application programmers will lack in-depth knowledge of the DOS implementation strategies, their reactions are an important test of the user-level conceptual models defined in the user manuals. Due to limited time and effort, only small-scale examples will be constructed during the extended system test, but these can nonetheless be expected to yield significant insight into the usefulness of the DOS design and implementation.

2.9.2 Integrity and Survivability

The test and evaluation of integrity and survivability of a system is one of the harder to perform. First, one must decide what constitutes appropriate behavior in this area, and then one must design (non-destructive) methods of test.

The first step in the test and evaluation procedure for system integrity and survivability is to ensure that the failure

modes identified in Section 4 can be artificially and easily induced in the ADM. For the failure of a processor, for example, this may mean simply that the processor can be either physically or logically disconnected from the network.

The monitoring capabilities of the DOS will include the maintenance of online error logs. These log files will be utilized during the extended test phase to record naturally occurring failures within the ADM, as the DOS is used routinely by the development team and application programmers. Errors which cannot be automatically recorded because of the nature or extent of the failure will be manually recorded in an offline log.

Finally we intend to build one or more reliable applications, and exercise the applications by means of aritificially induced failures.

2.9.3 System Scalability

There are two important facets to the evaluation of DOS scalability: function and performance. By scaling of function we mean the ability of the various DOS mechanisms to scale to larger configurations and user populations without regard to the effect of scaling on performance. Typically, different mechanisms have

different limits to scaling, which are determined by a sequence of decisions during design and implementation; in a conventional single-host operating system, these limits are often real constraints on the range of applicability of the system. For example, an operating system might limit the number of active users or the maximum file size. The first, and easiest, part of the evaluation of scalability is the identification and analysis of these maximum limits to growth.

Even if it is functionally possible to scale the system along some dimension, such as the number of active users, it may be undesirable to do so on performance grounds. A thorough evaluation of the effects of scaling on performance is not possible within the period of this contract; nonetheless, we expect to obtain some preliminary results by means of direct measurements and performance modeling.

We are interested in two primary dimensions of scaling:

1. **Workload scaling.** Given a fixed DOS configuration and a well-defined workload, how do the system response times for different classes of users change as the user population increases?

2. **Configuration scaling.** Given a well-defined workload and a fixed-size user population, how do the system response times for different classes of users change as the number of service hosts is scaled?

One important constraint on the evaluation of scalability is the size of the Advanced Development Model configuration. Because we expect functional limits to the number of hosts, for

example, to be on the order of 1,000 (15) , but will have only about 10 hosts (including DOS service hosts) in the ADM, empirical tests of configuration scaling will be possible only over a small portion of the DOS configuration space.

Our approach to the evaluation of scalability with respect to performance will be based on empirical performance data obtained from the ADM, used as the basis for system models which extrapolate to much larger workloads and configurations. By its nature, this type of performance modeling cannot be extremely precise, and tends to be more useful as a qualitative indicator of feasibility rather than a quantitative predictor of system performance. Analytic models can be constructed and evaluated very rapidly, so they are an inexpensive tool to apply. We believe they are the most appropriate modeling technique during the early life of a system, when decisions are more apt to concern gross changes in resource management strategies than fine-tuning of algorithm parameters.

The DOS system monitoring facilities will be designed to accumulate the performance data necessary for modeling during routine operation of a DOS cluster. This performance data can be collected during actual use of the system, or while system and application functions are exercised by artificially induced

(15) The Ethernet specification limits the number of attached hosts to approximately 1,000.

workloads. At this time, it is not known whether data from naturally-occurring workloads will suffice, or whether synthetic workload generators will be required; this issue should be clarified by the definition of the scalability test criteria during the design and implementation phases of the project.

2.10 Relation to OSI TAFIIS Report

The OSI report (16) serves as a baseline for command and control requirements pertaining to the development of a DOS. It proposes a multilayer operating system/network model for a TAFIIS (17) , composed of MAXI-DOS (global long-haul network) and MINI-DOS (local net) layers. The DOS we are developing in this project focuses primarily on the MINI-DOS issues.

2.10.1 General Aspects of the OSI Model

The OSI report identifies a set of services which should be provided by the TAFIIS system.

- Composition and editing of data.

(16) John R. Thompson, Enrique H. Ruspini, and Christine A. Montgomery. TAC CCC Distributed Operating System Study Final Report Technical Report OSI R79-045, Operating Systems, Inc., November 1979.
(17) Tactical Air Force Integrated Information System.

- Search for data records.
- Define/maintain data base.
- Retrieve and output data.
- Routing of data between users.
- User aids/computation.

We agree that this set of functions is typical of the application programs and processing load which the DOS will need to support.

The OSI report describes several aspects of the DOS which provide starting points for much of our system concept and design. In particular, they identify the following set of mechanisms as pertinet to the development of a DOS:

- Directory services.
- Allocation of resources shared by multiple nodes.
- Scheduling of tasks involving interprocessor interaction.
- Access to global system software.
- Performance monitoring.
- Degradation handling and system recovery.
- Interprocessor communication.
- Multi-level data security.

With the exception of multi-level data security, which is beyond the scope of this project, our system addresses each of these areas. The first two items, directory services and sharing of resources, are at the heart of our effort, since they are most

critical to the design of a local network operating system, if it is to operate as an integrated unit.

Our major extension to the OSI study is the detailed focus on the MINI-DOS aspects of the architecture. The inclusion of gateway functions to link instances of DOS clusters is a preliminary step to addressing MAXI-DOS issues. Differences in communication speed, delay, reliability and security in the MAXI-DOS area change the nature of the network integration task, making it distinct from MINI-DOS system integration.

2.10.2 OSI Identified Functions

The OSI report identifies a number of important functions of the DOS. In this section we briefly indicate our approach as to these functions, and contrast them with potential approaches suggested by the OSI report.

Interprocessor communication will be provided in the DOS using Ethernet together with DOD standard interprocess communication protocols. The Ethernet includes cable network hardware together with a local net CSMA/CD protocol. Above the Ethernet layer we will be using Internet Protocol (IP) Datagrams and, where reliable connection-based transport is required, Transmission Control Protocol (TCP). The use of IP and TCP within the cluster assures a degree of IPC compatibility with the

Internet community and with other DOD systems. We selected a high-bandwidth local network, since we believe high bandwidth, low delay transmission is necessary in order to enable the DOS to operate in an integrated fashion. The OSI report does not focus on MINI-DOS interprocessor communication. It suggests only that the MAXIDOS be capable of 10-50 Kb/second, similar to our ARPANET gateway but two orders of magnitude less than the speed of our local network.

Resource Management in the MINI-DOS is left unspecified in the OSI report, except for indications that resource management be tightly controlled and many appropriate strategies may require a high bandwidth communication medium. In addition, it is suggested in the report that resource management will probably be centralized. We are, of course, providing a high-bandwidth communication medium in the Ethernet. At higher levels of abstraction MINI-DOS resource management implementations must be distributed if the system is to survive component outages.

Security approaches within the MINI-DOS were not specified, since the OSI report felt it was dependent on the nature of the local net (cell, in the OSI terminology). Our system concept calls for a general purpose access control and authentication mechanism, which borrows from several traditional access control schemes. However, we are not planning to implement multi-level security.

Configuration management was regarded by the authors of the OSI report as a problem largely restricted to the MAXI-NET environment, where noisy channels might eliminate communication capability and isolate local networks from each other. While recognizing this problem, we believe that configuration management is also an important issue within the MINI-DOS, where individual host failures should not be allowed to disrupt the local network. In our system concept, there are two levels of configuration issues: the reconfiguration requirements resulting from failed components (and of components brought back into service), and reconfiguration resulting from scaling of the system, planned growth, and phasing in and out of generations of equipment.

The first type of reconfiguration, resulting from system faults, can to a great extent be handled by automatic procedures. These procedures require mechanisms which operate correctly despite outages of components, and of mechanisms which perform automatic reconfiguration when failures are recognized.

The second type of reconfiguration will be provided by manual intervention. Manual updates to configuration tables will be sufficient to accommodate many anticipated changes, and careful, modular system design should enable us to keep more radical configuration changes localized within modules.

Data Base Management is recognized as an important

function within the DOS cluster, but it is largely separable from the design of the DOS itself, and therefore is outside the scope of the present effort. The DOS will provide basic support for data storage and access, including reliable file mechanisms, which could provide a reasonable base for the implementation of data base management systems. In addition, an alternate approach to data base functions, dedicated data base machines, is now emerging. This approach fits in well with our DOS system concept of dedicated function components and we recommend that an instance of such a system be considered for inclusion in the DOS configuration. One of the issues we see concerns the potential conflict of the "black box" nature of these machines, and the desire for integration with other DOS system concepts (e.g. resource management, reliability).

2.11 DOS Glossary

Abstract Object Model
: Model of entities manipulated by the DOS which attempts to treat a wide variety of differing system and user entities in a unified manner. Types of DOS objects will include files, devices, and processes. Associated with each object is a unique identifier, and services for cataloging and controlling access.

Access Point
: Point of interface between the user and the DOS. The access point for a DOS user may be a Terminal Access Controller (TAC), a workstation, or a DOS application host.

Address
: Bit string representing the location at which an object

may be referenced. Addresses often consist of several concatenated fields, representing a hierarchy of containing "locations": Rome is in New York is in the United States of America. A field designates a unique location in the locale containing it; fields may be reused in different locales: Rome is in Italy.

Advanced Development Model (ADM)
Physical instance of the DOS to be developed under the DOS Design/Implementation contract; the ADM will initially be used by the system developers.

Application Host
DOS host on which application programs run. There are potentially many types of application hosts in the DOS; in the ADM, two important types are general-purpose timesharing hosts and GCE's dedicated to application programs.

Capability
If a process possesses a capability for an operation on an object, it may invoke the operation against the object. Possession of the capability is proof of authorization--no further access control check is made.

Cluster
The local network and its hosts. The cluster is the main focus of DOS integration activity. A primary characteristic of a cluster is its uniform high-speed, low delay communication.

Essential Service
Service of the DOS required for the continued operation of the DOS. Essential services are candidates for continuous availability, which is provided through redundancy.

Generic Computing Element (GCE)
A small computer system made up of interchangeable parts upon which many DOS functions will be built. In the Advanced Development Model of the DOS, GCE's will be built using 68000 processors in a Multibus backplane.

Integrity
Maintenance of system and application state information in a consistent state, meeting the system and application program functional specifications. Emphasis is on the maintenance of system integrity across failures, i.e., the phases of failure detection,

isolation, and recovery.

Process
: Model of the active agent or instruction execution in the DOS. Processes in the DOS are objects, and will provide a DOS-wide mechanism for addressing, invocation, and control.

Primitive Process
: Simple version of process which provides only a limited set of control functions. It is presumed that any host in the DOS will be able to provide a base for the implementation of at least one primitive process.

Scalability
: The capability of the DOS to grow or shrink in size, within reasonable bounds. Scalability will be supported by two means, the replacement of processors with more (less) capacity and the addition (deletion) of processors.

Security Envelope
: Boundary around the DOS cluster delimiting the region of the system within which security is ensured by the use of unforgeable addresses and trusted agents. Outside the security envelope, capabilities and passwords will be used to authenticate DOS access.

Survivability
: Ability of a system to continue to perform a given function despite expected failures, with only insignificant performance or functional degradation; synonymous with "continuous availability".

Symbolic Name
: Identification of a DOS object in a global name space independent of the object's location or the location of the reference. The symbolic name space is designed to consist of character strings, and is easily manipulated by the users of the system. A mapping is provided through the catalog mechanism for translating symbolic names to universal identifiers.

Universal Identifier (UID)
: A fixed-length bit-string which identifies, or names, a unique object. Every DOS object has a universal identifier; no two objects have the same identifier.

Workstation
: A computer which is dedicated to single-user-at-a-time

operation, which provides both computational services and an access point to the DOS. In the Advanced Development Model, Jerichos will fulfill this role.

3 Advanced Development Model Configuration Selection

This section reports on the activity preceding the selection of two important components of the cluster configurations: the local area network and the generic computing element. Some of the other elements of the initial configuration had been previously fixed by the statement of work.

3.1 Local Network Selection Criteria

Introduction

This document attempts to describe the factors relevant to the selection of the local network for the Distributed Operating System Design/Implementation contract. Because the Concept of Operations for the DOS is even now (October 1981) being written, there is perhaps less "top-down" motivation for the requirements than one might wish. Nevertheless, the local network must be procured promptly, and the procurement task is complex enough to warrant this semi-formal requirements document.

Although the terms "requirements" and "selection criteria" as used in the title of this document might seem to be redundant, they are not; rather, they express the separation of network attributes into two groups. The nature of the DOS in general, and the contract's Statement of Work in particular, impose some

very definite requirements on the local network. For example, we must have high bandwidth between DOS hosts. This requirement disqualifies any local network whose only host-to-host interface is by means of an RS-232 serial line (limited by the RS-232 standard to 19.2K bits/sec., maximum). On the other hand, we are free to choose a local network of almost any architecture: ring or bus, broadband or baseband transmission, and utilizing any transmission medium: twisted pair, coaxial cable, fiber optics, etc. The selection criteria will help us to choose among the possibilities, on the basis of the factors most relevant to the DOS.

The DOS Development Environment

The computers used in the DOS development environment fall into two categories: host computers, and Access Machines (AM's). Some of the DOS host computers will be directly connected to the local network, while others will be connected through the Access Machines. The Access Machines will be relatively inexpensive processors, such as LSI-11's or Motorola 68000's, and will be directly connected to the local network. The role of the Access Machines in the DOS is discussed in somewhat greater detail in Appendix A.

At this early juncture in the contract we cannot provide a definitive list of the equipment to be connected to the local network, nor can we give specific physical locations for the

various hosts. As an indication of the environment we expect to be constructed, however, we reproduce here the equipment list from our proposal in one plausible geographical arrangement. In this list, hosts connected to the local network through Access Machines are indicated in parentheses.

1. C/70-1 AM-1 (Honeywell Level 6) AM-2 (Gateway function)
2. AM-3 (Jericho-1) AM-4 (terminal MUX) Network Development Station lineprinter
3. C/70-2
4. AM-5 (Jericho-2)

Equipment within a group can be considered to reside within a room, or separated by no more than tens of feet; the distance from one group to another will be on the order of hundreds of feet.

The C/70 is a product of the BBN Computer Corporation. It is a microprogrammable machine with an instruction set oriented towards efficient execution of the C programming language; it operates under the UNIX operating system. A typical C/70 configuration includes memory up to 2MBytes, two 160MByte removable-media disk drives, and a 32-line terminal multiplexer. The Jericho is a personal computer developed by BBN to meet its internal needs for powerful research computing engines. It features 32-bit data paths, a bit-sliced processor, 16MBytes of virtual memory space, a bit-mapped graphics display, and a 200MByte Winchester-technology disk drive.

Requirements from the Statement of Work

The DOS contract's Statement of Work hands down several requirements:

1. The local network should be "high-speed".
2. It must be interfaced to all of the computer systems in the DOS demonstration environment.
3. "It is assumed that high-speed [local network hardware] will be a commercially available off-the-shelf system... This effort does not include development of the local [network] hardware."
4. "The contractor shall include ... interface units to interface the computers to the [local network]. These units shall be programmable such that computers other than those specified may also be added to the [network]."

The Statement of Work speaks specifically of an "interconnecting buss"; ellipses and brackets have been used in the quotes above to remove this apparent prejudice towards one particular local network technology.

Like most requirements, the points above are subject to interpretation. What, for example, should an "interface unit" consist of? The Statement of Work speaks specifically of "buss interface units". In a MITRENET-type network (a contention-bus network operating over standard CATV equipment providing terminal-to-terminal-port communication) a Bus Interface Unit or BIU is a microprocessor-based device that provides an RS-232 interface to a terminal or terminal port on a host computer on one side, handles all network protocol, and connects to the CATV

broadband modem on the other side. In the DOS, we require high-speed access from the hosts to the local network, and this tends to rule out RS-232 interfaces. More importantly, the DOS is primarily concerned with host-to-host communication, and the use of terminal interface standards (e.g., RS-232 or RS-449) for the host-to-network interface is suspect. Thus our "interface unit" cannot be the same object as the BIU of MITRENET. Instead, we must provide some analogous but more appropriate "interface unit" for the host-to-network connection.

DOS Requirements

Through distillation and interpretation of the requirements given in the contract's Statement of Work, we arrive at these requirements for the DOS local network:

R1 The local network must provide uniform functionality to each attached host. That is, the functionality visible to each host should be (at least potentially) the same. This functionality must include a datagram or message-style communication service.

At every point of attachment, there must be the potential to use all of the operations of the local network: to send and receive messages, perform control functions, etc. If it is possible to establish special operating modes that cater to particular devices (e.g., to operate a lineprinter at a well-known port), it must be possible to set or clear these modes for any point of attachment.

The local network may provide other services in addition to datagram service; some of these are listed below among the "selection criteria".

R2 The local network must provide interfaces to the equipment (hosts and/or Access Machines) of the development environment.

This requirement expressly permits the network vendor to use one "interface unit" to connect to the network some or all of the items within a physically proximate group; this may result in a cost saving. For example, using the cost figures given in Criterion below, if all of the equipment listed in Group 1 in Section 2 above could be connected to the network through a single interface unit, the cost of the unit could be as high as $16,875 without exceeding the allowable average cost per host interface.

R3 The network must be "off-the-shelf". As discussed above, we are not doing local network technology development under this contract.

The local network will be a critical component of the DOS implementation project; therefore, it is highly desirable that the network hardware be as reliable, stable, and understandable (modular and well documented) as possible. In particular, the existence of identical or extremely similar products in the field, and experiences of their users, will be weighted heavily during the selection process. Products with little or no proven track record will be down-rated accordingly.

Because of the presence of the local network at the lowest levels of the DOS architecture, it is extremely important that the network be available when needed and stable for the remainder of the contract. A sufficient portion of the local network hardware to begin testing Access-Machine-to-Access-Machine communications must be delivered by 1 March 1982, and the remainder must be delivered no later than 1 June 1982.

R4 The local network must provide an aggregate bandwidth of at least 0.5 Megabits/sec. Further, the network must provide a throughput of at least 0.5 Megabits/sec. between a pair of communicating hosts, with low latency.

(We assume that only two hosts are active, one transmitting data as rapidly as possible to the other, which is receiving the data as rapidly as possible. We assume that these hypothetical processors are very fast, and that throughput is restricted only by the host-to-interface-unit bandwidth, interface unit overhead, and network transmission bandwidth.)

(By "low latency" we mean that, on average,

messages experience little delay due to queuing, buffer-copying, etc. within the local network system. When greater-than-average delays occur they are due to heavy traffic loading on the network, and not, for example, to bandwidth allocation schemes that function independent of network loading.)

This requirement precludes virtual-circuit-only, RS-232-interface-only local network products. In addition, it implies that high-bandwidth interfaces are available for at least the C/70 hosts and the Access Machine (see below), assuming that hosts other than the C/70's connect to the network through an Access Machine.

DOS Selection Criteria

How do we choose among several local network products that meet our requirements? The criteria below define, in priority order, the attributes of local network products that we will evaluate.

C1 The local network product, and any custom-assembled interfaces, must be commercially available to our sponsor so that the DOS development environment can be replicated at one or more sites designated by the sponsor. Maintenance for the local network system (both hardware and software, if any) must be available as well.

The total cost of the vendor-supplied local network components and any special interfaces developed by the DOS project staff should not exceed $60,000.

(A specific local network may have a higher cost and still be acceptable, if it incorporates features that would permit cost savings elsewhere in the development environment.)

This figure is stated as a criterion rather than as a requirement because it is somewhat elastic. The project budget allocated this figure, based on the cost of a particular, representative local network system. This

representative system included:

a. two (2) C/70 interfaces

b. five (5) Access Machine interfaces

c. one (1) line printer (RS-232 or 8-bit parallel) interface

d. one (1) network development station

The network development station is a standalone microcomputer system consisting of a processor, floppy disks, terminal, and network interface. It is capable of supporting the development of any software internal to the vendor-supplied network components. Additionally, it might be used as the basis for network control and monitoring during the normal operation of the DOS (e.g., bringing up or taking down various components of the network).

Other local network systems might not include one physical component that performs all of these functions. However, all of these functions are required of the DOS, and must be supported; the cost of support will be included in the cost of the local network. For example, if the software development environment for the local network software is not on a machine dedicated to the DOS, the use of that machine by the DOS project must be estimated and its cost added to the cost of the local network.

Assuming a cost of $15,000 for the network development station, the remaining host interfaces have an average cost of $5,625. We do not require that the costs be broken down in this manner; the C/70 interfaces, for example, could be more expensive than interfaces to the Access Machine.

C3 To the extent possible, the network interface hardware for the primary DOS hosts should be available off-the-shelf.

The primary hosts in the DOS development environment are the C/70's and the Access Machines. In the case of the

Access Machines, a DMA interface to either the LSI QBUS or the Multibus (depending on the chcice of Access Machine) is required. It is doubtful whether any vendor will have a high-bandwidth C/70 interface at hand. C/70-to-local-network interfaces have been developed by BBN in the past, and the existence of a demonstrated interface design for connection to a specific local network will be considered prima facie evidence that the selection criterion can be met for the C/70 host.

Additionally, the existence of high-bandwidth interfaces between the local network and (1) the BBN Jericho computer, and (2) the Honeywell Level 6 computer family, is desirable. Their presence is less important, however, than that of interfaces to the primary DOS hosts.

To the extent that interfaces must be designed and/or constructed by the DOS project staff, in-house familiarity with the operation of the local network is a selection criterion.

C4 Network bandwidth in excess of that called for by Requirement is desirable.

In particular, it is highly desirable that the pairwise bandwidth approach the instantaneous processor-to-disk bandwidth of hosts in the development environment, which is on the order of 10 Megabits/sec. Capacity above this level is of lesser importance.

C5 The local network may provide functionality beyond simple datagram service, provided this additional functionality does not impair or degrade the performance of the datagram service called for by Requirement . Some of the services that might be useful to the DOS include:

a. broadcasting or multicasting of messages

b. virtual circuits

c. passive listening nodes

d. device handling (e.g., use of XON/XOFF)

e. access control to the network

f. flow control and buffering

g. terminal multiplexing

It is difficult to furnish a complete list in advance of the DOS Concept of Operations; the above list is merely an example of features which might influence the selection process. As was mentioned above, functionality beyond the basic datagram service should not, insofar as possible, alter the interface to or degrade the performance of the datagram service. Flexibility of the services is important, including the possibility of custom alterations or additions to network software during the course of the project.

C6 It is advantageous for the DOS local network to be compatible with the local network used for another BBN project, such as the Distributed Personal Computer (DPC) project (using BBN's Fibernet) or the Command Center Network project (using Ungermann-Bass's Net/One). It is also desirable that the DOS local network be compatible with one or more of the emerging local network standards, such as the DEC-Xerox-Intel Ethernet or the standard being formulated by the IEEE-802 committee.

Various degrees of compatibility are possible, and will be evaluated in the specific context of this project. For example, it is the intent of the DOS project staff to cooperate closely with the DPC project. The DPC project will be using Jericho personal computers connected by Fibernet; thus, compatibility with Fibernet is desirable. Ethernet compatibility is advantageous because of the substantial commercial interest in Ethernet-compatible products; some of these may eventually be included in the development environment.

Appendix A

Access Machine

The role of the Access Machine in the DOS can only be fully defined as the project proceeds. At a minimum, it provides a convenient way to connect a new host to the DOS. As such it provides a physical data path from the "interface unit" to a

host, through two interfaces:

1. host-to-AM interface

2. AM-to-local-network interface

The DOS Concept of Operation will decide to what extent the Access Machine has a larger role, performing more significant algorithmic tasks as part of the DOS.

We have identified two candidates for use as Access Machines in the DOS: the LSI-11 processors (either the LSI-11/2 or the LSI-11/23) and a Multibus board containing a 68000 microprocessor. Both are constructed around buses (the QBUS and Multibus, respectively) that support DMA access by peripherals to the processor's memory space. It is expected that a network-to-AM interface would consist of one or more boards performing DMA data transfer.

It is possible that either processor or both will be used in the DOS. To some extent, the decision depends upon the availability of interfaces to the local network. For purposes of network evaluation, it is important that we understand the issues involved in interfacing either type of system to the network.

The interface between the AM processor and the host it serves deserves attention, as well. It may be possible to use a standard communication line or bus interface, such as HDLC or the

IEEE-488 Instrumentation Bus. If so, this would be extremely desirable, as it would ease the task of connecting the AM's to the various DOS hosts.

3.2 DOS Local Network Selection

From August through October, Bill MacGregor and I along with other members of the DOS project team, pursued our search for a local network communications substrate for the Distributed Operating System. Significant milestones were:

- Presentation to RADC, 4 August
- Publication of DOS-12, "Requirements and Selection Criteria for the DOS Implementation Local Network", 24 August
- Establishment of three "finalists", mid-September
- Final selection, early October
- Review with RADC, 24 November

We have selected the 10 Mb/s Ethernet for the DOS implementation local network, using controllers (or "protocol modules") primarily from InterLAN, Inc. This section lists the candidates we considered, describes the selection process by

which we eliminated the non-finalists, and the specific reason for which we eliminated each one, discusses the three finalists, and, finally, presents our reasons for selecting ETHERNET technology, and InterLAN as a primary vendor.

The reader of this document is assumed to be familiar with the DOS-12 document noted above.

The Candidates

Our investigation unearthed seven candidate local network products. This may seem like a small number, considering the attention paid to local networking in the trade press, and the number of product announcements and advertisements in various journals. However, it is important to remember that most of these products are aimed at what we term terminal to terminal-port communications, providing serial terminal port (RS-232) interfaces for both terminal host connection. We believe that this type of network product is completely inappropriate for the DOS implementation network; we restricted our search to those network products that offered the capability of direct attachment to our DOS hosts.

In all, we investigated the following local network products:

- Net/One from Ungermann-Bass, Inc.
- ProNet from Proteon Associates

- PolyNet from Logica, Inc.
- Fibernet from the BBN Research Computer Center
- ETHERNET products from several manufacturers
- Hyperchannel and Hyperbus from Network Systems Corporation
- CATV-based systems from Sytek, Inc.

Relation to Air Force C^3 Goals

DOS is a research project in the area of distributed operating systems, not local network technology. Therefore, we are constrained to use commercially-available, "off-the-shelf" local network technology and products. At the same time, the research goals of DOS require that we employ the most advanced local network products currently available -- products that meet the functional, performance, interface capability, and cost goals that are outlined in DOS-12. We discovered, in the process of evaluating available local network products, that the products most advanced in these areas are aimed at the commercial marketplace and are designed to commercial, rather than military, standards. Essentially, this means that attributes traditionally sought in C^3 applications (such as robustness, physical ruggedness of the equipment and transmission media, provision for reconfiguration in the event of damage, etc.) have been given short shrift by the designers of these commercial products.

Products that do incorporate these attributes tend to be products of more limited capability, from the point of view of DOS needs.

This would be a matter of considerable concern, if DOS, as developed by BBN, were to be deployed in the field, either as a demonstration system or on an operational basis. However, DOS is to be a demonstration system operating in a laboratory environment. Because of this, and because of the fact that DOS is explicitly a distributed operating system, rather than a local network, research project, we believe it is both reasonable and proper to forego any attempt to meet the traditional C^3 requirements, in favor of the requirements established by the DOS research goals; this is, in fact, what we have done.

What would it take to produce a DOS local network implementation that does meet traditional military objectives? We do not forsee local network products with the appropriate characteristics becoming available in the commercial marketplace over the next few years. This is not because they are in any way technically infeasible; rather, we see the commercial marketplace, driven by quite different forces, moving in a direction different from that required to meet military objectives. We see two ways of providing suitable local network technology for a DOS implementation meeting C^3 goals:

1. Through an R&D effort, state-of-the-art local network

architectures, such as the contention bus or the token ring, could be adapted to provide the characteristics important for miliatry applications. This could readily be done in such a way as to be compatible with the computer hardware currently being selected for the DOS.

2. Through substitution of the Flexible Intraconnect. This approach, already contemplated in the DOS Statement of Work, would provide a local network specifically designed and engineered to meet C^3 objectives.

We consider either of these alternatives to be a "next logical step" in bringing DOS concepts out of a strictly laboratory environment, where functionality is to be demonstrated, closer to a deployable system.

The non-Finalists

Two candidates were eliminated from contention rather quickly, due to cost and complexity considerations:

Network Systems Corporation

Hyperchannel was eliminated for two reasons. First, it is expensive. Its data rate is in the tens of megabits range, beyond what we need for DCS, and it is priced accordingly. Second, its interfaces are complex. It is designed to interconnect IBM (and other) mainframe CPUs and peripheral devices, imitating channel interfaces and the like. Controllers

are available for PDP-11's and possibly several other minicomputers that are not part of the DOS host complement; development of interfaces for other hosts would be a very complex, costly, and risky proposition.

Hyperbus, said to be a less costly system, is not yet available. The Boston-area Network Systems Corporation sales office was not willing to release any information about it, or admit its existence.

Sytek's CATV-based products

Sytek's System 20 is primarily oriented towards terminal-to-terminal-port communication, rather than the host-to-host communication we desire for DOS; we eliminated it on functional grounds. System 40 makes us squarely face the question: Do we want a CATV-based local network for DOS? Our answer, after considerable thought, was a clear no; our reasons are given in the Postscript to this note. For now, suffice it to say that, as with the Network Systems Corporation products, cost and complexity argue against a CATV-based system.

Two other candidates were eliminated after more careful evaluation:

Fibernet (2 MB/s version) from the BBN Research Computer Center (RCC)

The current 2 Mb/s Fibernet system is at the low end of our data rate range. More importantly, it is currently interfaced only to Jerichoes, LSI-11's, and a special-purpose M68000-based terminal concentrator. Other candidates offer both higher data rates and a broader choice of interfaces. Moreover, the RCC is interested in upgrading Fibernet to a 10 Mb/s version compatible, in a functional sense, with ETHERNET. Therefore, the 2 Mb/s Fibernet may be regarded as a "lame duck" candidate with a limited future. Fibernet does have a place in the DOS local network, however, as it is the only network to which the Jericho machines are currently interfaced. Until this changes, the 2 Mb/s Fibernet will serve the Jerichoes in the DOS configuration.

Polynet from Logica, Inc.

Polynet is a commercial version of the Cambridge University Ring Network that has become quite popular in the United Kingdom; it was a promising candidate for DOS. Polynet offers network interfaces for PDP-11's (not relevant to DOS), LSI-11's (relevant to DOS) and Multibus systems (also relevant to DOS). In addition, Logica offers the "network node", or controller, by itself, without a host interface, which would meet our needs for the C/70. However, Polynet presents a few problems: First, there is no customer base in the U.S.; maintenance and support could present problems. While the Logica, Inc. office in New

York is very helpful, they are not the ultimate source of expertise in the product. Second, Polynet deals in "mini-packets" on the ring; each message is divided into packets of two bytes each. In our judgment this scheme is better suited to terminal-type traffic, where messages are likely to consist of small numbers of bytes, than it is to the file-transfer and transaction-type traffic that we expect to see in the DOS. Mini-packets introduce a large number of overhead bits into each message, compared to other network access methods; this has the effect of reducing the available network bandwidth. Thus, the 10 Mb/s Polynet mini-packet ring suffers in comparison to the 10 Mb/s Pronet token-controlled ring discussed below.

The Three Finalists

Net/One from Ungermann-Bass, Inc.

Net/One is a proven, accepted product in the local network field. We have used Net/One in a previous project at BBN, the Command Center Network for the U.S. Navy (NAVELEY). In particular, we have designed and built an interface for Net/One to the C/70; using the 4 Mb/s version of Net/One, we obtained a 2 Mb/s throughput from the C/70 onto the Net/One cable.

Net/One does have several disadvantages, however. It is primarily oriented toward terminal to terminal-port

communication; we used it for host-to-host communication in CCN with the aid of special software from Ungermann-Bass. The latest version of Net/One is compatible with the 10 Mb/s ETHERNET. However, Net/One's "Network Interface Unit", or NIU, contains a Z-80 processor, and this processor is, in essence, interposed between the terminal, or host, and the ETHERNET cable; this is what distinguishes Net/One from what we would term a "pure ETHERNET" system. The processor serves a useful purpose in the intended terminal to terminal-port application of Net/One; however, we seriously question its utility in an application such as DOS. In effect, it's "in the way"; we don't believe it's appropriate, for the DOS environment, to have a processor between the network and the host or access machine.

ETHERNET products

Three vendors, InterLAN, Inc., 3Com Corporation, and Intel, have announced ETHERNET board-level products for delivery in early 1982. ETHERNET offers a number of potential advantages for DOS: It's a *de facto* standard; we can expect more vendors to provide ETHERNET-compatible products as time goes on. This is a potential benefit as DOS evolves and, perhaps, grows. 3Com is marketing a PDP-11 interface and an LSI-11 interface; InterLAN is marketing PDP-11, LSI-11, and Multibus interfaces. Intel is currently marketing a Multibus interface somewhat oriented to its

iSBC series of microprocessor systems. As can be seen, there are multiple vendors for functionally-compatible interface products. In addition, InterLAN will be selling its "ETHERNET Protocol Module" daughter board as a supported product; this daughter board would form the basis of a C/70 ETHERNET interface.

Pronet from Proteon Associates

Pronet is a commercial version of the 10 Mb/s "Version II Local Network Interface" ring network developed under DARPA sponsorship by MIT and Proteon. It is a well-thought-out token-controlled ring network design that represents an alternative to the bus networks that are popular in this country today. Pronet is the cheapest of the alternatives, priced approximately $1000 per node less than either of the other finalists. As in the InterLAN ETHERNET product, Pronet's "network control" module is separate from its "host-specific" module, making possible the design and construction of a C/70 interface.

ETHERNET: The System of Choice

We eliminated Net/One because of its interposition of a Z-80 processor at a point at which we felt was inappropriate, and because of its standard terminal to terminal-port functionality. The choice was then between Pronet and ETHERNET. While Pronet is a very attractive product, we selected ETHERNET over Pronet for

three major reasons. First, ETHERNET is an emerging, albeit *de facto*, standard backed by three major corporations in the computer field. It is clear that, whatever happens in the field of local area networking, ETHERNET will have "staying power". Second, ETHERNET products are already being marketed by multiple vendors -- an advantage for continued DOS work. Third, we expect that direct connection to ETHERNET will be supported by a growing number of mainframe, minicomputer, and microcomputer system vendors -- again, an advantage for the future evolution of DOS.

In addition, the BBN RCC is committed to upgrading its 2 Mb/s Fibernet to a 10 Mb/s version compatible at the transceiver interface level with ETHERNET. Thus,"Fibernet-E", as it is termed, will use ETHERNET controllers, but will substitute a Fibernet tap and fiber-optic cable for the ETHERNET standard transceiver and coaxial cable. We believe this compatibility is an advantage in our development environment at BBN, since we will be using some of the services of the Research Computer Center. For example, the RCC will provide us with communication facilities for the Jerichoes over the 2 Mb/s "J-Net" Fibernet system, "bridged" to the DOS ETHERNET via an LSI-11.

Use of ETHERNET components for the DOS local network offers us a choice: we can either use "standard" ETHERNET cable and transceivers, or we can use the Fibernet-E system. Functionally,

from the DOS point of view, there is no difference between the two. Our final choice of transmission technology is likely to be based upon environmental considerations. Regardless of the choice we make, a replication of the DOS at another location could use standard ETHERNET transmission media exclusively, with no difference in functionality or in the ETHERNET controller hardware or software.

InterLAN and Intel

3Com does not currently plan to offer their ETHERNET controller as a stand-alone product, separate from the two host interfaces they have announced. InterLAN does, however, thus making it possible for us to construct a C/70 ETHERNET interface. InterLAN is also the only ETHERNET vendor so far committed to marketing all of the types of interfaces required for DOS. For this reason, we have selected InterLAN as our vendor for the ETHERNET "protocol module" (to use InterLAN's terminology) and for the interfaces to the DOS LSI-11's We will design and build a C/70 interface for use with the InterLAN protocol module. We have selected Intel as our vendor for the Multibus ETHERNET interface largely because it is available now; InterLAN does not expect to have their Multibus product on the market until the second quarter of 1982. We believe that this delivery schedule advantage outweighs the potential disadvantage of using ETHERNET

controllers of different manufacture, with slightly different programming characteristics.

There are several additional advantages to selecting InterLAN as our primary ETHERNET interface vendor. They are nearby, located in Chelmsford, Mass. Because their product is a new product, we believe they will be particularly responsive, should we experience any difficulties. Also, we believe they will be quite helpful to us in our effort to design and build the C/70 ETHERNET interface.

Postscript: Why not CATV?

CATV, or broadband, local networks are becoming quite popular, especially in military environments. They offer greater distance range, compatibility on a single transmission medium with other types of services, and well-understood electromagnetic radiation characteristics. These are all excellent arguments for the use of a broadband local network in an operational or system development environment.

However, along with the advantages of a broadband network come significant costs. Some of these costs are "per-system" costs; in a large network, these can be amortized over the large number of nodes served. When a broadband local network is added to an existing CATV system, these costs are essentially zero; most of

the per-system cost is associated with the CATV system itself, not the local network. However, per-node costs are also higher for a broadband network than for a baseband local net; one must pay for the ability to be compatible with other services. Finally, data rates are lower on broadband nets than on baseband networks; 2 Mb/s is an upper bound for products on the market today, with development underway for 5 Mb/s products. In contrast, all three of our "finalist" local networks, and some of our non-finalists, too, offer 10 Mb/s transmissiond rates.

As we discussed in Section 1 of this document, the DOS implementation local network does not require the particular features a broadband network offers: we don't need longer-distance transmission, we don't need compatibility with other services on the same transmission medium, and we have no special concern about the electromagnetic radiation characteristics of the system. Therefore, it is difficult for us to justify the added costs that a CATV-based system imposes. Essentially, these extra costs are:

- Higher-cost transmission medium (cable)
- Head-end equipment (power equipment, frequency translator, RF test equipment)
- More complex (and therefore higher-cost) modems (transceivers)

A final note: While this criticism of CATV-based local

networks may seem a little cavalier, it is important to note that the DOS local network is a communications substrate upon which the Distributed Operating System is to be built; it is a vehicle for development and not, itself, an object of development or a demonstration of technology well-suited to a particular environment.

3.3 Generic Computing Element for the DOS

Introduction

A large distributed computing system invariably contains a number of small, dedicated function processors or "computing elements" which perform a variety of tasks in communications, back-end storage management, real-time device control, I/O spooling, etc. These computing elements are typically tightly bound to a specific role within the system, both in terms of their hardware and software configurations. Cost-effectiveness of the hardware is a primary requirement, and the processor, memory, and peripheral resources committed to a particular computing element tend to be just sufficient to meet the needs of its role. For example, a small computing element often lacks secondary storage (disks or tape), a time-of-day clock, and the hardware to support memory management and virtual storage; these resources are taken for granted on larger, general-purpose hosts. A computing element may not contain any software distinguishable as an "operating system" apart from the application software. When it does, the operating system kernel is usually very small (a few thousand bytes), permanently memory resident, and tailored to real-time programming. These small computer elements are not generally "user programmable", but instead are considered integral and often invisible low-level components of the system.

As a consequence of their limited capacity and specialized roles, the computing elements do not usually support their own program development environment. Software is developed on a larger, general-purpose host and down line loaded into the computing elements in the form of binary program images.

The DOS Advanced Development Model (ADM) will contain several computing elements of this general type. It is highly desirable that the computing elements used enjoy a high degree of commonality in several respects:

1. The processor architecture (instruction set) of the computing elements.
2. The physical interfaces between the computing elements and the ADM local network.
3. The physical interfaces internal to computing elements, e.g., the processor-to-device-controller interface.
4. Software development tools, at least a compiler, linker, and debugging support.
5. The operating system kernel, device drivers, and communications support, e.g., Internet Protocol implementation.

A computing element which attains the compatibility goals above will be called a Generic Computing Element (GCE). Because of the requirement for adapting the computing element hardware and software to dedicated roles at minimal cost, a GCE is best described not as a particular computer but as a collection of interchangeable parts. The parts can be assembled into application-specific configurations as needed.

The need for standardized computing elements for dedicated applications has been recognized for some time. The Military Computer Family (MCF) project was developed to obtain this level of standardization for the widest possible range of military applications. Arguments in support of the MCF can be applied to the deployable systems which may evolve from the ADM, but the development schedule of the MCF Nebula computer architecture precludes its use in the ADM. Alternative hardware must be sought for the near term development work performed under the DOS Design/Implementation contract.

The benefits to be derived from the GCE concept fall in the two major areas of hardware and software commonality. Hardware commonality will assist the DOS project in several ways:

1. The ADM configuration will be alterable, by exchanging parts among the GCE's, to provide different test configurations. For example, a GCE employed as a "network front end" or "access machine" could be easily reconfigured by adding a disk controller and disk storage to serve as a network file server.

2. The presence of several GCE's with common parts implies the abilility to replace failed, critical components with similar parts from less critical GCE's. For example, the terminal multiplexer could be temporarily repaired with parts from a redundant file server, until the failed parts could be serviced by the manufacturer.

3. Common components mean simpler maintenance procedures and less time devoted to familiarization by project members. This is important both initially, while interfaces among components are developed, and throughout remainder of the project, as hardware failures are identified, diagnosed, and repaired.

The expected benefits from commonality of software are even

larger than those for commonality of hardware. The advantages to be gained are:

1. A direct savings in programming effort results since many software components will be developed only once, e.g., a GCE operating system kernel, device drivers, and protocol implementations.

2. An indirect (but nonetheless large) savings in effort results from the commonality of support tools, e.g., a compiler, linker, and debugging facilities.

3. Certain important reliability mechanisms are only feasible among processors with common instruction sets; the GCE's provide one instance of such commonality in the ADM (other instances exist).

The following sections will explain the goals of the GCE selection process and its outcome more fully.

Requirements for a Generic Computing Element

The Generic Computing Element should satisfy these requirements:

1. High performance at low cost. The GCE should utilize state-of-the-art components to achieve high instruction execution rates and provide a large physical memory.

2. Interface to the Ethernet. A high-speed interface between the GCE and the Ethernet transceiver cable should be commercially available.

3. Compatibility with standards. To the greatest extent practical, the GCE should be organized around a recognized standard for component interchangeability at the circuit board level.

4. Flexibility and Modularity. A large number of GCE component parts (e.g., disk and peripheral controllers, communication interfaces) should be commercially available, preferably from multiple vendors.

Requirement (1) is dictated by the computational tasks envisioned for GCE's (elaborated in a separate document) and by the desire to use GCE's as a system structuring concept. If a GCE is expensive relative to a timesharing host, for example, DOS function implementations would tend to migrate towards the timesharing environment. It is the intent that GCE cycles be much less expensive than those obtained from "general purpose" DOS hosts. (As an indication of the costs involved, a GCE processor, Ethernet interface, cabinet and power supply will represent an investment of about $10,000, almost an order of magnitude below the cost of a small timesharing host.)

Requirement (2) is the result of our selection of the Ethernet transceiver compatibility interface as a standard within the DOS ADM.

Requirement (3) insures that the GCE components have the longest possible life, and contributes to the possibility of technology transfer from other military, academic, and commercial projects to the DOS ADM. The selection of interface standards which are popular in industry allows the DOS ADM to benefit from competition among the vendors of compatible components, for example, in the areas of expansion memory and peripheral support. This factor is particularly important in areas where technology is moving very rapidly, as in the case of fixed-media Winchester disk drives.

Requirement (4) addresses the need to match GCE configurations to the specific function performed. Components should be readily available to produce GCE's with widely different memory capacities and peripheral complements. Ideally, a range of processor models with identical architectures but different execution rates would also be available.

The PDP-11: Its Merits and Limitations

The proposal for the DOS Design/Implementation contract suggested that the VLSI implementations of the PDP-11 processor, the LSI-11/2 and the LSI-11/23, were candidates for the GCE role. Indeed, the LSI-11's have been commonly used in circumstances requiring a GCE as defined above. The LSI-11 processors, together with the QBUS circuit board interconnection standard, satisfy most of the requirements for the GCE:

1. The LSI-11/2 and LSI-11/23 processors offer reasonably high execution rates at low cost (for a processor, memory, power supply, and packaging about $3,500 and $5,500, respectively).
2. Ethernet interfaces for the QBUS are available or under development by at least two manufacturers.
3. A de facto standard for compatibility with the QBUS exists, and many vendors produce add-in memory and plug-compatible peripheral components. The PDP-11 instruction set represents another form of standardization; tools for PDP-11 programming are readily available, and the popularity of the PDP-11 insures a pool of experienced programmers.
4. The LSI-11 has been used in the GCE role in the past, and a wide variety of QBUS-compatible parts exist, e.g., the 1822 interface to ARPANET IMP's manufactured by ACC.

Unfortunately, the LSI-11 processors are severely limited by the 16-bit address space of the PDP-11 architecture. Past experience has shown time and again that this limitation can be critical to the performance and functionality obtainable from the processor. The Hydra project at CMU encountered the problem, and coined the phrase "Small Address Problem" or "SAP" to refer to it; they believed the SAP was a major burden to the Hydra project. At BBN, long experience with the LSI-11 employed in dedicated roles (e.g., gateways, terminal concentrators, and the Packet Radio station) confirms that the problem is real and unavoidable. The use of the PDP-11 architecture as the GCE processor subelement would severely limit the utility of GCE's. Current LSI-11 based applications (e.g., gateways) consume nearly all of the address space; no space is left for enhancements required by the the DOS Concept of Operation. The limitation is sufficient to mandate the selection of a new processor architecture for the GCE.

Having made the decision not to employ the LSI-11 in the role of GCE, it nonetheless represents a useful benchmark for the evaluation of alternative processors. A replacement for the LSI-11 in the GCE role should offer a physical and logical address space many times larger than the LSI-11, in order to capitalize on the declining price of random access memory. The roles envisioned for the DOS GCE's will require much more than 64K bytes of random access memory per address space. It is

desirable, of course, that the GCE offer a higher instruction execution rate than the LSI-11's, but this property of the GCE is of lesser importance.

The DOS Generic Computing Element

The Internal GCE Module Interface Standard

There are currently three widely recognized commercial interface standards which might be considered to satisfy the requirements for the modular internal structure of the GCE:

1. IEEE 696 bus standard (the so-called "S-100" bus)
2. QBUS, the LSI-11 bus standard
3. IEEE 796 bus standard (the Intel "MULTIBUS" registered trademark of Intel Corporation)(18).

Standard (1) is the basis of a very competitive market, primarily oriented towards small business systems. All of the components necessary for the GCE are available within this market, or will be in the near future. The IEEE 696 standard is not the best choice for the GCE internal interface, however, because: the IEEE 696 bus is inferior to the IEEE 796 bus electrically and in the scope of the signals defined on the bus; Ethernet interfaces, although planned, are not yet available; the IEEE 696 circuit board form factor is more constraining than IEEE 796, requiring more boards and bus interface circuitry to

(18) MULTIBUS is a registered trademark of Intel Corporation.

implement similar functions; no processor with capabilities equivalent to those of the SUN workstation (see below) is available for the S-100 bus.

Standard (2) is the foundation of the LSI-11 computer family, which was eliminated as a candidate for the GCE processor in the previous section. There is little motivation for the use of the QBUS unless the LSI-11 is the processor of choice.

Standard (3), the IEEE 796 standard or "MULTIBUS", has several important advantages. High-performance processing elements are manufactured by several different vendors and are reasonably priced. The first commercially available Ethernet interface, produced by Intel, is compatible with the MULTIBUS; shipments of this product have begun, and initial reports from users are good. Like the other possibilities, there is an active market surrounding the standard, with over 50 vendors producing IEEE 796 compatible products with a wide range of functions available. Disk controllers, communication interfaces, real-time clocks, and many other MULTIBUS-@|compatible modules are available today. The introduction of new functional units compatible with the standard will continue, e.g., Intel has recently announced a first verson of the iAPX-432 processor on two circuit boards which speak to the MULTIBUS.

The Processor Subelement

Assuming that the GCE's are assembled around the MULTIBUS interconnect standard, the choice of processor is constrained. There are currently two candidates for this subelement of the GCE, given our requirement for large (much greater than 64 KB) address spaces:

1. Intel 8086 architecture (either 8086 or 8088 processor)
2. Motorola MC68000 architecture

Both architectures are available in several MULTIBUS-@|compatible forms. An MC68000 architecture has been selected for three major reasons: first, the MC68000 processor architecture has important technical advantages over the 8086 architecture@foot(Notably, the MC68000 views memory as a single large, linear address space; the segmentation approach used in the 8086 family makes code generation more difficult, and potentially less efficient, when random accesses to large data structures are required.); second, a particular MC68000 packaging (the SUN workstation processor board) is very well suited to the project needs; and third, important software development tools and functional modules (e.g., an assembler, a C compiler, a real-time OS kernel, and IP and TCP protocol implementations) are available and familiar to project members.

The MC68000-based SUN processor board chosen as the processor subelement of the GCE was designed at Stanford University, as a component for distributed system research. In

fact, the goals(19) stated by the SUN designers coincide precisely with the requirements for the DOS GCE, viz:

> The first goal was to use state-of-the-art technology to design a high-performance, yet affordable workstation. The result of this effort is a system with the raw computing power of about 1/3 of a Digital Equipment Company VAX11/780....
>
> The second goal was compatibility. We made a conscious effort to base our implementation on current or emerging industry standards....
>
> At the processor level, we have chosen the Motorola 68000 processor because of its 32-Bit architecture, its unsegmented address space, the performance of its initial implementation, and because of the large number of second sources. At the system bus level and packaging level, we have chosen the Intel MULTIBUS, which is being standardized as the IEEE 796 Bus. MULTIBUS, supported by more than 50 vendors, allows us to purchase commercially offered components such as disk controllers. Finally we have chosen the Ethernet as our local area network because it appears that it will be the dominant local network standard and because it is available now....
>
> Using a standard interconnection mechanism also makes the SUN workstation highly modular and extensible. This makes individual boards usable as standalone units or in conjunction with other computing equipment. At the same time, it becomes easy to configure systems for a wide variety of applications....

Quite apart from the similarity of GCE and SUN workstation design goals, the SUN processor board has the characteristics needed in the GCE processor element. The design details can be found in the Stanford technical report and the manufacturers' documentation; a summary will be given here. From the point of

(19) Contained in a draft of a Stanford technical report entitled "The SUN Workstation" by A. Bechtolsheim and F. Baskett.

view of the DOS, the most important features of the processor element are:

- an 8-MHz MC68000 CPU, executing with no wait states from on-board memory,
- 256K bytes of on-board random access MOS memory,
- a memory management unit providing the basis for mapping and protection in a multi-segment, multi-process operating system,
- provision for 32K bytes of PROM or EPROM non-volatile storage,
- an on-board monitor ROM which contains down-line load/dump routines as well as a primitive debugger,
- five timers, to support dynamic memory refresh and process management,
- two serial communication ports, capable of operation at up to 1 megabit,
- full MULTIBUS/IEEE 796 compatibility, including Multimaster capability.

The SUN workstation processor board is manufactured by at least four vendors; two of these, at the current time, sell the workstation only in completely assembled form (i.e., with disk controller and disks, bitmap graphics interface, local network interface, and packaging). Two vendors sell the processor board as a separate item:

Pacific Microcomputers, Inc. Post Office Box A81383 San Diego, California 92138 (714) 565-2727 (Model PM68K)

Forward Technology Inc. 2595 Martin Ave. Santa Clara, California 95050 (408) 988-2378 Attn: Paul Ray (Model FT68M)

There are no known functional differences between the

products of these two vendors.

Additional GCE Subelements

Additional MULTIBUS-compatible boards will be installed in GCE's as required by their specific roles in the ADM. Procurement has begun on an initial test system, consisting of the SUN workstation processor subelement, a MULTIBUS rack-mountable chassis (i.e., cabinet, MULTIBUS backplane, and power supply), and the following additional subelements described below:

- an Intel Ethernet controller
- a 256K byte dynamic memory board
- an 8-line RS-232C controller

These components will be assembled and tested by project members in order to gain experience with the GCE components and its software development environment; they will finally become the terminal multiplexer in the DOS ADM.

The Intel Ethernet Controller

As a consequence of the adoption of the Ethernet transceiver cable compatibility interface defining the physical interface of DOS hosts to the local network, the GCE's must contain Ethernet controllers. Only one controller is currently available, the

Intel iSBC 550 board set (consisting of two MULTIBUS-compatible circuit boards). The iSBC 550 performs data transfers between the Ethernet and the MULTIBUS address space via the MULTIBUS DMA facility.

The Ethernet interfaces to hosts other than GCE instances will probably be procured from InterLAN, Inc., for reasons explained in a separate document. InterLAN also intends to market a MULTIBUS-compatible Ethernet controller, and some advantage might accrue from its use in the DOS ADM. However, the product would not be available until March 1982 at the earliest, with a possibility of significant delays beyond that date. Because the Intel controller is available for early delivery (30 days ARO), and in light of the favorable reports from beta test sites, the Intel controller was chosen for inclusion in the first GCE.

The use of Ethernet controllers from different vendors is potentially advantageous, since it forces an early test of component interchangeability within the ADM.

The 256K Byte Memory Board

Because the Intel Ethernet controller performs packet transfers to and from MULTIBUS memory via DMA, and because the memory on the SUN workstation board is not accessible from the MULTIBUS, some additional MULTIBUS memory must be present in the

GCE. Due to the low cost of semiconductor memory relative to other components, a large increment is still economical; the particular 256K byte memory board chosen (model MM-8086D-256 from Micro Memory Inc.) is representative of a large number of suitable memory products.

The 8-Line RS-232C Controller

Like the memory board, the 8-line RS-232C controller (which permits up to 8 terminals to connect to the GCE) was chosen from several suitable RS-232C controllers from different vendors. The board chosen (model B1018 from Central Data) happens to accommodate 8 lines, the number deemed necessary for the ADM terminal multiplexer most similar controllers provide only 4 lines per board, and as a consequence the cost of 8 lines on two boards would be slightly higher than the cost of the 8-line board selected.

Budget for the First GCE

The purchasing of the components for the first GCE is still in progress, but the current price estimates (quantity one) are:

SUN processor board	$3,250
256K byte memory board	$1,350
Ethernet controller	$4,000

8-line RS-232C controller	$435
MULTIBUS chassis	$2,120

	$11,155

The Software Environment of the GCE

The choice of GCE hardware was preceded by a careful consideration of the existing programs and development tools for the LSI-11, 8086, and MC68000 processors, as such software relates to the DOS project. Although the most extensive in-house experience is with the PDP-11 family, recent projects at BBN have begun to make heavy use of the MC68000 processor. The relevant software environment for the MC68000 can be divided into three major areas discussed separately below: language support, operating system support, and application programs which might be incorporated into the DOS ADM.

Language Support

Assembly language and the C programming language are used for MC68000 programming at BBN. The assembler and C compiler run on the BBN C/70 computer under the UNIX operating system, and object code is down-line loaded into the target machines. The author of the MC68000 assembler is a DOS project member; thus the project has expertise in the use and internals of the MC68000 support tools.

Another document describes an approach to DOS software development which would use a subset of Ada for some of the GCE programming. This approach relies upon the existence of a code generator for the MC68000, integrated with a subset Ada compiler currently being produced at BBN but targeted to other machine architectures, including the PDP-11 family. The straightforward processor architecture of the MC68000, and its resemblance to the PDP-11 architecture, suggest that the production of an MC68000 code generator is a task commensurate with the language support effort budgeted for the DOS project. The feasibility of this approach to GCE programming is still being studied.

Operating System Support

GCE's will be configured to play different roles in the DOS, with varying memory capacity and peripheral complements. It is important that the GCE operating system be tailorable to the various roles; in particular, the small configurations will be GCE's without disks and with limited amounts of primary memory. The applications running on these GCE's, such as the terminal concentrator, are often performance sensitive.

The CMOS operating system(20) developed at BBN is well suited to the GCE environment. By virtue of a modular design and implementation in a high level language, C, CMOS in adaptable to

(20) Reference to CMOS manual.

a variety of assignments. Like the GCE hardware, CMOS can be tailored by including or excluding functional modules as required. Special attention has been paid to the use of CMOS in small configurations without secondary storage. Simplicity also contributes to performance, since the process switching and message transmission overhead is minimal; CMOS is useful in roles where larger, general-purpose operating systems are not appropriate.

The CMOS operating system currently provides four types of services to application programs:

1. process management,
2. input/output management,
3. memory management,
4. system clock management.

In each case, conservative choices were made in the selection features to be included, in order to maximize the adaptability and portability of CMOS to different hosts and applications. All of the CMOS functions are necessary to the DOS applications; for some of them, it may prove desirable to enhance CMOS. For example, CMOS does not currently support preemptive process scheduling, although it could be easily added.

Application Programs

Existing application programs for the MC68000, and also some

C programs from the UNIX environment, are of direct interest to the DOS project. The terminal concentrator built by the BBN Research Computing Center is based on the MC68000 architecture, and this software could be adapted for use in the DOS terminal concentrator GCE. This software includes TCP and IP protocol implementations, the protocols which will be widely used within the DOS ADM.

Because software development tools are available for the C programming language, the possibility of porting some software from the UNIX environment to the GCE's will be considered. (Notably, IP and TCP have been implemented in C by BBN.) This software transfer is not immediate, since the process environment in CMOS is quite different from UNIX, and much simpler; but for some purposes it may prove a useful technique.

Conclusion

The selection of the GCE hardware and software components involved the weighing of many system requirements, both technical and administrative. Our approach has been cautious, but has resulted in the definition of GCE components which will be useful, economical, and have a lifetime extending well beyond the period of performance of the DOS contract.

3.4 CMOS: A Constituent Operating System of Cronus

CMOS and Cronus: An Overview

The Cronus distributed operating system is based on a number of hardware as well as software components. One of the more important hardware components is the Generic Computing Element (GCE). When the concept of the GCE was formulated, the immediate question was what operating system would be appropriate for it. At that point in time there were two options: take an established time-sharing system like UNIX and try to mold it to fit the constraints of the GCE, or take a simple real-time operating system and enhance it to meet the requirements of Cronus. The latter option was chosen, and CMOS became the GCE operating system.

Prior to its introduction to the Cronus DOS project, CMOS was under development at BBN as a real-time operating system for several types of communication processors, such as gateways and network terminal concentrators. A number of GCE applications were similar to the applications CMOS was originally designed for, so it was a natural choice for the GCE OS. In addition, an extensive support environment for building and debugging CMOS applications had already been developed under UNIX.

Although CMOS had to be ported to the GCE, this procedure was greatly simplified by the fact that CMOS was designed to be a

portable operating system, plus the fact that it had already been ported to another 68000 based processor at BBN.

CMOS

CMOS is a small, simple operating system that provides the following basic features:

- multiple processes
- interprocess communication/coordination
- asynchronous I/O
- memory allocation
- system clock management

The design and programming of CMOS have been guided by goals of style, clarity, and consistency rather than a desire to achieve ultimate efficiency. This is not to say that efficiency issues have been ignored. CMOS is quite compact and efficient by virtue of its simplicity. Design principles and programming practices have not been compromised, however, for the sake of saving every possible byte or cpu cycle.

CMOS is designed to be an "open" operating system. This means that no distinct division exists between the operating system and tne application program. One can view the operating system as a collection of library routines. The operating system

can be easily extended by adding new routines and can be reduced by excluding unneeded routines. The programmer is not confined to the outermost interface presented by the operating system. If appropriate, the programmer can directly access lower-level interfaces.

CMOS is designed to be a portable operating system. The use of a high-level language is, of course, the principal factor in CMOS portability. Small size and simplicity are other important factors. The design attempts to minimize the amount of machine-dependent code and to segregate it into separate modules. The I/O system design allows for easy replacement of device-dependent modules.

CMOS Debugging

The first task in porting CMOS to the GCE was to establish a reasonable debugging environment. Fortunately, a cross machine debugger (XMD) had already been developed for 68000 target processors.

XMD is a display oriented debugger based on the PEN editor. Thus all of the commands and features of the editor are available to the user in addition to the debugger specific commands. Without going into great detail, PEN is a multi-window editor with extensive capabilities for manipulating multiple files and edit buffers. XMD displays a special configuration of windows

that are appropriate to debugging. This configuration consists of a source window, a register display window, a breakpoint window, and a window for displaying variables.

A specific module of XMD is responsible for communication with the target processor. Also, a low-level debugger must be resident in the target processor to interpret and execute commands sent to it over the communication path. The communication path in this case is a terminal line to the C70 UNIX host processor. The first step in getting XMD to work with the GCE was to implement the low-level debugger for it. This debugger was actually taken from another 68000 processor and made to work on the GCE. Once the resident debugger was in place, XMD was immediately functional with the GCE, and the task of porting CMOS could proceed.

Porting CMOS to the GCE

As was mentioned earlier, CMOS was designed to be a portable operating system. Apart from the devices, the machine dependent modules of CMOS are few and simple. The task of getting CMOS to work once XMD was functional was trivial compared to getting the GCE resident debugger to work. The only areas where problems needed to be fixed were in the console device and the machine dependent part of the timer code. Of course, the fact that CMOS had already been ported to a different 68000 processor helped considerably.

CMOS Enhancements

Some enhancements and changes were made to CMOS after it was added to Cronus. The first change was to the internal device structure. The purpose of the change was to give the I/O system more flexibility in dealing with the number of possible relationships between hardware devices and the interrupts generated by those devices. Without this change, the capability of writing simple device drivers for CMOS would have been severely compromised.

Another change was the addition of a name service capability. This service was viewed to be generally useful to CMOS for the run-time binding of string names to objects such as processes and devices. The name space is hierarchical and there is a notion of absolute and relative pathnames. In the presence of some form of mass storage, the names can be made non-volatile.

Everything discussed to this point has been accomplished. The remainder of this report will describe work in progress and future enhancements to CMOS.

CMOS and Networks

Access to networks will be provided to CMOS applications from three levels. At the highest level, the user will be able to open a TCP byte stream. The first application at this level will be Telnet and terminal concentration software. At the next

level will be an internet datagram service. This will be used to implement inter-process communication between hosts as part of Cronus development, as well as other standard internet protocols. The lowest level will be a virtual local network interface (see companion note). This interface will be local net independent, but will allow local network specific information and commands to pass through.

A CMOS EtherNet driver has been implemented for the GCE (see companion note). TCP and IP are being adapted to the current version of CMOS from older versions developed at Mitre. The virtual local network interface remains to be developed.

The communication module in XMD will be changed to use the EtherNet instead of a C70 terminal line. This will greatly increase the flexibility and usefulness of XMD. Downloading will be possible over the network, plus it will be easier to debug multiple GCE's from one site.

CMOS and Disks

Some Cronus services require that a mass storage device be available to the GCE and supported by CMOS. The CMOS disk device driver will provide physical storage block access for reading and writing. The typical mass storage device will be fixed disk, although above the driver, all mass storage devices will look the same. CMOS will provide routines and system processes that

support a "flat" (i.e non hierarhical) file system for inclusion in the Cronus distributed file system. At this level files will be accessed by a file identifier that resolves to a file descriptor block. File hierarchy is achieved by binding file identifiers to symbolic names in a hierarchical name space.

Cronus CMOS Applications

Since CMOS is the GCE operating system, it will be supporting all of those Cronus applications and services to be performed by the GCE. Included will be the following:

- Cronus Terminal Concentrator - A GCE having a number of terminal interfaces and an EtherNet interface will supply terminal access to other hosts on the network using Telnet protocols based on TCP-IP.
- Cronus Symbolic Catalog - A GCE with a network interface and a disk drive will provide a generalized name service for Cronus.
- Cronus Host Access Machine - A GCE with a network interface and HDLC standard communication interface will serve to connect non-EtherNet hosts to the Cronus cluster.

In addition to these specific services, the GCE may be a site for running other user developed Cronus applications. This will require a CMOS enhancement to be able to dynamically load processes, either from mass storage or over the network.

4 Initial System Design Acitvities

This section reports on a number of aspects of the Cronus system design which have preceded the general system design.

4.1 The Cronus Virtual Local Network

Purpose and Scope

This note defines the Cronus Virtual Local Network (VLN), a facility which provides interhost message transport to the Cronus Distributed Operating System. The VLN consists of a client interface specification and an implementation; the client interface is expected to be available on every Cronus host. Client processes can send and receive datagrams using specific, broadcast, or multicast addressing as defined in the interface specification.

From the viewpoint of other Cronus system software and application programs, the VLN stands in place of a direct interface to the physical local network (PLN). This additional level of abstraction is defined to meet two major system objectives:

* Compatibility. The VLN defines a communication facility which is compatible with the Internet Protocol (IP) developed by DARPA; by implication the VLN is compatible with higher-level protocols such as the Transmission Control

Protocol (TCP) based on IP.

* Substitutability. Cronus software built above the VLN is dependent only upon the VLN interface and not its implementation. It is possible to substitute one physical local network for another in the VLN implementation, provided that the VLN interface semantics are maintained.

(This note assumes the reader is familiar with the concepts and terminology of the DARPA Internet Program; reference [6] is a compilation of the important protocol specifications and other documents. Documents in [6] of special significance here are [5] and [4].)

The compatibility goal is motivated by factors relating to the Cronus design and its development environment. A large body of software has evolved, and continues to evolve, in the internet community fostered by DARPA. For example, the compatibility goal permits the Cronus design to assimilate existing software components providing electronic mail, remote terminal access, and file transfer in a straightforward manner. In addition to the roles of such services in the Cronus system, they are needed as support for the design and development process. The prototype Cronus cluster, called the Advanced Development Model (ADM), will be connected to the ARPANET, and it is important that the ADM conform to the standards and conventions of the DARPA internet community.

The substitutability goal reflects the belief that different instances of the Cronus cluster will utilize different physical

local networks. Substitution may be desirable for reasons of cost, performance, or other properties of the physical local network such as mechanical and electrical ruggedness. The existence of the VLN interface definition suggests a procedure for physical local network substitution, namely, re-implementation of the VLN interface on each Cronus host. Th implementations will be functionally equivalent but can be expected to differ along dimensions not specified by the VLN interface definition. Since different physical local networks are often quite similar, the task of "re-implementing" the VLN is probably much less difficult than building the first implementation; small modifications to an existing, exemplary implementation may suffice.

The concepts of the Cronus VLN, and in particular the VLN implementation based on Ethernet described in Section 4, have significance beyond their application in the Cronus system. Many organizations are now beginning to install local networks and immediately confront the compatibility issue. For a number of universities, for example, the compatibility problem is precisely the interoperability of the Ethernet and the DARPA internet. Although perhaps less immediate, the substitutability issue will also be faced by other organizations as local network technology advances, and the transfer of existing system and application software to a new physical local network base becomes an economic necessity.

Figure 5 shows the position of the VLN in the lowest layers of the Cronus protocol hierarchy. The VLN interface specification given in the next section is actually a meta-specification, like the specifications of IP and TCP, in that the programming details of the interface are host-dependent and unspecified. The precise representation of the VLN data structures and operations can be expected to vary from machine to machine, but the functional capabilities of the interface are the same regardless of the host.

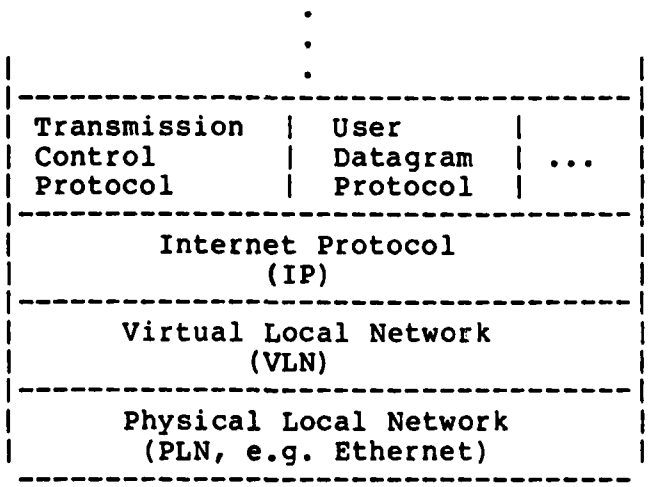


Figure 5 . Cronus Protocol Layering

The VLN is completely compatible with the Internet Protocol as defined in [5], i.e., no changes or extensions to IP are required to implement IP above the VLN. In fact, this was a

requirement on the VLN design; a consequence was the timely completion of the VLN design and avoidance of the lengthy delays which often accompany attempts to change or extend a widely-accepted standard.

The following sections define the VLN client interface and illustrate how the VLN implementation might be organized for an Ethernet PLN.

The VLN-to-Client Interface

The VLN layer provides a datagram transport service among hosts in a Cronus cluster, and between these hosts and other hosts in the DARPA internet. The hosts belonging to a cluster are directly attached to the same physical local network, but the VLN hides the peculiarities of the PLN from other Cronus software. Communication with hosts outside the cluster is achieved through some number of internet gateways, shown in Figure 6, connected to the cluster. The VLN layer is responsible for routing datagrams to a gateway if they are addressed to hosts outside the cluster, and for delivering incoming datagrams to the appropriate VLN host. A VLN is viewed as a network in the internet, and thus has an internet network number. (21)

(21) The PLN could possess its own network number, different from the network number of the VLN it implements, or the network numbers could be the same. Different numbers would complicate the gateways somewhat, but are consistent with the VLN and internet models.

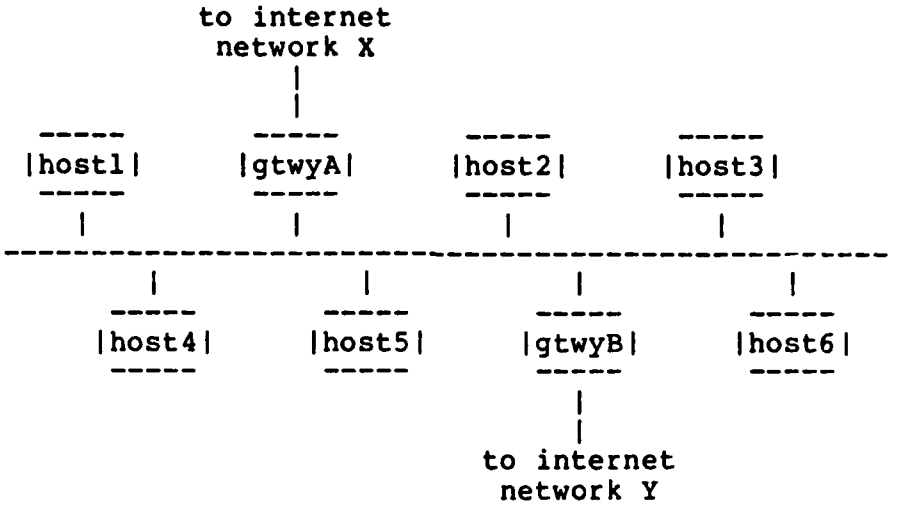


Figure 6 . A Virtual Local Network Cluster

The VLN interface will have one client process on each host, normally the host's IP implementation. The one "client process" may, in fact, be composed of several host processes; but the VLN layer will not distinguish among them, i.e., it performs no multiplexing/demultiplexing function. (22)

The structure of messages which pass through the VLN interface between client processes and the VLN implementation is identical to the structure of internet datagrams constructed in

(22) In the Cronus system, multiplexing/demultiplexing of the datagram stream will be performed above the IP level, primarily in conjunction with Cronus object management.

accordance with the Internet Protocol. Any representation for internet datagrams is also a satisfactory representation for VLN datagrams, and in practice this representation will vary from host to host. The VLN definition merely asserts that there is one well-defined representation for internet datagrams, and thus VLN datagrams, on any host supporting the VLN interface. The argument name "Datagram" in the VLN operation definitions below refers to this well-defined but host-dependent datagram representation.

The VLN guarantees that a datagram of 576 or fewer octets (i.e., the Total Length field of its internet header is less than or equal to 576) can be transferred between any two VLN clients. Larger datagrams may be transferred between some client pairs. Clients should generally avoid sending datagrams exceeding 576 octets unless there is clear need to do so, and the sender is certain that all hosts involved can process the outsize datagrams.

The representation of an VLN datagram is unconstrained by the VLN specification, and the VLN implementor has many reasonable alternatives. Perhaps the simplest representation is a contiguous block of memory locations, either passed by reference or copied across the VLN-to-client interface. It may be beneficial to represent a datagram as a linked list instead, however, in order to reduce the number of times datagram text is

copied as the datagram passes through the protocol hierarchy at the sending and receiving hosts. When a message is passing down (towards the physical layer) it is successively "wrapped" by the protocol layers. Addition of the "wrapper"--header and trailer fields--can be done without copying the message text if the header and trailer can be linked into the message representation. In the particular, when an IP implementation is the client of the VLN layer a linked structure is also desirable to permit reassembly of datagrams (the merger of several fragment datagrams into one larger datagram) inside the IP layer without copying data repeatedly. If properly designed, one linked list structure can speed up both wrapping/unwrapping and datagram reassembly in the IP layer.

Although the structure of internet and VLN datagrams is identical, the VLN-to-client interface places its own interpretation on internet header fields, and differs from the IP-to-client interface in significant respects:

1. The VLN layer utilizes only the Source Address, Destination Address, Total Length, and Header Checksum fields in the internet datagram; other fields are accurately transmitted from the sending to the receiving client.

2. Internet datagram fragmentation and reassembly is not performed in the VLN layer, nor does the VLN layer implement any aspect of internet datagram option processing.

3. At the VLN interface, a special interpretation is placed upon the Destination Address in the internet header, which allows VLN broadcast and multicast addresses to be encoded in the internet address structure.

4. With high probability, duplicate delivery of datagrams sent between hosts on the same VLN does not occur.

5. Between two VLN clients S and R in the same Cronus cluster, the sequence of datagrams received by R is a subsequence of the sequence sent by S to R; a stronger sequencing property holds for broadcast and multicast addressing.

VLN Addressing

In the DARPA internet an internet address is defined to be a 32 bit quantity which is partitioned into two fields, a network number and a local address. VLN addresses share this basic structure, and are perceived by hosts outside the Cronus system as ordinary internet addresses. A sender outside a Cronus cluster may direct an internet datagram into the cluster by specifying the VLN network number in the network number field of the destination address; senders in the cluster may transmit messages to internet hosts outside the cluster in a similar way. The VLN in a Cronus cluster, however, attaches special meaning to the local address field of a VLN address, as explained below.

Each network in the internet community is assigned a class, either A, B, or C, and a network number in its class. The partitioning of the 32 bit internet address into network number and local address fields is a function of the class of the network number, as follows:

	Width of Network Number	Width of Local Address
Class A	7 bits	24 bits
Class B	14 bits	16 bits
Class C	21 bits	8 bits

Table 1. Internet Address Formats

The bits not included in the network number or local address fields encode the network class, e.g., a 3 bit prefix of 110 designates a class C address (see [4]).

The interpretation of the local address field of an internet address is the responsibility of the network designated in the network number field. In the ARPANET (a class A network, with network number 10) the local address refers to a specific physical host; this is the most common use of the local address field. VLN addresses, in contrast, may refer to all hosts (broadcast) or groups of hosts (multicast) in a Cronus cluster, as well as specific hosts inside or outside of the Cluster. Specific, broadcast, and multicast addresses are all encoded in the VLN local address field. (23)

(23) The ability of hosts outside a Cronus cluster to transmit datagrams with VLN broadcast or multicast destination addresses into the cluster may be restricted by the cluster gateway(s), for reasons of system security.

The meaning of the local address field of a VLN address is defined in the table below.

Address Modes	VLN Local Address Values
Specific Host	0 to 1,023
Multicast	1,024 to 65,534
Broadcast	65,535

Table 2. VLN Local Address Modes

In order to represent the full range of specific, broadcast, and multicast addresses in the local address field, a VLN network should be either class A or class B. If a VLN is a class A internet network, a VLN local address occupies the low-order 16 bits of the 24 bit internet local address field, and the upper 8 bits of the internet local address are zero. If a VLN is a class B network, the internet local address field is fully utilized by the VLN local address.

VLN Operations

There are seven operations defined at the VLN interface and available to the VLN client on each host. An implementation of the VLN interface has wide latitude in the presentation of these operations to the client; for example, the operations may or may not return error codes.

A VLN implementation may define the operations to occur synchronously or asynchronously with respect to the client's computation. We expect that the ResetVLNInterface, MyVLNAddress, SendVLNDatagram, PurgeMAddresses, AttendMAddress, and IgnoreMAddress operations will usually be synchronous with respect to the client, but ReceiveVLNDatagram will usually be asynchronous, i.e., the client may initiate the operation, continue to compute, and at some later time be notified that a datagram is available. (The alternatives to asynchronous ReceiveVLNDatagram are A) a blocking receive operation; and B) a non-blocking but synchronous receive operation, which returns a failure code immediately if a datagram is not available. Either alternative may satisfy particular requirements, but an asynchronous receive subsumes these and is more generally useful.) At a minimum, the client must have fully synchronous access to each of the operations; more elaborate mechanisms may be provided at the option of the VLN implementation.

VLN OPERATIONS

ResetVLNInterface

> The VLN layer for this host is reset (e.g., for the Ethernet VLN implementation the operation ClearVPMap is performed, and a frame of type "Cronus VLN" and subtype "Mapping Update" is broadcast; see Section 4.2). This operation does not affect the set of attended VLN multicast addresses.

function MyVLNAddress()

> Returns the specific VLN address of this host; this can

always be done without communication with any other host.

SendVLNDatagram(Datagram)

When this operation completes, the VLN layer has copied the Datagram and it is either "in transmission" or "delivered", i.e., the transmitting process cannot assume that the message has been delivered when SendVLNDatagram completes.

ReceiveVLNDatagram(Datagram)

When this operation completes, Datagram is a representation of a VLN datagram sent by a VLN client and not previously received by the client invoking ReceiveVLNDatagram.

PurgeMAddresses()

When this operation completes, no VLN multicast addresses are registered with the local VLN component.

function AttendMAddress(MAddress)

If this operation returns True then MAddress, which must be a VLN multicast address, is registered as an "alias" for this host, and messages addressed to MAddress by VLN clients will be delivered to the client on this host.

IgnoreMAddress(MAddress)

When this operation completes, MAddress is not registered as a multicast address for the client on this host.

Whenever a Cronus host comes up, ResetVLNInterface and PurgeMAddresses are performed implicitly by the VLN layer before it will accept a request from the client or incoming traffic from the PLN. They may also be invoked by the client during normal operation. As described in Section 4.2 below, a VLN component may depend upon state information obtained dynamically from other hosts, and there is a possibility that incorrect information might enter a component's state tables. (This might happen, for

example, if the PLN address of a Cronus host were changed but its VLN address preserved--the old VLN-to-PLN address mappings held by other hosts would then be incorrect.) A cautious VLN client could call ResetVLNInterface at periodic intervals (every hour, say) to force the VLN component to reconstitute its dynamic tables.

A VLN component will place a limit on the number of multicast addresses to which it will simultaneously "attend"; if the client attempts to register more addresses than this, AttendMAddress will return False with no other effect. The actual limit will vary among VLN components, but it will usually be between 10 and 100 multicast addresses. Components may implement limits as large as the entire multicast address space (64,511 addresses).

The VLN layer does not guarantee any minimum amount of buffering for datagrams, at either the sending or receiving host(s). It does guarantee, however, that a SendVLNDatagram operation invoked by a VLN client will eventually complete; this implies that datagrams may be lost if buffering is insufficient and receiving clients are too slow. The VLN layer will do its best to discard packets for this reason very infrequently.

Reliability Guarantees

Guarantees are never absolute--there is always some

probability, however remote, that a catastrophe will occur and a promise be broken. Nevertheless, the concept of a guarantee is still valuable, because the improbability of a catastrophic failure influences the design and cost of the recovery mechanisms needed to overcome it. In this spirit, the word "guarantee" as used here implies only that the alternatives to correct function (i.e., catastrophic failures) are extremely rare events.

The VLN does not attempt to guarantee reliable delivery of datagrams, nor does it provide negative acknowlegements of damaged or discarded datagrams. It does guarantee that received datagrams are accurate representations of transmitted datagrams.

The VLN also guarantees that datagrams will not "replicate" during transmission, i.e., for each intended receiver, a given datagram is received once or not at all. (24)

Between two VLN clients S and R in the same cluster, the sequence of datagrams received by R is a subsequence of the sequence sent by S to R, i.e., datagrams are received in order, possibly with omissions.

A stronger sequencing property holds for broadcast and multicast transmissions. If receivers R1 and R2 both receive broadcast or multicast datagrams D1 and D2, either they both

(24) A protocol operating above the VLN layer (e.g., TCP) may employ a retransmission strategy; the VLN layer does nothing to filter duplicates arising in this way.

receive D1 before D2, or they both receive D2 before D1.

Desirable Characteristics of a Physical Local Network

While it is conceivable that a VLN could be implemented on a long-haul or virtual-circuit-oriented PLN, these networks are generally ill-suited to the task. The ARPANET, for example, does not support broadcast or multicast addressing modes, nor does it provide the VLN sequencing guarantees. If the ARPANET were the base for a VLN implementation, broadcast and multicast would have to be constructed from specific addressing, and a network-wide synchronization mechanism would be required to implement the sequencing guarantees. Although the compatibility and substitutability benefits might still be achieved, the implementation would be costly, and performance poor.

A good implementation base for a Cronus VLN would be a high-bandwidth local network with all or most of these characteristics:

1. The ability to encapsulate a VLN datagram in a single PLN datagram.
2. An efficient broadcast addressing mode.
3. Natural resistance to datagram replication during transmission.
4. Sequencing guarantees like those of the VLN interface.
5. A strong error-detecting code (datagram checksum).

Good candidates include Ethernet, the Flexible Intraconnect, and Pronet, among others.

A VLN Implementation Based on Ethernet

The Ethernet local network specification is the result of a collaborative effort by Digital Equipment Corp., Intel Corp., and Xerox Corp. The Version 1.0 specification [3] was released in September, 1980. Useful background information on the Ethernet internetworking model is supplied in [2].

The Ethernet VLN implementation begins with the assumption, in accordance with the model developed in [2], that the addresses of specific Ethernet hosts are arbitrary, 48 bit quantities, not under the control of DOS Design/Implementation Project. The VLN implementation must, therefore, develop a strategy to map VLN addresses to specific Ethernet addresses.

A second important assumption is that the VLN-address-to-Ethernet-address mapping should not be maintained manually in each VLN host. Manual procedures are too cumbersome and error-prone when a local network may consist of hundreds of hosts, and hosts may join and leave the network frequently. A protocol is described below which allows hosts to dynamically construct the mapping, beginning only with knowledge of their own VLN and Ethernet host addresses.

The succeeding sections discuss the VLN implementation based on the Ethernet PLN in detail, as designed for the Cronus prototype currently being assembled by Bolt Beranek and Newman,

Inc.

Datagram Encapsulation

An internet datagram is encapsulated in an Ethernet frame by placing the internet datagram in the Ethernet frame data field, and setting the Ethernet type field to "DoD IP".

To guarantee agreement by the sending and receiving VLN components on the ordering of internet datagram octets within an encapsulating Ethernet frame, the Ethernet octet ordering is required to be consistent with the IP octet ordering. Specifically, if IP(i) and IP(j) are internet datagram octets and i<j, and EF(k) and EF(l) are the Ethernet frame octets which represent IP(i) and IP(j) once encapsulated, then k<l. Bit orderings within octets must also be consistent. (25)

VLN Specific Addressing Mode

Each VLN component maintains a virtual-to-physical address map (the VPMap) which translates a 32 bit specific VLN host address (26) in this cluster to a 48 bit Ethernet address. (27)

---

(25) See [1] for a lively discussion of the problems arising from the failure of communicants to agree upon consistent orderings.
(26) Since the high-order 22 bits of the address are constant for all specific host addresses in a cluster, only the low-order 10 bits of the address are significant.
(27) The least significant bit of the first octet of the Ethernet address is always 0, since these are not broadcast or multicast addresses.

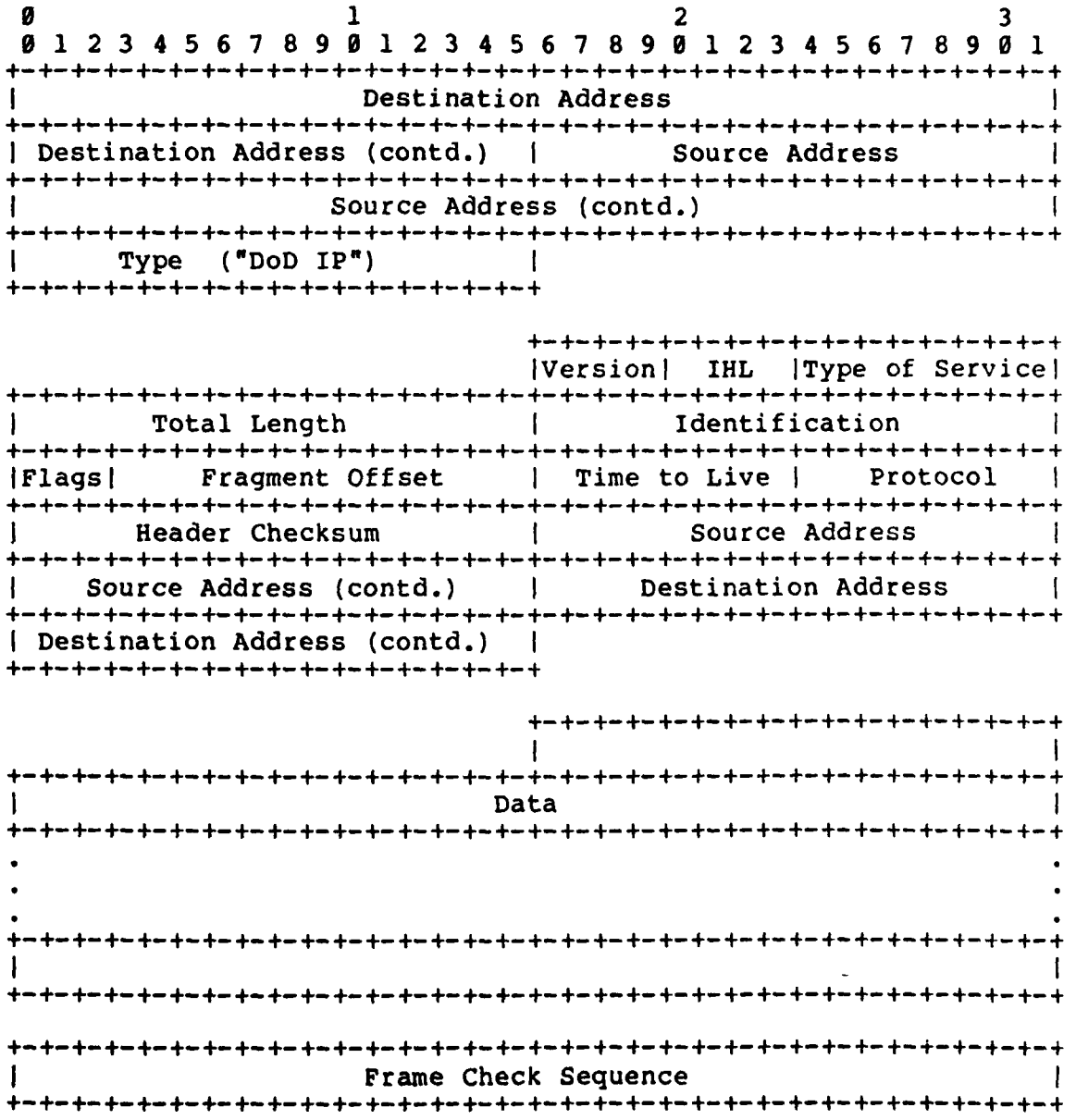


Table 3. An Encapsulated Internet Datagram

The VPMap data structure and the operations on it can be efficiently implemented using standard hashing techniques. Only three operations defined on the VPMap are discussed in this note: ClearVPMap, TranslateVtoP, and StoreVPPair.

Each host has an Ethernet host address (EHA) to which its controller will respond, determined by Xerox and the controller manufacturer (see Section 4.5.2). At host initialization time, the local VLN component establishes a second host address, the multicast host address (MHA), constructed from the host's VLN address. Represented as a sequence of octets in hexadecimal, the MHA has the form:

```
 A  B  C  D  E  F
09-00-08-00-hh-hh
```

A is the first octet transmitted, and F the last. The two octets E and F contain the host local address:

```
   E         F
000000hh  hhhhhhhh
     ^         ^
    MSB       LSB
```

When the VLN client invokes SendVLNDatagram to send a specifically addressed datagram, the local VLN component encapsulates the datagram in an Ethernet frame and transmits it without delay. The Source Address in the Ethernet frame is the

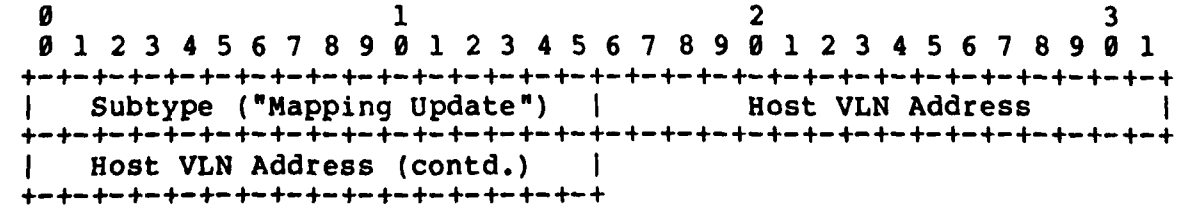
EHA of the sending host. The Ethernet Destination Address is formed from the destination VLN address in the datagram, and is either:

- the EHA of the destination host, if the TranslateVtoP operation on the VPMap succeeds,

or

- the MHA formed from the host number in the destination VLN address, as described above.

When a VLN component receives an Ethernet frame with type "DoD IP", it decapsulates the internet datagram and delivers it to its client. If the frame was addressed to the EHA of the receiving host, no further action is taken, but if the frame was addressed to the MHA of the receiving host the VLN component will broadcast an update for the VPMaps of the other hosts. This will permit the other hosts to use the EHA of this host for future traffic. The type field of the Ethernet frame containing the update is "Cronus VLN", and the format of the data octets in the frame is:



When a local VLN component receives an Ethernet frame with type "Cronus VLN" and subtype "Mapping Update", it performs a

StoreVPPair operation using the Ethernet Source Address field and the host VLN address sent as frame data.

This multicast mechanism could be extended to perform other address mapping functions, for example, to discover the addresses of a cluster's gateways. Suppose all gateways register the same Multicast Gateway Address (MGA, analogous to MHA) with their Ethernet controllers; the MGA then becomes a "logical name" for the gateway function in a Cronus cluster. If a host needs to send a datagram out of the cluster and doesn't know what specific gateway address to use, the host can multicast the datagram to all gateways by sending to MGA. One or more of the gateways can forward the datagram, and transmit a "Gateway Mapping Update" (containing the gateway's specific Ethernet address) back to the originating host. Specific gateway addresses could be cached in a structure similar to the VPMap, keyed to the destination network number. (28)

The approach just outlined suggests that all knowledge of the existence and connectivity of gateways would be isolated in the VLN layer of cluster hosts. Other mechanisms, e.g., based on the ICMP component of the Internet Protocol, could be used instead to disseminate information about gateways to cluster

(28) Because the Cronus Advanced Development Model will contain only one gateway, a simpler mechanism will be implemented initially; the specific Ethernet address of the gateway will be "well-known" to all VLN components.

hosts (see [7]). These would require, however, specific Ethernet addresses to be visible above the VLN layer, a situation the current design avoids.

VLN Broadcast and Multicast Addressing Modes

A VLN datagram will be transmitted in broadcast mode if the argument to SendVLNDatagram specifies the VLN broadcast address (local address = 65,535, decimal) as the destination. Broadcast is implemented in the most straightforward way: the VLN datagram is encapsulated in an Ethernet frame with type "DoD IP", and the frame destination address is set to the Ethernet broadcast address. The receiving VLN component merely decapsulates and delivers the VLN datagram.

The implementation of the VLN multicast addressing mode is more complex, for several reasons. Typically, each VLN host will define a constant called Max_Attended, equal to the maximum number of VLN multicast addresses which can be simultaneously "attended" by this host. Max_Attended should not be a function of the particular Ethernet controller(s) the host may be using, but only of the software resources (buffer space and processor time) that the host dedicates to VLN multicast processing. The protocol below permits a host to attend any number of VLN multicast addresses, from 0 to 64,511 (the entire VLN multicast address space), independent of the controller in use.

Understanding of the VLN multicast protocol requires some knowledge of the behavior of existing Ethernet controllers. The Ethernet specification does not specify whether a controller must perform multicast address recognition, or if it does, how many multicast addresses it must be prepared to recognize. As a result Ethernet controller designs vary widely in their behavior. For example, the 3COM Model 3C400 controller follows the first pattern and performs no multicast address recognition, instead passing all multicast frames to the host for further processing. The Intel Model iSBC 550 controller permits the host to register a maximum of 8 multicast addresses with the controller, and the Interlan Model NM10 controller permits a maximum of 63 registered addresses.

It would be possible to implement the VLN multicast mode using only the Ethernet broadcast mechanism. This would imply, however, that every VLN host would receive and process every VLN multicast, often only to discard the datagram because it is misaddressed. More efficient operation is possible if at least some Ethernet multicast addresses are used, since Ethernet controllers with multicast recognition can discard misaddressed frames more rapidly than their hosts, reducing both the processor time and buffer space demands upon the host.

The protocol specified below satisfies the design constraints and is especially simple.

A VLN-wide constant, Min_Attendable, is equal to the smallest number of Ethernet multicast addresses that can be simultaneously attended by any host in the VLN, or 64,511, whichever is smaller. A network composed of hosts with the Intel and Interlan controllers mentioned above, for example, would have Min_Attendable equal to 7; (29) a network composed only of hosts with 3COM Model 3C400 controllers would have Min_Attendable equal to 64,511, since the controller itself does not restrict the number of Ethernet multicast addresses to which a host may attend. (30)

The local address field of a VLN multicast address can be represented in two octets, in hexadecimal:

mm-mm

From Table 1, mm-mm considered as a decimal integer M is in the range 1,024 to 65,534. When SendVLNDatagram is invoked with a VLN multicast datagram, there are two cases:

1. (M - 1,023) <= Min_Attendable. In this case, the datagram is encapsulated in a "DoD IP" Ethernet frame, and multicast with the Ethernet address

   09-00-08-00-mm-mm

   A VLN component which attends VLN multicast addresses in this range should receive Ethernet multicast addresses in

(29) Min_Attendable is 7, rather than 8, because one multicast slot in the controller must be reserved for the host's MHA, as described in Section 4.2.
(30) For the Cronus Advanced Development Model, Min_Attendable is currently defined to be 60.

this format, if necessary by registering the addresses with its Ethernet controller.

2. (M - 1,023) > Min_Attendable. The datagram is encapsulated in a "DoD IP" Ethernet frame, and transmitted to the Ethernet broadcast address. A VLN component which attends VLN multicast addresses in this range must receive all broadcast frames, and filter them on the basis of frame type and VLN destination address (found in the IP destination address field).

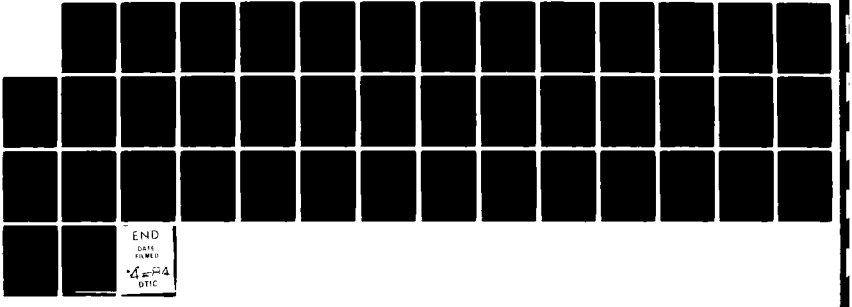
There are two drawbacks to this protocol that might induce a more complex design: 1) because Min_Attendable is the "lowest common denominator" for the ability of Ethernet controllers to recognize multicast addresses, some controller capabilities may be wasted. 2) small VLN addresses (less than Max_Attendable + 1,024) will probably be handled more efficiently than large VLN multicast addresses. The second factor complicates the assignment of VLN multicast addresses to functions, since the particular assignment affects multicast performance.

Reliability Guarantees

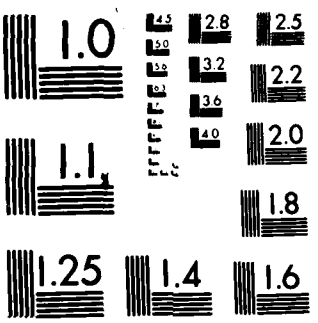
Delivered datagrams are accurate copies of transmitted datagrams because VLN components do not deliver incoming datagrams with invalid Frame Check Sequences. The 32 bit CRC error detecting code applied to Ethernet frames is very powerful, and the probability of an undetected error occurring "on the wire" is very small. The probability of an error being introduced before the checksum is computed or after it is checked is comparable to the probability of an error in a disk subsystem before a write operation or after a read; often, but not always,

AD-A139 588 CRONUS: A DISTRIBUTED OPERATING SYSTEM(U) BOLT BERANEK AND NEWMAN INC CAMBRIDGE MA R SCHANTZ ET AL. NOV 83 BBN-5086 RADC-TR-83-236 F30602-81-C-0132 3/3

UNCLASSIFIED F/G 9/2 NL



END
DATE FILMED
4-84
DTIC



it can be ignored.

Datagram duplication does not occur because the VLN layer does not perform datagram retransmissions, the primary source of duplicates in other networks. Ethernet controllers do perform retransmission as a result of "collisions" on the channel, but the "collision enforcement" or "jam" assures that no controller receives a valid frame if a collision occurs.

The sequencing guarantees hold because mutually exclusive access to the transmission medium defines a total ordering on Ethernet transmissions, and because a VLN component buffers all datagrams in FIFO order, if it buffers more than one datagram.

Use of Assigned Numbers

On a philosophical note, protocols such as IP and TCP exist to provide communication services to extensible sets of clients; new clients and usages continue to emerge over the life of a protocol. Because a protocol implementation must have some unambiguous knowledge of the "names" of the clients, sockets, hosts, networks, etc., with which it interacts, a need arises for the continuing administration of the assigned numbers related to the protocol. Typically the organization which declares a protocol to be a standard also becomes the administrator for its assigned numbers. The organization will designate an office to assign numbers to the clients, sockets, hosts, networks, etc.,

that emerge over time. The office will also prepare lists of number assignments that are distributed to protocol users; the reference [4] is a list of this kind.

There are three organizations responsible for number assignment related to the Ethernet-based VLN implementation: DARPA, Xerox, and the DOS Design/Implementation Project; their respective roles are described below.

DARPA

DARPA administers the internet network number and internet protocol number assignments. The Ethernet-based VLN implementation does not involve DARPA assigned numbers, but any particular instance of a Cronus VLN is expected to have a class A or B internet network number assigned by DARPA. For example, the prototype Cronus system (the Advanced Development Model) being constructed at Bolt Beranek and Newman, Inc., has class B network number 128.011.xxx.xxx.

Protocols built above the VLN will make use of other DARPA assigned numbers, e.g., the Cronus object-operation protocol requires an internet protocol number.

The Xerox Ethernet Address Administration Office

The Ethernet Address Administration Office at Xerox Corp. administers Ethernet specific and multicast address assignments,

and Ethernet frame type assignments.

It is the intent of the Xerox internetworking model that every Ethernet host have a distinct specific address, and that the address space be large enough to accommodate a very large population of inexpensive hosts (e.g., personal workstations). They have therefore chosen to delegate the authority to assign specific addresses to the manufacturers of Ethernet controllers, by granting them large blocks of addresses on request. Manufacturers are expected to assign specific addresses from these blocks densely, e.g., sequentially, one per controller, and to consume all of them before requesting another block.

The preceding paragraph explains the Xerox address assignment policy not because the DOS Design/Implementation Project intends to manufacture Ethernet controllers (!), but because Xerox has chosen to couple the assignment of specific and multicast Ethernet addresses. An assigned block is defined by a 23-bit constant, which specifies the contents of the first three octets of an Ethernet address, except for the broadcast/multicast bit (the least significant bit of the first octet). The possessor of an assigned block thus has in hand 2**24 specific addresses and 2**24 multicast addresses, to parcel out as necessary.

The block assigned for use in the Cronus system is defined by the octets 08-00-08 (hex). The specific addresses in this

block range from 08-00-08-00-00-00 to 08-00-08-FF-FF-FF (hex), and the multicast addresses range from 09-00-08-00-00-00 to 09-00-08-FF-FF-FF (hex).

The Ethernet Address Administration Office has designated a public frame type, "DoD IP", 08-00 (hex), to be used for encapsulated internet protocol datagrams. The Ethernet VLN implementation uses this frame type exclusively for datagram encapsulation. In addition, the Cronus system uses two private Ethernet frame types, assigned by the Ethernet Address Administration Office:

Name	Type
Cronus VLN	80-03
Cronus Direct	80-04

(The use of the "Cronus Direct" frame type is not described in this note.)

The same Ethernet address and frame type assignments will be used by every instance of a Cronus VLN; no further assignments from the Ethernet Address Administration Office are anticipated.

The DOS Design/Implementation Project

The DOS Design/Implementation Project assumes responsibility for the assignment of subtypes of the Ethernet frame type "Cronus VLN". No assignments of subtypes for purposes unrelated to the Cronus system design are expected, nor are assignments to other

organizations. The subtypes currently assigned are:

Name	Subtype
Mapping Update	00-01

REFERENCES

[1] "On holy wars and a plea for peace," Danny Cohen, Computer, V 14 N 10, October 1981, pp. 48-54.

[2] "48-bit absolute internet and Ethernet host numbers," Yogen K. Dalal and Robert S. Printis, Proc. of the 7th Data Communications Symposium, October 1981.

[3] "The Ethernet: a local area network, data link layer and physical layer specifications," Digital Equipment Corp., Intel Corp., and Xerox Corp., Version 1.0, September 1980.

[4] "Assigned numbers," Jon Postel, RFC 790, USC/Information Sciences Institute, September 1981.

[5] "Internet Protocol - DARPA internet program protocol specification," Jon Postel, ed., RFC 791, USC/Information Sciences Institute, September 1981.

[6] "Internet protocol transition workbook," Network Information Center, SRI International, Menlo Park, California, March 1982.

[7] "IP - Local Area Network Addressing Issues," Robert Gurwitz and Robert Hinden, Bolt Beranek and Newman Inc., (draft) August 1982.

4.2 Cronus Unique Numbers

Purpose and Scope

This note describes a facility in the Cronus operating system for the generation of unique numbers (UNO's). The UNO generator may be regarded as an abstract object which exists continuously throughout the life of a Cronus configuration, and is accessible to system and application processes. Processes may issue requests for UNO's at any time, from any of the hosts in a Cronus cluster. The purpose of the UNO facility is to guarantee that any requesting process is promptly supplied with a UNO.

There are several reasons for designing a unique number generator as a low-level component of the Cronus system. Unique numbers will be used to name objects, for example, files, processes, and hosts. They can be used as transaction identifiers to permit the detection of duplicate requests or responses resulting from retry mechanisms. Unique numbers are freely available to application programs, to serve as cluster-wide names for objects in the application domain.

The usefulness of UNO's stems from the simple property that no two requests by client processes ever obtain the same UNO, over the entire lifetime of a Cronus cluster. Considering that UNO's can be generated rapidly (on the order of 1,000 per second per host) and that a UNO is relatively small (64 bits), they form

an effectively inexhaustible supply of machine interpretable names for objects supported by Cronus hosts.

Two important consequences of this property are:

1. when UNO's are used to name objects, the object manager need never concern itself with the possibility that the name space will be fully consumed;
2. when UNO's are used to name objects of several kinds, any name can appear in any context with no further information required to disambiguate an object reference.

These are important advantages of UNO's for naming Cronus objects. The same arguments apply to objects realized within application programs, objects which may be invisible to Cronus. Given an implementation designed to satisfy Cronus system requirements, it is desirable and straightforward to provide access to the UNO facility to application programs. Doing so obviates the need for application programs to construct similar name generation facilities of their own (at considerable cost), and assists the integration of system and application concepts.

Because UNO's are expected to be widely used, it is important that they are easy to store and manipulate. For this reason the particular UNO generation scheme chosen produces fixed-length UNO's. With a length of 64 bits or 8 bytes, UNO's are comparable in size to short symbolic names, and can be easily stored and manipulated on byte-oriented, 16-bit-word, and 32-bit-word machine architectures.

The client processes which obtain UNO's from the facility

are distributed on different hosts, and thus the production of UNO's requires a generator which is logically centralized (i.e., two generators without knowledge of each other in the same Cronus cluster might generate the same UNO). Because we desire the facility to be continuously available with high probability, the implementation of the UNO generator facility is physically distributed.

The UNO facility is a very low-level component of Cronus because it is required to construct other relatively low-level facilities such as the Operation Switch (to be described in a forthcoming DOS Note).

As explained in the implementation section below, the UNO facility is composed of two types of software components, a relatively simple component, the SmallStepper, on every Cronus host, and a more complex component, the LargeStepper, residing on a few Cronus hosts called generator hosts. The production of a lengthy sequence of UNO's on one host is the result of cooperation between the SmallStepper component on that host and at least one LargeStepper component, usually remote.

One important goal of the UNO facility is to allow hosts without non-volatile storage to generate UNO's with a simple mechanism and with minimal delay. Because all Cronus hosts are expected to use the UNO facility, the implementation of the SmallStepper component is part of the integration cost of every

host. This cost is small because the SmallStepper component is a simple mechanism; the most difficult aspects of the reliability problem are treated in the design of the LargeStepper components. Delay in satisfying UNO requests is minimized by a design which requires synchronization between SmallStepper and LargerStepper components only infrequently; most requests can be satisfied locally and quickly.

High reliability is another important goal of the UNO facility, both in the sense of continuous availability and consistent restarts should all of the redundant generator hosts fail or be shut down at the same time. We will assume only that at least one generator host retains its non-volatile state (e.g., a disk file) across an outage of all generator hosts, in order to automatically resume the production of UNO's when that host is restarted. (31) A manual procedure will also be described to allow a restart of the UNO facility if all generator hosts lose non-volatile storage; initial UNO facility startup is a special case or this situation.

This note does not describe the way UNO's are stored and used by clients of the UNO generator, and in particular it does not define the way UNO's are used to refer to Cronus objects. A

(31) As detailed in the section on implementation, clients may continue to obtain UNO's from SmallStepper components for some period or time even if all generator hosts are unavailable; eventually, however, requests for UNO's must fail or block unless at least one generator host is operating.

Cronus Universal Identifer (UID) consists of a UNO and additional information related to the object, for example, the type of the object.

Unique Number Characteristics

A Unique Number is a 64 bit quantity whose concrete representation is dependent upon the host programming language and machine architecture. In C, for example, storage for two UNO's might be declared by

```
typedef struct
{
    char byte[8];
} UNO;

UNO a_UNO, another_UNO;
```

on systems which represent one character in eight bits. (32)

Viewing UNO's as abstract objects, they possess a small set of associated operations defined in the next section. One operation on UNO's, the "move" or "assign" operator, is not explicitly described below. Instead it is assumed that the host language provides one or more suitable primitives for this operation. In C, a possible coding is

```
for (i = 0; i < sizeof(UNO); i = i+1)
    a_UNO.byte[i] = another_UNO.byte[i]
```

(32) Note that the C/70 stores a character in ten bits; the C structures required to describe UNO's on the C/70 will be different from the structures given as examples in this note.

More efficient codings may exist.

The central property of the UNO abstraction is that two distinct invocations of the GenerateUNO entry point will never yield the same UNO as their results. By "distinct invocations" we mean successive invocations by the same DOS process, or invocations by two different DOS processes anywhere in the DOS cluster. Whether or not two UNO's are "the same" can be tested using the OrderOfUNOs operator (or perhaps by a direct bit-wise comparison, depending upon the UNO representation). Calls to the GenerateUNO entry point by processes in different DOS clusters may, in fact, yield the same bitstring; UNO's are universal only over the domain of a single cluster.

UNO's have two other useful properties. The internet host number of the machine which generated a given UNO is contained in a field of the UNO bitstring, and can be extracted with the OriginOfUNO operator. All UNO's generated by the same host are strictly ordered by time of creation, and can be compared using the OrderOfUNOs operator. UNO's generated by different hosts are not comparable; OrderOfUNOs will detect and indicate this situation to its caller.

The UNO size, 64 bits, was derived from assumptions about the maximum number of UNO's that could be generated over the lifetime of a Cronus cluster. We assume that the maximum number of hosts in a cluster is 1024, and the maximum lifetime of a DOS

cluster is 100 years. The implementation strategy will impose constraints upon the rate at which UNO's can be generated (fewer than 1000 per second per host) and on the rate at which a host can leave and reenter the cluster-wide UNO generation mechanism (about once every 10 seconds). The latter constraint effectively increases the boot-up delay of a Cronus host by a few seconds while it initializes its SmallStepper component.

Operations on UNO's

There are three primitive operations on UNO's (in addition to assignment, as mentioned above); these are the ONLY operations on UNO's which should be used by clients of the UNO generation facility. As will be described in the implementation section, UNO's have additional internal structure as a consequence of the generation technique, but this structure is implementation dependent, and should be regarded as subject to change in the future. The interface operations defined in this section will most often be available as procedure or system calls to a client process; in the C language they might appear as follows:

```
BOOLEAN GenerateUNO(unoptr) UNO *unoptr;
```

Generate a new UNO in the structure pointed to by unoptr and return TRUE, otherwise return FALSE.

```
HOSTNUM OriginOfUNO(unoptr) UNO *unoptr;
```

Return the internet address of the host which generated the UNO *unoptr, unless the UNO is well known, in which case return UNDEFINED.

```
UNOORD OrderOfUNOs(unoptr1,unoptr2)
UNO *unoptr1, *unoptr2;
```

Compare the UNO's *unoptr1 and *unoptr2, and return a result indicating equality, or the ordering between the UNO's, or declare them incomparable.

The types BOOLEAN, HOSTNUM, and UNOORD can be defined:

```
typedef int BOOLEAN;
#define TRUE        1
#define FALSE       0

typedef int HOSTADDRESS;     /* internet host address */
#define UNDEFINED   2**32-1      /* UNO is well known */

typedef int UNOORD;      /* UNO comparison scalar result */
#define UNOEQ       1            /* uno1=uno2 */
#define UNOLT       2            /* uno1<uno2 */
#define UNOGT       3            /* uno1>uno2 */
#define UNOINCOMP   4            /* incomparable */
```

The implementation strategy attempts to insure that these operations are continuously available, and will complete successfully unless the invoker's host fails during the call. GenerateUNO may fail (return FALSE) if all generator hosts are down or inaccessible for a long period of time; it is then at the discretion of the client to retry the operation immediately or after some delay, or to handle the failure in some other way. (A FALSE return from GenerateUNO is expected to be an extremely rare event.) An implementation of the SmallStepper component will guarantee that a GenerateUNO request always completes in a small, bounded amount of time, unless the client's host fails during the request.

A small portion of the UNO space is reserved for "well known

UNO's". These UNO's will never be returned by the GenerateUNO operation; some number of them will be statically associated with primitive objects in the Cronus system. For these UNO's the result of the OriginOfUNO operation is the value UNDEFINED, a 32 bit quantity which is not a valid internet address. When one or more or the arguments of OrderOfUNOs is a well known UNO, the result is always UNOINCOMP.

Implementation

The structure of a UNO as visible to the implementation has three fields: HostNumber, HostIncarnation, and SequenceNumber. In C, the structure might be declared:

```
typedef struct
{
    unsigned HostNumber:          10; /* bits */
    unsigned HostIncarnation:     32; /* bits */
    unsigned SequenceNumber:      22; /* bits */
}
```

A UNO with a HostIncarnation field equal to zero is a well known UNO. The HostNumber and SequenceNumber fields of a well known UNO are manually selected, arbitrary constants.

SmallStepper Components

When the GenerateUNO operation is invoked, control passes from the client to the SmallStepper component which resides on the client's host. The SmallStepper component enforces mutual exclusion (i.e., executes in a critical section) while responding

to client requests, and while performing incarnation number updates based on transmissions from LargeStepper components. The SmallStepper will create a new, properly formed UNO triple and return it to the client by storing the internet host number of this host (HostNumber), this host's current incarnation number (HostIncarnation), and a sequence number (SequenceNumber) into the UNO structure, provided that a current incarnation number is available. If no current incarnation number is available, the SmallStepper component will return FALSE.

Most invocations of GenerateUNO will cause the SmallStepper to increment a 22 bit sequence number stored in fast memory, and combine this with the host number and current incarnation number, also resident in fast memory, to form the UNO. This is a simple operation and can be done very rapidly.

Normally the SmallStepper maintains two (33) incarnation numbers in fast memory, the current incarnation number and the next incarnation number. If a GenerateUNO request causes the sequence number to overflow, the next incarnation number replaces the current incarnation number, and the sequence number is reset to zero. The next incarnation number will be "refilled" as soon as the SmallStepper receives a broadcast from a LargeStepper component. If the sequence number overflows and no next

(33) A SmallStepper may receive and queue more than two incarnation numbers in advance of need.

incarnation number is available, the current incarnation number becomes unavailable, and GenerateUNO returns FALSE.

It is an assumption of the Virtual Local Network (VLN) interface that internet host addresses are administratively assigned and "wired in" to each Cronus host (e.g., programmed into an EPROM or stored in Stable Storage on disk). A UNO generator software component obtains the host address of its host from the VLN interface.

The SmallStepper component obtains a new incarnation number passively, by listening for the next datagram transmitted on a well known multicast channel. This incarnation number becomes the current incarnation number if it was previously unavailable, else the next incarnation number if it was unavailable, else it is discarded. The LargeStepper components periodically (about once a second) transmit a new incarnation number on the channel; each number is guaranteed to be strictly greater than all previous incarnation numbers transmitted. Because UNO's generated on different hosts are distinguished by the HostNumber field, it is acceptable for several SmallStepper components to receive and use the same incarnation number from the multicast channel.

The separation of the SmallStepper and LargeStepper components removes the necessity for reliable, non-volatile storage at each host; the problem is now reduced to the

generation of a monotone incarnation number stream by the LargeStepper components. This problem is simpler because the LargeStepper components need only be present on a few hosts (the generator hosts), and it is reasonable to expect these hosts to have homogeneous architectures and operating systems, and non-volatile secondary storage. Note that this implementation strategy relies heavily on the sequencing and non-duplicating properties of the Virtual Local Network supporting host-to-host communication; see DOS Note 26 for further details.

LargeStepper Components

A subsequent DOS Note will discuss the implementation of LargeStepper components in depth. The paragraphs below convey the flavor of the approach, but they are not intended as a convincing argument that the LargeStepper components are, indeed, a reliable source of monotonically increasing incarnation numbers.

LargeStepper components can operate independently, i.e., without active acknowledgements from their peers. In its nominal, "homeostatic" condition, a LargeStepper component alternates between Passive and Active phases. The transition from Passive to Active is driven by a local timer, and the Passive phase will be about ten seconds in duration. The transition from Active to Passive occurs after a new incarnation number has been generated by this LargeStepper component, or if

the incarnation number generation is aborted by another, higher priority LargeStepper component, also in its Active phase. A static priority ordering for all LargeStepper components is derived from their host numbers.

In its Passive phase, a LargeStepper component receives two types of messages, and transmits none. One type of message received is the "I am here" message periodically emitted by every host; the LargeStepper component infers from the reception of an "I am here" message that its local network receiver is operable. The second type of message received is the "Intention" message produced by other LargeStepper components in their Active phases. Receipt of "Intention" causes the LargeStepper to increment its local copy of the incarnation number counter. (34)

When it enters the Active phase, a LargeStepper first tests to see that it received at least one "I am here" message since its last Active phase; if not, it enters an "Offline" phase awaiting manual intervention. If the receiver passes the test, the LargeStepper broadcasts a series of "Intention" messages containing the next incarnation number. Finally, if it has not received an "Abort" message from any other LargeStepper, this component updates its local copy of the incarnation number and multicasts the number on the channel received by SmallStepper

(34) As will be explained in detail in Part II, the LargeStepper does not increment it local incarnation number until it has received several "Intention" messages from the same source.

components. The LargeStepper then reenters the Passive phase.

If a LargeStepper is in its Active phase, and it receives an "Intention" message from a lower priority LargeStepper, it will transmit a series of "Abort" messages to the lower priority component.

The LargeStepper components will rely upon maximum elapsed time bounds for message transmission and LargeStepper transaction processing for correct operation. The UNO facility design as a whole relies on the sequencing and non-duplication properties of the Cronus Virtual Local Network.

This overview has omitted issues in several areas, such as assumptions about host and network failure modes, the specifics of phase transitions, and the startup of individual LargeStepper components. These issues will be treated in Part II.

A Justification of the 64 Bit UNO Length

The 64 bit length of UNO's is justified by the maximum rate at which UNO's might be generated and the expected maximum size and lifetime of a single Cronus cluster. We assume a maximum of 1,024 cluster hosts, hence the 10 bit HostNumber field. We assume a maximum incarnation number generation rate of 1 per second, and a maximum cluster lifetime of 100 years or slightly less than 2**32 seconds, hence the 32 bit HostIncarnation field.

The proper size of the SequenceNumber field is slightly more difficult to gauge. It should be large enough to guarantee that a SmallStepper component will not force a client to wait for a multicast incarnation numer as a result of sequence number overflow. Assuming a maximum GenerateUNO request rate of 1,000 per second, and a minimum HostIncarnation multicast rate of 0.1 per second, sequence numbers of at least 10,000 should be permitted. However, if the sequence number can grow larger a SmallStepper may continue to generate UNO's without frequent renewals of the incarnation number. A SequenceNumber field width of 22 bits allows SmallSteppers to generate UNO's at a maximum rate of 1,000 per second for about 1 hour, and for a much longer period of time at the lower request rates we expect in practice, without renewing the host incarnation number.

4.3 The Cronus Gateway

The Cronus Gateway forms an important, but low level, part of the Cronus system architecture. It provides a communication path between the hosts that make up a Cronus cluster and hosts in the rest of the Internet Catenet, including other Cronus clusters. The BBN-Cronus Gateway provides a direct connection between three networks, the Cronus Ethernet, the BBN Fibernet, and the ARPA-Net, and an indirect pather through other gateways to the rest of the Catenet. Creating the gateway involved

several considerations.

- o It must be a complete gateway. This means that it must communicate with the other gateways in the Catenet so as to be able to find the proper path to other networks and, conversely, to allow other hosts to find the path to the Cronus network. As gateway-gateway protocols are presently evolving this means that the code must also evolve in order to track the protocol changes.
- o It should require little software maintenance. The purpose of the Cronus project is to create a distributed operating system. The project is not involved with research on gateway architecture.

These, somewhat contradictory, goals have been achieved by basing the Cronus Gateway on the Cruuent Arpanet standard gateway code as off-the-shelf technology.

The gateway can be considered as consisting of two parts.

1. The standard gateway code, which process Internet packets, and can be regarded as a black box.
2. The local network interfaces, which convert local packets to Internet packets, and vise versa. These routines must be developed for each new network type the gateway is to be connected to.

In general protocol changes will affect only part 1. This means that the Cronus Gateway can track changes in the protocols (and improvements in the code) simply by rebuilding with the latest version of the standard code.

In the Cronus Gateway's case part 2 consists of three modules; one each for the Fibernet, Ethernet and Arpanet interfaces. The Arpanet module is also part of the standard code, so only the Fibernet and Ethernet and Ethernet modules had

to be developed.

To illustrate the functionality of these modules let us trace a packet from the fibernet interface to the Ethernet Interface. Note that the code is actually structured as a set of processes, one for each interface, and I/O occurs asynchronously; this discussion ignores the details of now packets are passed between the modules.

1. The packet is received by the hardware, and accepted at interrupt level. If a hardware error occurred during the receipt the buffer is flagged. The packet is passed to the fibernet input processor.

2. The packet is checked (at process level). If a hardware error occurred the packet is discarded, otherwise the local leader is checked. If the leader is not valid for an Internet packet the packet is also discarded, otherwise the local leader is removed and the packet is passed to the Gateway code (black box.)

3. In this case the gateway code decides the packet should be send out through the Ethernet interface. It therefore passes the packet, and the Internet address of its next destination, to the Ethernet output processor.

4. The output processor must transform the internet packet into a local packet. It does this by pre-pending a local leader.

   In the Ethernet case this involves a non-trivial operation -- determining the appropriate local destination. The problem is that Ethernet controller addresses are often set by the manufacturer, and there is no simple transformation between an Internet address (32 bits) and an Ethernet address (48 bits). Therefore the Ethernet routines maintain a translation table between the two. When the Ethernet output processor wishes to send a packet it checks the translatin table (using a hashing function). If a translation for that address exists, then it is used as the local destination address. Otherwise a multicast address is created by concatenating a 38 bit constant with the low 10 bits of the Internet

address(35). By convention each host on the network which wishes to receive Internet packets must listen for both its physical address, and the appropriate Multicast address.

The rest of the local leader is created straightforwardly using the gateway'[s Ethernet address as the source and the type field (assigned by XEROX) meaning the packet is of the Internet protocol.

5. Finally the packet is passed to the hardware, and output onto the network.

The reverse path (Ethernet to Fibernet) is similar, but differs in detail in steps 2 and 4.

When the Ethernet input processor receives a packet it may be one of 3 types.

1. The packet may be an Internet packet, with the correct physical destination address. In this case the packet is passed directly to the gateway.

2. The packet may be an Internet packet, but addressed to the gateway's Multicast address. This would happen if another Ethernet host wished to send a packet through the gateway, but didn't know the gateway host's Ethernet address. In this case a 'REDIRECT' packet is broadcast to all hosts. A REDIRECT packet is distinguished by its type field, and is used to inform other hosts on the network of the correct address translation for the gateway's Internet address. The original packet is then passed to the gateway.

3. The packet may be a REDIRECT originating from another host. In this case the information is used to update the translation table, and the packet is discarded.

The Fibernet output processor is considerably simpler than the Ethernets. Since a host's Fibernet address is a subset of

(35) It is sufficient to use only the low 10 bits, since an Ethernet is restricted to a maximum of 1024 hosts.

its Internet address there is a simple mapping from Internet to local addresses.

In conclusion: the primary goal in creating the Cronus Gateway was to get maximum functionally, from minimum effort. The only modules that were developed specially for this gateway were those required by the peculiar hardware environment. Later projects using the same hardware can avoid even that much effort. To further dimish maintenance needs, the modules developed will be turned over to the Arpanet gateway developers for integration into the standard sources.

5 Implementing the System Ethernet Interconnection

This section discusses aspects of the design and implementation of the Ethernet interface for the M68000 GCE and the C/70 UNIX.

5.1 Connecting Multibus-Based 68000 Systems to the Ethernet

INTRODUCTION

One important part of the Cronus Distributed Operating System is the "Generic Computing Element," or GCE. This is a Multibus-based microprocessor system, which in the current implementation uses a Motorola M68000 processor. GCEs are envisioned as low-cost hosts which can be used for a variety of applications, including terminal concentrators, file servers, and "Access Machines" which can be used to interface a new host into the Cronus system using standard commercial serial communication interfaces.

A current project task is to connect the GCE to the 10 MBit Ethernet which is being used as the communication substrate for the initial Cronus implementation. This note describes the features of the Multibus Ethernet controllers that were chosen as candidates for inclusion in the GCE component of the Cronus system. Because the GCE is a Multibus-based microcomputer, the choice of controller is limited to (currently) three

possibilities: the Intel iSBC 550, the Interlan NI3010, and the 3Com 3C400.

The Intel controller was originally chosen, as it was the only controller available at the time (early 1982). We obtained one of the Intel boards and began writing a driver for it. While working on the driver, we noticed some possible performance limitations in the Intel controller, and new product announcements which were coming out at that time caused us to look into possible alternatives. As a result of this, the 3Com controller was selected as an alternative, to be used in the remaining GCEs.

COMPARISON OF INTEL & 3COM CONTROLLERS

The Intel iSBC 550 and the 3Com 3C400 Ethernet controllers represent widely varying design philosophies and goals. The Intel controller consists of two boards, is microprocessor-based, and attempts to deal in a reasonable manner with multiple host processors on a single Multibus by using a message-passing scheme to communicate with other processors. This last may appear to be an advantage, but the result is that in single-processor systems (as the GCE is), the host software is much more complex than it is for the 3Com controller, which has no microprocessor controller, and is a much more straight-forward design for single host systems with its status register and transmit and receive buffers which are mapped directly into Multibus memory space.

There have been a few problems with the iSBC 550 controller besides the problems of complexity introduced by the message-based communication. One of these problems is that the controller was designed for use with Intel processor boards, which means that it orders bytes differently than the 68000, which is the processor used in the GCE. This means that the 68000 must swap bytes in a message before handing it to the Ethernet controller, and results in each packet being copied twice: once by the 68000 to swap the bytes, and once by the controller's 8088 to load the transmit buffer or deliver data from the receive buffer. The 3Com board, on the other hand, has a switch which indicates the byte-ordering scheme it is to use, and since its transmit and receive buffers are located in Multibus memory space, only a single copy operation need be performed on a packet. Thus, the achievable throughput should be higher with the 3Com board than with the Intel board.

Probably the most annoying problem with the Intel controller is the lack of proper documentation for communicating with the board. Again, because the controller was designed with only Intel hosts in mind, Intel will supply code for their processors to communicate with the controller. Other processors require custom implementations of the communication facility, but the documentation supplied with the board is wrong in several places and inadequate in others. Further, the iSBC 550's microprocessor does not do sufficient error checking on the messages passed to

it.

Other differences between the two vendors' boards are:

1. As noted, the Intel controller consists of two boards (though it has been referred to above in places as if it were a single board), while the 3Com controller is a single board. Because the 3Com controller has fewer parts and interconnections, it is likely to be more reliable in the long run. Also, the 3Com controller uses less power than the Intel controller (5A max vs. 9A max).

2. The Intel controller provides no way to change the board's Ethernet address. This will make building a system with multiple Ethernet controllers in the same Multibus chassis more difficult, as all such controllers should ideally have the same address. Also, swapping boards in order to debug or fix a hardware problem becomes more difficult in most applications, though this should not be a problem in the DOS, due to its use of Internet addresses and a dynamic lookup table to translate to Local network addresses. The 3Com controller requires that the host processor supply the Ethernet address, and provides a "hint" in ROM which is mapped into Multibus space, which the host may either choose to use or to ignore, so this board would not cause any problems in a system where it is important to be able to interchange controllers without changing Ethernet addresses.

3. Older versions of the Intel controller are sensitive to the type field of a received packet, requiring the user to specify which Data Link Types he is willing to accept. This is contrary to the Version 1.0 Ethernet specification, which states (in Section 6.2.2) that the type field is to be uninterpreted by the Data Link Layer (the controller implements the Physical and Data Link Layers of the network). This has been fixed in later versions, but the 3Com controller is (according to the documentation) completely insensitive to the type field.

4. The Intel controller requires that the system have memory in Multibus memory space. The 3Com controller contains its own memory for transmit and receive buffers, and this memory may be mapped into Multibus memory or I/O space. This reduces the cost and size of a minimum configuration, but could be a problem in large Multibus systems, where the necessary 8K byte block of contiguous

memory (I/O) locations might not be available.

5. The Intel board does Multicast address recognition, but only for a maximum of 8 addresses. The 3Com board does no Multicast recognition, and will simply deliver or not deliver all Multicast-addressed packets to the host. This means that the host must do any necessary Multicast recognition, which could preclude the use of protocols which make heavy use of the Multicast feature.

6. The 3Com board does not calculate the retransmission backoff interval, instead generating an interrupt to the host, which must calculate the backoff and load it into a register on the controller. 3Com claims that "The processor overhead to support backoff in software is minimal. Studies show that Ethernet packets typically experience collisions less than 0.03% of the time." The Intel controller does the backoff calculation and retransmission automatically.

   Both boards apparently have problems receiving a string of back-to-back packets. According to David Boggs of Xerox, the Intel board can handle no more than three back-to-back packets, while the 3Com board can handle only two.

CURRENT STATUS

We have had major problems in getting the Intel controller working. Intel has been unable to resolve these, and as of this date (July 1), the driver for the Intel controller is still not working. The problems we encountered with the Intel controller caused us to attempt to speed up delivery of the 3Com controllers. While awaiting receipt of the new controllers, a driver was written, based on the documentation supplied to us by 3Com when we placed the order. The driver interface was originally specified with multiple controllers in mind, and so looks the same to the application program whether the actual controller is an Intel or a 3Com. Besides the "expected" entry

points for sending and receiving packets, the driver also contains entries for reporting statistics, such as the number of packets received with bad checksum and the number of packets which experienced collisions on transmission.

Two 3Com controllers were received in late June, 1982, and within a week of receipt were sending and receiving packets. The driver for the controller is currently being debugged.

5.2 Connecting C/70 TCP to the Cronus Ethernet

Introduction

This note describes the effort underway as a part of the Cronus Distributed Operation System project, to connect a BBN Computer Corporation C/70 Unix system running TCP to standard 10 Megabit/second Ethernet. The BBN Cronus local area cluster includes a pair of C/70 UNIX systems interconnected to each other and other cluster hosts via the Ethernet. The C/70 is a general purpose microprogrammed minicomputer designed for efficient execution of the language C, and runs the UNIX operating system. An implemention of TCP has been written in C and is currently providing service to a number of network host machines at BBN. The C/70 currently has no hardware or software support for an Ethernet attachment. The C/70-Ethernet effort is therefore broken into two tasks: integrating the required interface

hardware and writing the software I/O drivers.

Hardware design

Before we go into the hardware design, the C/70 I/O design philosophy should be summarized. One of the goals in the design of the C/70 was to make the individual I/O controllers as simple as possible. Microcode would then be written to do much of the work normally found in I/O controllers. Since the machine has a fast microinstruction execution time (135ns) and extensive effort went into development of the C/70 microprogrammable architecture and microcode development and support tools [1,2,3], this was a reasonable approach to take.

The C/70 uses two methods for the transfer of information between I/O devices and the CPU. The first method is for the device to cause a microinterrupt and have the microcode read or write the data to or from the device. This method is analogous to programmed I/O in a traditional minicomputer. The second method is called pseudo-DMA. With this method, the I/O controller begins the transfer of a small block of data (16 words) by causing a microinterrupt. The service routine microcode tells the controller to proceed with pseudo-DMA. Then the I/O controller and the CPU enter a tight loop transfering data across the I/O bus. The number of words to be transferring is fixed in advance.

The decision as to which method is to be used has a definite impact on the type of I/O controller one designs for the C/70. Generally single-transfer I/O is used for "slow" device such as communication line interfaces while pseudo-DMA is used for high performance devices such as the disk controller.

The design of the Ethernet interface hardware was driven by several DOS project goals. The primary goal was the desire to use off-the-shelf local network technology. Two other goals were high throughput and short delivery time. Since no Ethernet interface board currently exists for the C/70, some amount of hardware development would be necessary to interface to the C/70 I/O bus. The first goal strongly suggested that we buy a commercially available Ethernet controller instead of designing one specifically for the C/70. We chose the Interlan NM10 Ethernet protocol module, based on availability, performance and price. It has been designed to be customizable for many of the microcomputers and minicomputers currently on the market and therefore was not locked into one vendor's I/O protocol. In particular, it is used as a "daugther board" in Interlan's own host-specific Ethernet interface products.

The NM10 interface presented us with two problems. The more important one was that its host parallel transfer protocol is not an etficient way to transfer data to and from the C/70. This problem was dealt with by designing a pseudo-DMA interface for

the C/70. The other problem is that the NM10 also doesn't specify the byte data transfer time. We were therefore required to design our interface to make all transfer, both data and control, asynchronous. Timing independence is handled in the hardware for psuedo-DMA; however, reading and writing NM10 registers require the microcode to handle the timing independence.

The interface board, called the MIENI (for MBB Interlan Ethernet Interface) has been designed, fabricated, and is being debugged. While the testing and debugging are not yet complete, the MIENI will currently perform many of its functions. After hardware debugging is complete, the microcode will be tested.

Concurrently with hardware design and testing, significant effort has gone into documenting the C/70 Ethernet interface to facilitate programming, production, and maintenance. This documentation is approximately 70% complete.

Microcode drivers

As stated earlier, the C/70 is designed to have the microcode do much of the work in an I/O transfer. Although the C/70 architecture and software tools enhance the microcode development process, the microcode for the MIENI is rather complex. Most of the complexity is caused by the timing independence problem mentioned earlier. To read or write one internal NM10 register

the microcode must inform the NM10 of the transfer and wait for a microcode interrupt when the operation is complete. Such transfers are very common in the MIENI packet transfer operations and this increases the microcode size and complexity.

The desire to minimize processor intervention in network transfers also adds complexity to the microcode. To decrease necessary intervention, the microcode is designed to allow the user to chain many buffers together for scatter-gather transfers. This approach greatly loosens the requirements for high level macrocode interrupt latency at the expense of microcode space and complexity.

The microcode is completely written and awaits the hardware for testing. Along with this microcode, a macrocode program has been written that can exercise all the features of the microcode in a controlled environment.

Macrocode drivers

The final phase of this task is develpopment of the macrocode software necessary to connect the microcode routines to TCP. This software is broken up into two modules, the device driver and the packet encapsulator. The job of the device driver is to handle the data coming from and going to the microcode. This involves servicing interrupts, keeping buffers queued on the input side and handing incoming packets to the higher levels.

This device driver has been completed.

The packet encapsulator is the software module that knows how to take TCP Internet packets and append Ethernet addresses before shipping them to the microcode. It also handles routing table updates. The basic problem is to map the 32 bits of an Internet address into the 48 bits of Ethernet. This is done by keeping routing tables with the correct Ethernet address for a given Internet address. There are simple protocols described in a companion note to get routing information when the network comes up. This second module is approximately 80 percent complete.

The debugging of the packet encapsulator and macrocode device driver are waiting for the microcode device driver to be tested and debugged.

References

[1] M. F. Kraley, et. al., "Design of a User Microprogrammable Building Block," Proc. 13th Annual Workshop on Microprogramming, December 1980.

[2] S. Geyer and A. Lake, "Development Tools for User Microprogramming," Proc. 14th Annual Workshop on Microprogramming, August 1981.

[3] R. Weissler, M. Kraley, and P.Herman, "MBB Microprogrammer's Handbook," BBN report No. 4268, August 1980

5.3 MBB Interlan Ethernet Interface Preliminary Specification

INTRODUCTION

This document provides a functional description of the hardware module used to interface the Interlan NM10 Ethernet protocol module to the Microprogrammable Building Block (MBB) series of BBN computers. The NM10 protocol module implements the 10 Mbps Ethernet version 1.0 data link and physical channel functions, and additionally performs data buffering, address recognition and filtering, diagnostic functions, and network statistics collection.

The MBB Interlan EtherNet Interface (MIENI) couples the NM10 into the MBB I/O system by providing MBB I/O bus address decoding, priority interrupt encoding, control signal mapping, and data buffering necessary to connect the NM10 to the MBB in a high performance configuration. In addition, diagnostic and self test features have been included to aid in testing and fault isolation. The MIENI, with the NM10 as a daughterboard, appears to the microcode as a standard MBB I/O interface, with addressable control, status, and data buffer registers, and microinterrupts to indicate important events.

BACKGROUND

The MIENI is being developed as part of the Distributed Operating System (DOS) project, where it will be used to connect C/70 UNIX

systems to other minicomputers through a high-speed local network for distributed operating system research. MIENI is one part of a larger system effort to effectively couple the C/70 Unix system into a high-speed local network, so that remote operations made on the user's behalf over the network do not incur large performance penalties relative to local operations.

Following a detailed evaluation of available technologies, the Ethernet local area network architecture was selected for use in DOS due to its high performance, cost-effectiveness, availability of interfaces for relevant host computers, and multi-vendor support. The Interlan NM10 was chosen as the starting point for a C/70 Ethernet interface due to its (relative) availability, and its use of a host-independent interface which could be merged into the underlying MBB I/O system. This interface, called the Interlan Standard Module Interface (ISMI), is an 8-bit bidirectional data path controlled by 5 handshaking lines. There is the likelihood that Interlan will produce a line of local network interfaces (ringnet, IEEE 802, improved Ethernet, etc.) which will be plug-compatible with the network-independent ISMI interface. By designing to the ISMI standard (rather than making the MIENI-to-ISMI interface specific to the NM10) we can take advantage of their future network product offerings.

DESIGN ISSUES

Here we simply list the major design issues and goals which must be addressed by the combined hardware, microcode, and Unix device driver system for C/70 Ethernet support in the DOS context.

- Programming compatibility with Unibus Ethernet interface used elsewhere in DOS on DEC VAX and PDP-11 hosts
- Minimum amount of CPU servicing necessary per data transfer
- Minimum propagation delay through MIENI buffering
- Full Ethernet 1.0 compatibility
- Need to run disk transfers and Unix while network I/O active, and therefore the need for interrupt throttling
- MBB design philosophy of minimum hardware for I/O
- Timescale for development (operational by June 1982)
- Minimum complexity of design for full-scale production
- Conversion from 8-bit NM10 data to 20-bit MBB format
- Maximum use of programmed logic for ease of change
- Small initial build
- Ease of testing and online diagnosis

Device Functions

The MIENI functions available to the microprogrammer can be broken into two groups: high-speed transfers of data blocks requiring optimized pseudo-DMA microcode, and control operations which generally involve reading and writing interface registers. Control operations are used to set up the block data transfers, and it is also possible to circumvent the high-speed data path hardware to manually transfer data, byte by byte, to and from the NM10. Both classes of functions use microinterrupts for efficient event signalling.

DESIGN ALTERNATIVES

Data Path Design

From issues listed in section 3, it is apparent that the primary tradeoff in the hardware design is performance versus development time and cost. The following discussion of design alternatives centers on the MIENI high-speed data path design, as the MBB and NM10 bus interface control logic is common to all designs.

Referring to the NM10 specification, it might be suggested that the minimal data path design of putting no buffering in the MIENI, and having all interface complexity handled by the microcode is the ideal solution. Due to the lack of critical timing information from Interlan, however, the interface would have to be polled or would interrupt the processor on a character-by-character basis, which would have severe performance implications. This is not a bug on Interlan's part however, as they have specified a general interface which should not be tied specifically to the NM10. For example, it is unspecified how long it will take for a new data word to be available from the NM10 Receive Buffer when a word is read by the MIENI. This means that the microcode cannot assume a priori how long to wait between data transfers, which is critical if multi-megabit transfers are to be handled while other Unix functions are in process (disk transfers, etc.)

The other extreme, putting a microprocessor in the interface between the NM10 and the MBB, is completely against the philosophy of the MBB I/O system, and is unnecessarily complex

for this application, due to the amount of processing already done by on-board the NM1Ø. It would eliminate the need for the microcode to manipulate the NM1Ø internal registers, but that is a small savings compared to the development costs.

Therefore we have implemented an intermediate solution, consisting of the minimum amount of FIFO buffering necessary in each direction of data transfer to allow efficient pseudo-DMA block accesses. The NM1Ø bus, FIFOs and MBB I/O bus interfaces will be controlled by a collection of programmable logic elements (PALs and PLAs). More elaborate versions of this approach have been used in the MBB disk interface (MDI) and MBB Ungermann-Bass interface (MUBI). Since we still need to access the NM1Ø control and status registers to issue commands, two basic modes of transfer are provided, referred to as Pseudo-DMA Block mode, and Transparent Character mode.

Byte Packing

Another issue which impacts the interface design is the need to convert the 8-bit data format of the NM1Ø to 2Ø-bit MBB words. The conversion could be done in hardware, microcode or software, and the design implements the conversion in hardware on the grounds of increased performance. To be specific, each 2Ø bit MBB word is broken into two 1Ø-bit half-words, and the data octets or bytes to be transferred over the Ethernet are packed two to a word, right-justified within each half-word.

The alternative of implementing a byte packing and unpacking routine in software has been judged to be an excessive computational task for even moderate network traffic loads. It would be nice to be able to do the conversion in microcode during the pseudo-DMA transfers, but having to manipulate the MBR effectively doubles the length of time to transfer two 8-bit datum. Details on microcode and interface control sequences which allow comparisons of the different byte-manipulation options is contained in an attached document. We have worked out a preliminary design for the control logic to do the multiplexing and demultiplexing in the middle of the FIFOs, at the cost of a 20-pin PAL in each direction.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.