A systems approach to synchronization and naming in a distributed computing environment

James Arthur Davis
Iowa State University

Follow this and additional works at: http://lib.dr.iastate.edu/rtd
Part of the Computer Sciences Commons

Recommended Citation
INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or “target” for pages apparently lacking from the document photographed is “Missing Page(s)”. If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of “sectioning” the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.
Davis, James Arthur

A SYSTEMS APPROACH TO SYNCHRONIZATION AND NAMING IN A DISTRIBUTED COMPUTING ENVIRONMENT

Iowa State University

University Microfilms International 300 N. Zeeb Road, Ann Arbor, MI 48106

Ph.D. 1984
A systems approach to synchronization and
naming in a distributed computing environment

by

James Arthur Davis

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY
Major: Computer Science

Approved:

Signature was redacted for privacy.

In Charge Of Major Work
Signature was redacted for privacy.
For the Major Department
Signature was redacted for privacy.
For the Graduate College

Iowa State University
Ames, Iowa
1984
TABLE OF CONTENTS

TRADEMARKS ........................................................................................................ vi

1. INTRODUCTION .................................................................................................. 1
   1.1 Introduction ....................................................................................................... 1
   1.2 Motivation ......................................................................................................... 6
   1.3 Problem Statement ........................................................................................... 7
   1.4 Thesis Plan ....................................................................................................... 8

2. REVIEW OF BACKGROUND MATERIAL ON DISTRIBUTED SYSTEMS ....... 10
   2.1 Motivation for Distributed Systems Research .................................................. 10
   2.2 Definition and Classification of Distributed Systems ........................................ 12
      2.2.1 Characteristics of Distributed Systems ...................................................... 13
      2.2.2 A Categorization of Computer Systems ..................................................... 16
   2.3 Philosophies on Replicated Data ...................................................................... 18
   2.4 The Consistency Problem .................................................................................. 22
      2.4.1 Mutual Consistency .................................................................................... 23
      2.4.2 Internal Consistency ................................................................................... 25
      2.4.3 Atomic Operations ...................................................................................... 27
      2.4.4 Serialization ............................................................................................... 28
      2.4.5 Deadlock ................................................................................................... 29
      2.4.6 Characteristics of a Good Solution ............................................................... 31
   2.5 Partitioning and Site Failures ............................................................................ 32
   2.6 Related Research ............................................................................................. 33

3. REVIEW OF BACKGROUND MATERIAL ON COMMUNICATION NETWORKS .... 35
   3.1 Introduction ....................................................................................................... 35
   3.2 The OSI Reference Model ................................................................................ 36
   3.3 Standard Gateway Types .................................................................................. 38
   3.4 Virtual Circuits and Datagrams ....................................................................... 40
   3.5 Relevant Communication Issues ....................................................................... 41
   3.6 Summary ........................................................................................................... 44

4. SYNCHRONIZATION AND CONSISTENCY OF DISTRIBUTED DATA .......... 45
   4.1 Introduction ....................................................................................................... 45
   4.2 Survey of Methods to Preserve Consistency ..................................................... 45
      4.2.1 Centralized Control of Updates ................................................................. 47
         4.2.1.1 Centralized Database Organization ................................................... 47
4.2.1.2 Primary Site Organization ........................................ 48
4.2.1.3 Summary of Centralized Control .................................. 49
4.2.2 Decentralized Control of Updates .................................... 50
4.2.2.1 Global Locking ...................................................... 50
4.2.2.2 Time Stamps and Event Counters ................................. 52
4.2.2.3 Summary of Decentralized Control ................................. 56
4.2.3 Summary ............................................................... 57
4.3 Proposed Algorithm ...................................................... 58
4.3.1 Introduction ............................................................ 58
4.3.2 Version Numbers ....................................................... 58
4.3.3 Reading and Updating Replicated Data ............................... 60
4.3.4 Distributed Updates of Replicated Data .............................. 63
4.3.5 Summary ............................................................... 65

5. NAMING IN A DISTRIBUTED ENVIRONMENT ................................ 66
5.1 Introduction ............................................................... 66
5.2 Related Research .......................................................... 68
5.3 Naming Conventions ....................................................... 70
5.4 The Proposed Name Management Method ................................. 70
5.5 The Name Table Implementation ........................................ 72
5.5.1 The Name Table Data Structure ...................................... 74
5.5.2 Name Table Entries .................................................... 77
5.5.2.1 The Gateway Name Table Entry ................................... 77
5.5.2.2 The Communication Port Name Table Entry ....................... 78
5.5.2.3 The Replicated Data Name Table Entry ............................... 79
5.5.2.4 The Group Name Table Entry ...................................... 80
5.6 Information Sharing Through Groups .................................... 81
5.7 Alternate Name Mappings ................................................ 83
5.8 Summary ............................................................... 84

6. THE INTELLIGENT MESSAGE TRANSPORT SYSTEM ...................... 85
6.1 Introduction ............................................................... 85
6.2 System Topology .......................................................... 86
6.3 Intersite Communication .................................................. 91
6.3.1 Communication Modes ................................................ 92
6.3.2 The Store-and-Forward Scheme ..................................... 93
6.3.3 The End-to-End Acknowledge Protocol ................................ 96
6.3.4 An Example of Intersite Communication .............................. 97
6.4 The Gateway Processor .................................................... 99
6.4.1 Introduction ............................................................ 99
6.4.2 The Architecture of the Gateway Processor ........................... 99
6.4.3 Gateway Message Formats ............................................ 105
6.4.3.1 The Independent Site Protocol (ISP) ............................... 106
6.4.3.2 The Independent Network Protocol (INP) .......................... 109
6.4.4 High-Level Gateway Functions ........................................ 110
   6.4.4.1 Gateway-to-Gateway Interface Functions ...................... 110
   6.4.4.2 Gateway-to-Site Interface Functions .......................... 114
6.5 Summary ............................................................................. 116

7. CONCLUSIONS ........................................................................ 117
   7.1 Research Summary ............................................................. 117
   7.2 Further Research ............................................................... 119

8. REFERENCES ........................................................................... 121

9. ACKNOWLEDGEMENTS ......................................................... 126
# LIST OF FIGURES

| FIGURE 1-1. Architecture of the intelligent message transport system | 3 |
| FIGURE 1-2. Architecture of a conventional distributed system | 5 |
| FIGURE 2-1. A taxonomy of computer systems | 17 |
| FIGURE 2-2. Evaluation sequence for interfering transactions | 26 |
| FIGURE 5-1. An example of 2-3 search tree insertions | 75 |
| FIGURE 6-1. A distributed system topology | 87 |
| FIGURE 6-2. Graph representation of Figure 6-1 | 88 |
| FIGURE 6-3. Optimizing the intersite path length | 90 |
| FIGURE 6-4. The end-to-end acknowledge protocol | 96 |
| FIGURE 6-5. Intersite communication example | 98 |
| FIGURE 6-6. A block diagram of the gateway processor | 101 |
| FIGURE 6-7. The gateway processor | 102 |
| FIGURE 6-8. The iSBC 186/51 Ethernet controller | 103 |
TRADEMARKS

PDP, UNIBUS, and DEC are trademarks of Digital Equipment Corporation.

MULTIBUS, iSBC, and COMMputer are trademarks of Intel Corporation.

Ethernet is a trademark of Xerox Corporation.

Ada is a trademark of the U.S. Department of Defense.

UNIX is a trademark of Bell Laboratories.
1. INTRODUCTION

1.1 Introduction

Recently, there has been considerable research into the areas of distributed systems, object-oriented architectures, and architectural support of database functions and high-level languages. Most often, these research areas have been isolated and investigated independently of each other. We believed that new contributions can be made to the study of distributed systems if certain key concepts are investigated within the framework of the total system. For example, while it is worthwhile to investigate synchronization methods for distributed data, the results of the research will become more significant if system-wide factors, such as communication bandwidth limitations, are also taken into account.

In this thesis, we will develop a distributed system architecture that supports loosely-coupled computing resources (sites) interconnected by heterogeneous communication networks. The underlying theme is that the difficult problems associated with decentralizing the control aspects of the system must be investigated and solved within the context of the total system. We found that we were able to develop relatively simple and efficient solutions to the problems of synchronization and naming of distributed data by moving the control of certain key functions from individual sites to specialized network processors. The solution to the synchronization problem is given in Chapter 4 and the solution to the naming problem is given in Chapter 5.

The first step, however, is to specify the topology of the system, identify its various components, and investigate the interactions between components and between system hardware and software.

In distributed system organizations that provide for sharing of named data, certain functions related to the consistency and naming of the shared data must be implemented by system-wide policies and conventions. These policies are typically
enforced by either: (1) centralizing the control of naming and synchronization by
designating one site as the primary site and requiring that all update and naming
requests be processed by that site; or (2) by decentralizing control and requiring all
sites to cooperate with one another and collectively implement these policies.

In the first case, a global name server or a primary site is not an attractive means
of organizing system control functions because the integrity of the entire system is
dependent on the integrity of the primary site. Also, the throughput of the system
may be limited by the bandwidth of the communication paths leading to the primary
site. If one envisions a system where sites are geographically dispersed, it is likely
that distant sites will be connected by a relatively slow speed network, such as stan­
dard telephone equipment. Because all transactions would need to be processed by the
primary site, including updates involving distant sites, throughput would be slowed
to the speed of the slowest communication path to the primary site.

In the second case, having individual sites enforce global control policies by
cooperation is also not the best solution. As for the primary site, the integrity of dis­
tributed data is dependent on the integrity of the site that coordinates the update
transaction. Thus, sites are not autonomous in that they must rely on the ability of
other sites to correctly process transactions. Also, because the same control functions
must be individually provided by all sites, it is implied that sites are nearly homo­
geneous, at least with respect to system control software.

In the proposed distributed system architecture, control functions for naming
and synchronization are provided by an intelligent message transport system. This
message transport system provides a super structure that connects the sites in a con­
sistent, logical fashion. It consists of the intersite communication networks and intel­
ligent network controllers, called gateway processors. The message transport system
does not play a role in the activities local to a site; it provides only system-wide con­
trol functions, such as:
Figure 1-1a. Site architecture

Figure 1-1b. One possible system topology

Figure 1-1. Architecture of the intelligent message transport system
- A consistent communication network interface that supports different types of communication media, including serial line protocols and local area networks (e.g., the Ethernet).
- Management of the control logic needed to synchronize updates of distributed replicated data.
- Management of the system-wide name space for shared data in a way that provides for a user-defined scope of reference for names.
- Message routing functions without the use of global site address tables.

The architecture of the proposed system is shown in Figure 1-1. For comparison, the architecture of a typical distributed system is shown in Figure 1-2. They differ both in the architecture of a site and in the system topology.

In a conventional distributed system organization, a site consists of processors, memory, and a network interface that are tightly-coupled by a high-speed system bus\(^1\) as shown in Figure 1-2a. The network interface will typically be capable of handling routine network protocols without processor intervention, however, the generation of control messages for the management of synchronization is handled by a local site processor that executes local system software. Failure to produce the correct control messages could result in total system failure. Because the processor is controlled locally, it can be affected by erroneous or mischievous user programs, as well as hardware failures and power interruptions. The architecture of a site in the proposed system is shown in Figure 1-1a. The gateway processors that connect sites to the network are external to the site in that they do not connect directly to the site system bus. Local processors do not have control over the gateway processor; instead, they must request services from the gateway processor by a small set of simple commands. This decoupling of the locally controlled processors from the network

\(^1\)An example would be a MULTIBUS system with an ISBC 186/51 COMMputer Ethernet controller.
Figure 1-2a. Site architecture

Figure 1-2b. Possible system topologies

Figure 1-2. Architecture of a conventional distributed system
interface enhances system reliability by isolating the network from site failures.

At the global level of the proposed system topology (Figure 1-1b), gateway processors form junctions that can interconnect several different types of communication networks to one site. The topology of the system has a tree-like structure\(^2\), where subtrees correspond to some logically connected geographic region, such as a laboratory or a group of offices. (As we will see later, the transport system makes use of these localities to limit message traffic to portions of the network that are involved with a particular transaction.) This is in contrast to a conventional distributed system organization (Figure 1-2b) where the network is a passive component and must be joined to other networks at a site. Because the system control functions are provided only by sites, the resulting topology must be point-to-point or a ring, and sites have the responsibility of routing messages and maintaining the consistency of site address tables.

1.2 Motivation

The motivation for this work arose from a desire to investigate the nature of hardware/software interactions and tradeoffs in a distributed computer system. From the beginning, we thought that a good understanding of the problems inherent in decentralizing control would allow us to design a distributed computer architecture that avoided or lessened many of these problems. This thesis focuses on two such problems, namely, preserving the consistency of replicated data and managing the name space in a distributed environment. There have been several solutions proposed for these problems (reviewed in Section 4.2), however, most are not candidates for implementation because of their inability to correctly handle communication systems that are less than totally reliable. We found that the performance and simplicity of such solutions could be greatly improved by designing the system specifically with

\(^2\)Strictly speaking, the topology is a directed graph.
these problems in mind. Our solution is a tree-like organization that limits message traffic to the portion of the system that is directly involved, and a specialized network interface processor that prevents inconsistency in replicated data by analyzing and rejecting certain update transactions.

1.3 Problem Statement

The problem to be solved in this thesis is the development of a distributed computer architecture that supports loosely-coupled heterogeneous computing resources in a way that mitigates two of the most difficult problems inherent to distributed systems, namely synchronization of updates to replicated data, and the naming of distributed shared data. The proposed architecture is novel in that it combines a tree-like topology with intelligent network processors that provide control functions typically implemented in each site. The benefits of the proposed system are:

- Each site is autonomous and does not need to maintain a list of other sites that it communicates with, nor do sites need to consult global address tables.
- Control of synchronization and naming is decentralized; there are no primary sites or global name servers.
- No assumptions have been made about the message transport medium; in particular, messages need not arrive in the order in which they were sent, they may be duplicated or lost, and arbitrary delays can be accommodated. Furthermore, we do not require that the network have a broadcast capability.
- Resiliency is enhanced by providing multiple paths between critical sites. The gateway processors that will be developed are capable of dynamically altering the communication path for messages should a regularly used path become inoperable.
- In conventional distributed systems, the system-wide name space is typically global. In the proposed system, users may export names to specific sites, and
restrict access of shared data to specific groups of users. Thus, the scope of refer-
ence for a name is limited to the sites specified and one identifier may have an
overlapped scope without confusion.

- The intelligent message transport system enhances performance by limiting the
  transmission of control messages to the portion of the network that connects the
  sites involved. Furthermore, the tree-like topology of the system allows inter-
site communications to be processed in parallel.

- The intelligent message transport system guarantees the delivery of messages
  on-line sites.

1.4 Thesis Plan

The remainder of this thesis describes the intelligent message transport system
and demonstrates how the problems of synchronization of replicated data and naming
of shared data are more easily solved within the context of the proposed system.

In Chapter 2, we provide background information on relevant issues in distrib-
uted computing, and discuss fundamental concepts and definitions that will be used
in later chapters. We will investigate the characteristics of a distributed system with
the goal of developing insight into the problems that arise when control is decentral-
ized.

In Chapter 3, we discuss relevant background material on communication net-
works including the OSI reference model, and the Xerox Pup internetwork.

In Chapter 4, we analyze various proposed algorithms for handling synchroni-
ization of replicated data, and offer a more efficient solution for the proposed system
that is based on the method described in Reed [Reed 1983]. In this algorithm, no glo-
bal locking is employed; instead, transactions are viewed as a sequence of immutable
versions, with each version resulting from an update transaction.
In Chapter 5, we show how the gateway processors manage the global name space in a way that allow user processes to create named data and selectively export it to particular sites or groups of users.

In Chapter 6, we discuss details about the implementation of the gateway processor, and the support for naming and synchronization. We will develop two simple communication protocols that provide standard gateway-to-site and gateway-to-gateway interfaces.

Finally, in Chapter 7 we summarize the work presented and outline future research areas.
2. REVIEW OF BACKGROUND MATERIAL ON DISTRIBUTED SYSTEMS

2.1 Motivation for Distributed Systems Research

In recent years, there has been much experimentation in industry as well as academia with novel computer architectures that consist of loosely-coupled processors interconnected by a communication network. Most often, these types of systems are grouped together and labeled with the catch-all title of distributed processing systems. The phrase distributed processing has been so casually used in the literature, that it no longer provides a useful description of the system that it is applied to. In Section 2.2 we will attempt to gain a better insight into the nature of distributed systems by identifying their basic characteristics and providing a means of comparing them to other types of computer architectures.

The continued interest in studying distributed systems demonstrates that (1) there exists a class of problems that cannot be conveniently solved using contemporary von Neumann architectures, and (2) despite all the past and present research in this area, there still exists technical problems for which a suitable solution has not been found. One such problem, the synchronization of concurrent updates of replicated data, will be discussed in Section 2.4 and also in Chapter 4. The remainder of this section will be concerned with a more detailed discussion of some of the motivations for continued research in distributed system architectures. These include:

- **Natural Solution.** For certain classes of problems, a distributed computing architecture has a close correspondence with the problem, and thus, provides a more natural solution than does a conventional von Neumann architecture. For example, some applications require that the data entry facility be physically separated from the storage or data processing facilities. This is the case for point of sale systems used in retail stores where the data entry sites consist of cash...
registers located on the sales floor, and the data processing facility is located elsewhere. Another example is an airline reservation system in which local terminals and processing facilities are one site in a network of many sites that may span coast to coast. For applications like these, a distributed organization has several advantages over a centralized system, such as enhanced processing speed and reliability; however, as we will see, decentralizing the control aspects of the system introduces some difficult problems.

• Open Problems. From an academic aspect, there are still some very interesting problems that need to be solved. Among these are the development of efficient management techniques for naming, and the control of concurrency; these specific problems are discussed in Chapters 4 and 5. Another problem, and a focus of this thesis, is the development of a network interconnection architecture that provides a consistent, high-level interface to users.

• Throughput Improvement. One way in which a distributed system organization can improve throughput is by allowing sites to make local copies of data objects that are frequently accessed. Depending on the characteristics of the interconnecting network, it is almost always the case that a local data access is much more efficient than an intersite access. Another way in which a distributed system organization can improve throughput is by taking advantage of the parallelism that is inherent in this type of organization. While fast processing of computation-bound problems is not a fundamental goal of most distributed systems, certain classes of problems can be computed more efficiently by partitioning the problem into small pieces that can be computed in parallel, or by specialized high-performance processors that are available on the network.

• Economy. Advancements in integrated circuit technology in the last decade have made the price/performance ratio for distributed systems very competitive to that of conventional systems. While distributed systems have been shown to
improve throughput for reasons discussed in an earlier section, the cost of implementing the control logic required to manage features like multiple copies of data, has been prohibitive. Fortunately, the current state-of-the-art of VLSI technology has made the manufacturing of custom logic components an economical way of overcoming difficult control problems. It is quite practical to use specialized VLSI components to, for example, assist time-critical algorithms by implementing them directly in hardware.

- **Reliability and Availability.** A distributed system architecture will enhance reliability in two ways. First, data objects can be replicated at many sites in the system, thus, unlike a primary or centralized system organization, single site failures may not affect other sites. Second, some systems (like the one proposed in this thesis) have a topology that provides for multiple paths between sites. This type of organization is especially resilient in that paths can be dynamically restructured if a communication link fails.

In summary, the primary motivating factor for continued research of distributed systems is the desire to build a distributed computer system that directly supports a class of problems in which data and resources must be geographically dispersed. Although this type of system has already been constructed, throughput and reliability are often sacrificed by overly complex solutions to the problems inherent in distributed systems. This thesis will outline a novel systems architecture in which some of these problems are mitigated by moving certain key control functions from a site to an intelligent network interface processor.

### 2.2 Definition and Classification of Distributed Systems

In this section, we will attempt to define the term *distributed system*. Unfortunately, the technical literature is full of such definitions that often contradict each other and that, in many cases, are so vague that one could not distinguish between a
distributed processing system and contemporary non-distributed system. The lack of a concise definition is owing to, in part, the realization that distributed processing systems are characterized by a combination of several logical and physical properties, some of which can be found in non-distributed systems. In the next two subsections, we will attempt to gain insight into the nature of distributed systems by looking at their characteristic properties and by demonstrating a classification method that will allow us to compare and contrast distributed systems with other types of computer architectures.

2.2.1 Characteristics of Distributed Systems

In this section, we develop a list of basic properties that a computer system should exhibit to be classified as a distributed system. These properties are discussed by Lampson, et al. [1981] and Enslow [1978] and are briefly reviewed below.

- **Physical Distribution.** The hardware system should be physically dispersed. While the components that comprise a site may be tightly-coupled, intersite connections are loosely-coupled and sites may be arbitrarily far apart. Our working definition for *site* is a grouping of logically or geographically related processors, memory, and data. The physical size of a site may vary from a multiple board set down to a single integrated chip. In this case, multiple sites could be implemented on a single printed circuit board thereby shortening the intersite communication delay. However, to meet this criteria for classification as a distributed system, it must be the case that these sites do not rely on their proximity to one another to function properly.

- **Communication.** Sites are interconnected by a communication network. The network is the only way in which sites can communicate; there is no shared memory or global tables. Using a communication network for all intersite communication introduces some problems that are not typically found in tightly-
coupled, bus-oriented systems. The most serious of these problems include: (1) low throughput because of communication channel bottlenecks and limited bandwidth, and (2) the need to handle the transmission errors that are an inherent problem with networks. Further, distributed system organizations that provide for multiple paths from one site to another cannot guarantee that messages will arrive in the order in which they were sent, nor can they rule out the possibility of messages becoming duplicated or lost in transit. These characteristics of networks tend to complicate the design of a synchronization algorithm and, in fact, are responsible for the failure of many previously proposed solutions.

- **Loose Coupling.** The concept of loosely-coupled processors is admittedly vague. For our purposes, the important point is that sites are geographically dispersed, and should be able to handle arbitrary delays in delivery of intersite messages. This constraint means that communicating sites should function properly regardless of the distance between sites, or the bandwidth of the interconnecting network.

- **Decentralized Control.** Each site in the system is autonomous in that all sites have an equal status and there is no master site that coordinates system-wide activity. During the process of enacting a transaction, sites may need to cooperate with each other, however, no two sites ever establish a master/slave relationship. The important implication is that no site in the system will ever have (or need) a consistent view of the system state.

- **System Transparency.** The location of dispersed resources in the system should be transparent at the process level. That is, processes should be able to reference resources by an established symbolic name, or by a local alias. Again, to preserve the autonomy of sites, management of the name space must not be accomplished with system-wide tables, nor should it be managed by on-site
software.

- **Heterogeneous Resources.** System components (i.e., data processors, memory processors, etc.) should not have to be homogeneous. One goal in constructing a distributed system is to simplify the management of specialized resources, such as high-speed numeric or string processors, in a way that allows dynamic allocation of a task to an appropriate processor.

- **Modular Design.** Distributed processing systems should be modular in design. It is important to be able to incrementally add, remove or upgrade resources without affecting other sites in the system.

These are the basic properties that most distributed systems exhibit. However, having one or more of these characteristics does not necessarily mean that a computer can be labeled as a distributed system. For example, consider a contemporary bus-oriented computer such as a Digital Equipment Corporation PDP-11 mini-computer [DEC 1978]. In a typical PDP-11 configuration, intelligent device controllers interface to the central processing unit (CPU) through a high-speed asynchronous bus, called the UNIBUS. The device controllers are intelligent in that they can perform complex operations, such as transferring data from a disk to memory, without the need to communicate with the CPU until the operation has completed. When communication is necessary, the device requesting the ability to communicate with another device is granted exclusive control of the UNIBUS by becoming the *bus master*, and other devices must yield control of the bus by becoming *bus slaves*. Although this configuration supports some degree of distributed processing, it does not meet the criteria for classification as a distributed processing system for at least two reasons: First, system components are tightly-coupled in that the maximum distance between components is dictated by bus timing constraints and is not arbitrary. Second, the master/slave relationship between components violates the notion that processors are of equal status and that they perform the task at hand by cooperating with each
Another example of a geographically distributed system that may not conform to the criteria for classification as a distributed system is a cluster of personal computer workstations sharing a centralized file server. Typically, in this type of organization, each microcomputer is treated as a stand-alone computer system that processes jobs that are totally disjoint from the other computers in the cluster. While workstations may share global read-only files (e.g., utility programs) stored at the file server, individual computers tend not to communicate or share data with each other. Thus, the system is distributed in a hardware sense, but does not exhibit the types of communication that we associate with distributed systems.

These examples show that the definition of the term *distributed system* is elusive at best. In the next section, we will attempt to refine the notion of distributed systems by reviewing a method of categorizing computer systems. Again, the goal is to develop an understanding about the characteristics of a distributed system so that we may gain insight into some of the problems inherent in decentralized control.

### 2.2.2 A Categorization of Computer Systems

Computer systems can be classified and compared by the number of concurrent instruction and data streams that it supports. For example, a contemporary microprocessor is classified as a SISD computer because it has a single instruction stream and a single data stream. An array processor contains many arithmetic processors that execute the same instruction on different data, so this type of organization would be an example of a SIMD computer. We can see that this classification scheme works well for certain types of architectures, but it does not accurately characterize non-von Neumann architectures. Both a data flow machine and a distributed system would be examples of MIMD organization, yet, data flow machines are typically tightly-coupled while distributed systems are loosely-coupled. The missing dimension
in this classification scheme is degree of decentralization for hardware, control, and data. Enslow [1978] classifies computer systems based on the degree of decentralization for these three characteristics. This is shown in Figure 2-1. The hardware decentralization axis characterizes systems based on the number of processors and the nature of the coupling between them. These vary from the simplest system that contains one CPU and one memory unit, to a distributed organization with many processors and memory units. The control decentralization axis characterizes systems in terms of
the general control strategy employed. These vary from a single, fixed point of control, to a distributed organization with multiple, fully cooperating control points. The database axis characterizes systems based on the organization of data. These vary from simple, centralized databases to distributed databases with replicated copies of data at all sites. The shaded region in Figure 2-1 represents a class of systems, such as the proposed system, that meet the criteria to be classified as a distributed system.

In this section, we have developed a list of the basic characteristics of distributed systems, and reviewed a classification method that allows us to compare and contrast distributed systems with other types of computer systems. We summarize this section by offering the following definition for a distributed system: A distributed system is a type of computer system organization in which resources are physically dispersed and are interconnected by a communication network of arbitrary length and speed, and questionable reliability, and there are multiple loci of control and processing activity. Further, data objects may be shared by replication at appropriate sites and the system provides for synchronization, consistency, and naming of these shared objects.

2.3 Philosophies on Replicated Data

In this section, we will discuss the impact and tradeoffs involved in supporting replicated data. Because supporting replicated data on conventional distributed systems has had such a negative impact on reliability and autonomy, there is no longer a favorable consensus in the computing community about replicated data. We will discuss both sides of the issue in this section.

As we have seen, the term replicated data refers to the ability to store copies of data at different sites in the system. The degree of redundancy varies from non-redundant (i.e. centralized database) to partially redundant to fully redundant. The problem to solve is ensuring the consistency of copies of data during a distributed
update transaction. There have been several proposed solutions to the multiple copy update problem. Unfortunately, none of these solutions exactly meets the design goals for the proposed system; therefore in Chapter 4, we develop our own variation of the two-phase protocol in combination with version numbers. One serious drawback to all the proposed solutions (including ours) is that these schemes will increase bus traffic by generating extra synchronization control messages. In the proposed system, this problem is lessened because the topology of the system allows communication between sites to be carried out in parallel. Another problem is that the algorithms, while conceptually easy to understand, are very detailed and tend to be difficult to implement. Furthermore, one needs to be careful not to introduce the possibility of deadlock into the system. The most serious drawback to most solutions is that they are not designed to be tolerant of communication failures. For example, most methods assume that transmitted control signals will successfully reach every intended destination; these algorithms do not provide for the possibility that not all sites will receive the message.

It would seem that there are tradeoffs between the simplicity of the system design and the ability to support consistent multiple copies of data. Because the primary reason for supporting replicated data is the desire to increase system throughput by moving data to the point of use, it would seem that the increased overhead required to support this feature is self-defeating.

In an actual implementation, the degree of improvement in throughput gained by decentralizing the control of data is dependent on the nature of the transactions that the system supports. Transactions can be characterized by the following properties:

- **Access Mode.** Accesses to data can be characterized as being read-mostly, write-mostly, or update. The first two categories are typical of the types of data accesses that are most prevalent in contemporary computer systems. For exam-
example, it is likely that the most frequently accessed programs on a timesharing system are the utility commands, such as the "directory list program." This is an example of read-mostly data because it is frequently accessed and seldom modified. Log files are examples of write-mostly data. Most operating systems keep accounting files that are updated when a user signs onto the system. The update usually involves appending a record onto the tail of the log file. Because the accounting file is usually processed only once a day, most of the accesses are write-mostly. An inventory application provides a good example of the update access mode. In this type of application, most of the transactions involve an update to an existing record. For example, the shipping department will decrease the quantity on hand of a stock item as it is shipped, and the receiving department will increase the quantity as the item is purchased. Applications in which transactions are read-mostly will benefit from making local copies of frequently accessed data because there is no network delay time involved in a local access. Write-mostly applications will not benefit because data objects are generated locally and there is no need or opportunity to share the data with other sites. Update transactions will benefit from multiple copies only if there is little interference.

- Interference. Interference refers to a scenario in which two sites attempt to update a replicated object at the same time. At the transaction level, two transactions interfere with each other when the input values (i.e. the readset) of one transaction overlap the output values (i.e. the writeset) of the other transaction. More precisely, interference occurs when one of the two following conditions are true:

\[
(\text{ReadSet}_1 \cup \text{WriteSet}_2) \cap \text{WriteSet}_2 \neq \emptyset \\
\text{WriteSet}_1 \cap (\text{ReadSet}_2 \cup \text{WriteSet}_2) \neq \emptyset
\]

When interference occurs, one site will typically terminate the transaction and try again later. In distributed systems where update transactions are prevalent,
throughput will decrease if interference is common. The degree of interference is influenced by the synchronization method chosen. Locking schemes that gain exclusive control of data tend to increase interference, while schemes based on a time ordering (e.g., time stamps or version numbers) tend to lessen interference by allowing reads to execute concurrently.

- **Volume.** The volume of transactions will also have an effect on the performance of the system. If the system reaches the point where the communication network becomes a bottleneck, it may be appropriate to further partition the system into smaller pieces, presuming that the data exhibits some degree of locality. As an example, consider an airline reservation system. At first thought, it would seem appropriate to use a fully-connected network because every site will eventually need to communicate with every other site. However, on closer inspection, it becomes apparent that there are zones of activity in which most of the transactions are local. It may be the case, for instance, that most of the transactions at Kennedy Airport in New York involve commuter flights to surrounding airports. By encapsulating this geographic area as a separate zone in the system organization, we can limit message traffic to the portions of the system that are involved with the transaction, thus eliminating much of the traffic on slower speed long-haul networks. In this example, message traffic that involved only New York sites would not need to be broadcast to all West Coast sites. This is precisely the approach taken in the proposed system, where levels of network support are tree-structured. This topology has the property of being able to create localities or scopes in which local message traffic is restricted to relevant portions of the network.

- **Geography.** The geography of the system has an impact on performance in that it may determine the type of communication network that must be used. For example, in cases where the system needs to be distributed over great distances
and satellite access in not possible, then slow speed telephone communications must be used. When sites are close to each other, a high-speed local area network may be feasible.

Clark and Svobodova [1980] argue that distributed systems should not support replicated data because the coordination of concurrent updates by other sites sacrifices the autonomy of a site in that it occasionally does not have control over a local copy of replicated data. Clark proposes that every data object have only one home site, namely, the site serving the person that is responsible for the information. Thus, in Clark’s system, sites can modify only locally accessible data. Sites are allowed to copy remote data, however, these copies will not be updated by the owner of the data. It is likely then, that these copies will soon become outdated and inconsistent. The authors correctly note that successful handling of the multiple copy update problem complicates the design of the system and may actually decrease performance owing to, in part, the complicated protocols of the control messages.

Another problem in allowing multiple copies is deciding what action to take when the system becomes partitioned because of a site or communication link failure. Should processing continue on both sides of a partitioned system, a very difficult problem arises, namely, how to reconcile differences in updated copies of an object. The post-partition consistency problem is currently an active area of research and is discussed in Section 2.5.

2.4 The Consistency Problem

As we have seen, the primary benefit of employing a distributed system architecture is enhanced resiliency and throughput that results from replication of data. These benefits extract a price by complicating the design of the system. The major problem to be solved is ensuring that the copies of replicated data remain consistent without sacrificing the robustness of the system. This problem has been thoroughly
researched in the literature [Ellis 1977, Reed and Kanodia 1977, Reed 1978, Thomas 1978, Chu and Gardarin 1979, Brereton 1983] and results will be reviewed in Section 4.2. Unfortunately, most of the proposed solutions are very complex and some will not function correctly in a production environment because they require an error-free message transport medium. Therefore, we propose a modified version of the algorithm proposed by Reed [1983] that is given in section Section 4.3.

The concept of consistency of replicated data has two distinct components, namely mutual consistency and internal consistency. Internal consistency is a stronger condition than mutual consistency, however, it is not necessarily the case that preserving internal consistency will also preserve mutual consistency; each must be addressed separately, but a complete solution to the consistency problem must account for both. These are discussed in the following two subsections.

2.4.1 Mutual Consistency

In order for replicated copies of data to be mutually consistent, all copies must represent the same value. The accepted definition of consistency [Thomas 1978] is somewhat weaker in that it guarantees only that copies will converge to the same value if all update activity were to cease. We prefer a stronger definition, namely that copies of replicated data objects are mutually consistent only if every accessible copy represents the same value. The distinction here is that we are acknowledging that updates are not instantaneous and at some point in time the value of two copies may differ because one has been updated and the other has not yet been updated. However, as we will see, no harm results from this event because the copies of data awaiting the update are obsolete and cannot successfully be used in transactions.

Mutual inconsistencies arise from the improper control of concurrent updates. The following example demonstrates what can happen when updates are not properly synchronized. Assume that there are two sites in the system and that each site can
query and update a local copy of common information that we will call "quantity on hand" (QoH). Initially we have the state shown at time 1:

<table>
<thead>
<tr>
<th>Time</th>
<th>Site 1</th>
<th>Site 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100 units</td>
<td>100 units</td>
</tr>
<tr>
<td>2</td>
<td>50 units</td>
<td>50 units</td>
</tr>
<tr>
<td>3</td>
<td>25 units</td>
<td>40 units</td>
</tr>
</tbody>
</table>

Either site may enact a transaction that first updates the local copy of QoH and then broadcasts the new value to the other site. Suppose that at time 1, site 1 reduces QoH by 50 units and broadcasts the new value of 50 units to site 2. At time 2, both sites are still mutually consistent. The problem occurs when both sites enact a transaction at the same time. At time 2, suppose that site 1 reduces QoH by 10 units and transmits the new value of 40 units to site 2. At the same time, site 2 reduces QoH by 25 units and transmits the new value of 25 units to site 1. The final result at time 3 will leave site 1 and site 2 mutually inconsistent and incorrect. In this simple example, we could have achieved the desired results by applying a synchronization algorithm that properly sequences update transactions. These algorithms are discussed in Section 4.2. However, it is interesting to note that in certain cases synchronization techniques are completely unnecessary. Two such cases occur when: (1) the update transaction order is not important, or (2) when updates consist only of storing values into the database. These are explained below.

In some applications, the ordering of update transactions is not important. Our prior example with the replicated QoH data can be modified so that the data remains mutually consistent without the use of synchronization mechanisms. This can be accomplished by transmitting the net effect of the update rather than the updated value. For example, at time 2 each site has 50 units QoH. At time 3, site 1 decreases its copy of QoH by 10 units and transmits the net change in QoH (i.e. -10) to site 2. At the same time, site 2 decreases its copy of QoH by 25 units and transmits the net change in QoH (i.e. -25) to site 1. At time 4, each site will again be consistent and
QoH will be 15. Unfortunately, this method is limited to problem domains in which net effect has meaning. This scheme would not work in problems where heterogeneous data structures (i.e. records) are replicated.

Another case in which synchronization mechanisms are not needed occurs when updates consist only of storing values into the database. In this case, transactions are tagged with a time stamp and after a transaction with time T has been performed, all subsequent requests with a time stamp less than T are ignored.

2.4.2 Internal Consistency

Whereas mutual consistency is concerned with the consistency of the copies of a single data object, internal consistency is concerned with the relationships of several objects and transactions that update more than one object at a time. Copies of replicated data objects are said to be internally consistent if they obey certain predicate conditions that have been established. These predicate conditions are developed by the database manager and become policy when the database is created. To a certain degree, predicates define the semantics of the database. While the update transaction mechanism should not need to understand these semantics, it can preserve these relationships by obeying the specified predicate expressions. As an example consider the following code segments:

\[
\begin{align*}
T_1: & \quad X &= X + 1 \\
& \quad Y &= X + Y \\
T_2: & \quad X &= X + 1 \\
& \quad Y &= X + Y
\end{align*}
\]

In this example, X and Y are replicated data objects (although they do not need to be replicated for this example) and T_1 and T_2 are tasks that execute their respective code segments. To examine the behavior of these transactions, we need to introduce some notation. We will let R_N(Z) denote a read operation by task N on variable Z, and W_N(Z) will denote the corresponding write operation. Thus, the read and write
sequences for \( T_1 \) and \( T_2 \) are:

\[
\begin{align*}
T_1: & \quad \text{R}_1(X) \ A(X) \ R_1(X) \ R_1(Y) \ W_1(Y) \\
T_2: & \quad \text{R}_2(X) \ A(X) \ R_2(X) \ R_2(Y) \ W_2(Y)
\end{align*}
\]

Initially, we let \( X=1 \) and \( Y=2 \). If we execute \( T_1 \) and \( T_2 \) sequentially so that \( T_2 \) follows \( T_1 \), the final values for \( X \) and \( Y \) will be \( X=3 \) and \( Y=7 \). Now, if we allow \( T_1 \) and \( T_2 \) to execute concurrently, we will get one of many interleaved execution sequences, such as:

\[
\begin{align*}
\text{R}_1(X) \ R_2(X) \ A_1(X) \ A_2(X) \ R_1(Y) \ R_2(Y) \ A_1(Y) \ A_2(Y)
\end{align*}
\]

If we evaluate this execution sequence by hand (Figure 2-2), we will arrive at an incorrect answer, namely \( X=2 \) and \( Y=6 \). The problem is caused by the interleaved execution sequence that allows tasks to read shared values while they are being updated. For example, in Figure 2-2, task \( T_1 \) begins its update of \( X \) at step 1 and completes the update at step 3. Note that \( T_2 \) reads the value of \( X \) at step 2, computes a new \( X \) based on that value, and writes the erroneous value in step 4. Some mechanism must be devised that either (1) rejects the write at step 4 because the value of \( X \) is outdated, or (2) suspends the read at step 2 until the new value for \( X \) is written at

\[
\begin{align*}
1. & \quad \text{R}_1(X): \quad \text{fetch } X: \quad X_1 = 1 \\
2. & \quad \text{R}_2(X): \quad \text{fetch } X: \quad X_2 = 1 \\
3. & \quad \text{W}_1(X): \quad \text{write } X: \quad X_1 = X_1 + 1 = 2 \\
4. & \quad \text{W}_2(X): \quad \text{write } X: \quad X_2 = X_2 + 1 = 2 \\
5. & \quad \text{R}_1(X): \quad \text{read } X: \quad X_1 = 2 \\
6. & \quad \text{R}_1(Y): \quad \text{read } Y: \quad Y_1 = 2 \\
7. & \quad \text{R}_2(X): \quad \text{read } X: \quad X_2 = 2 \\
8. & \quad \text{W}_1(Y): \quad \text{write } Y: \quad Y_1 = X_1 + Y_1 = 4 \\
9. & \quad \text{R}_2(Y): \quad \text{read } Y: \quad Y_2 = 4 \\
10. & \quad \text{W}_2(Y): \quad \text{write } Y: \quad Y_2 = Y_2 + X_2 = 6
\end{align*}
\]

Figure 2-2. Evaluation sequence for interfering transactions
step 3.

There are two basic methods for handling these types of synchronization problems, namely locks and time stamps. Locking is a form of mutual exclusion that can be used to prevent access to objects while they are being updated. Time stamps are relative time values or version numbers that are used to order transactions. Version numbers could be applied to the above example in the following way: The variable X would be tagged with version \( V_1 \) so that the reads of X in steps 1 and 2 would both read that version. The write of X in step 3 would increment the version on X to \( V_2 \) Now, the write at step 4 would be rejected because its new value for X is based on an obsolete version. Synchronization methods based on locks and time stamps are reviewed in Section 4.2.

2.4.3 Atomic Operations

The concept of an atomic operation is extremely important in systems that use decentralized control strategies. An atomic operation is a collection of computations that are viewed as a single, indivisible operation. We have seen that an update operation for replicated objects must be implemented as an atomic operation, otherwise it will be possible for copies of an object to become inconsistent during the update. An atomic operation either completely succeeds or it completely fails. That is, if it succeeds then we are assured that all composite computations have succeeded and that all accessible data objects have been correctly updated. If it fails, then all composite computations are voided and no object is modified. In either event, only the final result is visible to the external world; no intermediate computations are ever visible to other processes.

Atomic updates are described by Liskov and Zilles [1974] and elsewhere and are usually implemented with a two-step locking protocol. The first step locks all copies of the replicated object in a way that guarantees exclusive access. If any of the copies
cannot be locked (e.g., another update is in progress) then the operation is said to *abort* and all the locks that we have asserted are released. If the lock step succeeds, then each locked copy is updated with the new data value. Again, if any part of the update fails, then the entire update is aborted and all copies are restored to their original value and the locks are released. On receipt of an acknowledgment from each updated copy, the update operation is said to *commit*, and the new value is made accessible.

### 2.4.4 Serialization

In distributed systems that support replicated data, it is likely that update transactions involving a common set of data will occasionally overlap in time. To preserve the internal consistency of the data, it is sufficient that these transactions be *serialized* [Eswaran, et al. 1976]. This property requires that concurrent executions of transactions produces the same results as executing the transactions serially (i.e., one at a time). To be serializable, there must exist at least one such equivalent ordering of serial transactions. For example, consider the following transactions:

\[
\begin{align*}
T_1 &: \quad R_1(X) \quad W_1(Y) \\
T_2 &: \quad R_2(Z) \quad W_2(X) \\
T_3 &: \quad R_3(X) \quad W_3(Z)
\end{align*}
\]

Two possible execution sequences for these transactions are:

\[
\begin{align*}
E_1 &: \quad R_1(X) \quad R_2(Z) \quad W_1(Y) \quad R_3(X) \quad W_2(X) \quad W_3(Z) \\
E_2 &: \quad R_1(X) \quad R_2(Z) \quad R_3(X) \quad W_1(Y) \quad W_2(X) \quad W_3(Z)
\end{align*}
\]

Execution sequence $E_2$ is serial and represents $T_1 < T_2 < T_3$, where "$T_i < T_j$" means that $T_i$ is executed before $T_j$. Execution sequence $E_1$ is not serial, although, the resulting state is the same as that produced by $E_2$. Therefore, execution sequence $E_1$ can be serialized and represented by $E_2$. Some sequences cannot be serialized, as shown by
the following example.

\[ E_3: \text{R}_2(Z) \text{ R}_3(X) \text{ R}_1(X) \text{ W}_2(X) \text{ W}_3(Z) \text{ W}_1(Y) \]

Since \( R_2 \) precedes \( W_3 \), then \( T_2 \) must precede \( T_3 \) in any equivalent serialization. Also, since \( W_3 \) precedes \( W_1 \), then \( T_3 \) precedes \( T_1 \) in any equivalent serialization. However, because \( R_1 \) precedes \( W_2 \), then \( T_1 \) must precede \( T_2 \). Thus, we have a cyclic relationship between \( T_3 \) and \( T_1 \). Therefore, execution sequence \( E_3 \) is not serializable.

Fortunately, we need not be concerned with finding serializable execution sequences because the two common synchronization techniques, locking and time stamps, always generate serializable sequences. Locking serializes transactions by forcing them to execute one at a time. Time stamps or version numbers serialize transactions by providing a partial ordering of transactions with respect to time.

The property of serializability is fundamental to the concept of consistency. We have seen that if a synchronization technique produces serializable execution sequences, then these sequences are equivalent to another sequence in which transactions are executed one at a time. If every transaction preserves consistency, then by induction, so must a serialized execution sequence.

### 2.4.5 Deadlock

Deadlock describes a system state in which two or more transactions are blocked awaiting the allocation of a resource currently held by the other transaction. For example, consider two tasks, \( T_1 \) and \( T_2 \), which both use resources \( R_1 \) and \( R_2 \). Task \( T_1 \) requests \( R_1 \) first, then \( R_2 \). Task \( T_2 \) requests \( R_2 \) first, then \( R_1 \). At time 1, \( T_1 \) will request and receive \( R_1 \) and \( T_2 \) will request and receive \( R_2 \). At time 2, however, neither task will be able to proceed because the other task controls the needed resource. This event can arise whenever (1) tasks are allowed to lock (i.e. gain exclusive access) a resource, and (2) tasks are allowed to wait for locked resources.
The deadlock problem has been extensively researched over the past three decades, however, mostly in the context of a centralized system. The physical distance (manifested as communication delays) between sites in a distributed system has rendered accepted deadlock algorithms for centralized systems inefficient for distributed systems. Therefore, the issue of deadlock has again become an active area of research. There are two approaches to solving the deadlock problem, namely, deadlock prevention, and deadlock detection and recovery.

In systems that use deadlock detection and recovery, transactions are allowed to wait for locked data in an uncontrolled manner. Periodically, deadlock is checked for by constructing a waits-for graph. A waits-for graph contains nodes that represent the transactions currently being processed in the system, and arcs between these nodes if one transaction is waiting for a resource that the other transaction is holding. A cycle in the graph means that two transactions are deadlocked. This can be fixed by canceling one of the transactions and restarting it later. Unless all locking information is contained in one site (as in a centralized database), this method of detecting deadlocks is not practical owing to, in part, the volume of message traffic that will be generated.

In systems that use deadlock prevention algorithms, transactions are allowed to wait for resources in a controlled manner. There are many algorithms that prevent deadlock, a few of them are reviewed by Li [1982]. One example of a deadlock prevention algorithm is the Prioritized Transactions Algorithm [Rosenkrantz, et al. 1978]. In this scheme, deadlock is avoided by assigning a priority to each transaction, and requiring that when a conflict occurs, only higher priority transactions can wait for lower priority transactions. By enforcing this policy, each edge in the waits-for graph is in priority order and cycles are not possible since a lower priority transaction cannot wait for a higher priority transaction. Thus, there can be no cycles, and the system is deadlock free.
Deadlock must be considered in any system that uses locks. In the proposed system, we do not use a locking protocol, therefore deadlock is not possible.

2.4.6 Characteristics of a Good Solution

As a summary to Section 2.4, consider the characteristics of a good solution to the multiple copy problem:

- **Consistency.** All replicated copies of an object must remain mutually and internally consistent.

- **Deadlock Free.** A good solution should not introduce the possibility of deadlock into a deadlock free system. If this is not possible, the solution should provide a means of detecting deadlock and automatically recovering by terminating the offending request.

- **No Critical Blocking.** The solution must guarantee that a transaction will always complete (i.e. succeed or fail). It should not be possible for a transaction to become blocked indefinitely.

- **System Partitioning.** The solution must provide for the possibility that a portion of the system may fail, thus partitioning the system into several subsystems. Whether to allow subsystems to continue to operate in this mode of failure becomes an administrative as well as a technical issue. Our intent is to allow the system to continue to operate even if one or more of its sites are down.

- **Network Independence.** The solution must be independent of the transport medium that is used to transmit messages. This requirement has two subtle implications: (1) The protocol used must provide for variable time delays. Protocols that use timeout mechanisms as a means of protecting against lost messages must be carefully applied. (2) We cannot rely on the atomicity of messages. That is, if we transmit a message to replicated objects to update their values in some way, we cannot be guaranteed that all the objects received the message. It
may be the case that the first few objects correctly received the message, but somewhere along the network, the message became corrupted and the other objects did not receive it.

2.5 Partitioning and Site Failures

One of the more difficult problems to be solved in the area of distributed systems is the handling of site and communication link failures [Rolin and Boudenant 1982]. While one site is off-line, the remainder of the sites will continue to process transactions. After a short time, it is likely that copies of data stored at the off-line site will become inconsistent. The problem, then, is how to go about updating the local copies when the site is restarted. There have been at least two proposed solutions to this problem. One method, called *persistent communication*, [Rothnie and Goodman 1977] guarantees that the system will eventually deliver messages to the off-line site. Thus, the site may recover simply by processing these stored messages using the same mechanisms that the site would use when it is on-line. Another type of site failure recovery scheme uses a global log of updates to recreate the transactions that the site lost while it was off-line.

In the proposed system, the gateway processors are physically separated from the site. Therefore, site crashes do not affect the gateway's ability to process network transactions even though the attached site is down. Failures in the system must be almost catastrophic before a special method of recovery will need to be applied. We do, however, provide three methods for recovery of data:

1. The simplest solution is to do nothing; transactions that involve outdated copies of data will always fail (see Section 4.3) thus, forcing the gateway to obtain the current copy of the data.

2. The second method is to delete all local copies of replicated data and force the gateway to obtain current copies as needed. This scheme is not unlike filling
an empty cache memory. There will be an initial increase in message traffic as remote copies of the data objects are recopied to the site, however, once the data objects are restored, the message traffic will return to its volume before the crash.

(3) The third way of restarting a gateway is to recreate the data from the log file containing transactions at the gateway. The ability to log transactions is an optional gateway feature that should probably be used for gateways that process critical information.

A more difficult problem occurs when the system becomes partitioned because of a communication link failure. When this happens, there are usually two or more parts of the system that are disjoint, and each severed part may continue to process transactions. The problem then is how to rejoin the parts in a way that makes all replicated data consistent again. It is widely acknowledged in the literature that this is still an open problem for which a solution may not exist. In the proposed system, the regular structure of the topology makes it possible to detect partitioning, however, reconciliation of the data after this type of crash remains a difficult problem. A good administrative policy would be to either shut down the entire system until it can be repaired, or to assign the responsibility for maintenance of certain data to a particular portion of the system.

2.6 Related Research

There are hundreds of papers, dissertations, and books on the topic of distributed systems. The bibliography for this thesis would contain a list of at least one hundred publications if all reference material were cited. Furthermore, the topic of distributed systems is so broad that to study it correctly, one must also delve into related areas like operating systems, programming language support for concurrency, database applications, VLSI topics, and so on. We would like to acknowledge some ongoing
efforts that have influenced the work presented in this thesis.

The problems of naming and synchronization in a distributed system have received much attention in the literature [Reed 1978, Livesey 1979, Lampson, et al. 1981]. Starting with some ideas proposed by Thomas [1978] and others, Reed constructed a method that preserves consistency without using locks. Our algorithm for consistency presented in Chapter 4 is based on this work.

An excellent review of synchronization methods, serializability, and consistency issues is given by Teng-amnuay [Teng-amnuay 1983] and others [Brereton 1983, Chu and Gardarin 1979].

An ongoing discussion about the definition and nature of distributed systems can be found in the literature [Enslow 1978, Svobodova, et al. 1979, Clark and Svobodova 1980].

There is a wealth of material about atomic actions [Weihl and Liskov 1983, Reed 1983]. We were also influenced by the mechanisms in the language CLU [Liskov, et al. 1979] and the operating system kernel Hydra [Wulf, et al. 1975].
3. REVIEW OF BACKGROUND MATERIAL ON COMMUNICATION NETWORKS

3.1 Introduction

In this chapter, we review relevant background material on communication networks and protocols, and provide definitions for the many terms and acronyms that seem unavoidable when discussing networks. We will limit the scope of the material to include only topics that are directly applicable to the proposed intelligent message transport system.

We have already stated that the intelligent message transport system is an example of a network architecture that provides a means for reliable communication using heterogeneous networks. There has been much research in areas related to internetwork architectures and the system proposed in this thesis will build on this work. One of the most successful examples of an internetwork architecture is the Xerox Pup\(^1\) network [Boggs, et al. 1980]. We mention Pup here because it is one of the few working examples of an internet architecture, and it seems that many of the communication concerns that are applicable to our system are based on studies of Pup. The Pup internetwork provides an end-to-end media-independent connection between user processes in an environment where communication and computing facilities are heterogeneous. As of 1980, the Pup internetwork had grown to include over 1000 host systems attached to 25 networks of 5 different types using 20 internetwork gateways. Our system is similar to the Pup network in that it also uses gateway processors to join different types of networks and to convert between the various network-specific protocols. The proposed system differs from Pup and other internetwork architectures in several basic ways that will be discussed in Section 3.6.

\(^1\)The name Pup originally referred to the internetwork datagram (the PARC Universal Packet), but has been expanded in usage to denote a style of internetwork communication.
3.2 The OSI Reference Model

In 1977, the International Organization for Standardization (ISO) recognized the need to create a standard for interconnection of heterogeneous networks, and formed a subcommittee (SC16) to develop the Open Systems Interconnection (OSI) model. This model is described in detail by Zimmermann [1980]. We will briefly review the model in this section and compare the level at which our naming and synchronization methods are implemented with that of the OSI model.

The OSI reference model is based on a structure that is logically composed of layers, not unlike the modular approach in structured programming. Each layer provides a well-defined service and presents a standard interface to the next higher layer. Thus, the functions provided by layer N build on the functions provided by layer N-1; furthermore, users of the services at layer N cannot gain access to the services at layer N-1. The actual method of implementation for each layer is not specified by the OSI reference model. It is only necessary that each layer provides the specified interface to the next higher layer. Boggs, et al. [1980] lists some of the criteria that the ISO subcommittee considered when defining the OSI reference model. The result was a communication model composed of seven layers. These are listed below, in order from the highest (closest to the user) layer to the lowest layer.

1. The Application Layer. This is the highest layer in the model. Protocols at this layer are user-defined and provide direct support for application programs by managing the initiation and termination of transactions.

2. The Presentation Layer. The Presentation Layer provides a set of services that may be selected by the Application Layer to enable it to determine the meaning of the data exchanged. For example, the Presentation Layer might determine how to package data from the network in a way that allows the Application Layer to recognize it as some particular data type.
3. **The Session Layer.** The purpose of the Session Layer is to coordinate the interactions between cooperating presentation entities. To accomplish this, the Session Layer provides two categories of services, namely, the session administration service, and the session dialogue service. The session administration manages the binding and unbinding of two presentation entities. The session dialogue service manages aspects of the actual data exchange, such as delimiting data and synchronizing the transfer.

4. **The Transport Layer.** The Transport Layer masks all network details and provides a universal message transport service to the Session Layer. Only the source and destination sites are considered at this level. The details and concerns of routing messages through intermediate sites are handled by the Network Layer. Any decisions affecting the cost and reliability of a session are made at this level.

5. **The Network Layer.** The Network Layer provides the means to exchange messages between two transport entities over a network connection. Decisions about routing are made at this level. One of the more common protocols at this layer is level 3 of the X.25 interface [Davies, et al. 1979].

6. **The Data Link Layer.** The purpose of the Data Link Layer is to provide the means of establishing, maintaining, and releasing data links between two networks. This layer hides the network-specific details about establishing a connection with another site on the network, and provides for error detection and recovery. The ISO has defined standard protocols to be used at this layer, including *High-Level Data Link Control (HDLC)* [Davies, et al. 1979] and *Synchronous Data Link Control (SDLC)* [Green 1980].

7. **The Physical Layer.** The Physical Layer provides the mechanical, electrical, and functional capabilities to establish, maintain, and release physical connections between data link entities. The ISO has defined standard protocols to be
used at this layer including X.21, V.24\(^2\), and V.35 [Bertine 1980].

In the proposed system, the Application, Presentation, and Session Layers are part of the software local to a site. The remaining layers are part of the gateway software and hardware. A major difference between the proposed system and other distributed systems is that the naming and synchronization control logic is implemented at the Transport Layer rather than the Application Layer. This is in contrast to distributed systems that use schemes like predicate locks [Eswaran, et al. 1976] or structural locks [Lee and Yeh 1979] where the synchronization strategy is separated from the communication functions, and thus, must be implemented in the Application Layer. It is our belief that synchronization and naming belong in the lower protocols of the OSI model because it is appropriate at that level to deal with the detailed control signals that must be generated to coordinate an update transaction. Also, moving those functions from the site to the gateway ensures that each application process will be provided with a standard, consistent interface regardless of the type of application program or interconnecting network.

### 3.3 Standard Gateway Types

Gateways are devices that connect two or more possibly different networks in a way that provides a channel for messages to cross from one network to another. There are two recognized types of gateways, namely, media-conversion gateways, and protocol-conversion gateways.

The media-conversion gateways are the simplest in that they just pass messages from one network to another network. Incoming messages are "unwrapped" (i.e. the network packaging is removed), the next destination is computed, and the message is wrapped in the outgoing network's protocol packaging. The content of the message is not analyzed, it is simply forwarded to the next gateway.

\(^2\)The V.24 protocol is known as RS232C in the United States.
The protocol-conversion gateway is more complex in that it attempts to translate the protocol of the incoming network to the protocol of the outgoing network in a way that preserves the semantics of the message. The difference between these gateways is one of degree: the media-conversion gateway bridges the gap between different physical and link level protocols, while the protocol-conversion gateway bridges the gap between differing network and higher-level protocols. As noted by Postel [1980], the success in translating protocols seems to be inversely related to the level of the protocol. Low level protocols can be translated without much difficulty, but translation of higher level protocols is difficult because of the inability of the gateway to access status information used by the application process when it generated the message.

Other than providing for message routing, the media-conversion and protocol-conversion gateways provide no additional services to the system. That is, if it were physically possible to connect all sites by the same network, both types of gateways could be eliminated without a loss of services. This is the fundamental difference between the standard gateways and the gateway proposed in this thesis. We propose a gateway that plays an active role in supporting the functions of the system as a whole. In addition to the traditional functions of message routing, the proposed gateway will implement the basic services needed to support naming of distributed shared data, and synchronization for concurrent updates to replicated data. As in the media-conversion gateway, our gateway will simply unwrap incoming data and repackage it in a format suitable for transmission onto the outgoing network. It will not be necessary to perform high-level protocol conversions.

---

This is analogous to the well-known problem of symbolically converting a program written in one language to another language in that constructs that do not map directly into the target language must be analyzed to determine their semantics.
3.4 Virtual Circuits and Datagrams

Communication involves a pairwise interaction between two parties. These interactions can be classified as being either a virtual circuit type of communication, or a datagram. The datagram is the simplest type of communication. It is transaction-oriented and consists of a single message to be transmitted from one site to another. No reply is required from the receiving process, nor is an acknowledge control message sent back to the transmitting site. Thus, the detection and recovery of lost messages must be provided by an Application Layer protocol using some type of timeout mechanism.

The virtual circuit establishes a communication path between sites that can be used repeatedly until the entire message has been transmitted. It is helpful to think of virtual circuits as being session-oriented in that once the channel has been established, data may be streamed from the source site to the destination site without the need to reopen the channel. When the transaction is complete, the channel is deallocated. After establishing the virtual circuit, messages may refer to the circuit with a shortened address or circuit identifier, rather than an actual network address. The telephone system is an example of a virtual circuit. The path is established once and maintained until the caller hangs up. If telephone systems were designed to use datagrams, the path would need to be re-established for every word or sentence spoken.

The communication characteristics of the proposed system do not exactly map to either a virtual circuit or a datagram, instead, they exhibit a combination of both modes of communication. Recall that the primary application of the system is to support a distributed database, and that intersite communications will be transaction-oriented. Processes at sites will request a copy of some data object and then access the

---

*Datagrams may may also be sent to several sites if multicast (broadcast) communication is supported.*
object locally as if it were the only copy in the system. For example, when a process updates a replicated data object, the process needs only to present the update request to the gateway. The gateway then generates the appropriate synchronization control signals to manage the update. In one respect, this mode of communication is similar to datagrams in that user processes present single transactions to the gateway, rather than streams of data. It differs from datagrams in that the message transport system provides reliable message delivery with a positive acknowledgement signal, similar to a virtual circuit.

3.5 Relevant Communication Issues

In this section, we will review some of the design principles and issues for inter-network architectures. These are described in detail in the literature [Boggs, et al. 1980, Postel 1980].

- **Congestion Control.** Congestion control is sometimes confused with flow control, but actually attacks a different problem in packet-switched networks. Congestion control refers to a network-wide mechanism that attempts to balance the network message load so that individual gateways do not become saturated. In the proposed system, we use an end-to-end acknowledge protocol (Section 6.3.3) that enables the gateway to discard incoming messages after notifying the source gateway of this action. The source gateway will then wait for some period of time before retransmitting the message.

- **Flow control.** Flow control refers to a synchronization mechanism that regulates the transmission rate between a source and destination pair so that the source does not produce messages faster than the receiver can process them. Our end-to-end acknowledge protocol synchronizes the transfer rates between gateways by forcing the source to periodically retransmit the message until
the receiver is ready to accept it.

- **Packet Fragmentation.** The maximum size for a data packet is determined by the type of communication network that will transport the packet. Occasionally, a data object will be too large to send in one packet, so it must be fragmented into many packets on the source end and reconstructed by the receiver. Because individual sites are shielded from network realities like this, it is the responsibility of the gateway processors to handle the repackaging of the data object. In the proposed system, this turns out to be quite easily accomplished because the gateway processors have access to a large scratch-pad memory that can be used to collect the message fragments until the last one arrives.

- **Privacy and Security.** The present design of the intelligent message transport system does not contain provision for the security or authenticity of messages. The decision to ignore this type of issue was based in the desire to keep the design simple. Users of this system must realize that some of the interconnecting networks may be of the type that operates by broadcasting messages (e.g., the Ethernet) to all sites, thus introducing the potential for disclosure of the message contents. The final responsibility for encrypting messages rests with the user processes in the Application Layer.

- **Broadcast Packets.** The intelligent message transport system is an example of a *store and forward* network. This means that messages are propagated from gateway to gateway along the path from source site to destination site, in contrast to systems where messages are sent to every site and only the site with the matching address will process the message. Broadcast capability in store and forward networks is difficult to achieve. As noted by Boggs, et al. [1980], no existing store-and-forward network implements broadcast capability using any method except the brute-force transmission of the message to every node.
In the proposed system, broadcast is directly supported by the combination of the tree-structured topology and the intelligent gateway. Because there are no global resource tables, the responsibility for "locating" all on-line sites belongs to the gateways. Incoming broadcast messages are simply replicated and transmitted on all outgoing network paths (we call this technique branch and send in Chapter 6). The address field has a special code that designates that the message should be processed and forwarded by any site that receives it.

- **Detecting Path Cycles.** In some network topologies, failure of a site to properly process incoming messages may cause the messages to circulate throughout the network indefinitely. This problem becomes more difficult to solve if the network also allows cycles in its communication paths. One solution is to have a hop counter field in the message that is incremented each time the message passes through a gateway. When the hop counter becomes large, the message can be safely discarded. This is not a problem in the proposed system because it was specifically designed to support cycles in the communication path. As a message passes through a gateway, the gateway identification number is added to the head of the source gateway list field in the message. This enables the destination site to determine the exact path that the message traveled through the network. If a message passes through the same gateway a second time, the message is "wandering" and can be safely purged.

- **Reliable Transport.** As discussed earlier, the intelligent message transport system provides a reliable transport medium by using an end-to-end acknowledge protocol. Some networks already provide this type of reliability, but other networks based on probabilistic transmission (e.g., the Ethernet) cannot guarantee delivery. Even in networks that are designed to deliver correctly sequenced error-free packets, occasional anomalies may result from hardware or software failures. Our end-to-end protocol can detect these types of errors
and recover by requesting retransmission of the data. Of course, the system’s ability to detect errors is predicated on the belief that gateways are reliable, but because gateway processors are autonomous, it would take the instantaneous failure of most of the gateways in the system to create a scenario where some degree of error recovery could not be achieved.

3.6 Summary

In the previous section, we reviewed the important communication issues that are relevant to the proposed system, and briefly discussed the differences between our solutions to these problems and the solutions implemented by the Pup network. These differences stem from two factors:

1. The realization that our target applications are primarily database oriented, and thus, we need to support transaction-oriented processing efficiently. This is in contrast to the Pup internetwork that provides general process-to-process communication capabilities. This is not to say that we do not support generalized communication, however, it occurs infrequently and therefore does not need to be supported efficiently. This realization greatly simplifies the design of the system.

2. We consider the management of internetwork naming and synchronization of transactions involving replicated data to be functions that belong in the Transport Layer of the communication system. This implies that these functions must be implemented by the gateway processors rather than by protocols in the Application Layer. The decision to add these capabilities to the gateway turned out not to complicate their design, and even had beneficial system-wide side effects, such as alleviating the need for a global name server.
4. SYNCHRONIZATION AND CONSISTENCY OF DISTRIBUTED DATA

4.1 Introduction

In this chapter, we investigate various proposed algorithms that claim to maintain the consistency of replicated data. There has been much research in this area, and there are many proposed methods for managing concurrent updates. We will review the basic characteristics of some of these algorithms and provide a basis for comparing their expected performance in a real system.

One major problem with most algorithms is that they are not designed to be tolerant of unreliable communication networks. While some methods can properly handle a communication error while transmitting a data value, they often falsely assume that critical control messages will be received without error at every intended destination. Attempts to mask network errors at the Application Layer rather than at the Transport Layer seem to produce algorithms that are very complicated and whose reliability remains in question. In the proposed system, we were able to greatly simplify the synchronization algorithm because the intelligent message transport system guarantees the delivery of messages by the end-to-end acknowledge protocol.

In Section 4.2 we will review the basis for some common synchronization methods. In Section 4.3 we will discuss the method to be used in the proposed system.

4.2 Survey of Methods to Preserve Consistency

In this section, we discuss a few of the many proposed methods for maintaining the consistency of replicated data in a distributed computing environment. Synchronization methods are typically compared by the control strategy employed, namely centralized and decentralized control [Thomas 1978]. Others [Teng-amnuay 1983]
believe that these methods must be categorized on a theoretical basis using, for example, the concept of serializability as a guideline. As the goal in this section is simply to build a context for discussing our proposed algorithm, and we choose to compare existing methods from a quantitative perspective (rather than a theoretical perspective) by using the control strategy as a guideline. Actually, there is little choice in selecting a synchronization method because the structure of the system (i.e. the organization of the data) dictates the type of method that should be applied. In the proposed system, we wish to support the possibility of a fully-redundant database (i.e. one copy at each site), so, synchronization methods that use a centralized control strategy will not be applicable to our system.

As an aide in demonstrating the workings of these algorithms, we will derive some simple cost formulas for each method. These cost functions will attempt to quantify two important aspects of synchronization methods, namely (1) the volume of network traffic generated by read and update requests, and (2) the number of messages required to carry out the request. The following variables will be used in the analysis:

- \( N \) = the number of sites participating in the transaction
- \( C_s \) = the size (bits) of a control message
- \( D_s \) = the size (bits) of a data message (update or retrieval)
- \( T_d \) = an average end-to-end delivery time for a message
- \( V_m \) = the volume (bits) of message traffic generated by a read or update transaction
- \( N_m \) = the number of messages required to carry out the transaction
- \( T_m \) = an average time required to carry out the transaction

To simplify and standardize the calculations, we will assume that the delivery time of a message \( (T_d) \) is unity, and that the cost of transmitting a message between any pair of sites is the same. (This is not the case in a hierarchical network where messages sent to physically distant sites may have to pass through many gateways and transport mediums of varying speeds.) Also, we will ignore the slowdown effect that
interference has on distributed control algorithms, and that blocking has on central-
ized control algorithms. The results of the analysis of the various algorithms should
be used for comparison purposes only; it is likely that they have little relation to
real-world values.

4.2.1 Centralized Control of Updates

In the centralized control scheme, all the control aspects pertaining to the
maintenance of data are coordinated by one specialized site. The site has total respon-
sibility for verifying and executing requests, and for maintaining the consistency of
the database. There are two basic variations of the centralized control scheme,
namely, the centralized database organization and the primary site organization. The
first organization requires that all data storage and control functions be provided by
one central site, and the second organization centralizes the control aspects but allows
for data to be replicated at remote locations.

4.2.1.1 Centralized Database Organization

The centralized database organization is the familiar method in which all information is stored at one site in
the system. Because the data objects are not replicated, all queries and updates are
made on a single copy of the data located at the file server site. Synchronization is
easily achieved by providing a locking protocol that arbitrates simultaneous update
attempts. (Examples of such an arbitrator would include operating system abstrac-
tions such as semaphores or monitors.) The corresponding cost functions associated
with the centralized database organization are straightforward. An update request
can be analyzed as follows:
(1) send a request to update a data object along with the new value:
\[ V_m = D_s \quad N_m = 1 \]

(2) central database server processes request; sends an acknowledge
\[ V_m = C_s \quad N_m = 1 \]
\[ V_m = D_s + C_s \]
\[ N_m = 2 \]
\[ T_m = N_m \times T_d = 2 \]

A read request can be analyzed as follows:

(1) send a request to read a data object:
\[ V_m = C_s \quad N_m = 1 \]

(2) central database server sends data
\[ V_m = D_s \quad N_m = 1 \]
\[ V_m = C_s + D_s \]
\[ N_m = 2 \]
\[ T_m = N_m \times T_d = 2 \]

In a central database organization, the number of messages is constant, and the network delay time is nearly constant. This particular organization provides an optimal update time, but because the data objects are not replicated, we still need two messages to read the data. This will be improved in the primary site organization.

### 4.2.1.2 Primary Site Organization

This scheme differs from the centralized database organization in that sites are allowed to maintain local copies of the information stored in the primary site database. Local copies of data may be read without interaction with the primary database, however, update requests must still be processed by the primary facility. Once an update has been processed, the primary facility then transmits the new data value to each site that maintains a local copy.

The cost of an update operation can be analyzed as follows:
(1) send request to update along with the new data value to the primary facility: $V_m = D_s \quad N_m = 1$

(2) primary site processes request; sends acknowledge: $V_m = C_s \quad N_m = 1$

(3) the primary facility sends the new data to each site: $V_m = (N-1)*D_s \quad N_m = N-1$

(4) each site sends an acknowledge control message back to the primary site: $V_m = (N-1)*C_s \quad N_m = N-1$

$V_m = D_s + C_s + (N-1)*D_s + (N-1)*C_s = N*(C_s + D_s)$

$N_m = 2*N$

$T_m = N_m*T_d = 2*N$

Note that if the network supports a reliable broadcast facility, then step 3 will consist of only one message and step 4 can be eliminated. In this case we have:

(1) as before: $V_m = D_s \quad N_m = 1$

(2) as before: $V_m = C_s \quad N_m = 1$

(3) broadcast new value (acknowledge not required): $V_m = D_s \quad N_m = 1$

$V_m = D_s + C_s + D_s = 2*D_s + C_s$

$N_m = 3$

$T_m = N_m*T_d = 3$

As we can see, the volume and number of messages can be greatly reduced if the underlying network has a broadcast capability. This is misleading because the cost of broadcasting a message, in terms of message delay time may become too large if the system is geographically dispersed and interconnected by slow-speed networks.

4.2.1.3 Summary of Centralized Control

The centralized control scheme offers an optimal data organization with respect to the number and volume of messages required to read and update data. This is especially true for the primary site organization. From the perspective of a site, the cost of reading a local copy of the data is negligible, and the update transaction requires only two messages. Furthermore, the acknowledge message can be eliminated if the internetwork supports a reli-
able broadcast facility. Another reason the centralized control organization is attractive is that the detection of invalid update requests is made easier because the control logic is implemented at only one site. In distributed computer organizations, the control logic is replicated at every site and is difficult to coordinate. Also, crash recovery is simplified because at any point in time, there is only one master copy of the database.

The major disadvantage of using centralized control is that all query activity is suspended if the site maintaining the information becomes inaccessible. This one characteristic eliminates the centralized data organization as a viable system structure for systems that need to be resilient and have a high degree of availability. Another problem with this organization is that it is likely the central database site will become a system bottleneck both in terms of communication bandwidth and in the ability to service requests.

4.2.2 Decentralized Control of Updates

In algorithms that implement decentralized or distributed control, any site can initiate and coordinate an update transaction. The update procedure is similar to that of the centralized control scheme, except that the control signals are generated by the coordinating site rather than by a predetermined primary site. As we will see, the control signal protocols for distributed control algorithms are made complicated by the need to ensure that simultaneous updates initiated by different sites cannot introduce deadlock or create inconsistencies in the data. There are two basic classes of distributed control algorithms, namely global locking and time stamps. These are discussed in the following two sections.

4.2.2.1 Global Locking The global locking scheme is similar to the centralized database approach in that transactions are forced to execute sequentially with little concurrency. In some applications, even read operations must be locked to
prevent intervening updates in a "read-modify-write" sequence. The approach to global locking is to make each access an atomic operation. This means that we apply a two-phase locking protocol to gain exclusive access to every copy of the data object for the duration of the transaction. The details are described below.

The first step of the atomic operation is to gain control of all copies of the data object. This is accomplished by transmitting a "lock" command to each site containing a copy of the data. Sites will respond with either an "acknowledge" message indicating the lock was successful, or a "not-acknowledge" that indicates that the lock failed because a different update operation involving this data object is in progress. In the second case, the update will fail and all applied locks are released. After successful completion of the lock step, the coordinating site then transmits the new data value to each site involved with the update transaction. (In practice, the number of messages is reduced by transmitting the updated value and the lock command in a single message.) The receiving sites will create a temporary version of the new data object that is not yet accessible to other sites. The second step in the two-phase protocol is to commit the new value. After each site has accepted the updated object value, the coordinating site then transmits a commit message that directs each site to purge the old value and make the new value accessible. Again, if any site rejects the update, the entire transaction is terminated and the value of the data object is rolled-back to the previous value.

An analysis of the two-phase protocol follows. We will assume that each of the N sites in the system contains a copy of the data to be updated.
(1) send lock command and updated value:
   \[ V_m = D_s(N-1) \quad N_m = N-1 \]

(2) receive acknowledgement from each site:
   \[ V_m = C_s(N-1) \quad N_m = N-1 \]

(3) send commit command to each site:
   \[ V_m = C_s(N-1) \quad N_m = N-1 \]

   \[ V_m = D_s(N-1) + C_s(N-1) \times 2 = (N-1)D_s + (2N-2)C_s \]
   \[ N_m = 3\times N-3 \]
   \[ T_m = (3N-3)T_d = 3N-3 \]

If exclusive access to the data is not required for a read operation, then the local copy
may be read at any time. If exclusive access is required, then a read transaction will
include a lock step, as shown below:

(1) send lock command to each site with data:
   \[ V_m = C_s(N-1) \quad N_m = N-1 \]

(2) receive acknowledgements:
   \[ V_m = C_s(N-1) \quad N_m = N-1 \]

(3) access local copy — negligible time

(4) unlock all copies:
   \[ V_m = C_s(N-1) \quad N_m = N-1 \]

   \[ V_m = (3N-3)C_s \]
   \[ N_m = 3N-3 \]
   \[ T_m = (3N-3)T_d = 3N-3 \]

4.2.2.2 Time Stamps and Event Counters

We have seen that global locking serializes transactions by forcing them to execute sequentially. The class of
synchronization algorithms described in this section also serializes transactions, but
do so by ordering them in time instead of using locks. Each update transaction is
tagged with a time stamp [Bernstein, et al. 1978,1981] or an event counter [Reed and
Kanodia 1977] that can be used to systematically select one transaction over another
should a conflict arise.

A time stamp is a unique system-wide time that is assigned to each request to
update replicated copies of data. Obtaining a unique system-wide time stamp in a dis-
tributed computing environment is an interesting problem for which at least three
solutions have been proposed:

(1) The simplest solution is to have one site act as a time server that allocates unique time stamps to other sites that wish to initiate a transaction. There are two problems with this approach: First, the time server may become a bottleneck in the system if updates are frequent. Also, for distant sites, there may be an appreciable delay in obtaining the time stamp. Second, the integrity and availability of the system is dependent on the well being of one site. Should the time server site become isolated by a network failure, there is no alternative but to shut down the other sites in the system.

(2) Another approach is to have each site maintain a real-time clock. It has been suggested by Reed [Reed 1978] that sites could maintain accuracy by monitoring a time radio station, like WWV. One problem with this approach is that while time stamps do not need to correspond to real-world time, they do need to be unique. Thus, even though sites would be able to maintain an accurate time, there still remains the problem of choosing a time for a transaction and guaranteeing that no other transaction in the system uses that same time. It has been suggested that the case of multiple transactions with the same time stamp be handled by terminating all but one transaction. This will allow the one transaction to complete and force the sites originating the other transactions to resubmit them at a later time.

(3) A third approach to the problem is to have each site maintain a real-time clock as in the above solution, but to have the clocks synchronize with each other without the use of an external time source [Lamport 1978]. When a transaction arrives at a site, the time stamp on the transaction is inspected and if it is greater than the local site time, the site clock is advanced accordingly and the transaction is accepted. If the time stamp on the transaction is less than the local time, the transaction is rejected. Eventually, all site clocks will
become synchronized with each other. This technique becomes a little tricky because a site needs to date its transactions with a time that is sufficiently far enough into the future so that it will be greater than the local clock at the destination sites. It is not difficult to envision a scenario in which one site consistently chooses a time that is far into the future so that all transactions will be accepted. By continually moving the site clocks ahead, other valid transactions will be erroneously rejected.

Event counters are similar to time stamps in that they serialize transactions by ordering them in time, but they are more straightforward because they make no pretense about corresponding to real-world time. Event counters are simply monotonically increasing integer values that can be used to uniquely tag transactions. In their purest form, however, event counters have the same problem as time stamps. Without a global agent that guarantees the uniqueness of event counters, it is possible that more than one transaction could choose the same value. Here again, we would need to develop a strategy for selecting the transaction that would be allowed to complete.

A major advantage in using time stamps or event counters instead of locking is that transactions that do not interfere with each other are allowed to execute in parallel because mutual exclusion is not required. A disadvantage to using time stamps is that intermediate steps of the update are visible to other sites and the potential for mutual inconsistency exists unless some precautions are taken. For example, consider the case in which copies of an object are located at physically distant sites. As the commit message propagates through the network, transmission delays will allow sites closest to the coordinating site to commit the new value sooner than distant sites. The result is a temporary inconsistency that permits a distant site to initiate a second update request for a data object whose value is computed from the soon-to-be obsolete value. In this case, the new value for the second object is erroneous. This can be easily handled by assigning a time stamp to every data object in the
database when the object is created or updated. Instead of transmitting only the new value in an update request, we must also transmit the time stamps for each of the data objects used in the computation of the updated value. (These are called base variables.) Sites will accept the update request only if the time stamps of the base variables in the update message are the same as those of the local copies of the base variables. If any of the time stamps differ, the site will reject the update request thereby forcing the coordinating site to recompute the new value and try again.

As discussed in Section 4.3.2, our method will be based on time stamps, but will use version numbers for data objects rather than a time stamps or event counters. Version numbers have an advantage in that a site can easily generate the next version number by incrementing the current value. If a conflict arises because of simultaneous updates using the same version number, one or both of the transactions will be terminated and restarted by the originating site at a later time.

One synchronization algorithm that uses time stamps is the voting, or majority consensus algorithm [Thomas 1978]. The concept is to have each site vote on whether to accept or reject the update transaction. If all sites (or a majority of sites, depending on the algorithm) vote to accept the update, the coordinating site then transmits the commit message and the new value is made accessible; otherwise, the transaction is terminated. The basis for acceptance of the transaction is that the time stamps of the new value and the base variables must be the same as the time stamp on the local copies of the base variables.

While this method generates more message traffic than other algorithms do, it has several advantages. First, it is robust in that sites can crash without effecting the operation of this algorithm because only a majority of the accessible sites need to participate. Second, it handles transmission failures in that lost or duplicated messages do not cause mutual inconsistency. An analysis of a voting scheme follows. We will assume that all sites respond, and that the appropriate base variables can be sent in
one Ds message. Therefore, in order to update a replicated data object, it will require a message with a volume equivalent to two normal messages; one message for the new data value, and one for the base variables.

1) send update request, new value, base variables, and time stamps
   \[ V_m = (N-1) \times 2 \times Ds \]  \[ N_m = N-1 \]
2) receive acknowledgements
   \[ V_m = (N-1) \times Cs \]  \[ N_m = N-1 \]
3) send commit
   \[ V_m = (N-1) \times Cs \]  \[ N_m = N-1 \]
   \[ V_m = (2 \times N-2) \times (Cs + Ds) \]
   \[ N_m = 3 \times N-3 \]
   \[ T_m = (3 \times N-3) \times Td = 3 \times N-3 \]

4.2.2.3 Summary of Decentralized Control  As noted previously, solutions to the consistency problem in a system that supports decentralized control tend to be more complicated than consistency-preserving algorithms where centralized control is employed. The added complication is because multiple sites can simultaneously initiate update transactions, thereby introducing the possibility that replicated objects will become inconsistent. Two common methods were presented, namely global locking and time stamps. In analyzing both methods, we found that the volume and number of messages required to coordinate an update transaction are comparable. Furthermore, both methods improve the resiliency of the system because they are not dependent on the well being of one site. One major difference between global locking and time stamps is the degree of mutual consistency that each method provides. In global locking schemes, we can guarantee that replicated objects are mutually consistent in the strictest sense. Replicated objects will always denote the same value when they are not locked. We cannot make such a strong claim for replicated objects that are managed by time stamp methods. Since objects are not locked during the update process, it is possible that copies of an object will be inconsistent for a brief period of time until the update has completed. But because time stamps serial-
ize transactions, we can claim that if transactions are executed in the order that they are generated, then replicated objects will be mutually consistent when all update activity ceases.

One advantage that time-based synchronization methods have over locking is that read operations can execute concurrently. The penalty for this advantage is that the time stamps of the base variables need to be transmitted with the update message. Tagging each replicated data object with a version number or a time stamp has other benefits as well. One major benefit is that damaged data caused by other types of errors unrelated to the update process (i.e. software/hardware failures) can be detected and corrected in the normal course of operation.

4.2.3 Summary

We have reviewed the basic methods that have been proposed to maintain the consistency of a database. The goal was to obtain an insight into the different characteristics of each method, such as the amount of handshaking required to enact an update transaction. We have seen that all methods are comparable in terms of the volume of message traffic; and in the number of messages generated for a transaction. The choice of an algorithm for the proposed system is dictated by certain key characteristics of the system. We require that the synchronization method be capable of handling replicated data up to the case of a fully-redundant database. Furthermore, we will require a truly distributed control scheme, one in which any site can initiate and coordinate an update transaction. Our choices for a synchronization method are now limited global locking and time stamps. While global locking is the easiest to implement, the inability to handle concurrent read operations is a serious drawback. This coupled with the previously mentioned crash recovery benefits that time stamps provide has led us to decide that the synchronization method used in the proposed system will be based on time stamps.
4.3 Proposed Algorithm

4.3.1 Introduction

In this section, we will describe a solution to the consistency problem for the proposed system. Unfortunately, there is somewhat of a cyclic relationship between the material in this section and the material in Chapter 6. Our intent here is only to sketch the solution to the synchronization problem; specific implementation details, such as message formats and data structures, can be found in Chapter 6.

Our goal is to construct a solution to the multiple copy consistency problem that satisfies as many of the properties listed in Section 2.4.6 as possible. Our approach will be to use time stamps rather than a locking policy for reasons discussed in the last section.

4.3.2 Version Numbers

Our solution to the consistency problem is based on the use of time stamps to serialize transactions. We have adopted a method where version numbers are used instead of an actual time value [Reed 1978]. The use of version numbers is attractive because they are easier to manage than other popular methods, including system-wide a time resource that generates unique time stamps, or algorithms that attempt to synchronize a "pseudo-time" between sites. Version numbers begin at 1 for newly created data, and are monotonically increasing for successive versions. Because versions (or time stamps) create a partial ordering of transactions with respect to time, transactions are guaranteed to be serializable. Because individual update operations use a two-phase protocol which has been shown [Teng-amnuay 1983] to preserve consistency, then it follows that consistency is preserved for any sequence of transactions.

Version numbers allow us to view the state of a data object as a sequence of immutable version. That is, the value of an object is never modified; a new value is
created as a separate version. The highest version number contains the most recent value, the next highest is the previous value, and so on. The maintenance of version numbers is performed by the gateway processor and is transparent to user processes. The user process will begin a session by requesting from the gateway a private communication channel called a *port*. The port code assigned by the gateway will then be used to identify subsequent communication from that particular process. Note that it is our intent that each process acquire a different port; transactions generated by more than one process per port could potentially interfere with each other and produce undesirable results. The gateway maintains a name table that contains information about the replicated data that the attached site is using. Included in the name table entry for a particular data object is a list of ports that have accessed this data object, along with the last version of the object that was read by the process connected to the port. When we refer to a version number for a data object, we are referring to a pair of version numbers: the true version number that corresponds to the value stored in the gateway, and the port version number that corresponds to the highest version number read by a process attached to that port. Because the name table (and the version number) is stored in, and managed by the gateway, a user process cannot directly access or modify this information, except through the well-defined commands supported by the gateway interface. This organization improves the reliability of the system because user programs operating at a site cannot damage the integrity of the system by intentionally or otherwise corrupting this information.

Another advantage in using version numbers is that crash recovery is made easier because the version number identifies a particular instance of an object in a sequence of versions. The ordering of the sequence is based on the concept of a version, as opposed to time stamps that are based on clock time. With version numbers, it is easy to determine that only one version is between 7 and 9, but with time stamps,
there is no way to determine how many versions were created between time stamps 10:17:19 and 10:17:59. Because of the correspondence between version numbers and transactions that create versions of data, crash recovery is enhanced by periodically dumping the gateway name table and data storage space to a stable storage, such as disk.

4.3.3 Reading and Updating Replicated Data

The procedure for reading or updating replicated data is straightforward. The user process at the attached site presents the update or read service request to the gateway, and the gateway then manages all synchronization and other network-type functions. The user process will always receive one message from the gateway indicating whether the request could be serviced.

To read the present value, the user process sends a message through the assigned port requesting the gateway read service. The gateway will return the present value of the data object and update the port version number with the true version number. If the named data object was not found in the gateway name table, the request is terminated and the user process is notified of the error. Because we do not employ a locking strategy to manage synchronization, there is never any interference between read and update operations. Thus, a user process will never be denied read access to an object because it is being updated. If an update is in progress during a read operation, the version of the data read will be outdated. We will see later that the user process will not be allowed to use the outdated value in a computation involving replicated data. In some cases, however, it does not matter if the value is outdated. It is not hard to envision some applications in which the data are updated frequently and the user wishes to know only an approximate value. An example of this would be contacting your bank to determine the balance in your checking account; the quoted balance is approximate in that it is difficult to determine whether recent trans-
actions have been processed.

A user process can request that a named replicated data object be updated by sending the update request to the gateway along with the name and new value of the data object. Before accepting the transaction, the gateway will compare the port version number for the named data with the true version number and if they differ, the transaction will be rejected. (If they differ, then there is a possibility that the new value was created from an outdated value, as can happen in "$X \leftarrow X+1$" if an intervening update modifies X after the value was read but before the newly computed value is written.) If the version numbers are the same, then the gateway will proceed to coordinate a network update operation. If the update fails for reasons to be explained later, the transaction is terminated and the user process is notified of the cause of the error. If the transaction succeeds, the true version number and the port version number are incremented.

The update process becomes complicated slightly when more than one replicated data object is involved in the update transaction. Consider the following example where X and Y are replicated data, and Z is a local variable.

\begin{align*}
1. & \quad Z \leftarrow X + Y \\
2. & \quad X \leftarrow Z
\end{align*}

At line 1, the value of the local variable Z is replaced by the sum of the replicated data objects X and Y. At line 2, X is updated from Z. At first glance, it looks like this code segment will always have the same effect as "$X \leftarrow X+Y$", but in reality, it won't always. Again, the problem is that the code segment is not atomic with respect to X and Y. It is possible that during the processing of line 2, some other site has updated the value of Y, and thus, our new value for X in line 2 is outdated because it is based on an outdated value of Y. This interaction is easier to see if we put version numbers on the replicated data and repackage the example as follows:
IMPORT \( X_n \) and \( Y_m \)
\[
Z \leftarrow X_n + Y_m
\]
\[
X_{n+1} \leftarrow Z
\]
EXPORT \( X_{n+1} \)

We can see that the value of \( X_{n+1} \) depends on the values of the base variables \( X_n \) and \( Y_m \), and that \( X_{n+1} \) will not be correct unless the base variables are current. The problem of detecting outdated base variables can be handled if we expand the update message to include the names of the replicated data objects that were used in computing the new value. In this example, we would send an update message for \( X_n \) to the gateway and include the name of \( Y_m \). The gateway can then check the port version number of \( Y_m \) with the true version number and reject the transaction if there is a discrepancy. Note that we are able to detect transactions that will fail without actually sending the transaction to other sites. This is in contrast to other implementations of time-based synchronization methods [Thomas 1978] where remote copies of replicated data are consulted for every transaction. Our method has the potential for considerable communication bandwidth savings in applications where frequent updates cause user-maintained copies of data to become obsolete.

In the above example, we have extended the scope of the update message to include the relevant portions of the environment in which the new value was computed. To date, there has not been much research on mechanisms that can be embedded into languages or operating systems that assist the user in discovering these types of data dependencies. Reed [Reed 1978] offers a language construct called a pseudo-temporal region that creates a new naming environment, much like a block in PL/I. In this scheme, object versions are determined when the region is entered at run-time, and all updates within the region make use of those version numbers to reject transactions that are based on outdated values. Another approach to this problem would be to investigate the applicability of graph-theoretical techniques developed for code optimization that could assist a high-level language compiler in automatically generat-
ing the update message with the appropriate environment variables included. Because
we feel that a solution to this problem should be provided by the on-site Application
Layer software and not by the Transport Layer software in the gateway processor, we
will not investigate this problem further.

4.3.4 Distributed Updates of Replicated Data

The procedure to update replicated data that is stored at different gateways
throughout the network is implemented by a combination of the two-phase protocol
and version numbers. The first phase entails transmitting the new data value to the
gateways involved in the transaction. If every gateway accepts the transaction, the
coordinating site transmits the commit message and the new value becomes accessible.
If any site rejects the transaction, it is terminated by the coordinating site.

The first step involves transmitting a network update message to all gateways
that are now managing a copy of the replicated data. As in the case of the site update
message, the network message also includes the new value, version number, and the
name of the data object, as well as the names and version numbers of the replicated
data that were involved in computing the new value. On receiving the update mes-
sage, each gateway participating in the transaction will check the version numbers of
the named data for consistency with the local name table. Two error conditions
could occur that cause the transaction to be rejected:

(1) If the version number of the new value is less than or equal to the local ver-
sion number, or if there is an update in progress involving the named data
object, then the new value is rejected. This can happen when (1) concurrent
updates interfere with each other, or (2) the site originating the update has an
outdated database. In the first case, the scenario would be that two updates
from different sites attempt to modify the same replicated data object. Event-
tually, these updates will meet at a one of the gateways and all but the first
(2) The transaction will be rejected if any of the base variables are outdated. If the transaction is rejected at this point, the originating site is notified of the cause and may attempt to restart the transaction at a later time. If the version numbers are verified to be correct, the gateway will create a temporary copy of the new value. (It will not become accessible until the transaction has successfully completed.) If every participating gateway accepts the update, the originating gateway will then transmit a commit message that directs the gateways to install the temporary version as the new version. If any gateway rejects the transaction, the source site will transmit an abort messages that directs the gateways to delete the temporary copy and terminate the transaction. If for some reason a gateway does not correctly receive either the abort or commit message, the transaction will eventually timeout in each participating gateway and be terminated. This introduces the possibility of some gateways not being current because of a system crash during an update. This problem will eventually be corrected as the next transaction involving that data object will be rejected because it is outdated; this event forces the gateway to recopy the current value from an adjacent gateway.

We believe that the probability of losing control messages is very slight, however, in a real system, we need to anticipate that eventuality. It is precisely the inability to recover from lost control messages that causes many proposed update algorithms to fail. Some of these algorithms assume that control message are atomic when in fact they are not. That is, if a control message is lost or becomes corrupted, it is likely that some of the gateways have been visited while others have not. In the system proposed in this thesis, the end-to-end acknowledge protocol (Section 6.3.3) minimizes this possibility. However, as we have shown in the above discussion, data objects that have become inconsistent are automatically refreshed on the next transaction that it participates in.
4.3.5 Summary

In this section, we have given a solution to the multiple copy consistency problem within the context of the proposed system. The structure and properties of the proposed system greatly simplify the solution to the consistency problem. The end-to-end acknowledge protocol guarantees reliable delivery of messages and allows us to be confident that control messages will be received at their intended destination. If some unexpected circumstance prevents the delivery of control messages to a particular gateway, the copy at that gateway may become inconsistent for a time. However, no harm results from this situation because the copy will be updated the first time that it participates in an update operation. The implementation details of these mechanisms are given in Chapter 6.
5. NAMING IN A DISTRIBUTED COMPUTING ENVIRONMENT

5.1 Introduction

In this chapter, we will develop a method of managing the system-wide name space for the proposed system. By *name space* we are referring to the geographic area in the system in which a particular name is accessible (i.e. its *scope of reference*). It is our goal to develop a consistent and enforceable mechanism that allows sites to create names for different types of objects and to define the scope of reference for that name. The proposed method is independent of both site-specific naming conventions and on-site application software, such as operating systems and utility programs.

In every computer system, there exists an identification scheme that allows us to refer to various objects with some type of symbolic name. These objects include:

- Data files
- Records in a data file
- Fields in a record
- Procedures or tasks
- Sites
- Gateways
- Users
- Groups of users
- Resources (disk, terminals, etc.)
- Programs

To date, there is no commonly accepted naming scheme for even a subset of the above listed entities. This presents a problem when designing a system that interconnects heterogeneous computing resources in that any system-wide naming scheme is likely to conflict with naming conventions already established by application software at a site. Our solution to this problem was to establish a naming convention within the intelligent message transport system and to provide for local aliases to handle cases where established site naming conventions are preferred.
Names provide the fundamental means of accessing stored data. The process of converting the symbolic label into a machine-understandable label is called a name accessing function. In the simplest case, there is a one-to-one mapping between a name and the data object that it refers to. We will see that there are cases in which a one-to-many (broadcast) or a many-to-one (aliases) mapping is desirable. In addition to selecting the object referred to, names are often used to convey the location of the object and the route that must be traveled to reach the object. For example, in von Neumann architectures, the name of a memory cell is its address or location, and in UNIX [Ritchie and Thompson 1974], the fully-qualified name of a data file includes the path from the root node of the file system tree.

The naming scheme developed in this chapter provides for the following features:

- All references to replicated data are made with a symbolic name. Individual sites and user processes perform no address translations.
- Users may choose their own names for data objects. The only restriction is that they be composed from the standard ASCII character set. There are no implied naming conventions (e.g. "c" language files in UNIX have a "c" suffix). It is the responsibility of users sharing data to agree on naming conventions.
- The management of the name space is dynamic. That is, names are not bound to an object until a user process so directs. This is in contrast to task name management schemes in contemporary languages like Ada [Wegner 1980] where a user must specify task and communication port names at compile time.
- Names are route and location independent. There is no route information encoded in a name.
- The scope of a name can be defined by the process that creates it. At any point in time, the scope may be narrowed or amplified. It may be extended to particular sites or groups of people. This implies that one identifier can be used to name
different data objects without ambiguity as long as the scopes are disjoint.

- Management of the name space is provided by the gateway processors. There are no global tables or special name server sites. This increases the resiliency of the system because there is no one critical site that must be operational.

- A one-to-many name accessing function is provided so that a user process may conveniently broadcast messages to other processes.

- A many-to-one name accessing function is provided so that a user process may reference particular names via a local alias. This facilitates cases where local site software imposes naming conventions that may differ from those adopted by other sites.

5.2 Related Research

The problem of managing a global name space is a relatively recent problem. During the past decade, research in naming mechanisms in the areas of programming languages and operating systems has focused almost exclusively on encapsulating naming environments with the intent of restricting the scope of reference for names. Some recent work on managing global name spaces in programming languages has resulted in the package construct in Ada, and the module construct in Modula-2 [Wirth 1982]. Unfortunately, names in these languages are bound to objects statically (i.e. at compile or link time) and thus, these name mapping techniques cannot be applied to our problem.

There are two popular approaches to handling the naming problem in a distributed computing environment: a common operating system for each site, and a global name server. These are discussed below.

1. One approach to the problem is centered around interconnecting homogeneous computers running the UNIX operating system [Curtis and Wittie 1984]. The
The intent of this research is to extend the hierarchical UNIX file structure in a way that makes machine interconnections transparent to the user. Files can then be shared between remote computers using the standard UNIX commands, and processes can be scheduled to execute on any of the computers in the internetwork. The ability to interconnect systems in this fashion is a natural extension of UNIX. The management of the global name space (when treated as file names) is already part of the services provided by the operating system. This solution to the naming problem has been shown to be successful in experimental machine interconnections. The drawback to this approach is that the computing resources must be homogeneous, at least with respect to the software. In the proposed system, no assumptions can be made about the level of sophistication of the hardware or software for a site. One design goal is to provide internetwork access for site installations that range from a single board computer up to a timesharing system. This specification dictates that naming and synchronization must be provided by the gateway processors and not by common site software.

Another popular approach to the problem of naming in a distributed environment has been the use of global name servers, such as those in CSNET [Comer 1983]. Here, the concept is to designate one site as a clearinghouse for names. All requests to create new names or lookup old names must be sent to the name server. The primary advantage to this scheme is its simplicity. The enforcement of access policies and naming conventions is simplified when the control is localized to one site. The major disadvantage in this scheme is that the integrity of the system is dependent on the health of that one site.

In the proposed system, we chose to implement the control of the name space in the gateway processors. This decision complicated the logic of the gateway, however, the result will be a more robust, resilient system.
5.3 Naming Conventions

Names in the intelligent message transport system can be up to 24 characters in length. They may be composed of any printable ASCII characters except the "@" character. (Of course, common sense would restrict the choice to letters, numbers, and a few special characters like the underscore.) The "@" is used to delimit components in a network path specification. For example, the path specification "ISU@CSD@VAX1" indicates that the message should travel from gateway "ISU" through "CSD" to gateway "VAX1".

5.4 The Proposed Name Management Method

The proposed system supports two modes of communication, namely transaction-oriented and interprocess communication. Transaction-oriented communication refers to a datagram-like mode of communication that gateways use to synchronize replicated data. Interprocess communication refers to the virtual circuit, or conversational mode of communication between two processes. Each of these communication methods is directly supported by the intelligent message transport system. In all cases, initiating and using these services is accomplished by submitting a simple command to the attached gateway. The commands and response codes are encoded in a standard format\(^1\) using ASCII characters\(^2\).

In proposed system, there are four types of entities that can be referenced by a symbolic name at the internetwork level. These are: replicated data objects, gateway processors, communication ports, and group names.

1. **Gateway Names.** In Chapter 6, we discuss the message routing mechanisms of the proposed system. The basic concept is to uniquely label each gateway and then use the name to form a path specification for a message. For example, the

\(^1\)The independent site protocol (ISP) described in Chapter 6 is used.
\(^2\)Data messages can include any binary data and are not limited to the ASCII character set.
path "1@4@2" would indicate that the message should be routed from gateway 1 to gateway 2 by way of gateway 4. As a convenience to user processes, gateways may be referenced by predefined symbolic names so that path specifications can contain significant names that correspond to the location of the gateways. Instead of saying "1@4@2", a user could specify, for example, "IowaState@ComputerScience@vax1". It is our intent that gateway names or numeric identification codes be placed in read-only memory on-board the gateway processor. It seems unlikely that gateway names would be changed often, if ever. Note that we have chosen to name gateways and not sites. Because a gateway can have at most one attached site, there is no ambiguity.

(2) **Port Names.** User processes may communicate with the gateway through an abstraction called a communication port. A port is a named communication channel that is allocated on request to a process by the gateway, and is used to identify subsequent communications as belonging to that process. A process may, for example, use the port to send commands to the gateway, or the process may export the named port to other sites to facilitate process-to-process communication.

(3) **Replicated Data Names.** The intelligent message transport system provides for synchronization of multiple copies of data located at different sites in the system. To create a local copy of the data, a user process first establishes a port connection with the attached gateway processor, and then requests that the gateway locates and copies the named data object. From that point on, the user process may send read and update commands to the gateway processor through the port, and the gateway will manage system-wide synchronization with the other copies of the data.

(4) **Group Names.** Protection and control of information sharing is accomplished through the use of group names. After establishing a connection with the
gateway, a user process may send a command message to the gateway that lists the groups the process "belongs to", along with their passwords. All subsequent network messages generated by the gateway on behalf of the process will contain the group codes. In this way, user processes can be prevented from accessing public data or port names unless they belong to an appropriate group.

The names of ports and data objects may be made public to other sites. The user process requests that the gateway enter the name into the local name table and set the access protection so that only the specified groups of users can access the named object. The gateway processor then broadcasts the entry to all other gateways which check for scope conflicts with the new name. A scope conflict occurs if the same name is already accessible to any of the specified groups. In this case, the originating process must choose another name.

5.5 The Name Table Implementation

This section describes the implementation of the gateway name table. The gateway name table software provides the means for converting a symbolic user-defined name into an internal code that identifies, for example, a particular communication port or a network interface. We call this process name resolution. Because every entity in the system is addressed by a symbolic name, the efficiency of the name resolution process is of prime importance in the proposed system. Every message processed by a gateway will require at least one name resolution. In the simplest case, if the message should be routed to an adjacent gateway, then the gateway name needs to be converted into a network interface code so that the message is sent to the proper interface. Eventually the message will reach its destination and the target gateway will need to convert the symbolic name into an internal code representing a replicated data object or a port name.
There were four factors that influenced the structure of the name table and the search method chosen. These are:

1. **Memory Size.** In Chapter 6, we discuss two possible implementations for the gateway processor. The simplest configuration contains 256K bytes of read/write memory, and the extended design can support up to 16M bytes of read/write memory. Either gateway configuration will provide ample memory for the name table and data storage. Because we are not memory-limited, we chose to favor other factors over optimization of memory usage whenever possible. For example, names are stored in the table as ASCII strings without the benefit of space-saving data compression techniques.

2. **Name Table Size.** Because a system like the one proposed in this thesis does not exist, it is difficult to estimate an upper bound on the name table size. In the context of a transaction-oriented system, we would expect that no more than a few hundred names would be active at any point in time, and therefore, have selected a generous upper bound of 1024. We are not so much concerned with memory usage as we are with determining the most efficient representation (with respect to search time) for the name table.

3. **Speed Requirements.** Because every message processed by the gateway will require at least one name resolution, the speed of the lookup function will become important. We have chosen one millisecond as the upper bound for the name resolution process working on a full (1024 entries) name table. The 1 ms timing requirement is based on an estimate of the expected volume of message traffic through one gateway. A study of the performance characteristics of an Ethernet local area network was performed at the Xerox Palo Alto Research Center [Shoch and Hupp 1980]. The network supported over 120 host systems and carried more than 300 million bytes of information daily. The peak utilization approached 98 percent. An analysis of the data collected
has shown that the inter-message arrival time has a mean of 39.5 ms and a median of 8.5 ms. Because much of the traffic consists of request/response transactions with a server, it is common for the request message to be followed by the response without intervening messages. It was determined that 50 percent of the messages are followed by the next packet within 10 ms, 90 percent are followed within 64 ms, and 99 percent within 184 ms. With a name resolution time of 1 ms, it should be possible for a gateway to handle message traffic from two or three heavily loaded Ethernet networks. Similar gateway performance requirements can also be calculated for other modes of communication. For example, consider the case where a single network is dedicated to handling traffic for a file server. Performance monitoring on several production UNIX systems [McKusick, et al. 1983] has shown that the effective utilization of the transfer capacity of a disk is approximately 10 percent to 40 percent, or about 466K bytes/second for a 1.2M byte/second disk. This translates to about 61 page transfers per second, or one transfer every 16 ms. If we assume that each transfer would be packaged as one message, then a 1 ms name resolution time would mean that the symbolic name processing will not become the limiting factor for high throughput.

(4) \textit{Name Table Operations}. The only operations that must be supported by the name table organization are: \textit{insertion}, \textit{deletion}, and \textit{membership}. We expect that the insertion and deletion operations will occur infrequently, therefore, the choice of a data structure for the name table favors \textit{membership} over \textit{insertion} and \textit{deletions}, with the requirement that all operations meet the 1 ms criteria for a name table containing 1024 names.

\textbf{5.5.1 The Name Table Data Structure}

The name table is organized as a type of binary search tree known as a 2-3 tree [Aho, Hopcroft, and Ullman 1975]. In the 2-3 search tree, names are stored along the
leaf nodes in lexically ascending order. Each interior node keeps track of the largest left subtree value and the largest center subtree value, so it is easy to determine which path to follow towards a leaf node. An example is shown in Figure 5-1. The important aspect of the 2-3 tree organization is that insertions, deletions, and member operations will all execute in $O(\log_2 N)$ time, where $N$ is the number of leaf nodes in the tree (i.e. the number of entries in the name table). The choice of this particular organization is an example of speed and simplicity taking precedence over optimized

Insertion order: UNIX, VMS, CPM, FORTRAN, DBASE

![Diagram of 2-3 tree insertions]

Figure 5-1. An example of 2-3 search tree insertions
memory usage. In Figure 5-1, note that the names are stored at the leaf nodes and that the interior nodes contain only pointers. If the tree were to contain the maximum 1024 ($2^{10}$) entries, then in the worst case there would be 512 ($2^9$) interior nodes occupying about 6K bytes of memory.

To verify that the 1 ms time frame for name resolution can be met, we have calculated some search times for different numbers of name table entries. Our assumptions are:

1. The name resolution code is executed on an Intel iAPX 286 processor [Intel 1983] that uses a 10-MHz processor clock (20-MHz system clock) and has a basic clock cycle of 100 nanoseconds.

2. The timing of the character string compare instruction is $(5+9N) \times 100$ nanoseconds, where $N$ is the number of characters in the string. The compare instruction will terminate when a pair of corresponding characters are not the same, or when a preset count register decrements to zero.

3. While names can be up to 24 characters long, we expect the average name length to be around 8 to 16 characters. We will assume that in any string comparison, half of the characters in every name will be compared on the average. This fixes the time to compare two strings at $(5+9 \times 8) \times 100$ nanoseconds, or 7.7 microseconds. The worst-case timing would be 22.1 microseconds for all 24 characters.

4. The search time for the 2-3 tree is $\log_2 N$, where $N$ is the number of entries in the name table.

Based on these assumptions, we arrive at the following results:
<table>
<thead>
<tr>
<th>Entries N</th>
<th>Comparisons log₂N</th>
<th>Time (µs) 8 chars.</th>
<th>Time (µs) 24 chars.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>7.7</td>
<td>21.1</td>
</tr>
<tr>
<td>32</td>
<td>5</td>
<td>38.5</td>
<td>105.5</td>
</tr>
<tr>
<td>64</td>
<td>6</td>
<td>46.2</td>
<td>126.6</td>
</tr>
<tr>
<td>128</td>
<td>7</td>
<td>53.9</td>
<td>154.7</td>
</tr>
<tr>
<td>256</td>
<td>8</td>
<td>61.6</td>
<td>168.8</td>
</tr>
<tr>
<td>512</td>
<td>9</td>
<td>69.3</td>
<td>189.9</td>
</tr>
<tr>
<td>1024</td>
<td>10</td>
<td>77.0</td>
<td>211.1</td>
</tr>
</tbody>
</table>

We can see that the worst-case time for a name table search, insertion, or deletion using the iAPX 286 processor is only 0.2 ms which is safely below the 1 ms requirement. (In Chapter 6 we use the iAPX 186 and iAPX 286 interchangeably. The iAPX 186 is slightly slower, giving a worst-case performance of 0.5 ms.)

5.5.2 Name Table Entries

This section describes the format for name table entries. There are four formats for name table entries corresponding to the four types of entities that can be named. Because each entry contains fields that specifically support that type of entity, the name table is composed of different types of records. This complicates the storage management scheme slightly because of the need to occasionally coalesce adjacent blocks of free memory into a usable block of memory. The software required to perform these types of memory management functions will be needed for the gateway data memory also, so we incur little penalty for using them here. We estimate that a full name table would occupy about 80 to 100K bytes of memory. This is based on an average size of 82 bytes for the replicated data name table entry multiplied by 1024. The different types of name table entries are discussed below.

5.5.2.1 The Gateway Name Table Entry Gateway processors are referenced in a path by either their unique identification number, or by the unique symbolic name. A gateway needs to know the names of only the adjacent gateways,
which are determined during the power-up sequence. A name table entry for a gate-
way processor provides the translation from a symbolic reference to an internal code
identifying the software routine that controls communication on that network. The
numeric gateway name is treated like a character string and also has a name table
entry. We expect that a gateway will have no more than four network connections
to other gateways. Thus, there should be at most eight name table entries for adjacent
gateway processors. The format for the gateway name table entry is:

```
TYPE gateway IS RECORD
    rectype: (gateway, port, data, group);
    name: string;
    interface: interfacecode;
END RECORD;
```

The "rectype" field indicates that this record is a gateway name table entry. The
"name" field contains the symbolic name of the adjacent gateway, and the "interface" field contains an internal code indicating which physical network the gateway is con-
ected to.

**5.5.2.2 The Communication Port Name Table Entry** Communication ports are named virtual circuit connections between processes. The name table entry for a port is:

```
TYPE port IS RECORD
    rectype: (gateway, port, data, group);
    name: string;
    alias: string;
    idport: portcode;
    groups: ARRAY OF RECORD
        permissions: modes;
        groupname: address;
    END;
    copies: ARRAY OF INTEGER;
END RECORD;
```

The "rectype" field indicates that this is a port name table entry. The "name" field
contains the system-wide port name, the "alias" field contains the local alias for the name, and the "idport" field contains an internal code that is used to route message to the correct user port. Each "groups" record contains an entry for a group of users that are allowed to access this port. The "permissions" field contains a set of bits that represent the allowable access mode for this group (Section 5.5), and the "groupname" field contains a pointer to the corresponding group name table entry. The "copies" field is used in a heuristic that prevents the transmission of control information to gateways that are not involved with this name. Each element of the array corresponds to a particular out-going network interface (thus, the array usually contains 1 to 4 elements). The value of that array element is the number of copies of this replicated resource that can be found in that subtree. If the count is zero, then no control messages are propagated through that subtree. For ports, this scheme is used to limit broadcast messages involving the port. For replicated data objects, this scheme limits the propagation of synchronization control messages.

5.5.2.3 The Replicated Data Name Table Entry Each copy of a replicated data object that is managed by a gateway is referenced through a name table entry. The entry for replicated data is considerably larger than that of ports or gateways because of the version information that must be recorded. The format for the name table entry is:
TYPE data IS RECORD
  rectype: (gateway, port, data, group);
  name: string;
  alias: string;
  value: address;
  tempvalue: address;
  version: 1..2^32;
  portversion: ARRAY OF RECORD
    port: idport;
    version: 1..2^32;
  END
  groups: ARRAY OF RECORD
    permissions: modes;
    groupname: address;
  END;
  copies: ARRAY OF INTEGER.
END RECORD;

The "rectype", "name", "alias", and "groups" fields contain values similar to those described in the previous section. There are several additional fields: "version" contains the current version number, which can vary between 1 and 2^32. (An object could be updated once a second for 136 years before this counter will roll over to zero.) The "value" field contains a pointer to the value of the object stored in the gateway data space. The "tempvalue" field points to a temporary version of the object during an update operation. If the update succeeds, "tempvalue" will replace "value".

The "portversion" array contains one record for each process that is using this replicated object. The gateway uses this array to store the highest version number read by the process using the port. This information allows the gateway to reject transactions based on outdated values without creating network traffic. The "copies" field was described in the previous section.

5.5.2.4 The Group Name Table Entry

Group names provide a means for controlling access to replicated data or communication ports. The name table entry for a group name is:
TYPE group IS RECORD
  rectype: (gateway, port, data, group);
  name: string;
  alias: string;
  passwd: string;
  grpcde: internal_group_code;
  version: 1..2**32;
END RECORD;

The "rectype" field indicates that this is a group name table entry. The "name" field contains the system-wide group name, the "alias" field contains the local name, and the "grpcde" field contains an internal code that is used to uniquely identify the group name. The "passwd" field contains an encrypted password that the user process must know in order to belong to that group. Group entries are similar to replicated data entries in that modification of fields in the group entry must be synchronized with the other copies in the network. For this reason, group name table entries have a version number and the exact same mechanisms used to update data are also used to update group entries.

5.6 Information Sharing Through Groups

The intelligent message transport system allows users to selectively export names to particular sites or groups of users. A list of system-wide group names and passwords are maintained by each gateway. Usually, these types of protection functions are provided by an operating system kernel. In a distributed system, a unified protection scheme must be implemented by the message transport system or by one common operating system running at every site. The intent of the proposed system is to provide a network for interconnecting a variety of types of computing systems, including single board computers that are not controlled by the type of operating system that implements access protection. Therefore, we have added access protection features to the intelligent message transport system. The protection scheme is simple: users establish system-wide group names and then restrict access to replicated data or
ports by specifying the groups and their permissions when the data object is created. Members of the group with the appropriate permissions can modify the protection options on an entity at any time. The four access permissions are: *read*, *write*, *delete*, and *modify*. The four permissions and their corresponding effect on the four types of namable entities are summarized below.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Port Name</th>
<th>Data Name</th>
<th>Group Name</th>
<th>Gateway Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>N/A</td>
<td>read value</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Write</td>
<td>send message to</td>
<td>update value</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Delete</td>
<td>delete name</td>
<td>delete copy</td>
<td>delete group</td>
<td>N/A</td>
</tr>
<tr>
<td>Modify</td>
<td>permissions</td>
<td>permissions</td>
<td>permissions/password</td>
<td>N/A</td>
</tr>
</tbody>
</table>

There are four group code operations provided by the gateway:

- **MAKGRP** — create a group name, check its uniqueness, and export to selected sites
- **SETGRP** — allows a process to become a group member
- **MODGRP** — modify group permissions or passwords
- **DELGRP** — delete a group name

These are described below.

1. **Creating Group Names.** To create a system-wide group name, a user process sends the MAKGRP message to the gateway along with the name of the new group, the group password, and a list of groups and permissions of users that will be allowed to modify this group entry (e.g., the password). The gateway processor will query all gateways in the network to ensure that the group name is unique. If it is unique, the group name is entered into the symbol table and the user process is so notified; otherwise, the user is notified that another group name must be chosen.

2. **Using Group Names.** A user process establishes the groups that he belongs to by sending the gateway a SETGRP command message along with the group names and their passwords. If a group name is in the name table and the password is correct, all communication for that port will be tagged with the group name. If the group name is not in the name table, the gateway will
attempt to locate and copy the name and password from another gateway. The user will be notified if the name cannot be located, or the user-supplied password is incorrect.

(3) **Modifying Group Permissions.** A user process with sufficient permission to modify group permissions and passwords may do so with the MODGRP command. The gateway will then initiate an update request for that group name table entry using a method that is similar to the one used for updating replicated data objects. Because the group protection scheme is checked on every data or port access, a modification will take effect immediately at all sites. In this way, access privileges can be revoked at any time. (This is not the case in capability-based systems.)

(4) **Deleting Group Names.** A user process with sufficient permission can delete group names with the DELGRP gateway command. Only the local copy of the group name table entry is deleted, and only the process that originally created the group name can perform a system-wide purge of the name.

### 5.7 Alternate Name Mappings

As previously mentioned, many-to-one (aliases) and one-to-many (broadcast) name mappings are easy to achieve in the intelligent message transport system. When a process at the site requests that the gateway locate and copy a replicated data object, a local name (alias) can be specified. From that point onward, the site can reference the object with either the local name or the system-wide (global) name. This feature may be convenient to use when a site chooses to observe local naming conventions, such as those defined by an operating system.

Broadcast capability is also supported. When a process at a site transmits a message by way of a named communication port, every site that has opened a port by the same name will receive the message. In the proposed system, broadcast capability is
efficiently implemented by using the same reference count technique that limits the transmission of update control messages. In this way, broadcast messages do not actually broadcast throughout the network; instead, they are directly transmitted only to those sites that have permission to listen on that port.

5.8 Summary

In this chapter, we have developed a method for managing the system-wide name space in the proposed system. User processes are able to selectively export names to other sites or groups of users by using a small set of gateway processor functions. Because name management control functions and data storage are encapsulated in the gateway processor, sites can be heterogeneous with respect to both hardware and software. The intelligent message transport system provides a common naming structure for all sites in the network. It is likely that sites with sophisticated software systems (e.g. UNIX) will hide the internetwork naming conventions in a low-level driver, however, other less sophisticated sites (e.g. a single board computer) will be able to directly use the naming functions provided by the gateway processors.
6. THE INTELLIGENT MESSAGE TRANSPORT SYSTEM

6.1 Introduction

This chapter discusses the intelligent message transport system. The message transport system provides a super structure that combines heterogeneous computing facilities and networks into a unified system. It consists of the communications networks that interconnect sites and the gateway processors that provide for system-wide control functions as well as joining heterogeneous networks. The major functions of the intelligent message transport system include:

- Providing a means of interconnecting different types of communication networks. Messages that must travel through different networks will be packaged in accordance with the requirements of the host network. This reformatting of messages will be transparent to both source and destination sites.

- Providing a unified and consistent network interface to individual sites. The site interface hides all details about network concerns including protocols and message formats. It also provides a reliable means of communication with guaranteed delivery of messages (provided that the target site is accessible and on-line) even though the underlying network may support only probabilistic or "best-effort" delivery (e.g. the Ethernet).

- Providing a method of managing the system-wide name space in a way that allows users to specify the scope of reference for a name by controlling the sites that the name is exported to.

- Managing the details of ensuring the consistency of replicated distributed data in a way that is transparent to the user.

The remaining sections of this chapter will discuss the system topology and the gateway processor in more detail. Specific functions of the gateway processor will be
oulined, and a particular hardware implementation for the gateway will be offered.

6.2 System Topology

Many of the features that make the proposed system novel rely on the tree-like structure of the topology. Although we have called the structure "tree-like", it is actually a directed graph\(^1\) and not a tree. The distinction is that trees must have an identifiable root node (i.e. no predecessors), and must not contain cycles. We will continue to describe the system as being tree-like and use tree-related terminology because it is more familiar, and in this case, the distinction between a tree and a graph is not important.

The general structure of the system was shown in Figure 1-1b. The interior nodes in the tree represent gateway processors, the arcs are the interconnecting communication networks, and the terminal nodes are sites. The tree is organized so that there is a natural correspondence between a particular subtree and a geographic region. For example, a subtree might represent a local area network that serves a building, and a descendent of that subtree might serve a single laboratory. Eventually, through successive refinements, we will arrive at a terminal node of the tree that represents a site. The hierarchical structure blends well with the realization that networks serving a geographically small area (e.g. a building) generally provide a higher bandwidth than do long-haul networks. By organizing the system as a tree, we were able to create geographic regions where local messages are carried by a high bandwidth local area network. As we move up the tree (towards the root), the geographic area serviced by the system becomes broader. We believe that message traffic will exhibit a degree of *locality*, not unlike that exhibited by a string of memory references in a von Neumann computer. Thus, at the higher levels of the internet- work, message traffic is sparse and consists mostly of inter-region messages. It is the

\(^1\)A directed graph is often called a *mesh* in the network literature.
realization of the locality property for message traffic that suggested a hierarchical system topology. However, in practice, the path between communicating sites does not always fit well into a hierarchical organization. Occasionally, two sites that are not closely related in the hierarchy will have a need to carry on extensive communication. This type of anomaly is easily handled by constructing a cross-over (i.e. direct) link between two such sites. In effect, we have created a cycle in the directed graph representation of the internetwork. This is precisely the way that telephone networks are organized; most communication fits into a hierarchical structure, but occasional irregularities are patched with cross-over links.
In Figure 6-1, we show a distributed system that serves a small number of laboratories and offices. Sites are labeled with capital letters, and gateway processors are labeled with numbers. All gateways connect a site to the network, except 9 and 12, which act as "T" joints to interconnect three networks. In addition to providing an interface to sites, gateway processors are used to join different types of communication networks and may be inserted freely into the system without loss of performance. For example, gateways 9 and 12 could have been eliminated by joining gateways 8 and 10 directly to 11. Figure 6-2 shows the same system represented as a
graph. Gateways are uniquely labeled with symbolic names or numbers and paths 
are denoted by the gateway labels along the path. For example, one path from site $J$ 
to site $E$ would be 11-12-9-8-4-5. For consistency, we will always diagram the sys­
tem so that paths that ascend from leaf nodes (sites) towards the root correspond to a 
widening of the geographic area served by the system. There are many possible ways 
to draw the graph; it is only important to preserve the basic properties of the original 
topology, such as the distance between sites, measured in the number of interconnect­
ing networks and gateways. Also, it is not important which gateway we chose to be 
the root of the tree.

Once the system is represented as a graph, it becomes easier to estimate physical 
characteristics, such as communication delay times and message throughput. For 
example, a message traveling from site $J$ to site $I$ must pass through gateways 11,12,9 
and 10. Assuming a 1 millisecond delay for each gateway and a negligible delay for 
the interconnecting networks, the message will require 4 ms to travel from site $J$ to 
site $I$. Similarly, a message traveling from site $H$ to site $A$ will require 5 ms to travel 
the path 8-4-3-2-1, and 4 ms to travel the path 8-7-6-1. Of course, counting gateways 
will allow us to estimate only the minimum message delivery time. In an actual sys­
tem, we would need to consider the bandwidth of the interconnecting networks and 
the message load along those paths. As we saw in Chapter 5, the speed of the inter­
connecting is the limiting factor.

There has been much research in the area of internetwork organizations that 
focusing on minimizing communication delays [Davies, et al. 1979]. One of the 
approaches involves reshaping the system topology so that it is tree-structured and 
then transforming the tree so that it is low and broad, thereby reducing the number 
of paths between sites. As an example, consider the tree shown in Figure 6-3a. The 
average distance between sites can be reduced by transforming the tree so that it is 
broader, as shown in Figure 6-3b. If we carry this scheme further, we can derive the
Figure 6-3a. Original graph

Figure 6-3b. Reshaped graph

Figure 6-3c. Centralized system

Figure 6-3. Optimizing the intersite path length

\textit{N-ary} tree shown in Figure 6-3c. Although these transformations do reduce the average path length between sites, it is possible in some systems (like the proposed system) that throughput will actually decrease because of these transformations. The two main reasons for this are:

(1) We have seen that the proposed system is constructed from networks with varying bandwidth capabilities. This means that the paths in the tree representing networks are not interchangeable. For example, in Figure 6-3a site 3 is directly connected to site 6, however, in Figure 6-3b messages must
now be transferred through site 2. It could be the case that communication between sites 3 and 4 are critical to the stability of the Application Layer software (e.g. real-time data collection) and the introduction of an additional delay may cause the program to fail.

(2) As we reshape the structure of the topology to reduce path lengths, we negate the advantages of the local message domains created by the hierarchical structure. This is not surprising as the organization with the minimal path length in a configuration that is not fully-connected is simply a centralized system, shown in Figure 6-3c.

In summary, the optimization of system organization and message routing is an active area of research. Many techniques to shorten communication paths have been proposed, however, one must proceed with caution as seemingly sound algorithms may actually decrease throughput instead of improving it. The study of optimal system organizations is an example of research that needs to be investigated in the context of a real-life system because physical factors such as interconnecting network bandwidth, and the geography of sites must be accounted for in any proposed optimization algorithm. In the system proposed in this thesis, we do not offer an optimization algorithm, nor do we place constraints on the way in which the system topology is constructed. It is our belief that state-of-the-art communication networks provide sufficient bandwidth for our application and, therefore, do not feel the need to optimize either the system organization or message routing functions.

6.3 Intersite Communication

The gateway processors can be configured to be compatible with a variety of communication networks. Our intent is to use standard components to interface with common networks, like the Ethernet. We have purposely made no assumptions about the speed or reliability of the network, and we do not require a broadcast capability.
We assume only the ability to handle point-to-point communication of the type that could be supported by a simple direct *twisted-pair* connection\(^2\) between two sites. The gateway processors contain the software necessary to provide a high-level user interface, even if the underlying network is as simple as the twisted-pair.

### 6.3.1 Communication Modes

There are three modes of communication that are supported by the message transport system. They are:

1. Datagram (site-to-site or multicast)
2. Virtual Circuit
3. Transaction-oriented

The datagram mode of communication is appropriate to use when a process desires to send a *one-shot* message to another site. The message delivery is automatically acknowledged by the end-to-end protocol, however, the source process will generally not expect a reply. Multicast (broadcast) communication is directly supported by the gateway; the source site does not need to explicitly transmit the message to every site. The source will, however, receive an acknowledge from every site that received the broadcast message.

The virtual circuit mode of communication allows a pair of sites to exchange data without having to re-establish the communication path for each message. One of the sites must first locate the other site by requesting the "locate-site" gateway service, and then contact the other site using the address returned by the gateway. Because all messages that arrive at the destination site contain the path specification for the source site, both parties will then know each other's locations and will be able to communicate.

\(^2\)An example of a twisted-pair connection would be a RS232C connection between a terminal and a host computer.
The transaction-oriented mode of communication is a combination of the datagram and virtual circuit modes. As the intended application of the proposed system is to support distributed databases, we anticipate that this communication mode will be used most frequently. Transaction-oriented communication is used to support concurrent access to replicated data. Sites may read and update a local copy of a replicated object as if it were the only copy in the system because the gateways provide for system-wide synchronization in a way that is transparent to the source site. A site must first obtain a local copy of the data by requesting the "copy data" gateway service. From then on, application software running at the site may read the value of the object with the "access" gateway service request and update the object with the "update" service request.

These communication modes are just an abstraction that make the type of communication more natural to the application problem at hand. In some systems where global resource tables are consulted for every message routed, it is important to use the proper mode for performance reasons. In the intelligent message transport system, no global resource tables are used, and all messages are handled in the same way regardless of the communication mode used.

6.3.2 The Store-and-Forward Scheme

As discussed in Chapter 3, the intelligent message transport system routes messages using the store-and-forward technique. Store-and-forward means that the gateway extracts every message on the network, inspects the destination path, and either forwards the message onward to the next gateway, or to the attached site. In this scheme, message routing functions are completely controlled by the gateway software. As in most design decisions, there are tradeoffs involved in handling message routing in this way. The major disadvantage in using the store-and-forward scheme is that the message routing features provided by advanced networks cannot be utilized. For
these types of networks, our scheme will slightly reduce throughput and increase the delivery time for a message if more than two sites are involved in the transaction. For example, consider three sites interconnected by a broadcast network, like the Ethernet. (Label them 1, 2, and 3.) The most efficient way of delivering a message from site 1 to site 3 is to put the message onto the network where it will be broadcast to all accessible sites, and the site with the matching address will be able to extract the message from the network. In this case, the message traveled from source to destination in one hop. In the proposed system, gateway processors explicitly route messages, so it would be sent from site 1 to site 2, and then from site 2 to site 3, for a total of two hops. If the sites were interconnected by the Ethernet, then each hop would involve broadcasting the message to all sites connected to the network, which wastes some of the capacity of the network. It is not pleasing that the routing functions provided by certain high-level networks cannot be utilized, but it is simply a matter of tradeoffs.

We are willing to waste network capacity in order to satisfy the following objectives:

1. The store-and-forward technique provides the basis for the end-to-end acknowledge protocol that is used by a communicating pair to determine whether a message has been successfully received at the destination site. We felt that guaranteed delivery of messages was worth a decrease in throughput.

2. One stated objective was to accommodate all types of networks, including those that are less sophisticated than the Ethernet. Our "base" network is the simple twisted-pair which lacks a standard protocol and routing functions.

3. In order to lower maintenance and development costs, it is essential that the control logic implemented in software in the gateway processor be identical for every gateway processor in the system. This means that we cannot assume that any network has more intelligence than the simple serial connection. Although each different type of network will interface to the gateway processor by a different hardware and software interface, high-level functions such
as message routing and control of the name space must remain the same.

(4) A major objective in designing the intelligent message transport system was to enhance the autonomy and reliability of sites by alleviating the need for system-wide name servers and resource tables. The implication of not having a global resource table is two fold: (1) the path between gateways must be discovered dynamically, and (2) no gateway has knowledge of the location of other gateways in the system, except for those that it is directly connected to. For example, consider the system shown in Figure 6-1. A message traveling from gateway 6 to gateway 3 may use the path 6-1-2-3. As we have seen, the message will be transmitted to gateway 1, then onto gateway 2, and finally to gateway 3. It may be the case that all four gateways are interconnected by a broadcast network, in which case the message could be directly transmitted to gateway 3 in one hop. The problem is that while gateway 6 knows that the message path is 6-1-2-3, it has no knowledge of where gateway 3 is located, or even if gateway 3 is on the same network; it only knows that it is connected to gateway 1 and that the message should be forwarded to gateway 1 for subsequent processing. Here again, the objective (in this case autonomy) was achievable only if we were willing to accept lower throughput.

In summary, the store-and-forward technique for message propagation has many advantages, including the ability to guarantee message delivery, and the ability to combine a wide range of networks into one internetwork transport system that provides a single, consistent site interface. The disadvantage to this approach is that some network capacity will be wasted, however, we believe that contemporary local area networks will provide a sufficient bandwidth for good throughput, even with these modifications.
6.3.3 The End-to-End Acknowledge Protocol

One of the major advantages of the intelligent message transport system is its ability to guarantee the delivery of a message if the destination site is on-line and accessible. More importantly, the source site will always receive a response message regardless of whether the original message was deliverable. This feature was built into the message transport system to provide guaranteed delivery of messages even though one of the underlying networks may use a probabilistic delivery algorithm (e.g., the Ethernet) and cannot guarantee delivery. The ability to provide this function is a consequence of the combination of the store-and-forward scheme used to propagate messages through the network, and a low-level handshake protocol that is used for intergateway messages. As an example, consider a message traveling from gateway 1 to gateway 3 by way of gateway 2 (Figure 6-4). Recall that the message is actually forwarded by two independent transmissions, namely, gateway 1 to gateway 2 and gateway 2 to gateway 3. Each intergateway transmission is immediately

![Diagram of the end-to-end acknowledge protocol](image-url)

Figure 6-4. The end-to-end acknowledge protocol
acknowledged on receipt of the message. Thus, we have two levels of handshaking: at the highest level, the source site sends the message to the destination site and receives an acknowledge that the message has been successfully delivered (this is shown in the bottom diagram in Figure 6-4); at the gateway level, each hop of the message is acknowledged by a low-level handshaking signal. If the sending gateway fails to receive the low-level acknowledge in a short period of time, it will retransmit the message up to some fixed number of times. (The sequence number on messages makes them idempotent.) If subsequent retransmissions of the message continue to fail, the gateway will eventually purge the message and transmit a not-acknowledge to the original source site.

6.3.4 An Example of Intersite Communication

In this section, we will demonstrate how the store-and-forward technique is used with the end-to-end acknowledge in a fairly complex example. Assume that we are working with the system shown in Figure 6-5. Our viewpoint will be that of a user at site D. We would like to find a copy of a particular object that is located at some unknown site in the system (for our example, it is at site A). At our command, gateway 5 will initiate action that will search its own name table as well as the name tables of all accessible gateway processors in the system. We will eventually receive a message from gateway 5 that gives us the path to a copy of the object, or indicates that no such object is accessible. The message that we will trace through the system is the gateway "locate data" message described in Section 6.4.4.

The first step is for site D to send the locate command to gateway 5. The gateway receives the message and sends the RECVD acknowledge back to site D. The locate message is then encoded in INP (Section 6.4.3.2) and sent to adjacent gateways 6 and 3. As always, these gateways return the RECVD acknowledge to gateway 5. For the remainder of this example, we will assume that messages are correctly received
Figure 6-5. Intersite communication example

and will not explicitly mention the RECVD acknowledge. The locate message in
gateway 5 will be added to a list of messages that are waiting for a reply. With this
particular command, a gateway will return a not-acknowledge only if all adjacent
gateways indicate that a copy cannot be found. If one or more copies are located, the
gateway will chose the shortest path (stored in the source path slot of the message)
and forward this information to the originator of the locate command. For the
moment, we will concentrate on the message traveling from gateway 5 to gateway 3.
On arrival at gateway 3, the locate message is retransmitted to gateways 4 and 1, and
it is added to the list of messages awaiting a reply at gateway 3. At gateway 4, the
message is transmitted to gateways 6 and 2. The message sent to gateway 6 will
eventually be sent to gateway 5 where the cycle will be detected, the message will be
purged, and a not-acknowledge will be sent back to gateway 4. Gateway 2 will
respond with an acknowledge because site A has the desired object. Thus, gateway 4
will receive one acknowledge and one not-acknowledge, and will pass the ack-
knowledge back to gateway 3. Gateway 3 will eventually receive an acknowledge from the message sent to gateway 2 through gateway 1, and will forward an acknowledge back to gateway 5 arbitrarily choosing either the 3-1-2 or the 3-4-2 path. Gateway 5 will wait for a reply from the original message sent to gateway 6. Note that the path forked at gateway 4 and gateway 4 would chose the 4-2 path over the 3-1-2 path because it is shorter. Eventually, gateway 5 will receive both acknowledges and will decide between the 5-6-4-2 path from gateway 6, and either the 5-3-4-2 or the 5-3-1-2 paths from gateway 3.

6.4 The Gateway Processor

6.4.1 Introduction

The gateway processors are the "glue" that hold the system together. In addition to providing a consistent, intelligent network interface, the gateway provides high-level functions that directly support the naming and synchronization methods developed in Chapters 4 and 5. It is important to note that we have chosen to provide these functions in a component that is physically separated from the site. The decision to move interfacing functions to an autonomous processor provides the basis for a consistent intersite interface. The utility of this approach becomes apparent when the tree-structured topology of the system is factored in; it is precisely the combination of these two features that facilitates and efficient, resilient and consistent method of providing synchronization and naming for a loosely-coupled collection of heterogeneous processors.

6.4.2 The Architecture of the Gateway Processor

In this section, we will show specific hardware configurations for the gateway processor using standard off the shelf components. Readers will note that the choice of equipment is limited to products manufactured by Intel Corporation. Our intent is
to demonstrate that the intelligent message transport system can be easily constructed from standard readily-available equipment. We do not imply that Intel is the only manufacturer of communication components, nor do we imply that Intel products are better than those of other manufacturers.

A block diagram of the gateway processor is shown in Figure 6-6. The left side of the diagram depicts the path that an outgoing message follows. First, the message is delayed at the output queue until the output pipeline is ready to process it. At that time, the message is encoded in the INP format and sent to the network-specific handler which will wrap the message in the host network's protocol package. The destination and sequence number of the message are appended onto the list of transmitted messages that are waiting for an acknowledge. The right side of the diagram depicts the various checks that an incoming message is subjected to. It is checked for cycles and purged if the message is cycling. A not-acknowledge will cause the retransmission of the message that failed. An acknowledge will remove the message from the acknowledge list. The message is then queued for processing by the gateway control logic software. This software controls the name table and data storage functions, as well as routing message traffic. At the top of the diagram, the serial interface connects the site to the gateway. All messages that pass through this port are encoded in the ISP format.

The structure of the gateway processor is shown in Figure 6-7. Although every gateway processor provides a standard hardware and functional interface, it may be implemented by either of the two designs shown, depending on the types of networking capabilities required.

In Figure 6-7a, we show the simplest type of gateway processor. It can be used to connect a site to the Ethernet local area network. If it is desirable to use a different local area network, then the configuration shown in Figure 6-7b can be used. We believe that installations of the Ethernet are common enough that the design of a
Figure 6-6. A block diagram of the gateway processor
specialized gateway for Ethernet applications is warranted. The simplified gateway consists of an Intel iSBC 186/51 COMMputer board with an Intel iSBC 304 add-on memory board. The iSBC 186/51 shown in Figure 6-8 is described in the Intel docu-
The iSBC 186/51 Ethernet controller (from [Intel 1984])

mentation [Intel 1984]. It may be used as a complete stand-alone computer, or as a MULTIBUS-compatible board for use in a multimaster system environment. The Ethernet interface is provided by the on-board 82586 Local Communications Controller (LCC) and the processing power is provided by an on-board iAPX 186 (80186) 16-bit microprocessor operating at 8 MHz. Additionally, there are two serial ports based on the 8274 Multi-Protocol Serial Controller configured so that one port can support either RS232C or RS422A modes of communication, and the other port RS232C. There are 128K bytes of dual port read-write memory on-board that can be
externally expanded to 256K bytes with the iSBC 304 board. The memory can be
configured so that it can be shared with any attached MULTIBUS devices, thus facili­
tating DMA transfers of networked information. The 82586 LCC will, upon com­
mand from the 80186 CPU, read the specified buffer from memory, package the data
in the appropriate Ethernet wrappings, and transmit the data to the destination site.
Conversely, the 82586 LCC is able to read and unwrap packets from the Ethernet
and deposit them into memory without the assistance of the CPU. These and other
functions are directed by the 80130 on-board firmware. The operating system pro­
vides support for all Ethernet functions up through the Transport Layer of the OSI
Reference Model. Additionally, Application Layer software can be placed into on­
board read-only memory. In the simplified gateway processor, we envision naming
and synchronization functions to be placed in the read-only memory on the iSBC
186/51 board. With the iSBC 304 add-on memory board, there should be sufficient
read-write memory available to construct the site name tables.

In Figure 6-6b, we show a general gateway processor. The general gateway
processor organization can be used when networks other than the Ethernet must be
supported, when more than one network is joined by the gateway, or if special func­
tions are to be implemented, such as the ability to log message traffic. It is purposely
organized as a bus-oriented von Neumann computer so that it can be built from stan­
dard off-the-shelf components. This particular configuration is organized around the
industry-standard Intel MULTIBUS and includes network interfaces, a CPU, the site
interface, and a disk memory system that can be used to log messages. As in the
simplified gateway processor, the Ethernet interfaces (if needed) are provided by the
iSBC 186/51 COMMputer interface board. The difference in this configuration is that
the iSBC 186/51 will operate in a multimaster environment, rather than as a stand­
alone computer. Up to three Ethernet interfaces may be connected to the MULTIBUS.
Intel also supports other types of networks with single board products, such as the
8292 family of GPIB interfaces, and the 8273 HDLC/SDLC protocol controller. These
and other types of network interfaces may be used in the general gateway processor either with, or instead of the Ethernet controller. A portion of the on-board read-write memory will be mapped into the system address space provided by another MULTIBUS memory board. Thus, the 186/51 will continue to handle the Ethernet protocols, but the input will be read from an off-board memory space that is controlled by the system CPU; similarly, information read from the Ethernet will be stored in the off-board memory. The control for the gateway functions will be provided by a general purpose MULTIBUS processor, such as the iAPX 186, or the iAPX 8086. Either of these processors are compatible with the iSBC 186/51 CPU and will execute the Application Layer software that was developed for the Ethernet controller. An optional 82062 Winchester disk controller (compatible with ST506/ST412 disk interfaces) may be used to log message traffic for the purpose of off-line analysis of loading and other factors. As in the simple gateway processor configuration, a standard serial interface provides the connection between the gateway and the site.

6.4.3 Gateway Message Formats

As we have previously mentioned, the gateway processor provides a standard interface to a site in terms of both the high-level network functions provided, and in the protocol for messages that request those functions. Similarly, there is a standard protocol that is used to encode intergateway messages. These messages are self-contained packets of information and will be graphically depicted with the following structure:

<table>
<thead>
<tr>
<th>Field A</th>
<th>Field B</th>
<th>Field C</th>
<th>Field D</th>
<th>Field E</th>
</tr>
</thead>
</table>

All internetwork message packets will have the above structure. During the actual transmission process, the packet may be restructured as needed. For example, if the
interconnecting network used a synchronous time division multiplex scheme to share the network (like the DATANET [Davis and Pohm 1982, Pohm 1980]), then each packet would be transmitted byte-serially rather than one packet at a time. In this case, the packet would be reassembled on the receiving end and the gateway control software would again be working with a standard packet format.

In some of the message examples in this chapter, certain fields in the fixed-field message are not applicable to the example. In this case, the field is marked as "n/a". If the field will contain a value, but the value is unknown (e.g., a port number), then the field will be marked with "?".

We have invented two protocols that are used in communications with gateways. The independent network protocol (INP) is the standard message format for intergateway messages. Because the host network most likely has a format for messages, the INP messages are transmitted by inserting them into the data field of the host network protocol. The receiving gateway will extract the data field and present the message in INP format to higher levels of the gateway control logic. The other message protocol is the independent site protocol (ISP) and is used for messages that pass between the gateway and the attached site. Both of these protocols are described in detail in the following sections.

6.4.3.1 The Independent Site Protocol (ISP) The ISP specifies the format for messages that pass between a gateway and the attached site. It is purposely designed to be simple so that a site does not require much processing power to generate or interpret messages. The format for ISP messages is:

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Destination Path</th>
<th>Message ID</th>
<th>Process Port</th>
<th>Data Length</th>
<th>Data</th>
<th>CRC</th>
</tr>
</thead>
</table>

The fields for the ISP format are described below.
A. Function Code.

The function code identifies the type of the message, or request a particular service from the gateway processor. The codes that a site can send to the gateway are:

1. CPYDATA — locate and copy specified data
2. LOCSITE — locate a particular site in the network
3. UPDATE — update the specified replicated data
4. XMTMSG — send a message to the specified site
5. DELETE — delete specified name from gateway name table
6. ONLINE — the site is on-line (sent after a crash)
7. ACK — request successfully serviced
8. NACK — request couldn’t be serviced

The group codes described in Chapter 5:
9. MAKGRP — establish group name
10. SETGRP — set group name
11. MODGRP — change group permissions
12. DELGRP — delete group name

The codes that the gateway can send to a site or another gateway are:

13. ACK — request successfully serviced
14. NACK — request couldn’t be serviced
15. RECVD — message received
16. UP — the gateway is on-line (sent after a crash)

Not all of the fields are relevant to all function codes. The specific details of each function code will be explained in the following sections of this chapter.

B. Destination Path.

The destination path is the path that this particular transaction should take to arrive at the destination. The appropriate path to another site is discovered by the LOCSITE command. A “zero” destination path means that the message should be broadcast to every accessible site.

C. Process Port.

The process port is an abstraction that allows the site to multiplex the connection from the site to the gateway. Ports are simply channel numbers that are allocated by software in the gateway to processes that need to communicate with the gateway. Reply messages from the gateway will be directed towards the process that initiated the request through the specified port. In the case
where a site consists of a single computer dedicated to one task, all communication with the gateway can take place using the same port number. Port 0 is assumed by the gateway to be attached to a process that listens for urgent messages, such as a gateway restart after a crash. Port 0 is also initially used by processes to obtain a port number.

D. Message Id.

The message identification number is used to match request messages with response messages. In the case of gateway-to-site messages, every request presented to the gateway must have a message number that is unique (usually monotonically increasing) to that port. The reply message from the gateway will have the same message number as the original request. This scheme avoids the confusion that would result if replies were to arrive out of order. In gateway-to-gateway communications, the message identification number is generated by the source gateway (usually monotonically increasing) and is also used to match messages with replies.

E. Data Length.

The data length field specifies the number of bytes that is contained in the data field.

F. Data.

The data field contains the actual message data. It is organized as a sequence of 8-bit bytes which do not need to be encoded in any particular format. Note that the assumption that bytes contain 8 bits is natural for nearly all computers, but may present a conversion problem for the few machines that have a word size that is not a power of 2.

G. Cyclic Redundancy Code (CRC).

It is debatable whether there needs to be a CRC appended onto messages that pass between the site and the gateway. In practice, the distance between them
will be short, and the interface will be reliable. We include the CRC only for completeness. It is our intent that the CRC be very easy to compute, such as an 8-bit checksum generated by summing the bytes in the message.

6.4.3.2 The Independent Network Protocol (INP) The INP format is used to code messages that pass between gateway processors. It is similar to ISP, but contains a few additional fields that apply to networked messages. The format is:

<table>
<thead>
<tr>
<th>Function Code</th>
<th>Destination Path</th>
<th>Source Path</th>
<th>Sequence Number</th>
<th>Message ID</th>
<th>Process Port</th>
<th>Data Length</th>
<th>Data</th>
<th>CRC</th>
</tr>
</thead>
</table>

The additional fields in INP are:

A. Source Path.

When the message arrives at the destination site, the source path field will contain the path back to the source site. Each gateway preappends its identification number onto the head of the source path list.

B. Sequence Number.

The sequence number is used to identify the ordering of partial messages in the event that one large message must be fragmented into smaller messages. This will occur when the message to be transmitted is larger that the host network is capable of transmitting. The destination site uses the sequence number to reconstruct the original message. The sequence numbers begin at 1 and are monotonically increasing. The sequence number of the last packet is negated.

C. Cyclic Redundancy Code (CRC).

The CRC for intergateway messages will be generated using a standard method, such as those described by Peterson and Brown [1961]. For example, Peterson notes that a particular cyclic code having 22 check bits can encode 22,495 message bits in a way that detects any odd bit failures, two burst
errors of length 12 or less, a burst error of length 22 or less, and 99% of the
burst errors of length \( \geq 23 \). We anticipate using a long CRC with the intent
of detecting a many types of errors as possible. In some cases, the CRC will be
redundant as the protocol of the host network may provide CRC checks; how­
ever, we must provide the CRC because of our decision to make no assump­
tions about the various protocols (or lack of) of the interconnecting networks.

6.4.4 High-Level Gateway Functions

The gateway processor provides high-level functions that can be divided into
two groups, namely, those functions that support the site interface, and internal func­
tions that facilitate communication with other gateways. These are listed below and
discussed in detail in this section, except for naming and synchronization which are
discussed in Chapters 5 and 6, respectively.

Gateway-to-Gateway Interface Functions:

1. Gateway power-up sequence
2. Format messages in INP
3. Manage end-to-end acknowledge protocol
4. Error checking
5. Store and forward messages (check for cycles)

Gateway-to-Site Interface Support:

1. Site power-up sequence
2. Locate and copy the named data
3. Update the named data

6.4.4.1 Gateway-to-Gateway Interface Functions

The gateway-to-
gateway functions provided by the gateway processors are listed below.


When the gateway powers up, it performs a number of book keeping tasks prior
to establishing communication with the site or adjacent gateways. These include
determining its own unique identification number and name (stored in read/only memory), and clearing the name table and other internal data structures. Next, communication is established with each gateway connected by transmitting a message that announces the identification number of this gateway and requests the identification number of the receiver. The format is:

<table>
<thead>
<tr>
<th>funct</th>
<th>dest</th>
<th>src</th>
<th>seq</th>
<th>mid</th>
<th>port</th>
<th>len</th>
<th>data</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>UP</td>
<td>n/a</td>
<td>our id</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>0</td>
<td>n/a</td>
<td>?</td>
</tr>
</tbody>
</table>

The reply message format is:

<table>
<thead>
<tr>
<th>funct</th>
<th>dest</th>
<th>src</th>
<th>seq</th>
<th>mid</th>
<th>port</th>
<th>len</th>
<th>data</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>our id</td>
<td>their id</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>0</td>
<td>n/a</td>
<td>?</td>
</tr>
</tbody>
</table>

In a similar fashion, communication is established with the site by transmitting the following message:

<table>
<thead>
<tr>
<th>funct</th>
<th>dest</th>
<th>id</th>
<th>port</th>
<th>len</th>
<th>data</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONLINE</td>
<td>n/a</td>
<td>n/a</td>
<td>0</td>
<td>0</td>
<td>n/a</td>
<td>?</td>
</tr>
</tbody>
</table>

If the site responds with an acknowledge, then the power-up sequence is complete. If for some reason the site responds with a not-acknowledge, then the gateway will process internetwork traffic, but will reject any messages whose destination is the down site. At some later point in time, the site can come online by sending the gateway an ONLINE message.

2. Formatting Messages in INP.

All messages that pass between gateways are coded in the INP format. The network-specific software driver will package the INP message in the appropriate protocol for the host network. This is accomplished by simply treating the INP message as the data portion of the host network's protocol. Unfortunately, some network protocols have a limit on the size of the data field in a message. Because of this restriction, the INP coder in the gateway works with the network-specific driver to fragment the data portion of the INP message into pieces that are small enough to transmit. Each message piece is assigned a
sequence number, starting at 1, that will be used by the receiving gateway to reconstruct the message. For example, if we wanted to transmit a file of length 3*N to gateway 4 on path 12-2-4, the three fragment would look like:

<table>
<thead>
<tr>
<th>funct</th>
<th>dest</th>
<th>src</th>
<th>seq</th>
<th>mid</th>
<th>port</th>
<th>len</th>
<th>data</th>
<th>crc</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMTMSG</td>
<td>12-2-4</td>
<td>5</td>
<td>1</td>
<td>?</td>
<td>?</td>
<td>n</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>XMTMSG</td>
<td>12-2-4</td>
<td>5</td>
<td>2</td>
<td>?</td>
<td>?</td>
<td>n</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>XMTMSG</td>
<td>12-2-4</td>
<td>5</td>
<td>-3</td>
<td>?</td>
<td>?</td>
<td>n</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

The receiving gateway knows how many fragments to wait for because the largest sequence number is negative. This scheme allows the fragments to arrive at the destination in any order. Also, note that the sequence number makes messages idempotent; that is, they may be accidently transmitted twice without harmful effects.

3. **Manage End-to-End Acknowledge Protocol.**

The end-to-end acknowledge protocol was described in Section 6.3.3. It is used by a communicating pair to determine if a message has been successfully received at the destination site. This feature was built into the message transport system to provide guaranteed delivery of messages even though one of the underlying networks uses probabilistic delivery (e.g. Ethernet). This is easy to achieve because of the design decision to use the store-and-forward message delivery technique. When a message is transmitted to an adjacent gateway, the message is added to a list of messages that are waiting for an acknowledge from the adjacent gateway. Because the acknowledge is normally returned in a very short time, there should never be more than a few messages waiting. When the acknowledge arrives, the message is deleted from the gateway. If a waiting period expires before an acknowledge arrives, or if a not-acknowledge is
received, the message will be retransmitted. Continued failure means that the destination gateway is off-line and delivery of the message is not possible. In this case, a not-acknowledge is sent to the original source site and would be presumably handled by software in the site Application Layer.


Every message is checked for CRC errors at every gateway that the message passes through. When an error is detected, the message is purged. We do not bother to send a not-acknowledge message to the source (the source path may be corrupted anyway); the source site will timeout and retransmit the message.


The store-and-forward mechanism has been described previously. The scheme is to read the message from the network, possibly reconstructing it from fragments, and check for CRC errors. If the message is destined for the attached site, then it is reformated into the ISP format and dispatched to the site interface. Otherwise, the message is checked for cycles and forwarded onto the next gateway. The cycle test determines if the message has visited this gateway before. If it has, then the message is purged and a not-acknowledge message is sent to the original source site. For example, consider the following message encoded in the INP format:

<table>
<thead>
<tr>
<th>funct</th>
<th>dest</th>
<th>src</th>
<th>seq</th>
<th>mid</th>
<th>port</th>
<th>len</th>
<th>data</th>
<th>crc</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMTMSG</td>
<td>10-1-9-10-7</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

The originator is gateway 3, and the eventual destination gateway is 7. Assume that we are gateway 10. Each time this message passes through the named gateway, the gateway number is put on the head of the source path list, and it is removed from the head of the destination path list. After passing through gateway 10, the message is:
After passing through gateway 1:

<table>
<thead>
<tr>
<th>funct</th>
<th>dest</th>
<th>src</th>
<th>seq</th>
<th>mid</th>
<th>port</th>
<th>len</th>
<th>data</th>
<th>crc</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMTMSG</td>
<td>1-9-10-7</td>
<td>10-3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

After passing through gateway 9:

<table>
<thead>
<tr>
<th>funct</th>
<th>dest</th>
<th>src</th>
<th>seq</th>
<th>mid</th>
<th>port</th>
<th>len</th>
<th>data</th>
<th>crc</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMTMSG</td>
<td>10-7</td>
<td>9-1-10-3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

The message has returned to gateway 10. The gateway can detect a cycle by checking the source path to see if this message has visited here already. If the message is cycling, then the gateway will purge it and send a not-acknowledge message to the original source gateway (3 in this example). This type of cycling can occur because (1) sites provide the destination path which could be in error, or (2) broadcast messages will always cycle because the destination path is not specified.

6.4.4.2 Gateway-to-Site Interface Functions The gateway-to-site functions provided by the gateway processors are listed below.


   When a site is powered up, it sends the ONLINE message to the gateway, as shown below.

<table>
<thead>
<tr>
<th>funct</th>
<th>dest</th>
<th>id</th>
<th>port</th>
<th>len</th>
<th>data</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONLINE</td>
<td>n/a</td>
<td>n/a</td>
<td>0</td>
<td>0</td>
<td>n/a</td>
<td>7</td>
</tr>
</tbody>
</table>

   The gateway will immediately reply with an acknowledge on port 0. The next step is to carefully update the name table so that system-wide consistency is maintained with respect to replicated data. If the gateway has also crashed, then the name table will be cleared by the gateway power-up sequence. Otherwise, the name table will contain the names of the replicated data that the site
was using at the time of the crash. Each name descriptor will be inspected, and if the number of system-wide copies is greater than 1, then the data can be discarded because there is a current copy stored elsewhere. If the copy count is exactly 1 then this is the only copy in the system and we do not need to worry about consistency. In the event that more than one gateway has crashed and the power-up sequence has erased all copies of the data, then the data must be recreated from a gateway that maintains a transaction log.

2. Locate and Copy the Named Data.
   This gateway service creates a local copy of the named data. This is essentially a two-step process. In the first step, the gateway processor broadcasts a LOCDATA message that contains the name of the object to be copied along with the group codes for the requesting process. The gateway will eventually receive one response from each connected network. The response consists of either a path to a gateway that maintains a copy of the data object, or a not-acknowledge that indicates the data could not be found (or access was denied). If the data were found, the gateway then sends out a CPYDATA message that directs the remote site to transmit a copy of the data. As the copy travels back to this gateway, name table entries are created for the name at each gateway that the message passes through. If the name already exists, a reference counter field is incremented. At any point in time, the reference counters indicate the number of copies of a particular data object that are located in any subtree of the system.

3. Update the Named Data
   This gateway service coordinates the synchronization mechanism used to update a replicated data object, as described in Chapter 4. The user process initiates the update by presenting the update message to the gateway. The update message contains the name of the data object to be updated, and the names of other data objects that were used to calculate the replacement value. The gateway processes-
The gateway will check the version numbers on each name presented, and if any are outdated, the transaction will be rejected. If the version numbers are valid, the gateway will transmit an update message to each site that maintains a copy of the data. (The update message is simply broadcast to all gateways along paths that have a reference count greater than zero for the named object.) If all participating gateways acknowledge the update, then the coordinating gateway sends a commit message that instructs participating gateways to purge the outdated value and to make the new value accessible.

6.5 Summary

This chapter described the intelligent message transport system. The message transport system can be constructed from standard off-the-shelf components. The few software control functions provided by the gateway processor can be easily implemented. We expect that the controlling software could be coded in a high-level language because they are not time-critical in relation to the interconnecting networks. We have shown that these simple functions provided by the gateway processor are very powerful in that they simplify some of the inherent problems in distributed system, namely synchronization and naming.
7. CONCLUSIONS

7.1 Research Summary

In this thesis we have developed a distributed computer architecture that supports loosely-coupled computing resources interconnected by different types of communication networks. We have named the system the *intelligent message transport system*. Both datagram and virtual circuit communication is supported.

The structure of the system is hierarchical with the inner layers corresponding to small geographic areas interconnected by local area networks. The outer layers correspond to geographically wider areas of service interconnected by slower speed communication devices, such as standard telephone equipment. The intelligent message transport system uses the hierarchical system topology to limit the transmission of broadcast control messages to those portions of the system participating in the transaction. In this way, throughput is improved because the slower outer layer of networks is not involved unless necessary.

Networks are joined to each other and to computing resources (sites) by network controllers called *gateway processors*. The gateway processors handle protocol conversions between the different types of interconnecting networks by "wrapping" messages in the appropriate format for the host network. The message transport system encodes all internetwork messages in its own format called the *independent network protocol* (INP). If the underlying network provides a protocol, such as X.25, the INP messages are simply inserted into the data portion of the host protocol format.

The INP format provides a suitable stand-alone protocol for networks that do not provide a protocol, such as a simple twisted-pair mode of communication. The INP provides and end-to-end acknowledge that always notifies the message source site of the transmission outcome. This feature guarantees the delivery of messages to accessible sites even though one of the underlying networks may provide only a "best-effort"
Gateway processors provide a simple and consistent interface to computing sites. All networking details concerning message routing, error detection, and other network concerns, are managed by the gateway processor. Sites simply present a request for a particular service (e.g. "contact site X") and will eventually receive notification of the results of the requested service. All interaction between sites and gateways is encoded in a gateway format called the *independent site protocol* (ISP). The ISP format is a simple ASCII command format that provides a consistent gateway-to-site interface. No assumptions have been made about the relative sophistication of sites. A site is required only to have the capability to communicate with the gateway via a serial port, and to encode/decode messages in the ISP format - a task that can be performed by even the simplest of single board computers.

One of the more novel features of the intelligent message transport system is that the task of preserving the consistency of replicated data is provided by the gateway processor at the Transport Layer rather than by user processes at the Application Layer. In a similar fashion, the gateway processors also manage the system-wide name space for global data. Consistency of replicated data is preserved by synchronizing updates using a method that orders transactions in time, based on data object version numbers. Transactions involving temporarily outdated data (because of communication delays during the update process) are rejected.

The gateway processor also provides for the management of the system-wide name space. All gateways and global data objects are referenced by symbolic names. User processes at sites may create named objects and make them known to specific sites or groups of users elsewhere in the system. The group name is the mechanism by which access to global data objects is controlled. Group names are also managed by the gateway processor and may be created by user processes at any site.
In summary, the intelligent message transport system provides a simple and consistent interface to an internetwork system. Certain high-level services usually provided by the Application Layer have been moved to the Transport Layer and are implemented by the internetwork system. The expected result is a simpler, more reliable networking system that can easily be built from standard "off-the-shelf" components.

7.2 Further Research

The merits of the intelligent message transport system have been repeatedly stated in this thesis. However, there are aspects of the final system that turned out to be somewhat disappointing and need to be researched further.

The lack of some minimum level of sophistication for sites had a negative impact on the simplicity of the gateway processor design. The management of the system-wide name space may be more complicated as a result. Specifically, our attempt to allow selective exportation of names to particular groups of users is complicated, and the notion of user processes having to register group names is not pleasing. This type of access protection feature could be provided by the on-site operating system, however, that would require each site to use the same software which violates our goal of supporting heterogeneous computing resources. It is not clear where (or even if) access protection should be provided in an internetwork system.

A related issue is the management of synchronization of replicated data by the gateway processor. To provide synchronization, we tag data with version numbers. The control of these version numbers must be provided by the gateway, otherwise user processes could easily specify bogus version numbers that could damage the system's ability to reject erroneous transactions. The undesirable consequence of storing version numbers in the gateway is that replicated data must also be stored in the gateway, otherwise there would be no way to guarantee a correspondence between a
version number and a particular value of the data object. The problem is that large amounts of replicated data might require gateway processors to be equipped with secondary storage devices, such as a Winchester disk. The result of adding a disk would be to lower the overall reliability of the gateway processor and, thus, the intelligent message transport system.

We believe that these problems should be studied further by constructing a small test system. This way, we could investigate performance and other aspects of the system by implementing a real-world problem.
8. REFERENCES

Abraham, Steven M., and Yogen K. Dalal

Aho, Alfred V., John E. Hopcroft, and Jeffrey D. Ullman
1975 "The Design and Analysis of Computer Algorithms."
Addison-Wesley Publishing Company, Reading, Massachusetts.

Bernstein, Philip A., James B. Rothnie, Jr., Nathan Goodman, and
Christos A. Papadimitriou
1978 "The Concurrency Control Mechanism of SDD-1: A System for
Distributed Databases (The Fully Redundant Case)."
1981 "Concurrency Control in Distributed Database Systems."
Computing Surveys 13, No. 2: 185-221.

Bertine, H.V.
1980 "Physical Level Protocols." IEEE Transactions on
Communications COM-28, No. 4: 433-444.

Boggs, David R., John F. Shoch, Edward A. Taft, and Robert M. Metcalfe
1980 "Pup: An Internetwork Architecture."
IEEE Transactions on Communications COM-28, No. 4: 612-623.

Brereton, Pearl
1983 "Detection and Resolution of Inconsistencies Among Distributed
Replicates of Files." ACM SIGOPS 17, No.1: 10-15.

Chu, Wesley W., and George Gardarin
1979 "A Reliable Distributed Control Algorithm for Updating
Replicated Databases." Proceedings of the Sixth Data
Communication Symposium: 42-51.

Clark, David D., and Liba Svobodova
1980 "Design of Distributed Systems Supporting Local Autonomy."
Proceedings of the IEEE Spring COMPCON, San Francisco,
California: 438-444.

Comer, Douglas
1983 "A Computer Science Research Network CSNET: A History and

Curtis, Ronald, and Larry Wittie
1984 "Global Naming in Distributed Systems."
IEEE Transactions on Software 1, No. 3: 76-80.
Davies, D.W., D.L-A. Barber, W.L. Price, and C.M. Solomonides
1979 "Computer Networks and Their Protocols.”

Davis, James A., and Arthur V. Pohm
1982 "A Local Network for Experiment Support.”

Digital Equipment Corporation
Digital Equipment Corporation, Maynard, MA.

Ellis, Clarence A.
1977 "Consistency and Correctness of Duplicate Database Systems.”

Enslow, Philip H. Jr.
1978 "What is a Distributed Data Processing System?.”

Eswaran, K.P., J.N. Gray, R.A. Lorie, and LL. Traiger

Green, Paul E., Jr.
"An Introduction to Network Architectures and Protocols.”
IEEE Transactions on Communications COM-28, No. 4: 413-424.

Intel Corporation
Intel Corporation, Santa Clara, California.
1984 "iSBc 186/51 COMMputer Board Hardware Reference Manual.”
Number 122136-001, Intel Corporation, Santa Clara, California.

Lamport, Leslie
1978 "Time, Clocks, and the Ordering of Events in a Distributed System.”
Communications of the ACM 21, No. 7: 558-565.

Lampson, B.W., M. Paul, HJ. Siegert, editors
1981 "Distributed Systems - Architecture and Implementation.”
Lecture Notes in Computer Science 105, Springer-Verlag, New York, NY.

Lee, S., and R.T. Yeh
1979 "Structural Locking for Concurrency Control in Database Systems.”
Li, Victor  

Liskov, Barbara, and Stephen Zilles  

Liskov, Barbara, Russ Atkinson, Toby Bloom, Eliot Moss, Craig Schaffert, Bob Scheifler, and Alan Snyder  

Livesey, J.  

McKusick, Marshall K, William N. Joy, Samuel J. Leffler, and Robert S. Fabry  

Peterson, W.W, and D.T. Brown  

Pohm, Arthur V.  

Postel, Jonathan B.  

Reed, David P.  

1983  "Implementing Atomic Actions on Decentralized Data." ACM Transactions on Computer Systems 1, No. 1: 3-23.

Reed, David P., and Rajendra Kanodia  
Ritchie, Dennis M. and Ken Thompson
1974 "The UNIX Time-Sharing System."
Communications of the ACM 17, No. 7: 365-375.

Rolin, P. and J. Boudenant
1982 "Multiple Copy Consistency Faced to Network Partitioning."

Rosenkrantz, D.J., R.E. Stearns, and P.M. Lewis
1978 "System Level Concurrency Control for Distributed Database Systems."

Rothnie, James B. and Nathan Goodman
1977 "A Survey of Research and Development in Distributed Database Management."
IEEE Proceedings of the Third International Conference on Very Large Data Bases, Tokyo, Japan: 48-62.

Shoch, John F., and Jon A. Hupp
1980 "Measured Performance of an Ethernet Local Network."

Svobodova, Liba, Barbara Liskov, and David Clark
1979 "Distributed Computer Systems: Structure and Semantics."

Teng-amnuay, Yunyong
1983 "A Categorization Scheme for Concurrency Control Protocols in Distributed Databases."
Ph.D. Dissertation, Iowa State University, Ames, Iowa.

Thomas, Robert H.
1978 "A Solution to the Concurrency Control Problem for Multiple Copy Data Bases."

Wegner, Peter

Weihl, William, and Barbara Liskov

Wirth, Niklaus
1982 "Programming in Modula-2."
Springer-Verlag, New York, New York.
Wulf, William
1975 "HYDRA: The Kernel of a Multiprocessor Operating Systems."
Communications of the ACM 17, No. 6: 337-345.

Zimmermann, Hubert
1980 "OSI Reference Model - The ISO Model of Architecture for Open
Systems Interconnection." IEEE Transactions on
Communications COM-28, No. 4: 425-432.
I think that working towards a graduate degree is not unlike acting in a play. Ultimately you have to do the work and it's your show, but it takes a large cast of supporting players to make the final result meaningful. I have been very fortunate to receive welcomed advice and support from a number of people during my graduate career at Iowa State University. These people include Professors Dale Grosvenor, Clair Maple, Arch Oldehoeft, Arthur Pohm, Steffan Silverston, Terry Smay, George Strawn, and Charles Wright. Additionally, I would like to recognize Steve Christiansen, Gary Bridges, and Doug Jacobson for their support. Finally, I want to thank my parents who helped me make it through undergraduate school, and my wife Cheryl, who was extremely supportive during the long ordeal that goes with writing a dissertation.