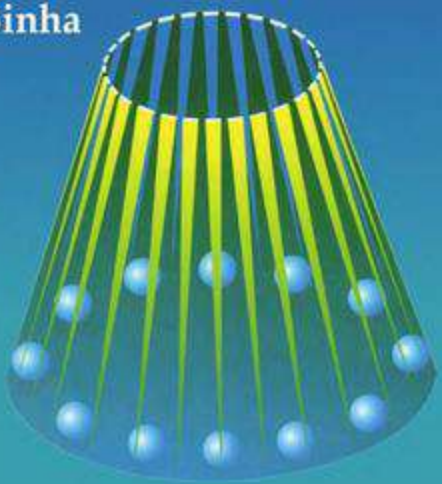
Eastern Economy Edition

Distributed Operating Systems

Concepts and Design

Pradeep K. Sinha



PHI

Distributed Operating Systems

IEEE Press
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331

Also from IEEE Press

Computer Communications and Networks, Second Edition
John Freer
Copublished with UCL Press
1996 Hardcover 400 pp IEEE Order No. PC5654 ISBN 0-7803-1179-5

Engineering Networks for Synchronization, CCS 7 and ISDN: Standards, Protocols, Planning, and Testing
P. K. Bhatnagar
1997 Hardcover 528 pp IEEE Order No. PC5628 ISBN 0-7803-1158-2

SONET-SDH: A Sourcebook of Synchronous Networking
edited by Curtis A. Siller, Jr. and Mansoor Shafi
1996 Hardcover 406 pp IEEE Order No. PC4457 ISBN 0-7803-1168-X

Distributed Operating Systems

Concepts and Design

Pradeep K. Sinha

Centre for Development of Advanced Computing

IEEE COMPUTER SOCIETY PRESS

IEEE PRESS

IEEE Communications Society, *Sponsor*

The Institute of Electrical and Electronics Engineers, Inc., New York

This book may be purchased at a discount from the publisher when ordered in bulk quantities. Contact:

IEEE Press Marketing
Attn: Special Sales
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331
Fax: (732) 981-9334

For more information on the IEEE Press, visit the IEEE home page:
http://www.ieee.org/

Printed in the United States of America

10 9 8 7 6 5 4 3

ISBN 0-7803-1119-1
IEEE Order Number: PC4705

Library of Congress Cataloging-in-Publication Data

Sinha, Pradeep K. (Pradeep Kumar)
Distributed operating systems/Pradeep K. Sinha.
p. cm.
Includes bibliographical references and index.
ISBN 0-7803-1119-1 (cloth)
1. Distributed operating systems (Computers) I. Title.
QA76.76.063S568 1996 96-26565
005.4'4—dc20 CIP

Contents

Preface xi

Acknowledgments xv

Abbreviations and Acronyms xvii

Chapter 1: Fundamentals 1

1.1 What Is a Distributed Computing System? 1
1.2 Evolution of Distributed Computing Systems 3
1.3 Distributed Computing System Models 5
1.4 Why Are Distributed Computing Systems Gaining Popularity? 12
1.5 What Is a Distributed Operating System? 16
1.6 Issues in Designing a Distributed Operating System 19
1.7 Introduction to Distributed Computing Environment (DCE) 34
1.8 Summary 38
Exercises 39
Bibliography 41
Pointers to Bibliographies on the Internet 44

Chapter 2: Computer Networks 46

2.1 Introduction 46
2.2 Networks Types 47
2.3 LAN Technologies 48
2.4 WAN Technologies 59
2.5 Communication Protocols 64
2.6 Internetworking 83
2.7 ATM Technology 91
2.8 Summary 104
Exercises 105
Bibliography 108
Pointers to Bibliographies on the Internet 112

Chapter 3: Message Passing 114

3.1 Introduction 114
3.2 Desirable Features of a Good Message-Passing System 115
3.3 Isses in IPC by Message Passing 118
3.4 Synchronization 120
3.5 Buffering 122
3.6 Multidatagram Messages 125
3.7 Encoding and Decoding of Message Data 126
3.8 Process Addressing 127
3.9 Failure Handling 130
3.10 Group Communication 139
3.11 Case Study: 4.3BSD UNIX IPC Mechanism 153
3.12 Summary 157
Exercises 160
Bibliography 163
Pointers to Bibliographies on the Internet 166

Chapter 4: Remote Procedure Calls 167

4.1 Introduction 167
4.2 The RPC Model 168
4.3 Transparency of RPC 170
4.4 Implementing RPC Mechanism 171
4.5 Stub Generation 174
4.6 RPC Messages 174
4.7 Marshaling Arguments and Results 177
4.8 Server Management 178
4.9 Parameter-Passing Semantics 183

4.10 Call Semantics 184
4.11 Communication Protocols for RPCs 187
4.12 Complicated RPCs 191
4.13 Client-Server Binding 193
4.14 Exception Handling 198
4.15 Security 198
4.16 Some Special Types of RPCs 199
4.17 RPC in Heterogeneous Environments 203
4.18 Lightweight RPC 204
4.19 Optimizations for Better Performance 208
4.20 Case Studies: Sun RPC, DCE RPC 212
4.21 Summary 222
Exercises 224
Bibliography 227
Pointers to Bibliographies on the Internet 230

Chapter 5: Distributed Shared Memory 231

5.1 Introduction 231
5.2 General Architecture of DSM Systems 233
5.3 Design and Implementation Issues of DSM 234
5.4 Granularity 235
5.5 Structure of Shared Memory Space 237
5.6 Consistency Models 238
5.7 Replacement Strategy 262
5.8 Thrashing 264
5.9 Other Approaches to DSM 266
5.10 Heterogeneous DSM 267
5.11 Advantages of DSM 270
5.12 Summary 272
Exercises 274
Bibliography 275
Pointers to Bibliographies on the Internet 281

Chapter 6: Synchronization 282

6.1 Introduction 282
6.2 Clock Synchronization 283
6.3 Event Ordering 292
6.4 Mutual Exclusion 299
6.5 Deadlock 305
6.6 Election Algorithms 332
6.7 Summary 336

Exercises 337
Bibliography 341
Pointers to Bibliographies on the Internet 345

Chapter 7: Resource Management 347

7.1 Introduction 347
7.2 Desirable Features of a Good Global Scheduling Algorithm 348
7.3 Task Assignment Approach 351
7.4 Load-Balancing Approach 355
7.5 Load-Sharing Approach 367
7.6 Summary 371
Exercises 372
Bibliography 374
Pointers to Bibliographies on the Internet 380

Chapter 8: Process Management 381

8.1 Introduction 381
8.2 Process Migration 382
8.3 Threads 398
8.4 Summary 414
Exercises 415
Bibliography 418
Pointers to Bibliographies on the Internet 420

Chapter 9: Distributed File Systems 421

9.1 Introduction 421
9.2 Desirable Features of a Good Distributed File System 423
9.3 File Models 426
9.4 File-Accessing Models 427
9.5 File-Sharing Semantics 430
9.6 File-Caching Schemes 433
9.7 File Replication 440
9.8 Fault Tolerance 447
9.9 Atomic Transactions 453
9.10 Design Principles 474
9.11 Case Study: DCE Distributed File Service 475
9.12 Summary 484
Exercises 486
Bibliography 489
Pointers to Bibliographies on the Internet 495

Chapter 10: Naming 496

10.1 Introduction 496
10.2 Desirable Features of a Good Naming System 497
10.3 Fundamental Terminologies and Concepts 499
10.4 System-Oriented Names 509
10.5 Object-Locating Mechanisms 512
10.6 Human-Oriented Names 515
10.7 Name Caches 541
10.8 Naming and Security 544
10.9 Case Study: DCE Directory Service 546
10.10 Summary 556
Exercises 558
Bibliography 560
Pointers to Bibliographies on the Internet 564

Chapter 11: Security 565

11.1 Introduction 565
11.2 Potential Attacks to Computer Systems 567
11.3 Cryptography 575
11.4 Authentication 586
11.5 Access Control 607
11.6 Digital Signatures 623
11.7 Design Principles 626
11.8 Case Study: DCE Security Service 627
11.9 Summary 630
Exercises 631
Bibliography 634
Pointers to Bibliographies on the Internet 640

Chapter 12: Case Studies 642

12.1 Introduction 642
12.2 Amoeba 643
12.3 V-System 659
12.4 Mach 674
12.5 Chorus 696
12.6 A Comparison of Amoeba, V-System, Mach, and Chorus 714
12.7 Summary 718
Exercises 722
Bibliography 725
Pointers to Bibliographies on the Internet 730

Index 731

Preface

Motivation

The excellent price/performance ratio offered by microprocessor-based workstations over traditional mainframe systems and the steady improvements in networking technologies have made distributed computing systems very attractive. While the hardware issues of building such systems have been fairly well understood for quite some time, the major stumbling block until now has been the availability of good distributed operating systems. Fortunately, recent research and development work in academic institutions and industries have helped us better understand the basic concepts and design issues in developing distributed operating systems. Distributed operating systems are no more only in research laboratories but are now commercially available.

With the proliferation of distributed computing systems, it has become increasingly important for computer science and computer engineering students to learn about distributed operating systems. As a result, a number of universities have instituted regular courses on distributed operating systems at the graduate level. Even in various undergraduate-level operating systems courses, the fundamental concepts and design principles of distributed operating systems have been incorporated.

However, there is still a lack of good textbooks that can provide a comprehensive and solid introduction to distributed operating systems in an orderly manner. Except for a few

recently published books, almost all books in this area are research monographs. Therefore, for both an educator and a student, the creation of an overall image of distributed operating systems is currently a complicated and time-consuming task. Furthermore, computer professionals and starting researchers who want to get an overall picture of distributed operating systems so as to identify the various research and design issues have difficulty in finding a good text for their purpose.

Motivated by these factors, I decided to do research toward the preparation of a textbook on distributed operating systems. My primary objective was to concisely present a clear explanation of the current state of the art in distributed operating systems so that readers can gain sufficient background to appreciate more advanced materials of this field.

Overview

The book is designed to provide a clear description of the fundamental concepts and design principles that underlie distributed operating systems. It does not concentrate on any particular distributed operating system or hardware. Instead, it discusses, in a general setting, the fundamental concepts and design principles that are applicable to a variety of distributed operating systems. However, case studies are included in the text to relate the discussed concepts with real distributed operating systems.

The material in the book has been drawn largely from the research literature in the field. Of the vast amount of research literature available in this field, effort was made to select and give more emphasis to those concepts that are of practical value in real systems, rather than those that are only of theoretical interest.

Each chapter contains carefully designed exercises that are meant to test the understanding of the materials in the text and to stimulate investigation.

An extensive set of references and a list of selected pointers to on-line bibliographies on the Internet have been provided at the end of each chapter to allow interested readers to explore more advanced materials dealing with finer details about each chapter's topics.

Throughout the book, the style of presentation used is motivational and explanatory in nature.

Contents

Chapter 1 provides an introduction to distributed computing systems, distributed operating systems, and the issues involved in designing distributed operating systems. It also provides a brief introduction to Distributed Computing Environment (DCE), whose components are described as case studies of key technologies in several chapters of the book.

Chapter 2 presents a brief introduction to computer networks and describes the current state of the art in networking technology.

Chapters 3, 4, and 5 describe the various communication techniques used for exchange of information among the processes of a distributed computing system. In particular, these three chapters deal with the issues involved in the design of interprocess communication mechanisms and the commonly used practical approaches to handle these issues. Chapter 3 deals with the message-passing mechanism. Chapter 4 deals with the remote procedure call mechanism, and Chapter 5 deals with the distributed shared-memory mechanism for interprocess communication.

Synchronization issues to be dealt with in a distributed system, such as clock synchronization, mutual exclusion, deadlock, and election algorithms, are discussed in Chapter 6.

Chapter 7 presents a discussion of the commonly used approaches for resource management in distributed systems.

Chapter 8 deals with the process management issues. In particular, it presents a discussion of process migration mechanisms and mechanisms to support threads facility.

A discussion of the issues and the approaches for designing a file system for a distributed system is given in Chapter 9.

Chapter 10 deals with the issues and mechanisms for naming and locating objects in distributed systems.

The security issues and security mechanisms for distributed systems are discussed in Chapter 11.

Finally, Chapter 12 contains case studies of four existing distributed operating systems to relate the concepts discussed in the preceding chapters with real distributed operating systems.

Audience

The book is suitable for anyone who needs a concise and informal introduction to distributed operating systems.

It can serve as an ideal textbook for a course on distributed operating systems. It can also be used for advanced undergraduate and postgraduate courses on operating systems, which often need to cover the fundamental concepts and design issues of distributed operating systems in addition to those of centralized operating systems.

The book can also be used as a self-study text by system managers, professional software engineers, computer scientists, and researchers, who either need to learn about distributed operating systems or are involved in the design and development of distributed operating systems or distributed application systems.

Advanced researchers will also find the rich set of references and the pointers to on-line bibliographies on the Internet provided at the end of each chapter very helpful in probing further on any particular topic.

Although full care has been taken to make the subject matter simple and easy to understand by a wide range of readers, I have assumed that the reader has a knowledge of elementary computer architecture and is familar with basic centralized operating systems concepts discussed in standard operating systems textbooks.

About Pointers to Bibliographies on the Internet

In addition to a good number of references provided in the end-of-chapter bibliographies, I have also provided lists of selected pointers to the on-line bibliographies of interest on the Internet. The purpose of these pointers is twofold:

1. The end-of-chapter bibliographies contain only selected references. A large number of references on the topics covered in a chapter are not included in the chapter's bibliography due to space limitations. The pointers may be used by interested readers to locate such references.
2. The end-of-chapter bibliographies contain references to only already published documents. Distributed operating systems is currently an active area of research, and a large volume of new documents are published almost every month. Since the on-line bibliographies on the Internet are updated from time to time, interested readers may use the pointers to locate those documents that are published after the publication of this book. Thus, in addition to the information contained in it, the book also provides a way for its readers to keep track of on-going research activities on the topics covered and related topics.

Note that the end-of-chapter lists of pointers to on-line bibliographies are by no means exhaustive. I have provided pointers for only those on-line bibliographies that I knew about and I felt would be useful in easily locating references of interest. Moreover, it is often the case that there are many mirrors for an on-line bibliography (the same bilbiography exists on multiple sites). For such cases, I have provided only one pointer for a bibliography.

Also note that most of the on-line bibliographies are not about on-line documents, but about on-line references to documents. A vast majority of documents referenced in the on-line bibliographies only exist in hard-copy form. However, a few of the referenced documents do have an on-line version on the Internet. For such documents, the bibliographies normally contain URLs (Uniform Resource Locators) pointing to the on-line version of the document. If you find a reference containing such a URL, just follow the URL to access the on-line version of the corresponding document.

Acknowledgments

Many people have contributed to this book, either directly or indirectly. To start with, I must thank all the researchers who have contributed to the fields of distributed computing systems and distributed operating systems because the book is based on their research results.

Mamoru Maekawa, my Ph.D. supervisor, provided me with the opportunity to do research in the area of distributed operating systems. Without this opportunity, this book would not have been possible.

Lively discussions with the members of the Galaxy distributed operating system project, including Kentaro Shimizu, Xiaohua Jia, Hyo Ashishara, Naoki Utsunomiya, Hirohiko Nakano, Kyu Sung Park, and Jun Hamano, helped a lot in broadening my knowledge of distributed operating systems.

Without the efforts of all the people who collected references and made them available on the Internet, it would not have been possible for me to provide the end-of-chapter pointers to the bibliographies on the Internet.

Several anonymous reviewers of my draft manuscript provided invaluable feedback concerning organization, topic coverage, typographical errors, and parts of the text that were not as clear as they are now.

My production editor at IEEE Press, Denise Phillip, did an excellent job in numerous ways to present the book in its current form. IEEE Press Director Dudley Kay, Senior

Acquisitions Editor John Griffin, and review coordinators Lisa Mizrahi and Lisa Dayne were of great help in improving the overall quality of the book and in bringing it out in a timely manner.

Finally, I thank my wife, Priti, for preparing the electronic version of the entire handwritten draft manuscript. I also thank her for her continuous patience and sacrifices during the entire period of this long project. Without her loving support and understanding, I would never have succeeded in completing this project.

Pradeep K. Sinha

Abbreviations and Acronyms

AAL	ATM Adaptation Layer
ACL	Access Control List
AFS	Andrew File System
ANSI	American National Standards Institute
API	Application Programming Interface
APPN	Advanced Peer-to-Peer Networking
ARP	Address Resolution Protocol
ARPANET	Advanced Research Projects Agency NETwork
ASN.1	Abstract Syntax Notation
ATM	Asynchronous Transfer Mode
BIOS	Basic Input Output System
B-ISDN	Broadband Integrated Services Digital Network
CBR	Constant Bit Rate
CCITT	International Telegraph and Telephone Consultative Committee
CDS	Cell Directory Service/Server
CERN	European Centre for Nuclear Research
CFS	Cedar File System
CICS	Customer Information Control System
CLP	Cell Loss Priority
CMH	Chandy-Misra-Hass
CMIP	Common Management Information Protocol
COOL	Chorus Object-Oriented Layer
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
CSRG	Computer Systems Research Group
DCE	Distributed Computing Environment
DDLCN	Distributed Double-Loop Computer Network
DEC	Digital Equipment Corporation
DES	Data Encryption Standard

DFS	Distributed File Service
DI	Directory Identifier
DIB	Directory Information Base
DIT	Directory Information Tree
DME	Distributed Management Environment
DN	Distinguished Name
DNS	Domain Name/Naming Service/System
DoD	Department of Defense
DSM	Distributed Shared Memory
DSVM	Distributed Shared Virtual Memory
DTS	Distributed Time Service
Email	Electronic Mail
ERB	Expanding Ring Broadcast
FDDI	Fiber Distributed Data Interface
FIFO	First-In First-Out
FLIP	Fast Local Internet Protocol
FTP	File Transfer Protocol
Gbps	Gigabits per second
GDA	Global Directory Agent
GDS	Global Directory Service/Server
GEOS	Geostationary Operational Environmental Satellites
GFC	Generic Flow Control
GNS	Global Name Service
GNU	Gnu's Not Unix
GPS	Global Positioning System
HEC	Header Error Control
HRPC	Heterogeneous Remote Procedure Call
IBM	International Business Machines
ICMP	Internet Control Message Protocol
IDL	Interface Definition Language
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IMS	Information Management System
INRIA	Institute National de Recherche en Informatique et Automatique
IP	Internet Protocol
IPC	Inter-Process Communication
ISDN	Integrated Services Digital Network
ISO	International Standards Organization
ITU	International Telecommunications Union
KDBMS	Kerberos Database Management Server
KDC	Key Distribution Center
LAN	Local Area Network
LEC	LAN Emulation Client
LES	LAN Emulation Server
LRPC	Lightweight Remote Procedure Call
MAN	Metropolitan Area Network
MBone	Multicast Backbone
Mbps	Megabits per second
MIG	Mach Interface Generator
MMU	Memory Management Unit
MTBF	Mean Time Between Failures
MTU	Maximum Transfer Unit
NCA	Network Computing Architecture
NCS	Network Computing System
NFS	Network File System
NIC	Network Information Center
NIST	National Institute for Standards and Technology
NRMB	Non-Replicated Migrating Block
NRNMB	Non-Replicated Non-Migrating Block
NSAP	Network Service Access Point
NTP	Network Time Protocol
NUMA	Non-Uniform Memory Access
OC-*n*	Optical Carrier level *n*
OLTP	On Line Transaction Processing
OPC	Output Port Controller
OSF	Open Software Foundation
OSI	Open System International
PCB	Process Control Block
PEM	Privacy Enhanced Mail

PKM	Public Key Manager
PMD	Physical Medium Dependent
POSIX	Portable Operating System Interface for Computer Environments
PRAM	Pipelined Random Access Memory
PSE	Packet Switching Exchange
PTI	Payload Type Identifier
RARP	Reverse Address Resolution Protocol
RDN	Relative Distinguished Name
RFS	Remote File Server
RFT	Request For Technology
RMB	Replicated Migrating Block
RNMB	Replicated Non-Migrating Block
RPC	Remote Procedure Call
RPCL	Remote Procedure Call Language
RR	Round-Robin
RSA	Rivest-Shamir-Adleman
RSS	Research Storage System
SDH	Synchronous Digital Hierarchy
SEAL	Simple and Efficient Adaptation Layer
SLIP	Serial Line Internet Protocol
SMTP	Simple Mail Transfer Protocol
SNA	System Network Architecture
SNMP	Simple Network Management Protocol
SONET	Synchronous Optical NETwork
STS-*n*	Synchronous Transport Signal level *n*
TC	Transmission Convergence
TCF	Transparent Computing Facility
TCP	Transport Control Protocol
TFTP	Trivial File Transfer Protocol
TI-RPC	Transport Independent-Remote Procedure Call
TP	Transport Protocol
TWFG	Transaction Wait-For-Graph
UCP	Unilateral Commit Protocol
UDP	User Datagram Protocol
UDS	Universal Directory Service
UNI	User Network Interface
UTC	Coordinated Universal Time
UUID	Universally Unique Identifier
VBR	Variable Bit Rate
VCI	Virtual Channel Identifier
VM	Virtual Memory
VMTP	Versatile Message Transfer Protocol
VPI	Virtual Path Identifier
WAIS	Wide Area Information Servers
WAN	Wide Area Network
WFG	Wait For Graph
WWW	World Wide Web
X-IPC	eXtented Inter-Process Communication
XDR	eXternal Data Representation
XDS	X/Open Directory Server
XNS	Xerox Networking System
XOM	X/Open Object Management

CHAPTER 1

Fundamentals

1.1 WHAT IS A DISTRIBUTED COMPUTING SYSTEM?

Over the past two decades, advancements in microelectronic technology have resulted in the availability of fast, inexpensive processors, and advancements in communication technology have resulted in the availability of cost-effective and highly efficient computer networks. The net result of the advancements in these two technologies is that the price-performance ratio has now changed to favor the use of interconnected, multiple processors in place of a single, high-speed processor.

Computer architectures consisting of interconnected, multiple processors are basically of two types:

1. *Tightly coupled systems.* In these systems, there is a single systemwide primary memory (address space) that is shared by all the processors [Fig. 1.1(a)]. If any processor writes, for example, the value 100 to the memory location x, any other processor subsequently reading from location x will get the value 100. Therefore, in these systems, any communication between the processors usually takes place through the shared memory.

2. *Loosely coupled systems.* In these systems, the processors do not share memory, and each processor has its own local memory [Fig. 1.1(b)]. If a processor writes the value

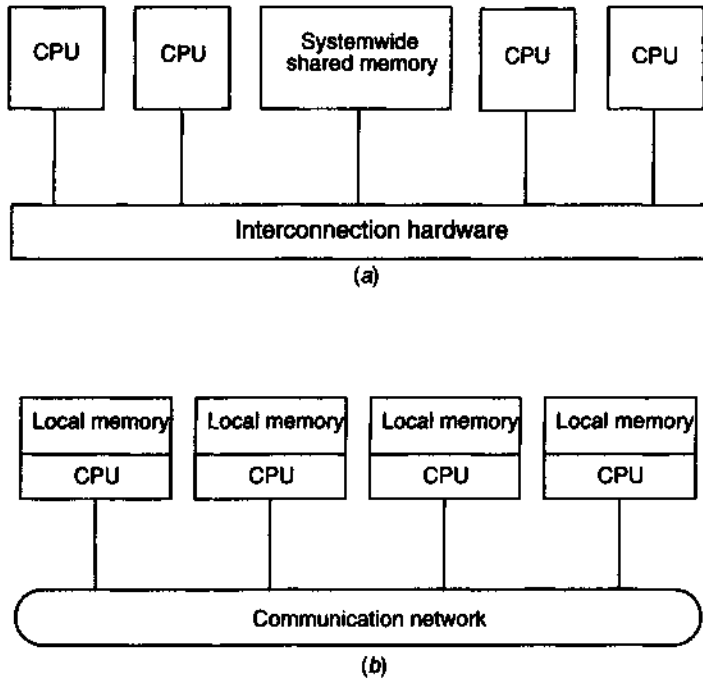


Fig. 1.1 Difference between tightly coupled and loosely coupled multiprocessor systems: (*a*) a tightly coupled multiprocessor system; (*b*) a loosely coupled multiprocessor system.

100 to the memory location x, this write operation will only change the contents of its local memory and will not affect the contents of the memory of any other processor. Hence, if another processor reads the memory location *x*, it will get whatever value was there before in that location of its own local memory. In these systems, all physical communication between the processors is done by passing messages across the network that interconnects the processors.

Usually, tightly coupled systems are referred to as *parallel processing systems*, and loosely coupled systems are referred to as *distributed computing systems*, or simply distributed systems. In this book, however, the term "distributed system" will be used only for true distributed systems—distributed computing systems that use distributed operating systems (see Section 1.5). Therefore, before the term "true distributed system" is defined in Section 1.5, the term "distributed computing system" will be used to refer to loosely coupled systems. In contrast to the tightly coupled systems, the processors of distributed computing systems can be located far from each other to cover a wider geographical area. Furthermore, in tightly coupled systems, the number of processors that can be usefully deployed is usually small and limited by the bandwidth of the shared memory. This is not the case with distributed computing systems that are more freely expandable and can have an almost unlimited number of processors.

In short, a distributed computing system is basically a collection of processors interconnected by a communication network in which each processor has its own local memory and other peripherals, and the communication between any two processors of the system takes place by message passing over the communication network. For a particular processor, its own resources are *local*, whereas the other processors and their resources are *remote*. Together, a processor and its resources are usually referred to as a *node* or *site* or *machine* of the distributed computing system.

1.2 EVOLUTION OF DISTRIBUTED COMPUTING SYSTEMS

Early computers were very expensive (they cost millions of dollars) and very large in size (they occupied a big room). There were very few computers and were available only in research laboratories of universities and industries. These computers were run from a console by an operator and were not accessible to ordinary users. The programmers would write their programs and submit them to the computer center on some media, such as punched cards, for processing. Before processing a job, the operator would set up the necessary environment (mounting tapes, loading punched cards in a card reader, etc.) for processing the job. The job was then executed and the result, in the form of printed output, was later returned to the programmer.

The job setup time was a real problem in early computers and wasted most of the valuable central processing unit (CPU) time. Several new concepts were introduced in the 1950s and 1960s to increase CPU utilization of these computers. Notable among these are batching together of jobs with similar needs before processing them, automatic sequencing of jobs, off-line processing by using the concepts of buffering and spooling, and multiprogramming. *Batching* similar jobs improved CPU utilization quite a bit because now the operator had to change the execution environment only when a new batch of jobs had to be executed and not before starting the execution of every job. *Automatic job sequencing* with the use of control cards to define the beginning and end of a job improved CPU utilization by eliminating the need for human job sequencing. *Off-line processing* improved CPU utilization by allowing overlap of CPU and input/output (I/O) operations by executing those two actions on two independent machines (I/O devices are normally several orders of magnitude slower than the CPU). Finally, *multiprogramming* improved CPU utilization by organizing jobs so that the CPU always had something to execute.

However, none of these ideas allowed multiple users to directly interact with a computer system and to share its resources simultaneously. Therefore, execution of interactive jobs that are composed of many short actions in which the next action depends on the result of a previous action was a tedious and time-consuming activity. Development and debugging of programs are examples of interactive jobs. It was not until the early 1970s that computers started to use the concept of *time sharing* to overcome this hurdle. Early time-sharing systems had several dumb terminals attached to the main computer. These terminals were placed in a room different from the main computer room. Using these terminals, multiple users could now simultaneously execute interactive jobs and share the resources of the computer system. In a time-sharing system, each user is given

the impression that he or she has his or her own computer because the system switches rapidly from one user's job to the next user's job, executing only a very small part of each job at a time. Although the idea of time sharing was demonstrated as early as 1960, time-sharing computer systems were not common until the early 1970s because they were difficult and expensive to build.

Parallel advancements in hardware technology allowed reduction in the size and increase in the processing speed of computers, causing large-sized computers to be gradually replaced by smaller and cheaper ones that had more processing capability than their predecessors. These systems were called *minicomputers*.

The advent of time-sharing systems was the first step toward distributed computing systems because it provided us with two important concepts used in distributed computing systems—the sharing of computer resources simultaneously by many users and the accessing of computers from a place different from the main computer room. Initially, the terminals of a time-sharing system were dumb terminals, and all processing was done by the main computer system. Advancements in microprocessor technology in the 1970s allowed the dumb terminals to be replaced by intelligent terminals so that the concepts of off-line processing and time sharing could be combined to have the advantages of both concepts in a single system. Microprocessor technology continued to advance rapidly, making available in the early 1980s single-user computers called *workstations* that had computing power almost equal to that of minicomputers but were available for only a small fraction of the price of a minicomputer. For example, the first workstation developed at Xerox PARC (called Alto) had a high-resolution monochrome display, a mouse, 128 kilobytes of main memory, a 2.5-megabyte hard disk, and a microprogrammed CPU that executed machine-level instructions at speeds of 2–6 μs. These workstations were then used as terminals in the time-sharing systems. In these time-sharing systems, most of the processing of a user's job could be done at the user's own computer, allowing the main computer to be simultaneously shared by a larger number of users. Shared resources such as files, databases, and software libraries were placed on the main computer.

Centralized time-sharing systems described above had a limitation in that the terminals could not be placed very far from the main computer room since ordinary cables were used to connect the terminals to the main computer. However, in parallel, there were advancements in computer networking technology in the late 1960s and early 1970s that emerged as two key networking technologies—*LAN* (*local area network*) and *WAN* (*wide-area network*). The LAN technology allowed several computers located within a building or a campus to be interconnected in such a way that these machines could exchange information with each other at data rates of about 10 megabits per second (Mbps). On the other hand, WAN technology allowed computers located far from each other (may be in different cities or countries or continents) to be interconnected in such a way that these machines could exchange information with each other at data rates of about 56 kilobits per second (Kbps). The first high-speed LAN was the Ethernet developed at Xerox PARC in 1973, and the first WAN was the ARPAnet (Advanced Research Projects Agency Network) developed by the U.S. Department of Defense in 1969. The data rates of networks continued to improve gradually in the 1980s, providing data rates of up to 100 Mbps for LANs and data rates of up to 64 Kbps for WANs. Recently (early 1990s) there has been another major advancement in networking technology—the *ATM* (*asynchronous*

transfer mode) technology. The ATM technology is an emerging technology that is still not very well established. It will make very high speed networking possible, providing data transmission rates up to 1.2 gigabits per second (Gbps) in both LAN and WAN environments. The availability of such high-bandwidth networks will allow future distributed computing systems to support a completely new class of distributed applications, called *multimedia applications*, that deal with the handling of a mixture of information, including voice, video, and ordinary data. These applications were previously unthinkable with conventional LANs and WANs.

The merging of computer and networking technologies gave birth to distributed computing systems in the late 1970s. Although the hardware issues of building such systems were fairly well understood, the major stumbling block at that time was the availability of adequate software for making these systems easy to use and for fully exploiting their power. Therefore, starting from the late 1970s, a significant amount of research work was carried out in both universities and industries in the area of distributed operating systems. These research activities have provided us with the basic ideas of designing distributed operating systems. Although the field is still immature, with ongoing active research activities, commercial distributed operating systems have already started to emerge. These systems are based on already established basic concepts. This book deals with these basic concepts and their use in the design and implementation of distributed operating systems. Several of these concepts are equally applicable to the design of applications for distributed computing systems, making this book also suitable for use by the designers of distributed applications.

1.3 DISTRIBUTED COMPUTING SYSTEM MODELS

Various models are used for building distributed computing systems. These models can be broadly classified into five categories—minicomputer, workstation, workstation-server, processor-pool, and hybrid. They are briefly described below.

1.3.1 Minicomputer Model

The *minicomputer model* is a simple extension of the centralized time-sharing system. As shown in Figure 1.2, a distributed computing system based on this model consists of a few minicomputers (they may be large supercomputers as well) interconnected by a communication network. Each minicomputer usually has multiple users simultaneously logged on to it. For this, several interactive terminals are connected to each minicomputer. Each user is logged on to one specific minicomputer, with remote access to other minicomputers. The network allows a user to access remote resources that are available on some machine other than the one on to which the user is currently logged.

The minicomputer model may be used when resource sharing (such as sharing of information databases of different types, with each type of database located on a different machine) with remote users is desired.

The early ARPAnet is an example of a distributed computing system based on the minicomputer model.

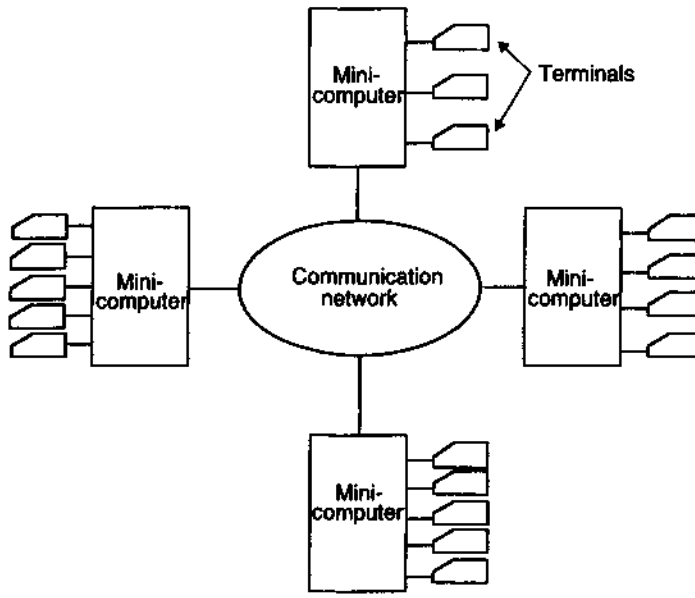


Fig. 1.2 A distributed computing system based on the minicomputer model.

1.3.2 Workstation Model

As shown in Figure 1.3, a distributed computing system based on the *workstation model* consists of several workstations interconnected by a communication network. A company's office or a university department may have several workstations scattered throughout a building or campus, each workstation equipped with its own disk and serving as a single-user computer. It has been often found that in such an environment, at any one time (especially at night), a significant proportion of the workstations are idle (not being used), resulting in the waste of large amounts of CPU time. Therefore, the idea of the workstation model is to interconnect all these workstations by a high-speed LAN so that idle workstations may be used to process jobs of users who are logged onto other workstations and do not have sufficient processing power at their own workstations to get their jobs processed efficiently.

In this model, a user logs onto one of the workstations called his or her "home" workstation and submits jobs for execution. When the system finds that the user's workstation does not have sufficient processing power for executing the processes of the submitted jobs efficiently, it transfers one or more of the processes from the user's workstation to some other workstation that is currently idle and gets the process executed there, and finally the result of execution is returned to the user's workstation.

This model is not so simple to implement as it might appear at first sight because several issues must be resolved. These issues are [Tanenbaum 1995] as follows:

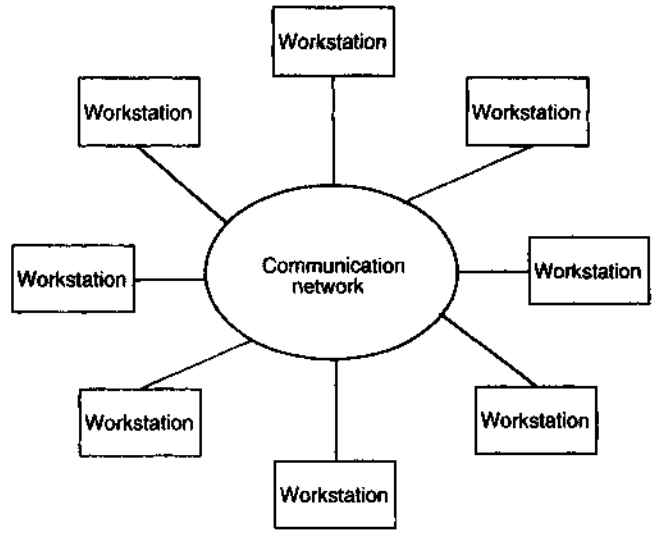


Fig. 1.3 A distributed computing system based on the workstation model.

1. How does the system find an idle workstation?
2. How is a process transferred from one workstation to get it executed on another workstation?
3. What happens to a remote process if a user logs onto a workstation that was idle until now and was being used to execute a process of another workstation?

Ways to handle the first two issues are described in Chapters 7 and 8, respectively. Three commonly used approaches for handling the third issue are as follows:

1. The first approach is to allow the remote process share the resources of the workstation along with its own logged-on user's processes. This method is easy to implement, but it defeats the main idea of workstations serving as personal computers, because if remote processes are allowed to execute simultaneously with the logged-on user's own processes, the logged-on user does not get his or her guaranteed response.

2. The second approach is to kill the remote process. The main drawbacks of this method are that all processing done for the remote process gets lost and the file system may be left in an inconsistent state, making this method unattractive.

3. The third approach is to migrate the remote process back to its home workstation, so that its execution can be continued there. This method is difficult to implement because it requires the system to support preemptive process migration facility. The definition of preemptive process migration and the issues involved in preemptive process migration are given in Chapter 8.

The Sprite system [Ousterhout et al. 1988] and an experimental system developed at Xerox PARC [Shoch and Hupp 1982] are two examples of distributed computing systems based on the workstation model.

1.3.3 Workstation-Server Model

The workstation model is a network of personal workstations, each with its own disk and a local file system. A workstation with its own local disk is usually called a *diskful workstation* and a workstation without a local disk is called a *diskless workstation*. With the proliferation of high-speed networks, diskless workstations have become more popular in network environments than diskful workstations, making the workstation-server model more popular than the workstation model for building distributed computing systems.

As shown in Figure 1.4, a distributed computing system based on the *workstation-server model* consists of a few minicomputers and several workstations (most of which are diskless, but a few of which may be diskful) interconnected by a communication network.

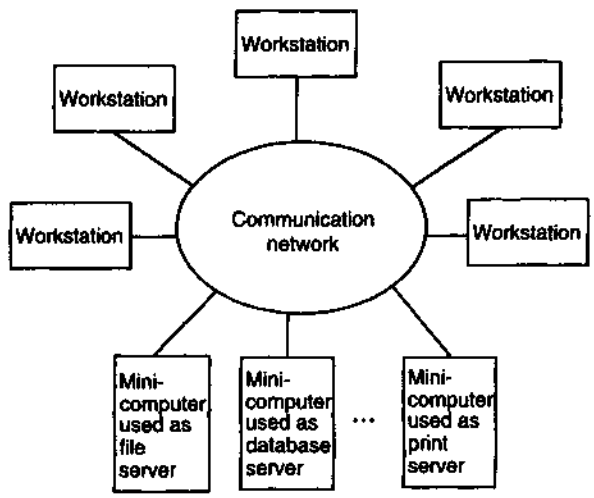


Fig. 1.4 A distributed computing system based on the workstation–server model.

Note that when diskless workstations are used on a network, the file system to be used by these workstations must be implemented either by a diskful workstation or by a minicomputer equipped with a disk for file storage. The minicomputers are used for this purpose. One or more of the minicomputers are used for implementing the file system. Other minicomputers may be used for providing other types of services, such as database service and print service. Therefore, each minicomputer is used as a server machine to provide one or more types of services. Hence in the workstation-server model, in addition to the workstations, there are specialized machines (may be specialized workstations) for running server processes (called *servers*) for managing and providing access to shared resources.

For a number of reasons, such as higher reliability and better scalability, multiple servers are often used for managing the resources of a particular type in a distributed computing system. For example, there may be multiple file servers, each running on a separate minicomputer and cooperating via the network, for managing the files of all the users in the system. Due to this reason, a distinction is often made between the services that are provided to clients and the servers that provide them. That is, a *service* is an abstract entity that is provided by one or more servers. For example, one or more file servers may be used in a distributed computing system to provide file service to the users.

In this model, a user logs onto a workstation called his or her home workstation. Normal computation activities required by the user's processes are performed at the user's home workstation, but requests for services provided by special servers (such as a file server or a database server) are sent to a server providing that type of service that performs the user's requested activity and returns the result of request processing to the user's workstation. Therefore, in this model, the user's processes need not be migrated to the server machines for getting the work done by those machines.

For better overall system performance, the local disk of a diskful workstation is normally used for such purposes as storage of temporary files, storage of unshared files, storage of shared files that are rarely changed, paging activity in virtual-memory management, and caching of remotely accessed data.

As compared to the workstation model, the workstation-server model has several advantages:

1. In general, it is much cheaper to use a few minicomputers equipped with large, fast disks that are accessed over the network than a large number of diskful workstations, with each workstation having a small, slow disk.

2. Diskless workstations are also preferred to diskful workstations from a system maintenance point of view. Backup and hardware maintenance are easier to perform with a few large disks than with many small disks scattered all over a building or campus. Furthermore, installing new releases of software (such as a file server with new functionalities) is easier when the software is to be installed on a few file server machines than on every workstation.

3. In the workstation-server model, since all files are managed by the file servers, users have the flexibility to use any workstation and access the files in the same manner irrespective of which workstation the user is currently logged on. Note that this is not true with the workstation model, in which each workstation has its local file system, because different mechanisms are needed to access local and remote files.

4. In the workstation-server model, the request-response protocol described above is mainly used to access the services of the server machines. Therefore, unlike the workstation model, this model does not need a process migration facility, which is difficult to implement.

The request-response protocol is known as the *client-server model* of communication. In this model, a client process (which in this case resides on a workstation) sends a

request to a server process (which in this case resides on a minicomputer) for getting some service such as reading a block of a file. The server executes the request and sends back a reply to the client that contains the result of request processing.

The client-server model provides an effective general-purpose approach to the sharing of information and resources in distributed computing systems. It is not only meant for use with the workstation-server model but also can be implemented in a variety of hardware and software environments. The computers used to run the client and server processes need not necessarily be workstations and minicomputers. They can be of many types and there is no need to distinguish between them. It is even possible for both the client and server processes to be run on the same computer. Moreover, some processes are both client and server processes. That is, a server process may use the services of another server, appearing as a client to the latter.

5. A user has guaranteed response time because workstations are not used for executing remote processes. However, the model does not utilize the processing capability of idle workstations.

The V-System [Cheriton 1988] is an example of a distributed computing system that is based on the workstation-server model.

1.3.4 Processor-Pool Model

The *processor-pool model* is based on the observation that most of the time a user does not need any computing power but once in a while he or she may need a very large amount of computing power for a short time (e.g., when recompiling a program consisting of a large number of files after changing a basic shared declaration). Therefore, unlike the workstation-server model in which a processor is allocated to each user, in the processor-pool model the processors are pooled together to be shared by the users as needed. The pool of processors consists of a large number of microcomputers and minicomputers attached to the network. Each processor in the pool has its own memory to load and run a system program or an application program of the distributed computing system.

As shown in Figure 1.5, in the pure processor-pool model, the processors in the pool have no terminals attached directly to them, and users access the system from terminals that are attached to the network via special devices. These terminals are either small diskless workstations or graphic terminals, such as X terminals. A special server (called a *run server*) manages and allocates the processors in the pool to different users on a demand basis. When a user submits a job for computation, an appropriate number of processors are temporarily assigned to his or her job by the run server. For example, if the user's computation job is the compilation of a program having n segments, in which each of the segments can be compiled independently to produce separate relocatable object files, n processors from the pool can be allocated to this job to compile all the n segments in parallel. When the computation is completed, the processors are returned to the pool for use by other users.

In the processor-pool model there is no concept of a home machine. That is, a user does not log onto a particular machine but to the system as a whole. This is in contrast

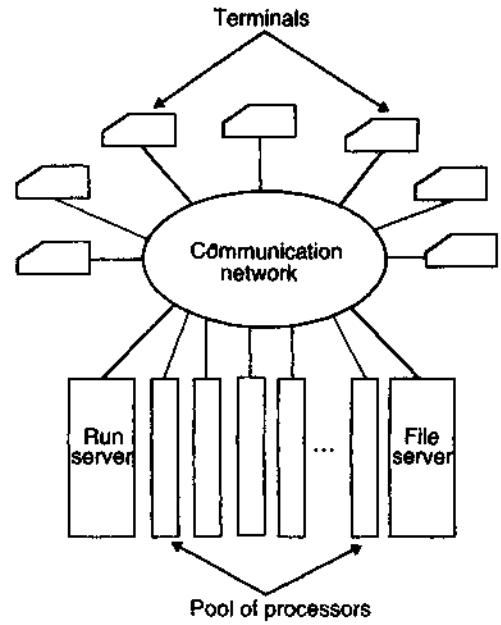


Fig. 1.5 A distributed computing system based on the processor-pool model.

to other models in which each user has a home machine (e.g., a workstation or minicomputer) onto which he or she logs and runs most of his or her programs there by default.

As compared to the workstation-server model, the processor-pool model allows better utilization of the available processing power of a distributed computing system. This is because in the processor-pool model, the entire processing power of the system is available for use by the currently logged-on users, whereas this is not true for the workstation-server model in which several workstations may be idle at a particular time but they cannot be used for processing the jobs of other users. Furthermore, the processor-pool model provides greater flexibility than the workstation-server model in the sense that the system's services can be easily expanded without the need to install any more computers; the processors in the pool can be allocated to act as extra servers to carry any additional load arising from an increased user population or to provide new services. However, the processor-pool model is usually considered to be unsuitable for high-performance interactive applications, especially those using graphics or window systems. This is mainly because of the slow speed of communication between the computer on which the application program of a user is being executed and the terminal via which the user is interacting with the system. The workstation-server model is generally considered to be more suitable for such applications.

Amoeba [Mullender et al. 1990], Plan 9 [Pike et al. 1990], and the Cambridge Distributed Computing System [Needham and Herbert 1982] are examples of distributed computing systems based on the processor-pool model.

1.3.5 Hybrid Model

Out of the four models described above, the workstation-server model, is the most widely used model for building distributed computing systems. This is because a large number of computer users only perform simple interactive tasks such as editing jobs, sending electronic mails, and executing small programs. The workstation-server model is ideal for such simple usage. However, in a working environment that has groups of users who often perform jobs needing massive computation, the processor-pool model is more attractive and suitable.

To combine the advantages of both the workstation-server and processor-pool models, a hybrid model may be used to build a distributed computing system. The hybrid model is based on the workstation-server model but with the addition of a pool of processors. The processors in the pool can be allocated dynamically for computations that are too large for workstations or that require several computers concurrently for efficient execution. In addition to efficient execution of computation-intensive jobs, the hybrid model gives guaranteed response to interactive jobs by allowing them to be processed on local workstations of the users. However, the hybrid model is more expensive to implement than the workstation-server model or the processor-pool model.

1.4 WHY ARE DISTRIBUTED COMPUTING SYSTEMS GAINING POPULARITY?

From the models of distributed computing systems presented above, it is obvious that distributed computing systems are much more complex and difficult to build than traditional *centralized systems* (those consisting of a single CPU, its memory, peripherals, and one or more terminals). The increased complexity is mainly due to the fact that in addition to being capable of effectively using and managing a very large number of distributed resources, the system software of a distributed computing system should also be capable of handling the communication and security problems that are very different from those of centralized systems. For example, the performance and reliability of a distributed computing system depends to a great extent on the performance and reliability of the underlying communication network. Special software is usually needed to handle loss of messages during transmission across the network or to prevent overloading of the network, which degrades the performance and responsiveness to the users. Similarly, special software security measures are needed to protect the widely distributed shared resources and services against intentional or accidental violation of access control and privacy constraints.

Despite the increased complexity and the difficulty of building distributed computing systems, the installation and use of distributed computing systems are rapidly increasing. This is mainly because the advantages of distributed computing systems outweigh their disadvantages. The technical needs, the economic pressures, and the major advantages that have led to the emergence and popularity of distributed computing systems are described next.

1.4.1 Inherently Distributed Applications

Distributed computing systems come into existence in some very natural ways. For example, several applications are inherently distributed in nature and require a distributed computing system for their realization. For instance, in an employee database of a nationwide organization, the data pertaining to a particular employee are generated at the employee's branch office, and in addition to the global need to view the entire database, there is a local need for frequent and immediate access to locally generated data at each branch office. Applications such as these require that some processing power be available at the many distributed locations for collecting, preprocessing, and accessing data, resulting in the need for distributed computing systems. Some other examples of inherently distributed applications are a computerized worldwide airline reservation system, a computerized banking system in which a customer can deposit/withdraw money from his or her account from any branch of the bank, and a factory automation system controlling robots and machines all along an assembly line.

1.4.2 Information Sharing among Distributed Users

Another reason for the emergence of distributed computing systems was a desire for efficient person-to-person communication facility by sharing information over great distances. In a distributed computing system, information generated by one of the users can be easily and efficiently shared by the users working at other nodes of the system. This facility may be useful in many ways. For example, a project can be performed by two or more users who are geographically far off from each other but whose computers are a part of the same distributed computing system. In this case, although the users are geographically separated from each other, they can work in cooperation, for example, by transferring the files of the project, logging onto each other's remote computers to run programs, and exchanging messages by electronic mail to coordinate the work.

The use of distributed computing systems by a group of users to work cooperatively is known as *computer-supported cooperative working* (CSCW), or *groupware*. Groupware applications depend heavily on the sharing of data objects between programs running on different nodes of a distributed computing system. Groupware is an emerging technology that holds major promise for software developers.

1.4.3 Resource Sharing

Information is not the only thing that can be shared in a distributed computing system. Sharing of software resources such as software libraries and databases as well as hardware resources such as printers, hard disks, and plotters can also be done in a very effective way among all the computers and the users of a single distributed computing system. For example, we saw that in a distributed computing system based on the workstation-server model the workstations may have no disk or only a small disk (10–20 megabytes) for temporary storage, and access to permanent files on a large disk can be provided to all the workstations by a single file server.

1.4.4 Better Price-Performance Ratio

This is one of the most important reasons for the growing popularity of distributed computing systems. With the rapidly increasing power and reduction in the price of microprocessors, combined with the increasing speed of communication networks, distributed computing systems potentially have a much better price-performance ratio than a single large centralized system. For example, we saw how a small number of CPUs in a distributed computing system based on the processor-pool model can be effectively used by a large number of users from inexpensive terminals, giving a fairly high price-performance ratio as compared to either a centralized time-sharing system or a personal computer. Another reason for distributed computing systems to be more cost-effective than centralized systems is that they facilitate resource sharing among multiple computers. For example, a single unit of expensive peripheral devices such as color laser printers, high-speed storage devices, and plotters can be shared among all the computers of the same distributed computing system. If these computers are not linked together with a communication network, each computer must have its own peripherals, resulting in higher cost.

1.4.5 Shorter Response Times and Higher Throughput

Due to multiplicity of processors, distributed computing systems are expected to have better performance than single-processor centralized systems. The two most commonly used performance metrics are response time and throughput of user processes. That is, the multiple processors of a distributed computing system can be utilized properly for providing shorter response times and higher throughput than a single-processor centralized system. For example, if there are two different programs to be run, two processors are evidently more powerful than one because the programs can be simultaneously run on different processors. Furthermore, if a particular computation can be partitioned into a number of subcomputations that can run concurrently, in a distributed computing system all the subcomputations can be simultaneously run with each one on a different processor. Distributed computing systems with very fast communication networks are increasingly being used as parallel computers to solve single complex problems rapidly. Another method often used in distributed computing systems for achieving better overall performance is to distribute the load more evenly among the multiple processors by moving jobs from currently overloaded processors to lightly loaded ones. For example, in a distributed computing system based on the workstation model, if a user currently has two processes to run, out of which one is an interactive process and the other is a process that can be run in the background, it may be advantageous to run the interactive process on the home node of the user and the other one on a remote idle node (if any node is idle).

1.4.6 Higher Reliability

Reliability refers to the degree of tolerance against errors and component failures in a system [Stankovic 1984]. A reliable system prevents loss of information even in the event of component failures. The multiplicity of storage devices and processors in a distributed computing system allows the maintenance of multiple copies of critical information within

the system and the execution of important computations redundantly to protect them against catastrophic failures. With this approach, if one of the processors fails, the computation can be successfully completed at the other processor, and if one of the storage devices fails, the information can still be used from the other storage device. Furthermore, the geographical distribution of the processors and other resources in a distributed computing system limits the scope of failures caused by natural disasters.

An important aspect of reliability is *availability*, which refers to the fraction of time for which a system is available for use. In comparison to a centralized system, a distributed computing system also enjoys the advantage of increased availability. For example, if the processor of a centralized system fails (assuming that it is a single-processor centralized system), the entire system breaks down and no useful work can be performed. However, in the case of a distributed computing system, a few parts of the system can be down without interrupting the jobs of the users who are using the other parts of the system. For example, if a workstation of a distributed computing system that is based on the workstation-server model fails, only the user of that workstation is affected. Other users of the system are not affected by this failure. Similarly, in a distributed computing system based on the processor-pool model, if some of the processors in the pool are down at any moment, the system can continue to function normally, simply with some loss in performance that is proportional to the number of processors that are down. In this case, none of the users is affected and the users cannot even know that some of the processors are down.

The advantage of higher reliability is an important reason for the use of distributed computing systems for critical applications whose failure may be disastrous. However, often reliability comes at the cost of performance. Therefore, it is necessary to maintain a balance between the two.

1.4.7 Extensibility and Incremental Growth

Another major advantage of distributed computing systems is that they are capable of incremental growth. That is, it is possible to gradually extend the power and functionality of a distributed computing system by simply adding additional resources (both hardware and software) to the system as and when the need arises. For example, additional processors can be easily added to the system to handle the increased workload of an organization that might have resulted from its expansion. Incremental growth is a very attractive feature because for most existing and proposed applications it is practically impossible to predict future demands of the system. Extensibility is also easier in a distributed computing system because addition of new resources to an existing system can be performed without significant disruption of the normal functioning of the system. Properly designed distributed computing systems that have the property of extensibility and incremental growth are called *open distributed systems*.

1.4.8 Better Flexibility in Meeting Users' Needs

Different types of computers are usually more suitable for performing different types of computations. For example, computers with ordinary power are suitable for ordinary data processing jobs, whereas high-performance computers are more suitable for complex

mathematical computations. In a centralized system, the users have to perform all types of computations on the only available computer. However, a distributed computing system may have a pool of different types of computers, in which case the most appropriate one can be selected for processing a user's job depending on the nature of the job. For instance, we saw that in a distributed computing system that is based on the hybrid model, interactive jobs can be processed at a user's own workstation and the processors in the pool may be used to process noninteractive, computation-intensive jobs.

Note that the advantages of distributed computing systems mentioned above are not achieved automatically but depend on the careful design of a distributed computing system. This book deals with the various design methodologies that may be used to achieve these advantages.

1.5 WHAT IS A DISTRIBUTED OPERATING SYSTEM?

Tanenbaum and Van Renesse [1985] define an *operating system* as a program that controls the resources of a computer system and provides its users with an interface or virtual machine that is more convenient to use than the bare machine. According to this definition, the two primary tasks of an operating system are as follows:

1. To present users with a virtual machine that is easier to program than the underlying hardware.
2. To manage the various resources of the system. This involves performing such tasks as keeping track of who is using which resource, granting resource requests, accounting for resource usage, and mediating conflicting requests from different programs and users.

Therefore, the users' view of a computer system, the manner in which the users access the various resources of the computer system, and the ways in which the resource requests are granted depend to a great extent on the operating system of the computer system. The operating systems commonly used for distributed computing systems can be broadly classified into two types—*network operating systems* and *distributed operating systems*. The three most important features commonly used to differentiate between these two types of operating systems are system image, autonomy, and fault tolerance capability. These features are explained below.

1. *System image.* The most important feature used to differentiate between the two types of operating systems is the image of the distributed computing system from the point of view of its users. In case of a network operating system, the users view the distributed computing system as a collection of distinct machines connected by a communication subsystem. That is, the users are aware of the fact that multiple computers are being used. On the other hand, a distributed operating system hides the existence of multiple computers and provides a single-system image to its users. That is, it makes a collection

of networked machines act as a *virtual uniprocessor*. The difference between the two types of operating systems based on this feature can be best illustrated with the help of examples. Two such examples are presented below.

In the case of a network operating system, although a user can run a job on any machine of the distributed computing system, he or she is fully aware of the machine on which his or her job is executed. This is because, by default, a user's job is executed on the machine on which the user is currently logged. If the user wants to execute a job on a different machine, he or she should either log on to that machine by using some kind of "remote login" command or use a special command for remote execution to specify the machine on which the job is to be executed. In either case, the user knows the machine on which the job is executed. On the other hand, a distributed operating system dynamically and automatically allocates jobs to the various machines of the system for processing. Therefore, a user of a distributed operating system generally has no knowledge of the machine on which a job is executed. That is, the selection of a machine for executing a job is entirely manual in the case of network operating systems but is automatic in the case of distributed operating systems.

With a network operating system, a user is generally required to know the location of a resource to access it, and different sets of system calls have to be used for accessing local and remote resources. On the other hand, users of a distributed operating system need not keep track of the locations of various resources for accessing them, and the same set of system calls is used for accessing both local and remote resources. For instance, users of a network operating system are usually aware of where each of their files is stored and must use explicit *file transfer* commands for moving a file from one machine to another, but the users of a distributed operating system have no knowledge of the location of their files within the system and use the same command to access a file irrespective of whether it is on the local machine or on a remote machine. That is, control over file placement is done manually by the users in a network operating system but automatically by the system in a distributed operating system.

Notice that the key concept behind this feature is "transparency." We will see later in this chapter that a distributed operating system has to support several forms of transparency to achieve the goal of providing a single-system image to its users. Moreover, it is important to note here that with the current state of the art in distributed operating systems, this goal is not fully achievable. Researchers are still working hard to achieve this goal.

2. *Autonomy.* A network operating system is built on a set of existing centralized operating systems and handles the interfacing and coordination of remote operations and communications between these operating systems. That is, in the case of a network operating system, each computer of the distributed computing system has its own local operating system (the operating systems of different computers may be the same or different), and there is essentially no coordination at all among the computers except for the rule that when two processes of different computers communicate with each other, they must use a mutually agreed on communication protocol. Each computer functions independently of other computers in the sense that each one makes independent decisions about the creation and termination of their own processes and management of local

resources. Notice that due to the possibility of difference in local operating systems, the system calls for different computers of the same distributed computing system may be different in this case.

On the other hand, with a distributed operating system, there is a single systemwide operating system and each computer of the distributed computing system runs a part of this global operating system. The distributed operating system tightly interweaves all the computers of the distributed computing system in the sense that they work in close cooperation with each other for the efficient and effective utilization of the various resources of the system. That is, processes and several resources are managed globally (some resources are managed locally). Moreover, there is a single set of globally valid system calls available on all computers of the distributed computing system.

The set of system calls that an operating system supports are implemented by a set of programs called the *kernel* of the operating system. The kernel manages and controls the hardware of the computer system to provide the facilities and resources that are accessed by other programs through system calls. To make the same set of system calls globally valid, with a distributed operating system identical kernels are run on all the computers of a distributed computing system. The kernels of different computers often cooperate with each other in making global decisions, such as finding the most suitable machine for executing a newly created process in the system.

In short, it can be said that the degree of autonomy of each machine of a distributed computing system that uses a network operating system is considerably high as compared to that of machines of a distributed computing system that uses a distributed operating system.

3. *Fault tolerance capability.* A network operating system provides little or no fault tolerance capability in the sense that if 10% of the machines of the entire distributed computing system are down at any moment, at least 10% of the users are unable to continue with their work. On the other hand, with a distributed operating system, most of the users are normally unaffected by the failed machines and can continue to perform their work normally, with only a 10% loss in performance of the entire distributed computing system. Therefore, the fault tolerance capability of a distributed operating system is usually very high as compared to that of a network operating system.

The following definition of a distributed operating system given by Tanenbaum and Van Renesse [1985] covers most of its features mentioned above:

> A distributed operating system is one that looks to its users like an ordinary centralized operating system but runs on multiple, independent central processing units (CPUs). The key concept here is transparency. In other words, the use of multiple processors should be invisible (transparent) to the user. Another way of expressing the same idea is to say that the user views the system as a "virtual uniprocessor," not as a collection of distinct machines. [P. 419].

A distributed computing system that uses a network operating system is usually referred to as a *network system*, whereas one that uses a distributed operating system is

usually referred to as a *true distributed system* (or simply a distributed system). In this book, the term *distributed system* will be used to mean a true distributed system.

Note that with the current state of the art in distributed operating systems, it is not possible to design a completely true distributed system. Completely true distributed systems are the ultimate goal of researchers working in the area of distributed operating systems.

1.6 ISSUES IN DESIGNING A DISTRIBUTED OPERATING SYSTEM

In general, designing a distributed operating system is more difficult than designing a centralized operating system for several reasons. In the design of a centralized operating system, it is assumed that the operating system has access to complete and accurate information about the environment in which it is functioning. For example, a centralized operating system can request status information, being assured that the interrogated component will not change state while awaiting a decision based on that status information, since only the single operating system asking the question may give commands. However, a distributed operating system must be designed with the assumption that complete information about the system environment will never be available. In a distributed system, the resources are physically separated, there is no common clock among the multiple processors, delivery of messages is delayed, and messages could even be lost. Due to all these reasons, a distributed operating system does not have up-to-date, consistent knowledge about the state of the various components of the underlying distributed system. Obviously, lack of up-to-date and consistent information makes many things (such as management of resources and synchronization of cooperating activities) much harder in the design of a distributed operating system. For example, it is hard to schedule the processors optimally if the operating system is not sure how many of them are up at the moment.

Despite these complexities and difficulties, a distributed operating system must be designed to provide all the advantages of a distributed system to its users. That is, the users should be able to view a distributed system as a virtual centralized system that is flexible, efficient, reliable, secure, and easy to use. To meet this challenge, the designers of a distributed operating system must deal with several design issues. Some of the key design issues are described below. The rest of the chapters of this book basically contain detailed descriptions of these design issues and the commonly used techniques to deal with them.

1.6.1 Transparency

We saw that one of the main goals of a distributed operating system is to make the existence of multiple computers invisible (transparent) and provide a single system image to its users. That is, a distributed operating system must be designed in such a way that a collection of distinct machines connected by a communication subsystem

appears to its users as a virtual uniprocessor. Achieving complete transparency is a difficult task and requires that several different aspects of transparency be supported by the distributed operating system. The eight forms of transparency identified by the International Standards Organization's Reference Model for Open Distributed Processing [ISO 1992] are access transparency, location transparency, replication transparency, failure transparency, migration transparency, concurrency transparency, performance transparency, and scaling transparency. These transparency aspects are described below.

Access Transparency

Access transparency means that users should not need or be able to recognize whether a resource (hardware or software) is remote or local. This implies that the distributed operating system should allow users to access remote resources in the same way as local resources. That is, the user interface, which takes the form of a set of system calls, should not distinguish between local and remote resources, and it should be the responsibility of the distributed operating system to locate the resources and to arrange for servicing user requests in a user-transparent manner.

This requirement calls for a well-designed set of system calls that are meaningful in both centralized and distributed environments and a global resource naming facility. We will see in Chapters 3 and 4 that due to the need to handle communication failures in distributed systems, it is not possible to design system calls that provide complete access transparency. However, the area of designing a global resource naming facility has been well researched with considerable success. Chapter 10 deals with the concepts and design of a global resource naming facility. The distributed shared memory mechanism described in Chapter 5 is also meant to provide a uniform set of system calls for accessing both local and remote memory objects. Although this mechanism is quite useful in providing access transparency, it is suitable only for limited types of distributed applications due to its performance limitation.

Location Transparency

The two main aspects of location transparency are as follows:

1. *Name transparency.* This refers to the fact that the name of a resource (hardware or software) should not reveal any hint as to the physical location of the resource. That is, the name of a resource should be independent of the physical connectivity or topology of the system or the current location of the resource. Furthermore, such resources, which are capable of being moved from one node to another in a distributed system (such as a file), must be allowed to move without having their names changed. Therefore, resource names must be unique systemwide.

2. *User mobility.* This refers to the fact that no matter which machine a user is logged onto, he or she should be able to access a resource with the same name. That is, the user should not be required to use different names to access the same resource from two

different nodes of the system. In a distributed system that supports user mobility, users can freely log on to any machine in the system and access any resource without making any extra effort.

Both name transparency and user mobility requirements call for a systemwide, global resource naming facility.

Replication Transparency

For better performance and reliability, almost all distributed operating systems have the provision to create replicas (additional copies) of files and other resources on different nodes of the distributed system. In these systems, both the existence of multiple copies of a replicated resource and the replication activity should be transparent to the users. That is, two important issues related to replication transparency are naming of replicas and replication control. It is the responsibility of the system to name the various copies of a resource and to map a user-supplied name of the resource to an appropriate replica of the resource. Furthermore, replication control decisions such as how many copies of the resource should be created, where should each copy be placed, and when should a copy be created/deleted should be made entirely automatically by the system in a user-transparent manner. Replica management issues are described in Chapter 9.

Failure Transparency

Failure transparency deals with masking from the users' partial failures in the system, such as a communication link failure, a machine failure, or a storage device crash. A distributed operating system having failure transparency property will continue to function, perhaps in a degraded form, in the face of partial failures. For example, suppose the file service of a distributed operating system is to be made failure transparent. This can be done by implementing it as a group of file servers that closely cooperate with each other to manage the files of the system and that function in such a manner that the users can utilize the file service even if only one of the file servers is up and working. In this case, the users cannot notice the failure of one or more file servers, except for slower performance of file access operations. Any type of service can be implemented in this way for failure transparency. However, in this type of design, care should be taken to ensure that the cooperation among multiple servers does not add too much overhead to the system.

Complete failure transparency is not achievable with the current state of the art in distributed operating systems because all types of failures cannot be handled in a user-transparent manner. For example, failure of the communication network of a distributed system normally disrupts the work of its users and is noticeable by the users. Moreover, an attempt to design a completely failure-transparent distributed system will result in a very slow and highly expensive system due to the large amount of redundancy required for tolerating all types of failures. The design of such a distributed system, although theoretically possible, is not practically justified.

Migration Transparency

For better performance, reliability, and security reasons, an object that is capable of being moved (such as a process or a file) is often migrated from one node to another in a distributed system. The aim of migration transparency is to ensure that the movement of the object is handled automatically by the system in a user-transparent manner. Three important issues in achieving this goal are as follows:

1. Migration decisions such as which object is to be moved from where to where should be made automatically by the system.
2. Migration of an object from one node to another should not require any change in its name.
3. When the migrating object is a process, the interprocess communication mechanism should ensure that a message sent to the migrating process reaches it without the need for the sender process to resend it if the receiver process moves to another node before the message is received.

Chapter 7 deals with the first issue. The second issue calls for a global resource naming facility, which is described in Chapter 10. Ways to handle the third issue are described in Chapter 8.

Concurrency Transparency

In a distributed system, multiple users who are spatially separated use the system concurrently. In such a situation, it is economical to share the system resources (hardware or software) among the concurrently executing user processes. However, since the number of available resources in a computing system is restricted, one user process must necessarily influence the action of other concurrently executing user processes, as it competes for resources. For example, concurrent update to the same file by two different processes should be prevented. Concurrency transparency means that each user has a feeling that he or she is the sole user of the system and other users do not exist in the system. For providing concurrency transparency, the resource sharing mechanisms of the distributed operating system must have the following four properties:

1. An event-ordering property ensures that all access requests to various system resources are properly ordered to provide a consistent view to all users of the system.

2. A mutual-exclusion property ensures that at any time at most one process accesses a shared resource, which must not be used simultaneously by multiple processes if program operation is to be correct.

3. A no-starvation property ensures that if every process that is granted a resource, which must not be used simultaneously by multiple processes, eventually releases it, every request for that resource is eventually granted.

4. A no-deadlock property ensures that a situation will never occur in which competing processes prevent their mutual progress even though no single one requests more resources than available in the system.

Chapter 6 deals with the above-mentioned issues of concurrency transparency.

Performance Transparency

The aim of performance transparency is to allow the system to be automatically reconfigured to improve performance, as loads vary dynamically in the system. As far as practicable, a situation in which one processor of the system is overloaded with jobs while another processor is idle should not be allowed to occur. That is, the processing capability of the system should be uniformly distributed among the currently available jobs in the system.

This requirement calls for the support of intelligent resource allocation and process migration facilities in distributed operating systems. Chapters 7 and 8 deal with these two issues.

Scaling Transparency

The aim of scaling transparency is to allow the system to expand in scale without disrupting the activities of the users. This requirement calls for open-system architecture and the use of scalable algorithms for designing the distributed operating system components. Section 1.6.3 of this chapter and Section 2.6 of Chapter 2 focus on the issues of designing an open distributed system. On the other hand, since every component of a distributed operating system must use scalable algorithms, this issue has been dealt with in almost all chapters of the book.

1.6.2 Reliability

In general, distributed systems are expected to be more reliable than centralized systems due to the existence of multiple instances of resources. However, the existence of multiple instances of the resources alone cannot increase the system's reliability. Rather, the distributed operating system, which manages these resources, must be designed properly to increase the system's reliability by taking full advantage of this characteristic feature of a distributed system.

A *fault* is a mechanical or algorithmic defect that may generate an error. A fault in a system causes system failure. Depending on the manner in which a failed system behaves, system failures are of two types—fail-stop [Schlichting and Schneider 1983] and Byzantine [Lamport et al. 1982]. In the case of *fail-stop failure*, the system stops functioning after changing to a state in which its failure can be detected. On the other hand, in the case of *Byzantine failure*, the system continues to function but produces wrong results. Undetected software bugs often cause Byzantine failure of a system. Obviously, Byzantine failures are much more difficult to deal with than fail-stop failures.

For higher reliability, the fault-handling mechanisms of a distributed operating system must be designed properly to avoid faults, to tolerate faults, and to detect and

recover from faults. Commonly used methods for dealing with these issues are briefly described next.

Fault Avoidance

Fault avoidance deals with designing the components of the system in such a way that the occurrence of faults is minimized. Conservative design practices such as using high-reliability components are often employed for improving the system's reliability based on the idea of fault avoidance. Although a distributed operating system often has little or no role to play in improving the fault avoidance capability of a hardware component, the designers of the various software components of the distributed operating system must test them thoroughly to make these components highly reliable.

Fault Tolerance

Fault tolerance is the ability of a system to continue functioning in the event of partial system failure. The performance of the system might be degraded due to partial failure, but otherwise the system functions properly. Some of the important concepts that may be used to improve the fault tolerance ability of a distributed operating system are as follows:

1. *Redundancy techniques.* The basic idea behind redundancy techniques is to avoid single points of failure by replicating critical hardware and software components, so that if one of them fails, the others can be used to continue. Obviously, having two or more copies of a critical component makes it possible, at least in principle, to continue operations in spite of occasional partial failures. For example, a critical process can be simultaneously executed on two nodes so that if one of the two nodes fails, the execution of the process can be completed at the other node. Similarly, a critical file may be replicated on two or more storage devices for better reliability.

Notice that with redundancy techniques additional system overhead is needed to maintain two or more copies of a replicated resource and to keep all the copies of a resource consistent. For example, if a file is replicated on two or more nodes of a distributed system, additional disk storage space is required, and for correct functioning, it is often necessary that all the copies of the file are mutually consistent. In general, the larger is the number of copies kept, the better is the reliability but the larger is the system overhead involved. Therefore, a distributed operating system must be designed to maintain a proper balance between the degree of reliability and the incurred overhead. This raises an important question: How much replication is enough? For an answer to this question, note that a system is said to be *k-fault tolerant* if it can continue to function even in the event of the failure of k components [Cristian 1991, Nelson 1990]. Therefore, if the system is to be designed to tolerate k fail–stop failures, $k+1$ replicas are needed. If k replicas are lost due to failures, the remaining one replica can be used for continued functioning of the system. On the other hand, if the system is to be designed to tolerate k Byzantine failures, a minimum of $2k+1$ replicas are needed. This is because a voting mechanism can be used to believe the majority $k+1$ of the replicas when k replicas behave abnormally.

Replication and consistency control mechanisms for memory objects are described in Chapter 5 and for file objects are described in Chapter 9.

Another application of redundancy technique is in the design of a stable storage device, which is a virtual storage device that can even withstand transient I/O faults and decay of the storage media. The reliability of a critical file may be improved by storing it on a stable storage device. Stable storage devices are described in Chapter 9.

2. *Distributed control.* For better reliability, many of the particular algorithms or protocols used in a distributed operating system must employ a distributed control mechanism to avoid single points of failure. For example, a highly available distributed file system should have multiple and independent file servers controlling multiple and independent storage devices. In addition to file servers, a distributed control technique could also be used for name servers, scheduling algorithms, and other executive control functions. It is important to note here that when multiple distributed servers are used in a distributed system to provide a particular type of service, the servers must be independent. That is, the design must not require simultaneous functioning of the servers; otherwise, the reliability will become worse instead of getting better. Distributed control mechanisms are described throughout this book.

Fault Detection and Recovery

The fault detection and recovery method of improving reliability deals with the use of hardware and software mechanisms to determine the occurrence of a failure and then to correct the system to a state acceptable for continued operation. Some of the commonly used techniques for implementing this method in a distributed operating system are as follows:

1. *Atomic transactions.* An atomic transaction (or just *transaction* for short) is a computation consisting of a collection of operations that take place indivisibly in the presence of failures and concurrent computations. That is, either all of the operations are performed successfully or none of their effects prevails, and other processes executing concurrently cannot modify or observe intermediate states of the computation. Transactions help to preserve the consistency of a set of shared data objects (e.g., files) in the face of failures and concurrent access. They make crash recovery much easier, because a transaction can only end in two states: Either all the operations of the transaction are performed or none of the operations of the transaction is performed.

In a system with transaction facility, if a process halts unexpectedly due to a hardware fault or a software error before a transaction is completed, the system subsequently restores any data objects that were undergoing modification to their original states. Notice that if a system does not support a transaction mechanism, unexpected failure of a process during the processing of an operation may leave the data objects that were undergoing modification in an inconsistent state. Therefore, without transaction facility, it may be difficult or even impossible in some cases to roll back (recover) the data objects from their current inconsistent states to their original states. Atomic transaction mechanisms are described in Chapter 9.

2. *Stateless servers.* The client-server model is frequently used in distributed systems to service user requests. In this model, a server may be implemented by using any one of

the following two service paradigms—stateful or stateless. The two paradigms are distinguished by one aspect of the client-server relationship, whether or not the history of the serviced requests between a client and a server affects the execution of the next service request. The stateful approach does depend on the history of the serviced requests, but the stateless approach does not depend on it. Stateless servers have a distinct advantage over stateful servers in the event of a failure. That is, the stateless service paradigm makes crash recovery very easy because no client state information is maintained by the server. On the other hand, the stateful service paradigm requires complex crash recovery procedures. Both the client and server need to reliably detect crashes. The server needs to detect client crashes so that it can discard any state it is holding for the client, and the client must detect server crashes so that it can perform necessary error-handling activities. Although stateful service becomes necessary in some cases, to simplify the failure detection and recovery actions, the stateless service paradigm must be used wherever possible. Stateless and stateful servers are described in Chapters 4 and 9.

3. *Acknowledgments and timeout-based retransmissions of messages.* In a distributed system, events such as a node crash or a communication link failure may interrupt a communication that was in progress between two processes, resulting in the loss of a message. Therefore, a reliable interprocess communication mechanism must have ways to detect lost messages so that they can be retransmitted. Handling of lost messages usually involves return of acknowledgment messages and retransmissions on the basis of timeouts. That is, the receiver must return an acknowledgment message for every message received, and if the sender does not receive any acknowledgment for a message within a fixed timeout period, it assumes that the message was lost and retransmits the message. A problem associated with this approach is that of duplicate messages. Duplicate messages may be sent in the event of failures or because of timeouts. Therefore, a reliable interprocess communication mechanism should also be capable of detecting and handling duplicate messages. Handling of duplicate messages usually involves a mechanism for automatically generating and assigning appropriate sequence numbers to messages. Use of acknowledgment messages, timeout-based retransmissions of messages, and handling of duplicate request messages for reliable communication are described in Chapter 3.

The mechanisms described above may be employed to create a very reliable distributed system. However, the main drawback of increased system reliability is potential loss of execution time efficiency due to the extra overhead involved in these techniques. For many systems it is just too costly to incorporate a large number of reliability mechanisms. Therefore, the major challenge for distributed operating system designers is to integrate these mechanisms in a cost-effective manner for producing a reliable system.

1.6.3 Flexibility

Another important issue in the design of distributed operating systems is flexibility. Flexibility is the most important feature for open distributed systems. The design of a distributed operating system should be flexible due to the following reasons:

1. *Ease of modification.* From the experience of system designers, it has been found that some parts of the design often need to be replaced/modified either because some bug is detected in the design or because the design is no longer suitable for the changed system environment or new-user requirements. Therefore, it should be easy to incorporate changes in the system in a user-transparent manner or with minimum interruption caused to the users.

2. *Ease of enhancement.* In every system, new functionalities have to be added from time to time to make it more powerful and easy to use. Therefore, it should be easy to add new services to the system. Furthermore, if a group of users do not like the style in which a particular service is provided by the operating system, they should have the flexibility to add and use their own service that works in the style with which the users of that group are more familiar and feel more comfortable.

The most important design factor that influences the flexibility of a distributed operating system is the model used for designing its kernel. The *kernel* of an operating system is its central controlling part that provides basic system facilities. It operates in a separate address space that is inaccessible to user processes. It is the only part of an operating system that a user cannot replace or modify. We saw that in the case of a distributed operating system identical kernels are run on all the nodes of the distributed system.

The two commonly used models for kernel design in distributed operating systems are the monolithic kernel and the microkernel (Fig. 1.6). In the *monolithic kernel* model, most operating system services such as process management, memory management, device management, file management, name management, and interprocess communication are provided by the kernel. As a result, the kernel has a large, monolithic structure. Many distributed operating systems that are extensions or imitations of the UNIX operating system use the monolithic kernel model. This is mainly because UNIX itself has a large, monolithic kernel.

On the other hand, in the *microkernel* model, the main goal is to keep the kernel as small as possible. Therefore, in this model, the kernel is a very small nucleus of software that provides only the minimal facilities necessary for implementing additional operating system services. The only services provided by the kernel in this model are interprocess communication, low-level device management, a limited amount of low-level process management, and some memory management. All other operating system services, such as file management, name management, additional process, and memory management activities, and much system call handling are implemented as user-level server processes. Each server process has its own address space and can be programmed separately.

As compared to the monolithic kernel model, the microkernel model has several advantages. In the monolithic kernel model, the large size of the kernel reduces the overall flexibility and configurability of the resulting operating system. On the other hand, the resulting operating system of the microkernel model is highly modular in nature. Due to this characteristic feature, the operating system of the microkernel model is easy to design, implement, and install. Moreover, since most of the services are implemented as user-level server processes, it is also easy to modify the design or add new services. This also allows

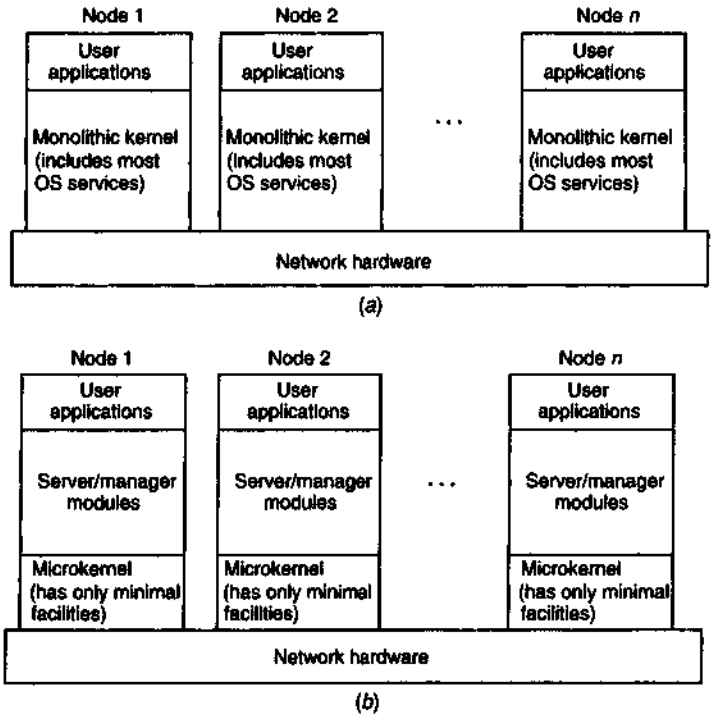


Fig. 1.6 Models of kernel design in distributed operating systems: (*a*) The monolithic kernel model. The level above the kernel level can be partitioned by the users into whatever hierarchical levels are appropriate. (*b*) The microkernel model. Although the figure shows a two-level heirarchy above the kernel level, users can extend the hierarchy to whatever levels are appropriate.

those users who do not like a particular service provided by the operating system to implement and use their own service. Furthermore, for adding or changing a service, there is no need to stop the system and boot a new kernel, as in the case of a monolithic kernel. Therefore, changes in the system can be incorporated without interrupting the users.

The modular design of a system based on the microkernel model, however, is potentially subject to a performance penalty. This is because in the microkernel model each server is an independent process having its own address space. Therefore, the servers have to use some form of message-based interprocess communication mechanism to communicate with each other while performing some job. Furthermore, message passing between server processes and the microkernel requires context switches, resulting in additional performance overhead. In the monolithic kernel model, however, since all the services are provided by the kernel, the same address space is shared by all of them. Therefore, no message passing and no context switching are required while the kernel is performing the job. Hence a request may be serviced faster in the monolithic kernel model than in the microkernel model.

In spite of its potential performance cost, the microkernel model is being preferred for the design of modern distributed operating systems. The two main reasons for this are as follows:

1. The advantages of the microkernel model more than compensate for the performance cost. Notice that the situation here is very similar to the one that caused high-level programming languages to be preferred to assembly languages. In spite of the better performance of programs written in assembly languages, most programs are written in high-level languages due to the advantages of ease of design, maintenance, and portability. Similarly, the flexibility advantages of the microkernel model previously described more than outweigh its small performance penalty.

2. Some experimental results have shown that although in theory the microkernel model seems to have poorer performance than the monolithic kernel model, this is not true in practice. This is because other factors tend to dominate, and the small overhead involved in exchanging messages is usually negligible [Douglis et al. 1991].

Details of several distributed operating systems whose design is based on the microkernel model are presented in Chapter 12.

1.6.4 Performance

If a distributed system is to be used, its performance must be at least as good as a centralized system. That is, when a particular application is run on a distributed system, its overall performance should be better than or at least equal to that of running the same application on a single-processor system. However, to achieve this goal, it is important that the various components of the operating system of a distributed system be designed properly; otherwise, the overall performance of the distributed system may turn out to be worse than a centralized system. Some design principles considered useful for better performance are as follows:

1. *Batch if possible.* Batching often helps in improving performance greatly. For example, transfer of data across the network in large chunks rather than as individual pages is much more efficient. Similarly, piggybacking of acknowledgment of previous messages with the next message during a series of messages exchanged between two communicating entities also improves performance.

2. *Cache whenever possible.* Caching of data at clients' sites frequently improves overall system performance because it makes data available wherever it is being currently used, thus saving a large amount of computing time and network bandwidth. In addition, caching reduces contention on centralized resources.

3. *Minimize copying of data.* Data copying overhead (e.g., moving data in and out of buffers) involves a substantial CPU cost of many operations. For example, while being transferred from its sender to its receiver, a message data may take the following path on the sending side:

(a) From sender's stack to its message buffer
(b) From the message buffer in the sender's address space to the message buffer in the kernel's address space
(c) Finally, from the kernel to the network interface board

On the receiving side, the data probably takes a similar path in the reverse direction. Therefore, in this case, a total of six copy operations are involved in the message transfer operation. Similarly, in several systems, the data copying overhead is also large for read and write operations on block I/O devices. Therefore, for better performance, it is desirable to avoid copying of data, although this is not always simple to achieve. Making optimal use of memory management often helps in eliminating much data movement between the kernel, block I/O devices, clients, and servers.

4. *Minimize network traffic.* System performance may also be improved by reducing internode communication costs. For example, accesses to remote resources require communication, possibly through intermediate nodes. Therefore, migrating a process closer to the resources it is using most heavily may be helpful in reducing network traffic in the system if the decreased cost of accessing its favorite resource offsets the possible increased cost of accessing its less favored ones. Another way to reduce network traffic is to use the process migration facility to cluster two or more processes that frequently communicate with each other on the same node of the system. Avoiding the collection of global state information for making some decision also helps in reducing network traffic.

5. *Take advantage of fine-grain parallelism for multiprocessing.* Performance can also be improved by taking advantage of fine-grain parallelism for multiprocessing. For example, threads (described in Chapter 8) are often used for structuring server processes. Servers structured as a group of threads can operate efficiently because they can simultaneously service requests from several clients. Fine-grained concurrency control of simultaneous accesses by multiple processes to a shared resource is another example of application of this principle for better performance.

Throughout the book we will come across the use of these design principles in the design of the various distributed operating system components.

1.6.5 Scalability

Scalability refers to the capability of a system to adapt to increased service load. It is inevitable that a distributed system will grow with time since it is very common to add new machines or an entire subnetwork to the system to take care of increased workload or organizational changes in a company. Therefore, a distributed operating system should be designed to easily cope with the growth of nodes and users in the system. That is, such growth should not cause serious disruption of service or significant loss of performance to users. Some guiding principles for designing scalable distributed systems are as follows:

1. *Avoid centralized entities.* In the design of a distributed operating system, use of centralized entities such as a single central file server or a single database for the entire system makes the distributed system nonscalable due to the following reasons:

(a) The failure of the centralized entity often brings the entire system down. Hence, the system cannot tolerate faults in a graceful manner.
(b) The performance of the centralized entity often becomes a system bottleneck when contention for it increases with the growing number of users.
(c) Even if the centralized entity has enough processing and storage capacity, the capacity of the network that connects the centralized entity with other nodes of the system often gets saturated when the contention for the entity increases beyond a certain level.
(d) In a wide-area network consisting of several interconnected local-area networks, it is obviously inefficient to always get a particular type of request serviced at a server node that is several gateways away. This also increases network traffic. Local area and wide-area networking concepts are described in Chapter 2.

Therefore, the use of centralized entities should be avoided in the design. Replication of resources and distributed control algorithms are frequently used techniques to achieve this goal. In fact, for better scalability, as far as practicable, a functionally symmetric configuration should be used in which all nodes of the system have a nearly equal role to play in the operation of the system.

2. *Avoid centralized algorithms.* A centralized algorithm is one that operates by collecting information from all nodes, processing this information on a single node and then distributing the results to other nodes. The use of such algorithms in the design of a distributed operating system is also not acceptable from a scalability point of view. The reasons for this are very similar to those mentioned in the use of centralized entities. For example, a scheduling algorithm that makes scheduling decisions by first inquiring from all the nodes and then selecting the most lightly loaded node as a candidate for receiving jobs has poor scalability factor. Such an algorithm may work fine for small networks but gets crippled when applied to large networks. This is because the inquirer receives a very large number of replies almost simultaneously and the time required to process the reply messages for making a host selection is normally too long. Moreover, since the complexity of the algorithm is $O(n^2)$, it creates heavy network traffic and quickly consumes network bandwidth. Therefore, in the design of a distributed operating system, only decentralized algorithms should be used. In these algorithms, global state information of the system is not collected or used, decision at a node is usually based on locally available information, and it is assumed that a systemwide global clock does not exist (the clocks of all nodes are not synchronized).

3. *Perform most operations on client workstations.* If possible, an operation should be performed on the client's own workstation rather than on a server machine. This is because a server is a common resource for several clients, and hence server cycles are more precious than the cycles of client workstations. This principle enhances the scalability of the system, since it allows graceful degradation of system performance as the system grows in size, by reducing contention for shared resources. Caching is a frequently used technique for the realization of this principle.

Throughout the book, we will come across the use of these design principles in the design of the various distributed operating system components.

1.6.6 Heterogeneity

A heterogeneous distributed system consists of interconnected sets of dissimilar hardware or software systems. Because of the diversity, designing heterogenous distributed systems is far more difficult than designing homogeneous distributed systems in which each system is based on the same, or closely related, hardware and software. However, as a consequence of large scale, heterogeneity is often inevitable in distributed systems. Furthermore, often heterogeneity is preferred by many users because heterogeneous distributed systems provide the flexibility to their users of different computer platforms for different applications. For example, a user may have the flexibility of a supercomputer for simulations, a Macintosh for document processing, and a UNIX workstation for program development.

Incompatibilities in a heterogeneous distributed system may be of different types. For example, the internal formatting schemes of different communication and host processors may be different; or when several networks are interconnected via gateways, the communication protocols and topologies of different networks may be different; or the servers operating at different nodes of the system may be different. For instance, some hosts use 32-bit word lengths while others use word lengths of 16 or 64 bits. Byte ordering within these data constructs can vary as well, requiring special converters to enable data sharing between incompatible hosts.

In a heterogeneous distributed system, some form of data translation is necessary for interaction between two incompatible nodes. Some earlier systems left this translation to the users, but this is no longer acceptable. The data translation job may be performed either at the sender's node or at the receiver's node. Suppose this job is performed at the receiver's node. With this approach, at every node there must be a translator to convert each format in the system to the format used on the receiving node. Therefore, if there are n different formats, $n - 1$ pieces of translation software must be supported at each node, resulting in a total of $n(n - 1)$ pieces of translation software in the system. This is undesirable, as adding a new type of format becomes a more difficult task over time. Performing the translation job at the sender's node instead of the receiver's node also suffers from the same drawback.

The software complexity of this translation process can be greatly reduced by using an intermediate standard data format. In this method, an intermediate standard data format is declared, and each node only requires a translation software for converting from its own format to the standard format and from the standard format to its own format. In this case, when two incompatible nodes interact at the sender node, the data to be sent is first converted to the standard format, the data is moved in the format of the standard, and finally, at the receiver node, the data is converted from the standard format to the receiver's format. By choosing the standard format to be the most common format in the system, the number of conversions can be reduced.

Various techniques to deal with heterogeneity in distributed systems are described in Chapters 2, 4, 5, and 8.

1.6.7 Security

In order that the users can trust the system and rely on it, the various resources of a computer system must be protected against destruction and unauthorized access. Enforcing security in a distributed system is more difficult than in a centralized system because of the lack of a single point of control and the use of insecure networks for data communication. In a centralized system, all users are authenticated by the system at login time, and the system can easily check whether a user is authorized to perform the requested operation on an accessed resource. In a distributed system, however, since the client-server model is often used for requesting and providing services, when a client sends a request message to a server, the server must have some way of knowing who is the client. This is not so simple as it might appear because any client identification field in the message cannot be trusted. This is because an intruder (a person or program trying to obtain unauthorized access to system resources) may pretend to be an authorized client or may change the message contents during transmission. Therefore, as compared to a centralized system, enforcement of security in a distributed system has the following additional requirements:

1. It should be possible for the sender of a message to know that the message was received by the intended receiver.
2. It should be possible for the receiver of a message to know that the message was sent by the genuine sender.
3. It should be possible for both the sender and receiver of a message to be guaranteed that the contents of the message were not changed while it was in transfer.

Cryptography (described in Chapter 11) is the only known practical method for dealing with these security aspects of a distributed system. In this method, comprehension of private information is prevented by encrypting the information, which can then be decrypted only by authorized users.

Another guiding principle for security is that a system whose security depends on the integrity of the fewest possible entities is more likely to remain secure as it grows. For example, it is much simpler to ensure security based on the integrity of the much smaller number of servers rather than trusting thousands of clients. In this case, it is sufficient to only ensure the physical security of these servers and the software they run. Chapter 11 deals with the commonly used techniques for designing secure distributed systems.

1.6.8 Emulation of Existing Operating Systems

For commercial success, it is important that a newly designed distributed operating system be able to emulate existing popular operating systems such as UNIX. With this property, new software can be written using the system call interface of the new operating system

to take full advantage of its special features of distribution, but a vast amount of already existing old software can also be run on the same system without the need to rewrite them. Therefore, moving to the new distributed operating system will allow both types of software to be run side by side.

We will see in Chapter 12 how some of the existing distributed operating systems have been designed to support UNIX emulation facility.

1.7 INTRODUCTION TO DISTRIBUTED COMPUTING ENVIRONMENT (DCE)

Chapter 12 of the book presents case studies of four distributed operating systems: Amoeba, V-System, Mach, and Chorus. In addition, examples of key technologies of individual distributed operating system components that have either become or are poised to become de facto international standards are presented in individual chapters wherever such industry examples are available. In particular, the following technologies are presented as case studies:

- Ethernet, IEEE Token Ring, the Internet Protocol suite, and the Internet are presented as case studies of networking technologies in Chapter 2.
- The 4.3BSD UNIX interprocess communication mechanism is presented as a case study of message-passing technology in Chapter 3.
- SUN RPC and DCE RPC are presented as case studies of Remote Procedure Call (RPC) technology in Chapter 4.
- IVY and Munin are presented as case studies of Distributed Shared Memory (DSM) technology in Chapter 5.
- DCE Distributed Time Service (DTS) is presented as a case study of clock synchronization technology in Chapter 6.
- The DCE threads package is presented as a case study of threads technology in Chapter 8.
- DCE Distributed File Service (DFS) is presented as a case study of distributed file system technology in Chapter 9.
- The various components of DCE naming facility are presented as case studies of naming technology in Chapter 10.
- The Kerberos authentication system and DCE Security Service are presented as case studies of security technology in Chapter 11.

Notice from the above list that almost half of the key technologies presented as case studies in the various chapters of this book are tools and services that belong to DCE. This is because of the way in which DCE was created (described next). Therefore, for a better understanding of these key technologies, it will be useful to know something about DCE before going into the details of its key components. This section presents a brief introduction to DCE.

1.7.1 What Is DCE?

A vendor-independent distributed computing environment, DCE was defined by the Open Software Foundation (OSF), a consortium of computer manufacturers, including IBM, DEC, and Hewlett-Packard. It is not an operating system, nor is it an application. Rather, it is an integrated set of services and tools that can be installed as a coherent environment on top of existing operating systems and serve as a platform for building and running distributed applications.

A primary goal of DCE is vendor independence. It runs on many different kinds of computers, operating systems, and networks produced by different vendors. For example, some operating systems to which DCE can be easily ported include OSF/1, AIX, DOMAIN OS, ULTRIX, HP-UX, SINIX, SunOS, UNIX System V, VMS, WINDOWS, and OS/2. On the other hand, it can be used with any network hardware and transport software, including TCP/IP, X.25, as well as other similar products.

As shown in Figure 1.7, DCE is a middleware software layered between the DCE applications layer and the operating system and networking layer. The basic idea is to take a collection of existing machines (possibly from different vendors), interconnect them by a communication network, add the DCE software platform on top of the native operating systems of the machines, and then be able to build and run distributed applications. Each machine has its own local operating system, which may be different from that of other machines. The DCE software layer on top of the operating system and networking layer hides the differences between machines by automatically performing data-type conversions when necessary. Therefore, the heterogeneous nature of the system is transparent to the applications programmers, making their job of writing distributed applications much simpler.

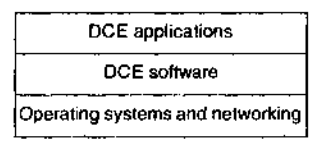


Fig. 1.7 Position of DCE software in a DCE-based distributed system.

1.7.2 How Was DCE Created?

The OSF did not create DCE from scratch. Instead, it created DCE by taking advantage of work already done at universities and industries in the area of distributed computing. For this, OSF issued a request for technology (RFT), asking for tools and services needed to build a coherent distributed computing environment. To be a contender, a primary requirement was that actual working code must ultimately be provided. The submitted bids were carefully evaluated by OSF employees and a team of outside experts. Finally, those tools and services were selected that the members of the evaluation committee believed provided the best solutions. The code comprising the selected tools and services, almost entirely written in C, was

then further developed by OSF to produce a single integrated package that was made available to the world as DCE. Version 1.0 of DCE was released by OSF in January 1992.

1.7.3 DCE Components

As mentioned above, DCE is a blend of various technologies developed independently and nicely integrated by OSF. Each of these technologies forms a component of DCE. The main components of DCE are as follows:

1. *Threads package.* It provides a simple programming model for building concurrent applications. It includes operations to create and control multiple threads of execution in a single process and to synchronize access to global data within an application. Details are given in Chapter 8.

2. *Remote Procedure Call (RPC) facility.* It provides programmers with a number of powerful tools necessary to build client-server applications. In fact, the DCE RPC facility is the basis for all communication in DCE because the programming model underlying all of DCE is the client-server model. It is easy to use, is network- and protocol-independent, provides secure communication between a client and a server, and hides differences in data requirements by automatically converting data to the appropriate forms needed by clients and servers. Details are given in Chapter 4.

3. *Distributed Time Service (DTS).* It closely synchronizes the clocks of all the computers in the system. It also permits the use of time values from external time sources, such as those of the U.S. National Institute for Standards and Technology (NIST), to synchronize the clocks of the computers in the system with external time. This facility can also be used to synchronize the clocks of the computers of one distributed environment with the clocks of the computers of another distributed environment. Details are given in Chapter 6.

4. *Name services.* The name services of DCE include the Cell Directory Service (CDS), the Global Directory Service (GDS), and the Global Directory Agent (GDA). These services allow resources such as servers, files, devices, and so on, to be uniquely named and accessed in a location-transparent manner. Details are given in Chapter 10.

5. *Security Service.* It provides the tools needed for authentication and authorization to protect system resources against illegitimate access. Details are given in Chapter 11.

6. *Distributed File Service (DFS).* It provides a systemwide file system that has such characteristics as location transparency, high performance, and high availability. A unique feature of DCE DFS is that it can also provide file services to clients of other file systems. Details are given in Chapter 9.

The DCE components listed above are tightly integrated. It is difficult to give a pictorial representation of their interdependencies because they are recursive. For example, the name services use RPC facility for internal communication among its

various servers, but the RPC facility uses the name services to locate the destination. Therefore, the interdependencies of the various DCE components can be best depicted in tabular form, as shown in Figure 1.8.

Component name	Other components used by it
Threads	None
RPC	Threads, name, security
DTS	Threads, RPC, name, security
Name	Threads, RPC, DTS, security
Security	Threads, RPC, DTS, name
DFS	Threads, RPC, DTS, name, security

Fig. 1.8 Interdependencies of DCE components.

1.7.4 DCE Cells

The DCE system is highly scalable in the sense that a system running DCE can have thousands of computers and millions of users spread over a worldwide geographic area. To accommodate such large systems, DCE uses the concept of cells. This concept helps break down a large system into smaller, manageable units called cells.

In a DCE system, a *cell* is a group of users, machines, or other resources that typically have a common purpose and share common DCE services. The minimum cell configuration requires a cell directory server, a security server, a distributed time server, and one or more client machines. Each DCE client machine has client processes for security service, cell directory service, distributed time service, RPC facility, and threads facility. A DCE client machine may also have a process for distributed file service if a cell configuration has a DCE distributed file server. Due to the use of the method of intersection for clock synchronization (described in Chapter 6), it is recommended that each cell in a DCE system should have at least three distributed time servers.

An important decision to be made while setting up a DCE system is to decide the cell boundaries. The following four factors should be taken into consideration for making this decision [Tanenbaum 1995, Rosenberry et al. 1992, OSF 1992]:

1. *Purpose.* The machines of users working on a common goal should be put in the same cell, as they need easy access to a common set of system resources. That is, users of machines in the same cell have closer interaction with each other than with users of machines in different cells. For example, if a company manufactures and sells various

types of products, depending on the manner in which the company functions, either a product-oriented or a function-oriented approach may be taken to decide cell boundaries [Tanenbaum 1995]. In the product-oriented approach, separate cells are formed for each product, with the users of the machines belonging to the same cell being responsible for all types of activities (design, manufacturing, marketing, and support services) related to one particular product. On the other hand, in the function-oriented approach, separate cells are formed for each type of activity, with the users belonging to the same cell being responsible for a particular activity, such as design, of all types of products.

2. *Administration.* Each system needs an administrator to register new users in the system and to decide their access rights to the system's resources. To perform his or her job properly, an administrator must know the users and the resources of the system. Therefore, to simplify administration jobs, all the machines and their users that are known to and manageable by an administrator should be put in a single cell. For example, all machines belonging to the same department of a company or a university can belong to a single cell. From an administration point of view, each cell has a different administrator.

3. *Security.* Machines of those users who have greater trust in each other should be put in the same cell. That is, users of machines of a cell trust each other more than they trust the users of machines of other cells. In such a design, cell boundaries act like firewalls in the sense that accessing a resource that belongs to another cell requires more sophisticated authentication than accessing a resource that belongs to a user's own cell.

4. *Overhead.* Several DCE operations, such as name resolution and user authentication, incur more overhead when they are performed between cells than when they are performed within the same cell. Therefore, machines of users who frequently interact with each other and the resources frequently accessed by them should be placed in the same cell. The need to access a resource of another cell should arise infrequently for better overall system performance.

Notice from the above discussion that in determining cell boundaries the emphasis is on purpose, administration, security, and performance. Geographical considerations can, but do not have to, play a part in cell design. For better performance, it is desirable to have as few cells as possible to minimize the number of operations that need to cross cell boundaries. However, subject to security and administration constraints, it is desirable to have smaller cells with fewer machines and users. Therefore, it is important to properly balance the requirements imposed by the four factors mentioned above while deciding cell boundaries in a DCE system.

1.8 SUMMARY

A distributed computing system is a collection of processors interconnected by a communication network in which each processor has its own local memory and other peripherals and communication between any two processors of the system takes place by message passing over the communication network.

The existing models for distributed computing systems can be broadly classified into five categories: minicomputer, workstation, workstation-server, processor-pool, and hybrid.

Distributed computing systems are much more complex and difficult to build than the traditional centralized systems. Despite the increased complexity and the difficulty of building, the installation and use of distributed computing systems are rapidly increasing. This is mainly because the advantages of distributed computing systems outweigh its disadvantages. The main advantages of distributed computing systems are (a) suitability for inherently distributed applications, (b) sharing of information among distributed users, (c) sharing of resources, (d) better price-performance ratio, (e) shorter response times and higher throughput, (f) higher reliability, (g) extensibility and incremental growth, and (h) better flexibility in meeting users' needs.

The operating systems commonly used for distributed computing systems can be broadly classified into two types: network operating systems and distributed operating systems. As compared to a network operating system, a distributed operating system has better transparency and fault tolerance capability and provides the image of a virtual uniprocessor to the users.

The main issues involved in the design of a distributed operating system are transparency, reliability, flexibility, performance, scalability, heterogeneity, security, and emulation of existing operating systems.

EXERCISES

1.1. Differentiate among the following types of operating systems by defining their essential properties:
(a) Time sharing
(b) Parallel processing
(c) Network
(d) Distributed

1.2. In what respect are distributed computing systems better than parallel processing systems? Give examples of three applications for which distributed computing systems will be more suitable than parallel processing systems.

1.3. What were the major technological, economical, and social factors that motivated the development of distributed computing systems? What are some of the main advantages and disadvantages of distributed computing systems over centralized ones?

1.4. Discuss the relative advantages and disadvantages of the various commonly used models for configuring distributed computing systems. Which model do you think is going to become the most popular model in future? Give reasons for your answer.

1.5. Consider the case of a distributed computing system based on the processor-pool model that has P processors in the pool. In this system, suppose a user starts a computation job that involves compilation of a program consisting of F source files ($F < P$). Assume that at this time this user is the only user using the system. What maximum gain in speed can be hoped for this job in this system as compared to its execution on a single-processor system (assume that all the processors we are talking about are of equal capability)? What factors might cause the gain in speed to be less than this maximum?

1.6. Explain the difference between the terms *service* and *server*. In the design of a distributed operating system, discuss the relative advantages and disadvantages of using a single server and multiple servers for implementing a service.

1.7. Why are distributed operating systems more difficult to design than operating systems for centralized time-sharing systems?

1.8. What is *groupware*? Why is it considered to be a promising technology for software development?

1.9. What are the main differences between a network operating system and a distributed operating system?

1.10. What are the major issues in designing a distributed operating system?

1.11. A distributed operating system makes a collection of networked machines to act like a virtual uniprocessor. What are the main advantages of this virtual-machine architecture for a user? What issues are important for a distributed operating system designer in achieving this goal?

1.12. Concurrency transparency is an important issue in the design of a distributed operating system. Is it also an important issue in the design of an operating system for a centralized system? If no, explain why. If yes, list some mechanisms that are commonly used in operating systems for centralized systems to support this feature.

1.13. Discuss some of the important concepts that a distributed operating system designer might use to improve the reliability of his or her system. What is the main problem in making a system highly reliable?

1.14. Differentiate between the monolithic kernel and microkernel approaches for designing a distributed operating system. Discuss their relative advantages and disadvantages.

1.15. In the microkernel approach for designing a distributed operating system, what are the primary tasks that the kernel must perform?

1.16. Figure 1.6 indicates that a layered approach is used to design a distributed system. What are the main advantages of using this approach?

1.17. Discuss the main guiding principles that a distributed operating system designer must keep in mind for the good performance of his or her system.

1.18. Why is scalability an important feature in the design of a distributed system? Discuss some of the guiding principles for designing a scalable distributed system.

1.19. Why is heterogeneity unavoidable in many distributed systems? What are some of the common types of incompatibilities encountered in heterogeneous distributed systems? What are the common issues with which the designer of a heterogeneous distributed system must deal?

1.20. Suppose a component of a distributed system suddenly crashes. How will this event inconvenience the users when:

(a) The system uses the processor-pool model and the crashed component is a processor in the pool.

(b) The system uses the processor-pool model and the crashed component is a user terminal.

(c) The system uses the workstation-server model and the crashed component is a server machine.

(d) The system uses the workstation-server model and the crashed component is a user workstation.

1.21. Compare the following types of systems in terms of cost, hardware complexity, operating system complexity, potential parallelism, and programmability (how easily users can write efficient programs):

(a) A multiprocessor system having a single shared memory.

(b) A multiprocessor system in which each processor has its own local memory in addition to a shared memory used by all processors in the system.

(c) A multiprocessor system in which each processor has its own memory. All processors are kept in a big hall and are interconnected by a high-capacity communication line forming a network. Each processor can communicate with other processors only by exchanging messages.

(d) A multiprocessor system in which each processor has its own memory. The processors are located far from each other (may be in different cities of a country) and are interconnected by a low-capacity communication line forming a network. Each processor can communicate with other processors only by exchanging messages.

For comparing the systems, consider three cases—(a) number of processors is small (2–8); (b) number of processors is large (16–32); and (c) number of processors is very large (more than 100).

BIBLIOGRAPHY

[Accetta et al. 1986] Accetta, M., Baron, R., Golub, D., Rashid, R., Tevanian, A., and Young, M., "Mach: A New Kernel Foundation for UNIX Development," In: *Proceedings of the Summer 1986 USENIX Technical Conference*, pp. 93–112 (July 1986).

[Avresky and Pradhan 1996] Avresky, D., and Pradhan, D. (Eds.), *Fault-Tolerant Parallel and Distributed Systems*, IEEE Computer Society Press, Los Alamitos, CA (1996).

[Black et al. 1992] Black, D. L., Golub, D. B., Julin, D. P., Rashid, R. F., Draves, R. P., Dean, R. W., Forin, A., Barrera, J., Tokuda, H., Malan, G., and Bohman, D., "Microkernel Operating System Architecture and Mach," In: *Proceedings of the USENIX Workshop on Microkernels and Other Kernel Architectures*, USENIX, pp. 11–30 (1992).

[Boykin and LoVerso 1990] Boykin, J., and LoVerso, J., "Recent Developments in Operating Systems," *IEEE Computer*, pp. 5–6 (May 1990).

[Brazier and Johansen 1993] Brazier, F., and Johansen, D. (Eds.), *Distributed Open Systems*, IEEE Computer Society Press, Los Alamitos, CA (1993).

[Butler 1993] Butler, M., *Client Server*, Prentice-Hall, London, UK (1993).

[Casavant and Singhal 1994] Casavant, T. L., and Singhal, M. (Eds.), *Readings in Distributed Computing Systems*, IEEE Computer Society Press, Los Alamitos, CA (1994).

[Cheriton 1984] Cheriton, D. R., "The V Kernel: A Software Base for Distributed Systems," *IEEE Software*, Vol. 1, No. 2, pp. 19–42 (1984).

[Cheriton 1988] Cheriton, D. R., "The V Distributed System," *Communications of the ACM*, Vol. 31, No. 3, pp. 314–333 (1988).

[Coulouris et al. 1994] Coulouris, G. F., Dollimore, J., and Kindberg, T., *Distributed Systems Concepts and Design*, 2nd ed., Addison-Wesley, Reading, MA (1994).

[Cristian-1991] Cristian, F., "Understanding Fault-Tolerant Distributed Systems," *Communications of the ACM*, Vol. 34, pp. 56–78 (February 1991).

[Critchley and Batty 1993] Critchley, T., and Batty, K., *Open Systems: The Reality*, Prentice-Hall, London, UK (1993).

[Deitel 1990] Deitel, H. M., *An Introduction to Operating Systems*, 2nd ed., Addison-Wesley, Reading, MA (1990).

[Douglis et al. 1991] Douglis, F., Ousterhout, J. K., Kaashoek, M. F., and Tanenbaum, A. S., "A Comparison of Two Distributed Systems: Amoeba and Sprite," *Computing Systems*, Vol. 4, pp. 353–384 (1991).

[Ghafoor and Yang 1993] Ghafoor, A., and Yang, J., "A Distributed Heterogeneous Supercomputing Management System," *IEEE Computer*, Vol. 26, No. 6, pp. 78–86 (1993).

[Gien 1990] Gien, M., "Micro-Kernel Architecture: Key to Modern Operating Systems Design," *UNIX Review*, p. 10 (November 1990).

[Gien and Grob 1992] Gien, M., and Grob, L., "Microkernel Based Operating Systems: Moving UNIX on to Modern System Architectures," In: *Proceedings of the UniForum'92 Conference*, USENIX, pp. 43–55 (1992).

[Golub et al. 1990] Golub, D., Dean, R., Forin, A., and Rashid, R., "UNIX as an Application Program," In: *Proceedings of the Summer 1990 USENIX Conference*, USENIX, pp. 87–95 (June 1990).

[Goscinski 1991] Goscinski, A., *Distributed Operating Systems, The Logical Design*, Addison-Wesley, Reading, MA (1991).

[Hariri et al. 1992] Hariri, S., Choudhary, A., and Sarikaya, B, "Architectural Support for Designing Fault-Tolerant Open Distributed Systems," *IEEE Computer*, Vol. 25, No. 6, pp. 50–62 (1992).

[Hunter 1995] Hunter, P., *Network Operating Systems: Making the Right Choice*, Addison-Wesley, Reading, MA (1995).

[Islam 1996] Islam, N., *Distributed Objects: Methodologies for Customizing Operating Systems*, IEEE Computer Society Press, Los Alamitos, CA (1996).

[ISO 1992] *Basic Reference Model of Open Distributed Processing, Part 1: Overview and Guide to Use*, ISO/IEC JTC1/SC212/WG7 CD10746–1, International Standards Organization (1992).

[Jalote 1994] Jalote, P., *Fault Tolerance in Distributed Systems*, Prentice-Hall, Englewood Cliffs, NJ (1994).

[Khanna 1994] Khanna, R. (Ed.), *Distributed Computing: Implementation and Management Strategies*, Prentice-Hall, Englewood Cliffs, NJ (1994).

[Khokhar et al. 1993] Khokhar, A., Prasanna, V. K., Shaaban, M. E., and Wang, C. L., "Heterogeneous Computing: Challenges and Opportunities," *IEEE Computer*, Vol. 26, No. 6, pp. 18–27 (1993).

[Lampson 1983] Lampson, B. W., "Hints for Computer System Design," In: *Proceedings of the 9th Symposium on Operating Systems Principles* (October 1983).

[Lamport et al. 1982] Lamport, L., Shostak, R., and Pease, M., "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 3, pp. 382–401 (1982).

[Lelann 1981] Lelann, G., "Motivations, Objectives, and Characterization of Distributed Systems," *Distributed Systems—Architecture and Implementation, Lecture Notes in Computer Science*, Vol. 105, Springer-Verlag, New York, NY (1981).

[Lockhart Jr. 1994] Lockhart Jr., H. W., *OSF DCE: Guide to Developing Distributed Applications*, IEEE Computer Society Press, Los Alamitos, CA (1994).

[Marca and Bock 1992] Marca, D., and Bock, G. (Eds.), *Groupware: Software for Computer-Supported Cooperative Work*, IEEE Computer Society Press, Los Alamitos, CA (1992).

[Martin et al. 1991] Martin, B. E., Pedersen, C. H., and Roberts, J. B., "An Object-Based Taxonomy for Distributed Computing Systems," *IEEE Computer*, Vol. 24, No. 8 (1991).

[Milenkovic 1992] Milenkovic, M., *Operating Systems: Concepts and Design*, 2nd ed., McGraw-Hill, New York (1992).

[Mullender 1987] Mullender, S. J.,"Distributed Operating Systems," *Computer Standards and Interfaces*, Vol. 6, pp. 37–44 (1987).

[Mullender 1993] Mullender, S. J. (Ed.), *Distributed Systems*, 2nd ed., Addison-Wesley, Reading, MA (1993).

[Mullender and Tanenbaum 1984] Mullender, S. J., and Tanenbaum, A. S., "Protection and Resource Control in Distributed Operating Systems," *Computer Networks*, Vol. 8, pp. 421–432 (1984).

[Mullender et al. 1990] Mullender, S. J., Van Rossum, G., Tanenbaum, A. S., Van Renesse, R., and Van Staverene, H., "Amoeba: A Distributed Operating System for the 1990s," *IEEE Computer*, Vol. 23, No. 5, pp. 44–53 (1990).

[Needham and Herbert 1982] Needham, R. M., and Herbert, A. J., *The Cambridge Distributed Computing System*, Addison-Wesley, Reading, MA (1982).

[Nelson 1990] Nelson, V. P., "Fault-Tolerant Computing: Fundamental Concepts," *IEEE Computer*, Vol. 23, No. 7, pp. 19–25 (1990).

[Nicol et al. 1993] Nicol, J. R., Wilkes, C. T., and Manola, F. A., "Object Orientation in Heterogeneous Distributed Computing Systems," *IEEE Computer*, Vol. 26, No. 6, pp. 57–67 (1993).

[Notkin et al. 1987] Notkin, D., Hutchinson, N., Sanislo, J., and Schwartz, M., "Heterogeneous Computing Environments: Report on the ACM SIGOPS Workshop on Accommodating Heterogeneity," *Communications of the ACM*, Vol. 30, No. 2, pp. 132–140 (1987).

[Nutt 1991] Nutt, G. J., *Centralized and Distributed Operating Systems*, Prentice-Hall, Englewood Cliffs, NJ (1991).

[Nutt 1992] Nutt, G. J., *Open Systems*, Prentice-Hall, Englewood Cliffs, NJ (1992).

[OSF 1992] *Introduction to OSF DCE*, Prentice-Hall, Englewood Cliffs, NJ (1992).

[Ousterhout et al. 1988] Ousterhout, J. K., Cherenson, A. R., Douglis, F., Nelson, M. N., and Welch, B. B., "The Sprite Network Operating System," *IEEE Computer*, Vol. 21, No. 2, pp. 23–36 (1988).

[Pike et al. 1990] Pike, R., Presotto, D., Thompson, K., and Trickey, H., "Plan 9 from Bell Labs," In: *Proceedings of the Summer 1990 UKUUG (UK Unix Users Group) Conference*, pp. 1–9 (July 1990).

[Popek and Walker 1985] Popek, G., and Walker, B., *The LOCUS Distributed System Architecture*, MIT Press, Cambridge, MA (1985).

[Rashid 1985] Rashid, R. F., "Network Operating Systems," In: *Local Area Networks: An Advanced Course, Lecture Notes in Computer Science*, Vol. 184, pp. 314–340, Springer-Verlag, New York, NY (1985).

[Ritchie and Thompson 1974] Ritchie, D., and Thompson, K., "The UNIX Time-Sharing System," Communications of the ACM, Vol. 17, No. 7, pp. 365–375 (1974).

[Rosenberry et al. 1992] Rosenberry, W., Kenney, D., and Fisher, G., *OSF DISTRIBUTED COMPUTING ENVIRONMENT. Understanding DCE*, O'Reilly, Sebastopol, CA (1992).

[Schlichting and Schneider 1983] Schlichting, R. D., and Schneider, F. B., "Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems," *ACM Transactions on Computer Systems*, Vol. 1, No. 3, pp. 222–238 (1983).

[Shoch and Hupp 1982] Shoch, J. F., and Hupp, J. A., "The Worm Programs: Early Experiences with a Distributed Computation," *Communications of the ACM*, Vol. 25, No. 3, pp. 172–180 (1982).

[Silberschatz and Galvin 1994] Silberschatz, A., and Galvin, P. B., *Operating Systems Concepts*, 4th ed., Addison-Wesley, Reading, MA (1994).

[Singhal and Shivaratri 1994] Singhal, M., and Shivaratri, N. G., *Advanced Concepts in Operating Systems*, McGraw-Hill, New York, NY (1994).

[Stalling 1995] Stalling, W., *Operating Systems*, 2nd ed., Prentice-Hall, Englewood Cliffs, NJ (1995).

[Stankovic 1984] Stankovic, J. A., "A Perspective on Distributed Computer Systems," *IEEE Transactions on Computers*, Vol. C-33, No. 12, pp. 1102–1115 (1984).

[Suri et al. 1994] Suri, N., Walter, C. J., and Hugue, M. M. (Eds.), *Advances in Ultra-Dependable Distributed Systems*, IEEE Computer Society Press, Los Alamitos, CA (1994).

[Tanenbaum 1995] Tanenbaum, A. S., *Distributed Operating Systems*, Prentice-Hall, Englewood Cliffs, NJ (1995).

[Tanenbaum and Van Renesse 1985] Tanenbaum, A. S., and Van Renesse, R., "Distributed Operating Systems," *ACM Computing Surveys*, Vol. 17, No. 4, pp. 419–470 (1985). © ACM, Inc., 1985.

[Umar 1993] Umar, A., *Distributed Computing: A Practical Approach*, Prentice-Hall, Englewood Cliffs, NJ (1993).

[Vaughn 1994] Vaughn, L. T., *Client/Server System Design and Implementation*, IEEE Computer Society Press, Los Alamitos, CA (1994).

[Wittie 1991] Wittie, L. D., "Computer Networks and Distributed Systems," *IEEE Computer*, pp. 67–75 (1991).

POINTERS TO BIBLIOGRAPHIES ON THE INTERNET

Bibliography containing references on *Operating Systems* can be found at:

ftp:ftp.cs.umanitoba.ca/pub/bibliographies/Os/os.html

Bibliography containing references on *Taxonomies for Parallel and Distributed Systems* can be found at:

ftp:ftp.cs.umanitoba.ca/pub/bibliographies/Parallel/taxonomy.html

Bibliography containing references on *Distributed Computing* can be found at:

ftp:ftp.cs.umanitoba.ca/pub/bibliographies/Distributed/Osser.html

Bibliographies containing references on *Distributed Systems* can be found at:

ftp:ftp.cs.umanitoba.ca/pub/bibliographies/Distributed/Dcs-1.0.html

ftp:ftp.cs.umanitoba.ca/pub/bibliographies/Distributed/dist.sys.1.html

Bibliography containing references on *Open Systems* and *Open Computing* can be found at:

ftp:ftp.cs.umanitoba.ca/pub/bibliographies/Os/opencomp.html

Bibliography containing references on *Fault Tolerant Distributed Systems* can be found at:

ftp:ftp.cs.umanitoba.ca/pub/bibliographies/Distributed/fault.tolerant.html

Bibliographies containing references on *Computer Supported Cooperative Work* (*CSCW*) can be found at:

ftp:ftp.cs.umanitoba.ca/pub/bibliographies/Distributed/CSCWBiblio.html

ftp:ftp.cs.umanitoba.ca/pub/bibliographies/Distributed/CSCW92.html

List of publications of the MIT Parallel & Distributed Operating Systems (*PDOS*) group can be found at:

http:www.pdos.lcs.mit.edu/PDOS-papers.html

List of publications of the Stanford Distributed Systems Group (DSG) can be found at:

http:www-dsg.stanford.edu/Publications.html

List of publications of the Distributed Systems Research Group (DSRG) at Oregon Graduate Institute can be found at:

http:www.cse.ogi.edu/DSRG/osrg/osrg.html#Current Paper

List of publications of the Distributed Systems Group at Trinity College, Dublin, can be found at:

http:www.dsg.cs.tcd.ie/dsgpublications/bibs

CHAPTER 2

Computer Networks

2.1 INTRODUCTION

A computer network is a communication system that links end systems by communication lines and software protocols to exchange data between two processes running on different end systems of the network. The end systems are often referred to as *nodes*, *sites*, *hosts*, *computers*, *machines*, and so on. The nodes may vary in size and function. Sizewise, a node may be a small microprocessor, a workstation, a minicomputer, or a large supercomputer. Functionwise, a node may be a dedicated system (such as a print server or a file server) without any capability for interactive users, a single-user personal computer, or a general-purpose time-sharing system.

As already mentioned in Chapter 1, a distributed system is basically a computer network whose nodes have their own local memory and may also have other hardware and software resources. A distributed system, therefore, relies entirely on the underlying computer network for the communication of data and control information between the nodes of which they are composed. Furthermore, the performance and reliability of a distributed system depend to a great extent on the performance and reliability of the underlying computer network. Hence a basic knowledge of computer networks is required for the study of distributed operating systems. A comprehensive treatment of computer networks will require a complete book in itself, and there are many good books available on this subject [Tanenbaum 1988, Black 1993, Stallings 1992b, Ramos et al. 1996].

Therefore, this chapter deals only with the most important aspects of networking concepts and designs, with special emphasis to those aspects that are needed as a basis for designing distributed operating systems.

2.2 NETWORKS TYPES

Networks are broadly classified into two types: *local area networks* (*LANs*) and *wide-area networks* (*WANs*). The WANs are also referred to as *long-haul networks*. The key characteristics that are often used to differentiate between these two types of networks are as follows [Abeysundara and Kamal 1991]:

1. *Geographic distribution*. The main difference between the two types of networks is the way in which they are geographically distributed. A LAN is restricted to a limited geographic coverage of a few kilometers, but a WAN spans greater distances and may extend over several thousand kilometers. Therefore, LANs typically provide communication facilities within a building or a campus, whereas WANs may operate nationwide or even worldwide.

2. *Data rate*. Data transmission rates are usually much higher in LANs than in WANs. Transmission rates in LANs usually range from 0.2 megabit per second (Mbps) to 1 gigabit per second (Gbps). On the other hand, transmission rates in WANs usually range from 1200 bits per second to slightly over 1 Mbps.

3. *Error rate*. Local area networks generally experience fewer data transmission errors than WANs do. Typically, bit error rates are in the range of 10^{-8}–10^{-12} with LANs as opposed to 10^{-5}–10^{-7} with WANs.

4. *Communication link*. The most common communication links used in LANs are twisted pair, coaxial cable, and fiber optics. On the other hand, since the sites in a WAN are physically distributed over a large geographic area, the communication links used are by default relatively slow and unreliable. Typical communication links used in WANs are telephone lines, microwave links, and satellite channels.

5. *Ownership*. A LAN is typically owned by a single organization because of its limited geographic coverage. A WAN, however, is usually formed by interconnecting multiple LANs each of which may belong to a different organization. Therefore, administrative and maintenance complexities and costs for LANs are usually much lower than for WANs.

6. *Communication cost*. The overall communication costs of a LAN is usually much lower than that of a WAN. The main reasons for this are lower error rates, simple (or absence of) routing algorithms, and lower administrative and maintenance costs. Moreover, the cost to transmit data in a LAN is negligible since the transmission medium is usually owned by the user organization. However, with a WAN, this cost may be very high because the transmission media used are leased lines or public communication systems, such as telephone lines, microwave links, and satellite channels.

Networks that share some of the characteristics of both LANs and WANs are sometimes referred to as *metropolitan area networks* (*MANs*) [Stallings 1993a]. The MANs usually cover a wider geographic area (up to about 50 km in diameter) than LANs and frequently operate at speeds very close to LAN speeds. A main objective of MANs is to interconnect LANs located in an entire city or metropolitan area. Communication links commonly used for MANs are coaxial cable and microwave links.

We saw in Chapter 1 that the performance of a distributed system must be at least as good as a centralized system. That is, when a particular application is run on a distributed system, its overall performance should not be appreciably worse than running the same application on a single-processor system. The data transmission rates of LANs and MANs are usually considered to be adequate to meet this requirement for many applications. However, with the current technology, the transmission rates of WANs cannot fully meet this requirement of distributed systems. Therefore, WAN-based distributed systems are used mainly for those applications for which performance is not important. Several inherently distributed applications that require information sharing among widely distributed users/computers belong to this category.

Although current WANs cannot fully meet the performance requirements of distributed systems, with the emergence of Broadband Integrated Services Digital Network (B-ISDN) [Kawarasaki and Jabbari 1991] and Asynchronous Transfer Mode (ATM) technologies [Vetter 1995], future WANs are expected to have data transmission rates that will be adequate for the construction of WAN-based distributed systems and the implementation of a wide range of applications on these distributed systems. ISDN [Helgert 1991] refers to telecommunication networks that transfer many types of data, such as voice, fax, and computer data, in digital form at data transmission rates that are multiples of a basic channel speed of 64 kilobits per second (Kbps). B-ISDN are ISDN networks that provide point-to-point data transmission speeds of 150 Mbps and above. B-ISDN networks are considered to be suitable for high-bandwidth applications, such as applications involving high-quality video and bulk data transmissions. ATM technology can provide data transmission rates of up to 622 Mbps. (ATM technology is described later in this chapter.)

2.3 LAN TECHNOLOGIES

This section presents a description of topologies, principles of operation, and case studies of popular LANs.

2.3.1 LAN Topologies

The two commonly used network topologies for constructing LANs are multiaccess bus and ring. In a simple *multiaccess bus network*, all sites are directly connected to a single transmission medium (called the bus) that spans the whole length of the network (Fig. 2.1). The bus is passive and is shared by all the sites for any message transmission in the network. Each site is connected to the bus by a drop cable using a T-connection or tap. Broadcast communication is used for message transmission.

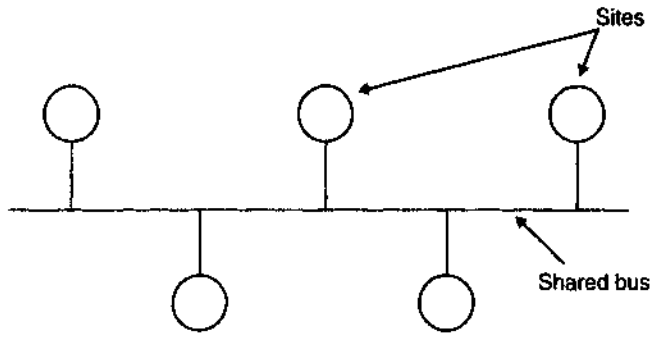


Fig. 2.1 Simple multiaccess bus network topology.

That is, a message is transmitted from one site to another by placing it on the shared bus. An address designator is associated with the message. As the message travels on the bus, each site checks whether it is addressed to it and the addressee site picks up the message.

A variant of the simple multiaccess bus network topology is the multiaccess branching bus network topology. In such a network, two or more simple multiaccess bus networks are interconnected by using repeaters (Fig. 2.2). *Repeaters* are hardware devices used to connect cable segments. They simply amplify and copy electric signals from one segment of a network to its next segment.

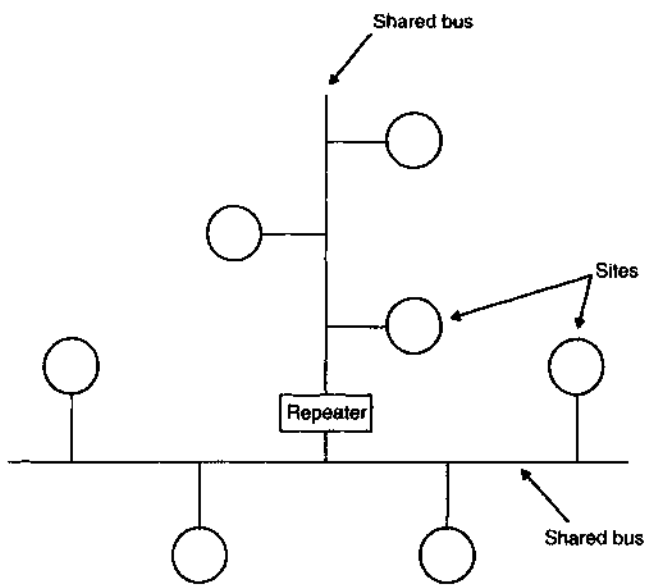


Fig. 2.2 Multiaccess branching bus network topology.

The connection cost of a multiaccess bus network is low and grows only linearly with the increase in number of sites. The communication cost is also quite low, unless there is heavy contention for the shared bus and the bus becomes a bottleneck. A disadvantage of multiaccess bus topology is that message signals, transmitted over the single shared medium, suffer more attenuation and distortion compared to the shorter point-to-point links of other topologies. Therefore, especially in the case of fiber-optic bus networks, only a few sites can usually be supported.

In a *ring network*, each site is connected to exactly two other sites so that a loop is formed (Fig. 2.3). A separate link is used to connect two sites. The links are interconnected by using repeaters. Data is transmitted in one direction around the ring by signaling between sites. That is, to send a message from one site to another, the source site writes the destination site's address in the message header and passes it to its neighbor. A site that receives the message checks the message header to see if the message is addressed to it. If not, the site passes on the message to its own neighbor. In this manner, the message circulates around the ring until some site removes it from the ring. In some ring networks, the destination site (to which the message is addressed) removes the message from the ring, while in others it is removed by the source site (which sent the message). In the latter case, the message always circulates for one complete round on the ring. Generally, in ring networks, one of the sites acts as a *monitor site* to ensure that a message does not circulate indefinitely (that is, in case the source site or the destination site fails). The monitor site also perform other jobs, such as housekeeping functions, ring utilization, and handling other error conditions.

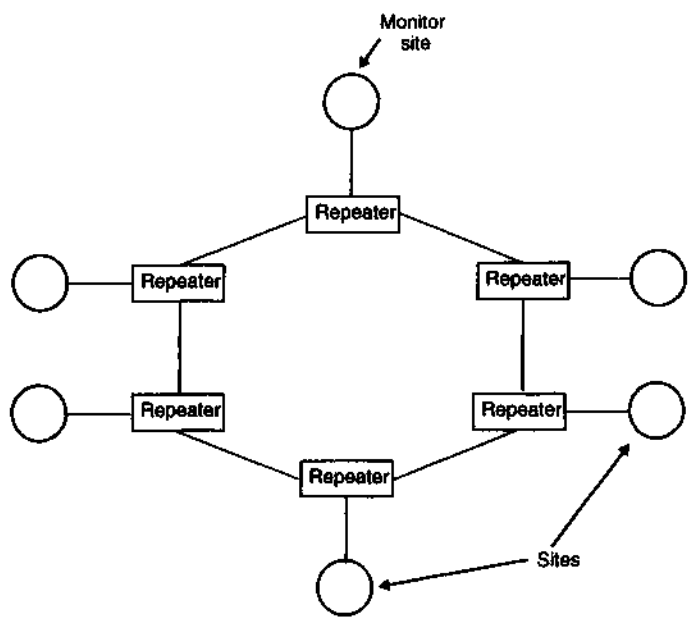


Fig. 2.3 Ring network topology.

The connection cost of a ring network is low and grows only linearly with the increase in number of sites. The average communication cost is directly proportional to the number of sites in the network. If there are n sites, at most $(n-1)$ links have to be traversed by a message to reach its destination. An often-quoted disadvantage of ring topology is the vulnerability of ring networks due to site or link failures; the network is partitioned by a single link failure. Variations of the basic ring topology, such as using bidirectional links, providing double links between two neighbors, and site skipping links with each site joined to its two immediate predecessors, have been considered to improve network reliability.

2.3.2 Medium-Access Control Protocols

In case of both multiaccess bus and ring networks, we saw that a single channel is shared by all the sites of a network, resulting in a multiaccess environment. In such an environment, it is possible that several sites will want to transmit information over the shared channel simultaneously. In this case, the transmitted information may become scrambled and must be discarded. The concerned sites must be notified about the discarded information, so that they can retransmit their information. If no special provisions are made, this situation may be repeated, resulting in degraded performance. Therefore, special schemes are needed in a multiaccess environment to control the access to a shared channel. These schemes are known as *medium-access control protocols*.

Obviously, in a multiaccess environment, the use of a medium having high raw data rate alone is not sufficient. The medium-access control protocol used must also provide for efficient bandwidth use of the medium. Therefore, the medium-access control protocol has a significant effect on the overall performance of a computer network, and often it is by such protocols that the networks differ the most. The three most important performance objectives of a medium-access control protocol are high throughput, high channel utilization, and low message delay. In addition to meeting the performance objectives, some other desirable characteristics of a medium-access control protocol are as follows [Abeysundara and Kamal 1991]:

1. For fairness, unless a priority scheme is intentionally implemented, the protocol should provide equal opportunity to all sites in allowing them to transmit their information over the shared medium.
2. For better scalability, sites should require a minimum knowledge of the network structure (topology, size, or relative location of other sites), and addition, removal, or movement of a site from one place to another in the network should be possible without the need to change the protocol. Furthermore, it should not be necessary to have a knowledge of the exact value of the end-to-end propagation delay of the network for the protocol to function correctly.
3. For higher reliability, centralized control should be avoided and the operation of the protocol should be completely distributed.

4. For supporting real-time applications, the protocol should exhibit bounded-delay properties. That is, the maximum message transfer delay from one site to another in the network must be known and fixed.

It is difficult to achieve all of the previously mentioned characteristics at the same time while achieving the performance objectives of high throughput, high channel utilization, and low message delay.

Several protocols have been developed for medium-access control in a multiaccess environment. Of these, the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) protocol is the one most commonly used for multiaccess bus networks, and the token ring and slotted ring are the two commonly used protocols for ring networks. These medium-access control protocols are described next.

Note that for efficient and fair use of network resources, a message is often divided into packets prior to transmission. In this case, a *packet* is the smallest unit of communication. It usually contains a header and a body. The body contains a part of the actual message data, and the header contains addressing information (identifiers of the sender and receiver sites) and sequencing information (position of the packet data within the entire message data).

The CSMA/CD Protocol

The CSMA/CD scheme [IEEE 1985a] employs decentralized control of the shared medium. In this scheme, each site has equal status in the sense that there is no central controller site. The sites contend with each other for use of the shared medium and the site that first gains access during an idle period of the medium uses the medium for the transmission of its own message. Obviously, occasional collisions of messages may occur when more than one site senses the medium to be idle and transmits messages at approximately the same time. The scheme uses collision detection, recovery, and controlled retransmission mechanisms to deal with this problem. Therefore, the scheme is comprised of the following three mechanisms and works as described next:

1. *Carrier sense and defer mechanism.* Whenever a site wishes to transmit a packet, it first listens for the presence of a signal (known as a *carrier* by analogy with radio broadcasting) on the shared medium. If the medium is found to be free (no carrier is present on the medium), the site starts transmitting its packet. Otherwise, the site defers its packet transmission and waits (continues to listen) until the medium becomes free. The site initiates its packet transmission as soon as it senses that the medium is free.

2. *Collision detection mechanism.* Unfortunately, carrier sensing does not prevent all collisions because of the nonzero propagation delay of the shared medium. Obviously, collisions occur only within a short time interval following the start of transmission, since after this interval all sites will detect that the medium is not free and defer transmission. This time interval is called the *collision window* or *collision interval* and is equal to the amount of time required for a signal to propagate from one end of the shared medium to the other and back again. If a site attempts to transmit a packet, it must listen to the shared

medium for a time period that is at least equal to the collision interval in order to guarantee that the packet will not experience a collision.

Collision avoidance by listening to the shared medium for at least the collision interval time before initiation of packet transmission leads to inefficient utilization of the medium when collisions are rare. Therefore, instead of trying to avoid collisions, the CSMA/CD scheme allows collisions to occur, detects them, and then takes necessary recovery actions.

A decentralized collision detection mechanism is used when a site transmits its packet through its output port, it also listens on its input port and compares the two signals. A collision is detected when a difference is found in the two signals. On detection of a collision, the site immediately stops transmitting the remaining data in the packet and sends a special signal, called a *jamming signal*, on the shared medium to notify all sites that a collision has occurred. On seeing the jamming signal, all sites discard the current packet. The sites whose packets collided retransmit their packets at some later time.

Note that, for ensuring that all collisions are detected, a lower bound on packet length is needed. To illustrate this, let us assume that the maximum propagation delay between two sites of a network is t. If the two sites start transmitting their packets almost at the same time, it will take at least time t for the sites to start receiving the other site's packet data and to detect the collision. Hence if the packet size is so small that it takes less than time t for a site to pump all its data on the network, the sites will not detect the collision because the two sites complete their packet transmission before they see the other site's packet data. However, any other site on the same network for which the propagation time from the two sites is less than t will receive scrambled data of the packets of both the sites.

3. *Controlled retransmission mechanism.* After a collision, the packets that became corrupted due to the collision must be retransmitted. If all the transmitting stations whose packets were corrupted by the collision attempt to retransmit their packets immediately after the jamming signal, collision will probably occur again. To minimize repeated collisions and to achieve channel stability under overload conditions, a controlled retransmission strategy is used in which the competition for the shared medium is resolved using a decentralized algorithm.

Retransmission policies have two conflicting goals: (a) scheduling a retransmission quickly to get the packet out and maintain use of the shared medium and (b) voluntarily backing off to reduce the site's load on a busy medium. A retransmission algorithm is used to calculate the delay before a site should retransmit its packet. After a collision takes place, the objective is to obtain delay periods that will reschedule each site at times quantized in steps at least as large as a collision interval. This time quantization is called the *retransmission slot time*. To guarantee quick use of the medium, this slot time should be short; yet to avoid collisions it should be larger than a collision interval. Therefore, the slot time is usually set to be a little longer than the round-trip time of the medium. The real-time delay is the product of some retransmission delay D (a positive integer) and the retransmission slot time (S_t).

A good example of the controlled retransmission mechanism is the binary exponential back-off algorithm used in Ethernet. This algorithm is described later in this chapter during the description of Ethernet (a case study of LAN technology).

The CSMA/CD scheme works best on networks having a bus topology with bursty asynchronous transmissions. It has gained favor for transmission media that have relatively low speeds (around 10 Mbps) mainly because of its ease of implementation and its channel utilization efficiency. Notice that the performance of the CSMA/CD scheme depends on the ratio of packet length to propagation delay. The higher the ratio, the better the performance because the propagation delay is the interval during which a packet is vulnerable to collision. After that interval, all sites will have "heard" the transmission and deferred. As this ratio diminishes, the collision frequency increases, causing significant performance degradation. Because this performance becomes more pronounced when the ratio of packet length to propagation delay is too low, CSMA/CD is unsuitable for high data rates and/or long distances. For instance, for a transmission medium having photonic speeds (hundreds of megabits per second or gigabits per second) the efficiency of CSMA/CD is often unacceptable.

In addition to being unsuitable for systems having high data rates, small-size packets, and long cable lengths, the CSMA/CD scheme has the following drawbacks: (a) It does not possess a bounded-delay property. Because the loading of the shared medium is variable, it is impossible to guarantee the delivery of a given message within any fixed time, since the network might be fully loaded when the message is ready for transmission. (b) It is not possible to provide priorities for the use of the shared transmission medium. Since all sites are equal, none have priority over others, even though some sites may require greater use of the facilities due to the nature of a particular application.

The Token Ring Protocol

This scheme also employs decentralized control of the shared medium. In this scheme, access to the shared medium is controlled by using a single token that is circulated among the sites in the system. A *token* is a special type of message (having a unique bit pattern) that entitles its holder to use the shared medium for transmitting its messages. A special field in the token indicates whether it is free or busy. The token is passed from one site to the adjacent site around the ring in one direction. A site that has a message ready for transmission must wait until the token reaches it and it is free. When it receives the free token, it sets it to busy, attaches its message to the token, and transmits it to the next site in the ring. A receiving site checks the status of the token. If it is free, it uses it to transmit its own message. Otherwise, it checks to see if the message attached to the busy token is addressed to it. If it is, it retrieves the message attached to the token and forwards the token without the attached message to the next site in the ring. When the busy token returns to the sending site after one complete round, the site removes it from the ring, generates a new free token, and passes it to the next site, allowing the next site to transmit its message (if it has any). The free token circulates from one site to another until it reaches a site that has some message to transmit. To prevent a site from holding the token for a very long time, a token-holding timer is used to control the length of time for which a site may occupy the token.

To guarantee reliable operation, the token has to be protected against loss or duplication. That is, if the token gets lost due to a site failure, the system must detect the loss and generate a new token. This is usually done by the monitor site. Moreover, if a site i crashes, the ring must be reconfigured so that site $i-1$ will send the token directly to site $i+1$.

An advantage of the token ring protocol is that the message delay can be bounded because of the absence of collisions. Another advantage is that it can work with both large and small packet size as well as variable-size packets. In principle, a message attached to the token may be of almost any length. A major disadvantage, however, is the initial waiting time to receive a free token even at very light loads. This initial waiting time could be appreciable, especially in large rings.

A variant of the token ring protocol described above is the IEEE 802.5 standard Token Ring protocol [IEEE 1985c]. This protocol is described later in this chapter during the description of IEEE Token Ring (a case study of LAN technology).

The Slotted-Ring Protocol

In this scheme, a constant number of fixed-length message slots continuously circulate around the ring. Each slot has two parts—control and data. The control part usually has fields to specify whether the slot is full or empty, the source and destination addresses of the message contained in a full slot, and whether the message in it was successfully received at the destination. On the other hand, the data part can contain a fixed-length message data.

A site that wants to send a message first breaks down the message into packets of size equal to the size of the data part of the slots. It then waits for the arrival of an empty slot. As soon as an empty slot arrives, it grabs it, places one packet of its message in its data part, sets the source and destination addresses properly, sets the full/empty field to full, and puts it back on the ring. This slot then circulates on the ring and the site again waits for the arrival of another empty slot. The site continues doing this until it has transmitted all the packets of the message.

Each site inspects every full slot to check if it contains a packet addressed to it. If not, it simply forwards the slot to the next site on the ring. Otherwise, it removes the packet from the slot, properly alters the field in the slot showing that the message in the slot was successfully received at the destination, and then forwards the slot to the next site in the ring. When the slot returns back to its sender after completing its journey round the ring, the sender changes its full/empty field to *empty*, making it available for transmission of any other message in the system.

This scheme prevents hogging of the ring and guarantees fair sharing of the bandwidth of the shared medium among all sites. It also allows messages from multiple sites to be simultaneously transmitted (in the token ring scheme, only one site can transmit at a time). However, it requires a long message to be broken into smaller packets (this is not required in the token ring scheme).

The slotted-ring scheme is used in the Cambridge Ring [Wilkes and Wheeler 1979], which was developed at Cambridge University in the 1970s and is widely used in Britain. Further details of this protocol can be found in [King and Mitrani 1987].

2.3.3 LAN Technology Case Studies: Ethernet and IEEE Token Ring

Ethernet

Ethernet is the most widely used LAN technology for building distributed systems because it is relatively fast and economical. It was introduced by DEC (Digital Equipment Corporation), Intel, and Xerox in 1980, and subsequently, a slightly modified version of it was adopted by the IEEE as a standard LAN technology, known as the IEEE 802.3 standard [IEEE 1985a].

The network topology used for Ethernet is a simple multiaccess bus or a multiaccess branching bus topology. The communication medium used is low-loss coaxial cable having a data transfer rate of 10 Mbps.

A message is transmitted from one site to another by breaking it up into packets (called *frames* in Ethernet) and then by broadcasting the packets to the bus. An address designator is associated with each packet. As a packet travels on the bus, each site checks whether the packet is addressed to it and the addressee site picks up the message.

For medium-access control, Ethernet uses the CSMA/CD protocol with a *binary exponential back-off algorithm* as the controlled retransmission mechanism. In the binary exponential back-off algorithm of Ethernet, the value of retransmission delay (D) is selected as a random number from a particular retransmission interval between zero and some upper limit (L). That is, if the packets of two sites collide, one will retransmit after an interval of $X \times S_t$ (S_t is the retransmission slot time) and the other will retransmit after an interval of $Y \times S_t$, where X and Y are likely to be different since they were chosen randomly. To control the shared medium and keep it stable under high load, the value of L is doubled with each successive collision, thus extending the range for the random selection of the value of D. In Ethernet, on first collision, the value of D is randomly chosen to be 0 or 1; on the second collision, it is randomly chosen to be 0, 1, 2, or 3; and on the ith successive collision, it is randomly chosen to be an integer in the range 0 and $2^i - 1$ (both inclusive). Thus, the more often that a sender fails (due to repeated collisions), the longer potential period of time it will defer before attempting to retransmit. This algorithm has very short retransmission delays at the beginning but will back off quickly, preventing the medium from becoming overloaded.

Notice that after some number of back-offs, the retransmission interval becomes large. To avoid undue delays and slow response to improved medium characteristics, the doubling of the retransmission interval is usually stopped at some point, with additional retransmissions still being drawn from this interval, before the transmission is finally aborted. This is referred to as the *truncated binary exponential back-off algorithm.*

The structure of a packet in Ethernet is shown in Figure 2.4. The destination and source addresses occupy 6 bytes each. The destination address may be a single-site address (specifies a single site), a multicast address (specifies a group of sites), or a broadcast address (specifies all sites). The address consisting of all 1's is the broadcast address. The sites that belong to a multicast address have their network interfaces configured to receive all packets addressed to the multicast address. Moreover, to distinguish multicast addresses from single-site addresses, the higher

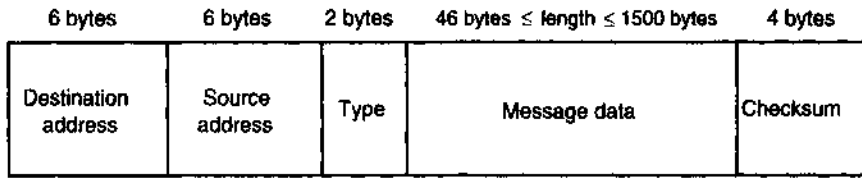


Fig. 2.4 Structure of a packet in Ethernet.

order bit of a multicast address is always 1, whereas this bit of a single-site address is always 0.

The type field is used to distinguish among multiple types of packets. This field is used by higher layer communication protocols (communication protocol layers are described later in this chapter).

The message data field is the only field in the packet structure that may have variable length. It contains the actual message data to be transmitted. The length of this field may vary from 46 to 1500 bytes. The minimum length of 46 bytes for this field is necessary to ensure that collisions are always detected in the CSMA/CD scheme used for medium-access control. On the other hand, the maximum length of 1500 bytes for this field is used to allow each site in the network to allocate buffer space for the largest possible incoming packet and to avoid a sender site waiting for a long time for the communication channel to become free. Since there is no field to indicate the length of the message data field, an interval of 9.6 μs is used between the transmission of two packets to allow receiving sites to detect the end of transmission of a packet.

The last 4 bytes of a packet always contain a checksum generated by the sender and used by the receiver(s) to check the validity of the received packet. A receiver simply drops a packet that contains an incorrect checksum. Due to this, message delivery is not guaranteed in Ethernet. This is the reason why simple datagram protocols used in local networks are potentially unreliable. If guaranteed message delivery is needed, the upper layer of the communication protocol (communication protocol layers are described later in this chapter) must use acknowledgments for the receipt of each packet and timeout-based retransmissions for unacknowledged packets.

Every Ethernet hardware interface is assigned a unique address by the manufacturer. This allows all the sites of a set of interconnected Ethernets to have unique addresses. IEEE acts as an allocation authority for Ethernet addresses. A separate range of 48-bit addresses is allocated to each manufacturer of Ethernet hardware interfaces.

IEEE Token Ring

Another commonly used LAN technology for building distributed systems is the IEEE Token Ring technology, known as the IEEE 802.5 standard [IEEE 1985b]. IBM has adopted this technology as a basis for its distributed system products.

The network topology used in the IEEE Token Ring technology is ring topology, and the medium-access control protocol used is the token ring protocol. Initially it operated at

a speed of 4 Mbps but was later upgraded to 16 Mbps. It can use a cheap twisted pair or optical fiber as the communication medium and has almost no wasted bandwidth when all sites are trying to send.

A single token of 3 bytes keeps circulating continuously around the ring. The token may either be busy or free. A site willing to send a message attaches its message to the token and changes its status to busy, when the circulating token arrives at the sender's site with free status. Therefore, a busy token has a message packet attached to it whose structure is shown in Figure 2.5.

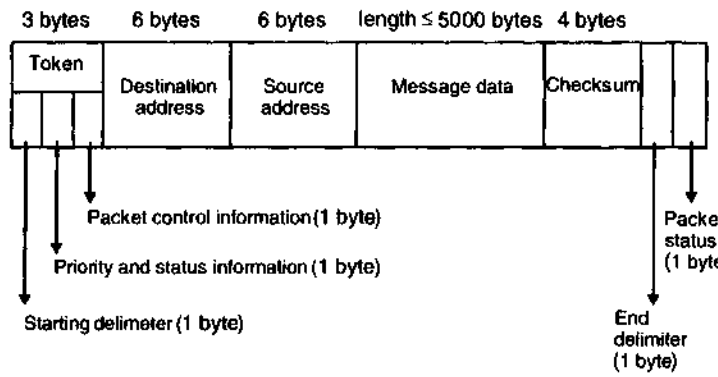


Fig. 2.5 Structure of a packet in IEEE Token Ring.

The first byte of the token contains a fixed bit pattern that enables sites to recognize the start of a packet and synchronize to the data transmission rate. The second byte contains priority and token status (free/busy) information. The third byte contains packet control information. The priority field can be used to implement a variety of methods for sharing the channel capacity among the sites on the network.

In addition to the token, a packet in the IEEE Token Ring has fields for source and destination addresses (each 6 bytes long), message data (*length* ≤ 5000 *bytes*), checksum (4 bytes), end delimiter (1 byte), and packet status (1 byte). The source address and destination address fields respectively contain the addresses of the sending and receiving sites. The message data field contains the data to be transmitted. This field is of variable length and allows packets to be of almost any length. The upper bound of 5000 bytes for the length of this field is a default value for a parameter that can be configured on a per-installation basis. The checksum field contains a checksum generated by the sender and used by the receiver to check the validity of the received packet. The end-delimiter field contains a fixed bit pattern that enables sites to recognize the end of a packet. Finally, the packet status field specifies whether the packet was successfully received by the receiving site. This field helps the sending site in knowing whether its packet was received by the receiver. The sending site is responsible for removing its packet from the ring when it returns after one rotation.

2.4 WAN TECHNOLOGIES

A WAN of computers is constructed by interconnecting computers that are separated by large distances; they may be located in different cities or even in different countries. In general, no fixed regular network topology is used for interconnecting the computers of a WAN. Moreover, different communication media may be used for different links of a WAN. For example, in a WAN, computers located in the same country may be interconnected by coaxial cables (telephone lines), but communications satellites may be used to interconnect two computers that are located in different countries.

The computers of a WAN are not connected directly to the communication channels but are connected to hardware devices called *packet-switching exchanges* (*PSEs*), which are special-purpose computers dedicated to the task of data communication. Therefore, the communication channels of the network interconnect the PSEs, which actually perform the task of data communication across the network (Fig. 2.6). A computer of a WAN only interacts with the PSE of the WAN to which it is connected for sending and receiving messages from other computers on the WAN.

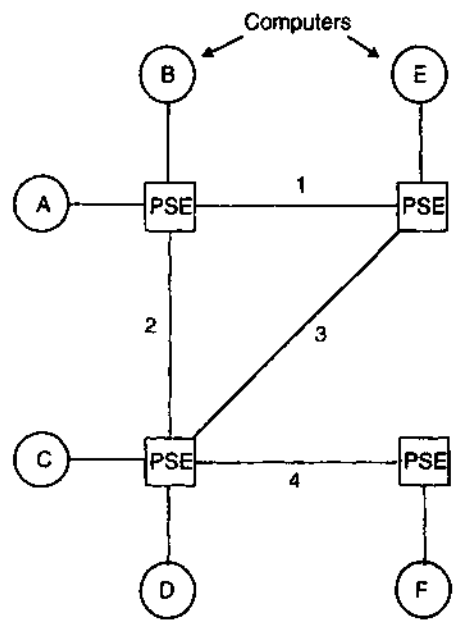


Fig. 2.6 A WAN of computers.

To send a message packet to another computer on the network, a computer sends the packet to the PSE to which it is connected. The packet is transmitted from the sending computer's PSE to the receiving computer's PSE, possibly via other PSEs. The actual mode of packet transmission and the route used for forwarding a packet from its sending computer's PSE to its receiving computer's PSE depend on the switching and routing

techniques used by the PSEs of the network. Various possible options for these techniques are described next. When the packet reaches its receiving computer's PSE, it is delivered to the receiving computer.

2.4.1 Switching Techniques

We saw that in a WAN communication is achieved by transmitting a packet from its source computer to its destination computer through two or more PSEs. The PSEs provide switching facility to move a packet from one PSE to another until the packet reaches its destination. That is, a PSE removes a packet from an input channel and places it on an output channel. Network latency is highly dependent on the switching technique used by the PSEs of the WAN. The two most commonly used schemes are circuit switching and packet switching. They are described next.

Circuit Switching

In this method, before data transmission starts, a physical circuit is constructed between the sender and receiver computers during the circuit establishment phase. During this phase, the channels constituting the circuit are reserved exclusively for the circuit; hence there is no need for buffers at the intermediate PSEs. Once the circuit is established, all packets of the data are transferred in the data transfer phase. Since all the packets of a message data are transmitted one after another through the dedicated circuit without being buffered at intermediate sites, the packets appear to form a continuous data stream. Finally, in the circuit termination phase, the circuit is torn down as the last packet of the data is transmitted. As soon as the circuit is torn down, the channels that were reserved for the circuit become available for use by others. If a circuit cannot be established because a desired channel is busy (being used), the circuit is said to be blocked. Depending on the way blocked circuits are handled, the partial circuit may be torn down, with establishment to be attempted later.

This scheme is similar to that used in the public telephone system. In this system, when a telephone call is made, a dedicated circuit is established by the telephone switching office from the caller's telephone to the callee's telephone. Once this circuit is established, the only delay involved in the communication is the time required for the propagation of the electromagnetic signal through all the wires and switches. While it might be hard to obtain a circuit sometimes (such as calling long distance on Christmas Day), once the circuit is established, exclusive access to it is guaranteed until the call is terminated.

The main advantage of a circuit-switching technique is that once the circuit is established, data is transmitted with no delay other than the propagation delay, which is negligible. Furthermore, since the full capacity of the circuit is available for exclusive use by the connected pair of computers, the transmission time required to send a message can be known and guaranteed after the circuit has been successfully established. However, the method requires additional overhead during circuit establishment and circuit disconnection phases, and channel bandwidths may be wasted if the channel capacities of the path forming the circuit are not utilized efficiently by the connected pair of computers.

Therefore, the method is considered suitable only for long continuous transmissions or for transmissions that require guaranteed maximum transmission delay. It is the preferred method for transmission of voice and real-time data in distributed applications.

Circuit switching is used in the Public Switched Telephone Network (PSTN).

Packet Switching

In this method, instead of establishing a dedicated communication path between a sender and receiver pair (of computers), the channels are shared for transmitting packets of different sender-receiver pairs. That is, a channel is occupied by a sender-receiver pair only while transmitting a single packet of the message of that pair; the channel may then be used for transmitting either another packet of the same sender-receiver pair or a packet of some other sender-receiver pair.

In this method, each packet of a message contains the address of the destination computer, so that it can be sent to its destination independently of all other packets. Notice that different packets of the same message may take a different path through the network and, at the destination computer, the receiver may get the packets in an order different from the order in which they were sent. Therefore, at the destination computer, the packets have to be properly reassembled into a message. When a packet reaches a PSE, the packet is temporarily stored there in a packet buffer. The packet is then forwarded to a selected neighboring PSE when the next channel becomes available and the neighboring PSE has an available packet buffer. Hence the actual path taken by a packet to its destination is dynamic because the path is established as the packet travels along. Packet-switching technique is also known as *store-and-forward* communication because every packet is temporarily stored by each PSE along its route before it is forwarded to another PSE.

As compared to circuit switching, packet switching is suitable for transmitting small amounts of data that are bursty in nature. The method allows efficient usage of channels because the communication bandwidth of a channel is shared for transmitting several messages. Furthermore, the dynamic selection of the actual path to be taken by a packet gives the network considerable reliability because failed PSEs or channels can be ignored and alternate paths may be used. For example, in the WAN of Figure 2.6, if channel 2 fails, a message from computer A to D can still be sent by using the path 1–3. However, due to the need to buffer each packet at every PSE and to reassemble the packets at the destination computer, the overhead incurred per packet is large. Therefore, the method is inefficient for transmitting large messages. Another drawback of the method is that there is no guarantee of how long it takes a message to go from its source computer to its destination computer because the time taken for each packet depends on the route chosen for that packet, along with the volume of data being transferred along that route.

Packet switching is used in the X.25 public packet network and the Internet.

2.4.2 Routing Techniques

In a WAN, when multiple paths exist between the source and destination computers of a packet, any one of the paths may be used to transfer the packet. For example, in the WAN of Figure 2.6, there are two paths between computers E and F: 3–4 and

1–2–4—and any one of the two may be used to transmit a packet from computer E to F. The selection of the actual path to be used for transmitting a packet is determined by the routing technique used. An efficient routing technique is crucial to the overall performance of the network. This requires that the routing decision process must be as fast as possible to reduce the network latency. A good routing algorithm should be easily implementable in hardware. Furthermore, the decision process usually should not require global state information of the network because such information gathering is a difficult task and creates additional traffic in the network. Routing algorithms are usually classified based on the following three attributes:

- Place where routing decisions are made
- Time constant of the information upon which the routing decisions are based
- Control mechanism used for dynamic routing

Note that routing techniques are not needed in LANs because the sender of a message simply puts the message on the communication channel and the receiver takes it off from the channel. There is no need to decide the path to be used for transmitting the message from the sender to the receiver.

Place Where Routing Decisions Are Made

Based on this attribute, routing algorithms may be classified into the following three types:

1. *Source routing*. In this method, the source computer's PSE selects the entire path before sending the packet. That is, all intermediate PSEs via which the packet will be transferred to its destination are decided at the source computer's PSE of the packet, and this routing information is included along with the packet. The method requires that the source computer's PSE must have fairly comprehensive information about the network environment. However, the routing decision process is efficient because the intermediate PSEs need not make any routing decision. A drawback of the method is that the path cannot be changed after the packet has left the source computer's PSE, rendering the method susceptible to component failures.

2. *Hop-by-hop routing*. In this method, each PSE along the path decides only the next PSE for the path. That is, each PSE maintains information about the status of all its outgoing channels and the adjacent PSEs and then selects a suitable adjacent PSE for the packet and transmits it to that PSE. The routing decisions are typically based on the channel availability and the readiness of the adjacent PSEs to receive and relay the packet. The method requires that each PSE must maintain a routing table of some sort. However, as compared to the static routing method, this method makes more efficient use of network bandwidth and provides resilience to failures because alternative paths can be used for packet transmissions.

3. *Hybrid routing*. This method combines the first two methods in the sense that the source computer's PSE specifies only certain major intermediate PSEs of the complete path, and the subpaths between any two of the specified PSEs are decided by the method of hop-by-hop routing.

Static and Dynamic Routing

Depending on when the information used for making routing decisions is specified and how frequently it is modified, routing algorithms are classified into the following two types:

1. *Static routing*. In this method, routing tables (stored at PSEs) are set once and do not change for very long periods of time. They are changed only when the network undergoes major modifications. Static routing is also known as *fixed* or *deterministic routing*. Static routing is simple and easy to implement. However, it makes poor use of network bandwidth and causes blocking of a packet even when alternative paths are available for its transmission. Hence, static routing schemes are susceptible to component failures.

2. *Dynamic routing*. In this method, routing tables are updated relatively frequently, reflecting shorter term changes in the network environment. Dynamic routing strategy is also known as *adaptive routing* because it has a tendency to adapt to the dynamically changing state of the network, such as the presence of faulty or congested channels. Dynamic routing schemes can use alternative paths for packet transmissions, making more efficient use of network bandwidth and providing resilience to failures. The latter property is particularly important for large-scale architectures, since expanding network size can increase the probability of encountering a faulty network component. In dynamic routing, however, packets of a message may arrive out of order at the destination computer. This problem can be solved by appending a sequence number to each packet and properly reassembling the packets at the destination computer.

The path selection policy for dynamic routing may either be *minimal* or *nonminimal*. In the minimal policy, the selected path is one of the shortest paths between the source and destination pair of computers. Therefore, every channel visited will bring the packet closer to the destination. On the other hand, in the nonminimal policy, a packet may follow a longer path, usually in response to current network conditions. If the nonminimal policy is used, care must be taken to avoid a situation in which the packet will continue to be routed through the network but never reach the destination.

Control Mechanisms for Dynamic Routing

In the dynamic routing strategy, routing tables are constantly updated. One of the following three approaches may be used for controlling the update action:

1. *Isolated manner*. In this approach, individual PSEs update the information in their local routing table in an isolated manner, perhaps by periodically trying various routes and observing performance.

2. *Centralized manner.* In this method, changes in the network environment, connectivity, or performance are constantly reported to one centralized PSE. Based on the information received, the global routing table maintained at the centralized PSE is constantly updated. The updated routing table information is periodically sent from the centralized PSE to the source PSEs (in the source-routing strategy) or to all PSEs (in the hop-by-hop routing strategy).

The main advantages of the centralized approach are that the routes are globally optimal and other PSEs are not involved in the information gathering of the global network status. However, the centralized approach suffers from poor performance in situations where the system is large or where traffic changes are frequent. Also, it has poor reliability since the table construction is performed at a single PSE.

3. *Decentralized manner.* To overcome the shortcomings of the centralized approach, several systems use the distributed control mechanism. In this method, each PSE maintains a routing table and the routing table updates are performed by mutual interaction among the PSEs. Often a PSE piggybacks the routing table update information along with some other message being sent to another PSE.

2.5 COMMUNICATION PROTOCOLS

In the last several sections of this chapter we saw that several types of agreements are needed between the communicating parties in a computer network. For example, it is important that the sender and receiver of a packet agree upon the positions and sizes of the various fields in the packet header, the position and size of actual data in the packet, the position and size of the checksum field, the method to calculate the checksum for error detection and so on. For transmission of message data comprised of multiple packets, the sender and receiver must also agree upon the method used for identifying the first packet and the last packet of the message, and since the last packet may only be partially filled, a method is needed to identify the last bit of the message in this packet. Moreover, agreement is also needed for handling duplicate messages, avoiding buffer overflows, and assuring proper message sequencing. All such agreements, needed for communication between the communicating parties, are defined in terms of rules and conventions by network designers. The term *protocol* is used to refer to a set of such rules and conventions.

Computer networks are implemented using the concept of *layered protocols.* According to this concept, the protocols of a network are organized into a series of layers in such a way that each layer contains protocols for exchanging data and providing functions in a logical sense with peer entities at other sites in the network. Entities in adjacent layers interact in a physical sense through the common interface defined between the two layers by passing parameters such as headers, trailers, and data parameters. The main reasons for using the concept of layered protocols in network design are as follows:

- The protocols of a network are fairly complex. Designing them in layers makes their implementation more manageable.

- Layering of protocols provides well-defined interfaces between the layers, so that a change in one layer does not affect an adjacent layer. That is, the various functionalities can be partitioned and implemented independently so that each one can be changed as technology improves without the other ones being affected. For example, a change to a routing algorithm in a network control program should not affect the functions of message sequencing, which is located in another layer of the network architecture.
- Layering of protocols also allows interaction between functionally paired layers in different locations. This concept aids in permitting the distribution of functions to remote sites.

The terms *protocol suite*, *protocol family*, or *protocol stack* are used to refer to the collection of protocols (of all layers) of a particular network system.

2.5.1 Protocols for Network Systems

In Chapter 1 we saw that distributed systems are basically different from network systems. Therefore, the requirements of communication protocols of these two types of systems are also different. The basic goal of communication protocols for network systems is to allow remote computers to communicate with each other and to allow users to access remote resources. On the other hand, the basic goal of communication protocols for distributed systems is not only to allow users to access remote resources but to do so in a transparent manner.

Several standards and protocols for network systems are already available. However, protocols for distributed systems are still in their infancy and no standards are yet available. Some standard network protocol models are described next. The protocols for distributed systems are presented in the next section.

The ISO/OSI Reference Model

The number of layers, the name of each layer, and the functions of each layer may be different for different networks. However, to make the job of the network communication protocol designers easier, the International Standardization Organization (ISO) has developed a reference model that identifies seven standard layers and defines the jobs to be performed at each layer. This model is called the *Open System International Reference Model* (abbreviated *OSI model*) [DePrycker et al. 1993, Larmouth 1993, Stallings 1993c]. It is a guide, not a specification. It provides a framework in which standards can be developed for the services and protocols at each layer. Note that adherence to standard protocols is important for designing open distributed systems. This is because if standard protocols are used, separate software components of distributed systems can be developed independently on computers having different architectures (different code ordering and data representations). To provide an understanding of the structure and functioning of layered network protocols, a brief description of the OSI model is presented next. There are many sources for more detail on this model [Tanenbaum 1988, Stallings 1993c, Larmouth 1993].

The architecture of the OSI model is shown in Figure 2.7. It is a seven-layer architecture in which a separate set of protocols is defined for each layer. Thus each layer has an independent function and deals with one or more specific aspects of the communication. The roles of the seven layers are briefly described below.

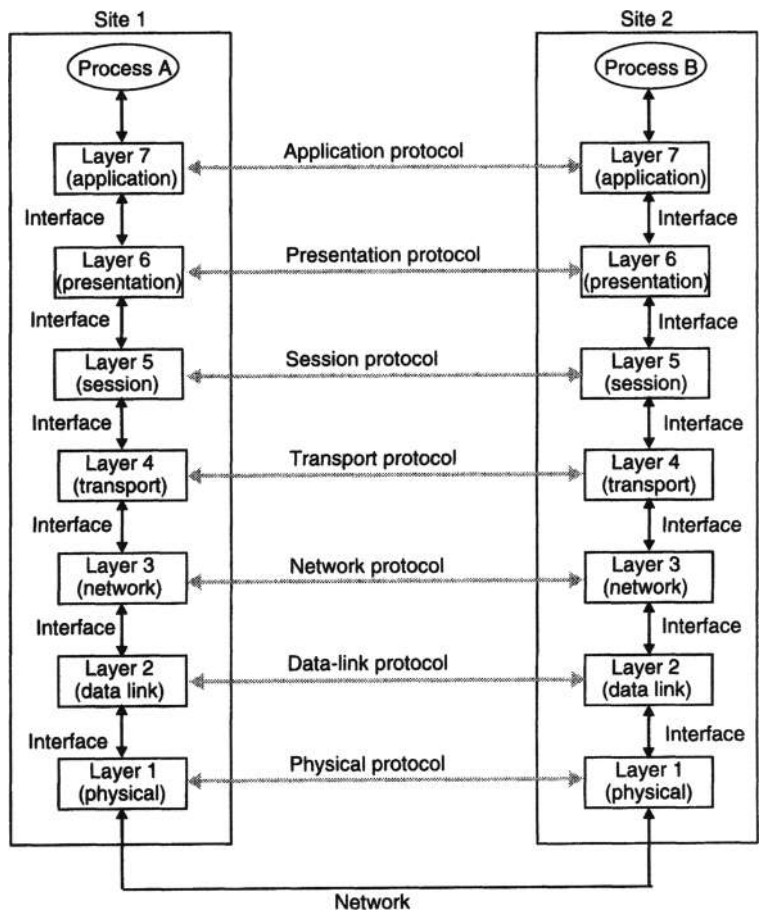


Fig. 2.7 Layers, interfaces, and protocols in the OSI model.

Physical Layer. The physical layer is responsible for transmitting raw bit streams between two sites. That is, it may convert the sequence of binary digits into electric signals, light signals, or electromagnetic signals depending on whether the two sites are on a cable circuit, fiber-optic circuit, or microwave/radio circuit, respectively. Electrical details such as how many volts to use for 0 and 1, how many bits can be sent per second, and whether transmission can take place only in one direction or in both directions

simultaneously are also decided by the physical layer protocols. In addition, the physical layer protocols also deal with the mechanical details such as the size and shape of the connecting plugs, the number of pins in the plugs, and the function of each pin. In short, the physical layer protocols deal with the mechanical, electrical, procedural, and functional characteristics of transmission of raw bit streams between two sites. RS232–C is a popular physical layer standard for serial communication lines.

Data-Link Layer. The physical layer simply transmits the data from the sender's site to the receiver's site as raw bits. It is the responsibility of the data-link layer to detect and correct any errors in the transmitted data. Since the physical layer is only concerned with a raw bit stream, the data-link layer partitions it into frames so that error detection and correction can be performed independently for each frame. The data-link layer also performs flow control of frames between two sites to ensure that a sender does not overwhelm a receiver by sending frames at a rate faster than the receiver can process. Therefore, the error control and flow control mechanisms of a network form the data-link layer protocols in the OSI model. Notice that the data-link layer and physical layer protocols establish an error-free communication of raw bits between two sites.

Network Layer. The network layer is responsible for setting up a logical path between two sites for communication to take place. It encapsulates frames into packets that can be transmitted from one site to another using a high-level addressing and routing scheme. That is, routing is the primary job of the network layer and the routing algorithm forms the main part of the network layer protocols of the network.

Two popular network layer protocols are the *X.25 Protocol* and the *Internet Protocol* (called *IP*). The X.25 is a *connection-oriented protocol* that is based on the concept of establishing a virtual circuit between the sender and receiver before the actual communication starts between them. In this protocol, a request for connection is first sent to the destination, which can either be accepted or rejected. If the connection is accepted, the requesting party is given a connection identifier to use in subsequent requests. During the connection establishment phase, a route between the two parties is also decided that is used for the transmission of subsequent traffic.

On the other hand, IP is a *connectionless protocol* in which no connection is established between the sender and receiver before sending a message. Therefore, each packet of the message is transmitted independently and may take a different route. IP is part of the DoD (U.S. Department of Defense) protocol suite.

Notice that the functions performed at the network layer are primarily required in WANs. In a single LAN, the network layer is largely redundant because packets can be transmitted directly from any site on the network to any other site. Therefore the network layer, if present, has little work to do.

Transport Layer. The job of the transport layer is to provide site-to-site communication and to hide all the details of the communication subnet from the session layer by providing a network-independent transport service. Using this service, all the details of the communication subnet are sealed and one subnet can be replaced with another without disturbing the layers above the transport layer.

In particular, the transport layer accepts messages of arbitrary length from the session layer, segments them into packets, submits them to the network layer for transmission, and finally reassembles the packets at the destination. Some packets may be lost on the way from the sender to the receiver, and depending on the routing algorithms used in the network layer, packets may arrive at the destination in a sequence that is different from the order in which they are sent. The transport layer protocols include mechanisms for handling lost and out-of-sequence packets. For this, the transport layer records a sequence number in each packet and uses the sequence numbers for detecting lost packets and for ensuring that messages are reconstructed in the correct sequence.

The ISO model provides five classes of transport protocols (known as *TP0* through *TP4*) which basically differ in their ability to handle errors. TP0 is the least powerful one and TP4 is the most powerful one. The choice of which one to use depends on the properties of the underlying network layer.

The two most popular transport layer protocols are the *Transport Control Protocol* (TCP) and the *User Datagram Protocol* (UDP). Both are implemented in the DARPA protocol suite of DARPA Internet. TCP is a connection-oriented transport protocol that provides the same services as TP4 of the ISO model. It uses end-to-end mechanisms to ensure reliable, ordered delivery of data over a logical connection. These goals are basically achieved by using packet sequence numbers and positive acknowledgments with timeout and retransmission.

The UDP is a connectionless transport protocol. It is an unreliable protocol because, when it is used, message packets can be lost, duplicated, or arrive out of order. Therefore, only those applications that do not need reliable communication should use UDP.

Session Layer. The purpose of the session layer is to provide the means by which presentation entities can organize and synchronize their dialog and manage their data exchange. It allows the two parties to authenticate each other before establishing a dialog session between them. It also specifies dialog type—one way, two way alternate, or two way simultaneous–and initiates a dialog session if the message is a connection request message. The other services of the session layer include quarantine service, dialog control, and priority management. The quarantine service buffers a group of messages on the receiving side until the session layer on the sending side explicitly releases them. This is useful in database applications where a transaction (consisting of a group of messages) needs to be an atomic unit. The dialog control is useful for dialog sessions in which the user primitives used for sending and receiving messages are of the nonblocking type. In this case, the user may have multiple requests outstanding on the same session, and replies may come back in an order different from that in which the requests were sent. The dialog control reorders replies according to the order of requests. The priority management service is useful for giving priority to important and time-bound messages over normal, less-important messages. The session layer is not required for connectionless communication.

Presentation Layer. The purpose of this layer is to represent message information to communicating application layer entities in a way that preserves meaning while resolving syntax differences. For this, the presentation layer may perform one or more of the following types of transformations on message data:

- A message usually contains structured information that may include any of the data types used in programming languages—integers, characters, arrays, records, and so on, including user-defined data types. Translation is therefore required where language systems or application programs in the source and destination computers use different representations for these data types.
- Data format conversions are also needed to transfer data between computers when the hardware of the sending and receiving computers uses different data representations. In this case, the presentation layer software in the sending computer transforms message data from the formats used in its own computer to a set of standard network representations called *eXternal Data Representation* (*XDR*) before transmission. The presentation layer software in the receiving computer transforms the message data from the network representations to the formats used in its own computer.
- For applications dealing with confidential or secret data, the presentation layer software in the sending computer encrypts message data before passing it to the session layer. On the receiver side, the encrypted message data is decrypted by the presentation layer before being passed on to the application layer.
- In a similar manner, when message data is large in volume (such as multimedia data) or with networks that are slow or heavily loaded, message data may be compressed and decompressed by the presentation layer software in the sending and receiving computers, respectively.

Application Layer. The application layer provides services that directly support the end users of the network. Obviously, the functionality implemented at this layer of the architecture is application-specific. Since each application has different communication needs, no fixed or standard set of application layer protocols can meet the needs of all applications. Therefore, the application layer is basically a collection of miscellaneous protocols for various commonly used applications such as electronic mail, file transfer, remote login, remote job entry, and schemas for distributed databases. Some popular application layer protocols are X.400 (Electronic Mail Protocol), X.500 (Directory Server Protocol), FTP (File Transfer Protocol), and rlogin (Remote Login Protocol).

In actual implementation, of the seven layers, the first three layers are likely to be in hardware, the next two layers in the operating system, the presentation layer in library subroutines in the user's address space, and the application layer in the user's program.

Example of Message Transfer in the OSI Model. To illustrate the functions of the various layers of the OSI model, let us consider a simple example of message transmission. With reference to Figure 2.8, let us assume that a process at the sending site wants to send a message M to a process at the receiving site. The sending site's process builds the message M and passes it to the application layer (layer 7) on its machine. The application layer software adds a header (H_7) to M and passes the resulting message to the presentation layer (6) via the interface between layers 7 and 6. The presentation layer software performs text compression, code conversion, security encryption, and so on, on the received message, and after adding a header (H_6) to it,

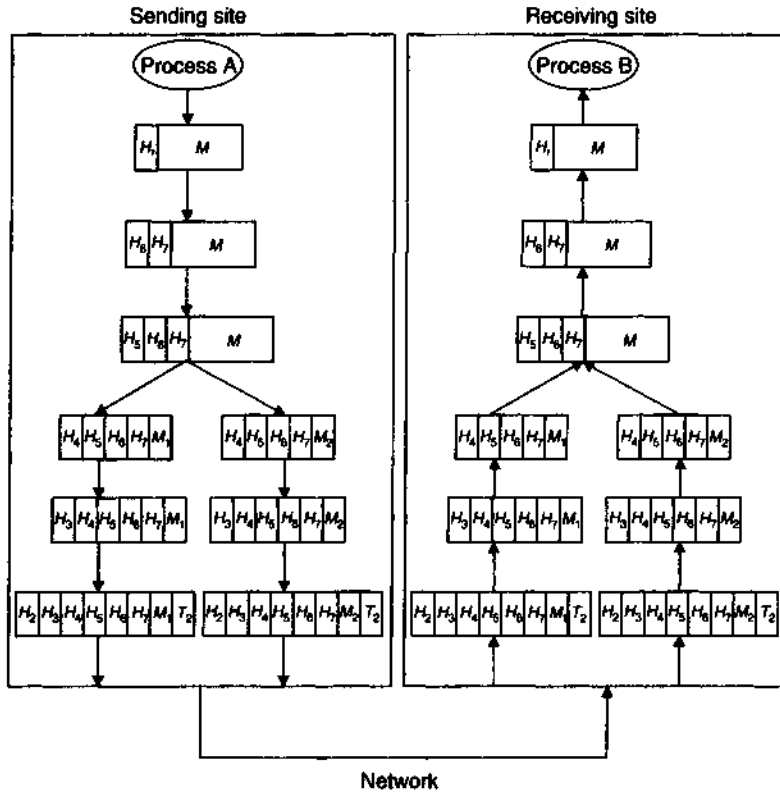


Fig. 2.8 An example illustrating transfer of message M from sending site to receiving site in the OSI model: H_n, header added by layer n; T_n, trailer added by layer n.

it passes the resulting message on to the session layer (5). Depending on the type of dialog, the session layer software establishes a dialog between the sender and the receiver processes. It also regulates the direction of message flow. A header (H_5) is added to the message at this layer, and the resulting message is passed on to the transport layer (4). The transport layer software now splits the message into smaller units (M_1 and M_2) called packets and adds a header (H_4) to each packet. These headers contain the sequence numbers of the message packets. The packets are then passed on to the network layer (3). The network layer software makes routing decisions for the received packets and sets up a logical path between the sending and receiving sites for transmission of the packets. It then adds a header (H_3) to each packet and passes them on to the data-link layer (2). The data-link layer software adds a header (H_2) and a trailer (T_2) to each of these packets. The trailers contain the checksum of the data in the corresponding packets. The resulting message units are called frames, which are passed on to the physical layer (1). The physical layer software simply transmits the raw bits from the sender's machine to the receiver's machine using the physical connection between the two machines.

On the receiver's machine, the message data traverses up from the physical layer to the application layer. As the message data traverses to higher level layers, each layer performs the functions assigned to it and strips off the headers or trailers added by its peer layer at the sending site. For example, the data-link layer at the receiving machine performs error detection by recalculating the checksum for each frame and comparing it with the checksum in the trailer of the frame. It strips off the header (H_2) and the trailer (T_2) from the frames before passing them on to the network layer. The application layer of the receiver's machine finally passes on the message in its original form to the communicating process on the receiver's site. Notice that the software of a particular layer on the sending machine conceptually communicates only with its peer layer on the receiving machine, although physically it communicates only with the adjacent layers on the sending machine. This abstraction is crucial to network design.

The IEEE 802 LAN Reference Model

The ISO model is oriented toward WANs rather than LANs because it was conceived as a model for computer networking primarily in the point-to-point packet-switching environment. In a LAN, the host computers are connected directly to a network circuit by relatively simple interface hardware. The interface hardware and network driver software in each site can send and receive data at high speeds with low error rates and without switching delays. These important characteristics of LANs give considerable advantages in cost, speed, and reliability in comparison to WANs. Due to these differences in characteristics between LANs and WANs, and also because of the following differences between the OSI model and LAN concepts, the OSI model is generally considered to be unsuitable for LAN environments:

- In the OSI model, information is exchanged between two communicating entities only after they have entered into an agreement about exchanging information. But in a LAN, no such prior agreement is needed for communication to take place, and information may be delivered to a destination from a number of different sources within a short time interval.
- In the OSI model, the model of communication is generally one to one and/or one to many. But in a LAN, the model of communication is generally many to many.
- The OSI model is often said to be connection oriented. But communications in a LAN is mostly connectionless.

This implies that a modified reference model particularly suited to LANs is needed. This problem was realized long ago, and a reference model suitable for LANs was built by IEEE in 1983, the *IEEE 802 LAN Reference Model* (abbreviated IEEE 802 LAN) [IEEE 1990]. The IEEE 802 LAN model was built with an aim to use as much as possible of the OSI model while providing compatibility between LAN equipments made by different manufacturers such that data communication can take place between these equipments with the minimum effort on the part of the users or builders of LANs. Therefore, the IEEE 802 LAN model modifies only the lowest two layers of the OSI

model and does not include specifications for higher layers, suggesting the use of the OSI model protocols at higher layers. The modifications of the lowest two layers are mainly concerned with the most important features of LANs that result from the fact that the physical medium is a resource shared by all sites connected to it. The relationship between the IEEE 802 LAN model and the OSI model is shown in Figure 2.9.

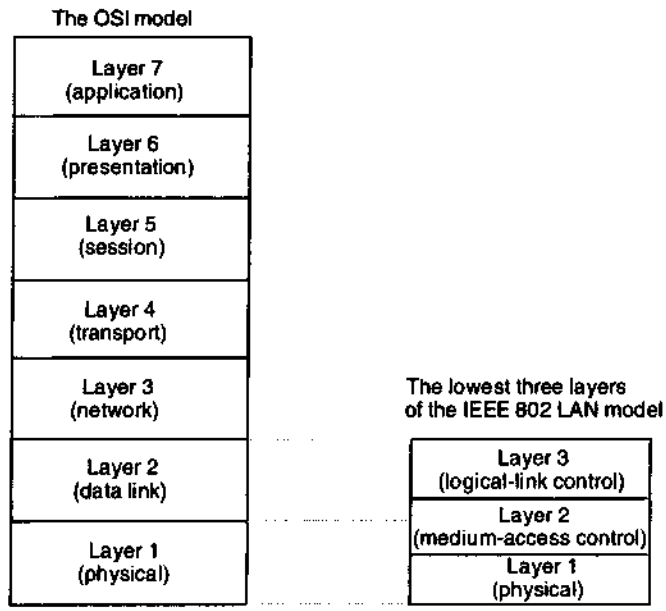


Fig. 2.9 Relationship between the IEEE 802 LAN model and the OSI model.

As shown in the figure, the lowest three layers of the IEEE 802 LAN model are the physical layer, the medium-access-control layer, and the logical-link-control layer. The physical layer defines interface protocols for the following four types of media that are commonly used in LANs: baseband, broadband, fiber optics, and twisted pair. As the name implies, the medium-access-control layer deals with the medium-access-control protocols for LANs. This layer includes functions associated with both the physical and data-link layers of the OSI model. It includes the following four standards:

1. The IEEE 802.3 standard, which defines protocols for a LAN having bus topology that uses the CSMA/CD method for medium-access control.
2. The IEEE 802.4 standard, which defines protocols for a LAN having bus topology that uses the token-passing method for medium-access control. The sites connected to the bus are arranged in a logical ring to use the token-passing method.

3. The IEEE 802.5 standard, which defines protocols for a LAN having ring topology that uses the token-passing method for medium-access control.
4. The IEEE 802.6 standard, which defines protocols for a MAN.

The third layer, that is, the logical-link-control layer, contains the IEEE 802.2 standard. This standard basically defines a common logical-link-control protocol that can be used in conjunction with each of the four standards defined at the media-access-control layer.

Finally, the relationship between the protocols defined in the OSI model and the standards defined in the IEEE 802 LAN model is described in the IEEE 802.1 standard. Further details on the IEEE 802 LAN model can be found in [IEEE 1990, 1985a,b,c].

Network Communication Protocol Case Study: The Internet Protocol Suite

Several protocol suites for network systems, such as the IP suite of the U.S. Department of Defense, Xerox Networking System (XNS) of Xerox, System Network Architecture (SNA) of IBM, Advanced Peer-to-Peer Networking (APPN) of IBM, and NetBIOS of IBM, are available today. Of the available protocol suites for network systems, IP is the most popular and widely used one because it has several attractive features. For instance, it is suitable for both LANs and WANs; it can be implemented on all types of computers, from personal computers to the larger supercomputers; and it is not vendor specific. It is an open standard governed by the nonaligned (vendor-independent) Internet Engineering Task Force (IETF). Every major vendor supports IP, making it the lingua franca on networking. Moreover, IP is such a dominant networking standard that companies in a position to use it as their backbone protocol will be able to move quickly to high-speed internetworking technologies like ATM, FDDI, switched Ethernet and Token Ring, or 100-Mbps Ethernet. Owing to its importance and wide popularity, IP is described below as a case study of protocol suites for network systems.

Figure 2.10 shows the structure of the IP suite. It consists of five layers and several protocols at each layer. The existence of multiple protocols in one layer allows greater flexibility to the users due to different protocols for different applications having different communication requirements. The protocols of each layer are briefly described next.

Layers	Protocols at each layer
Application	FTP, TFTP, TELNET, SMTP, DNS, others
Transport	TCP, UDP
Network	IP, ICMP
Data link	ARP, RARP, others
Physical	SLIP, Ethernet, Token Ring, others

Fig. 2.10 The Internet Protocol suite structure.

Physical Layer Protocols. Most systems using the IP suite for a LAN use the Ethernet protocol at the physical layer. However, LAN products using the IP suite with the Token Ring protocol used at the physical layer are also available. Moreover, implementations using the *SLIP* (*Serial Line Internet Protocol*), which uses an RS-232 serial line protocol (having speeds from 1200 bits per second to 19.2 Kbps), also exist. Networks using the IP suite with physical layer protocols for satellite and microwave links also exist.

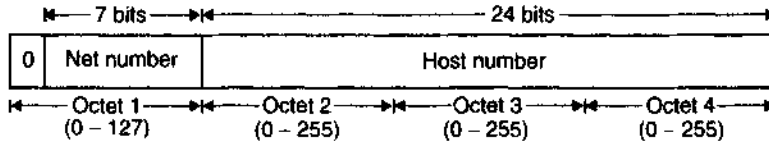
Data-link Layer Protocols. Every protocol suite defines some type of addressing for uniquely identifying each computer on a network. In the IP suite, a computer's address (called *Internet address* or *IP address*) consists of the following information:

1. *Net number.* Each network using the Internet protocol suite has a unique net number that is assigned by a central authority—the Network Information Center (NIC) located at SRI International.
2. *Subnet number.* This is an optional number representing the subnet to which the computer being addressed is attached. This number is assigned by a user to the subnet when the subnet is being set up. The subnet number is stated in the IP address of a computer only when the computer is on a subnet.
3. *Host number.* This number uniquely identifies the computer being addressed on the network identified by a net number.

All this information is represented by a 32-bit address divided into four 8-bit fields. Each field is called an *octet*. Each octet is separated from the next octet by a period. The decimal number represents 1 byte of the IP address, which can have a value of 0–255. Therefore, a typical IP address is 190.40.232.12.

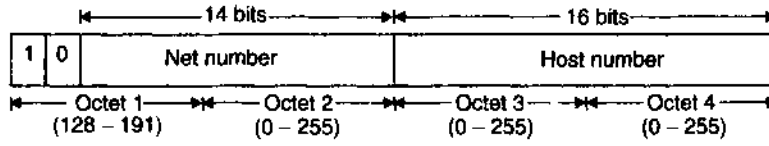
There are three classes of Internet address—A, B, and C—plus a provision for Internet multicast communication that is useful for applications that need multicast facility. The format and the permissible values for each octet for each class of Internet address are shown in Figure 2.11. Due to the increasing importance of multicast-based applications, a group of user organizations has established a multicast network called *MBone* (*Multicast Backbone*) that is a virtual network in the Internet that multicasts audio, video, white-board, and other streams that need to be distributed to large groups simultaneously [Macedonia and Brutzman 1994]. MBone addresses such network issues as bandwidth constraints, routing protocols, data compression, and network topology. Many application tools are available free of cost for a wide variety of host platforms. Today, hundreds of researchers use MBone to develop protocols and applications for group communication.

The three classes of Internet addresses are designed to meet the requirements of different types of organizations. Thus class A addresses are used for those networks that need to accommodate a large number of computers (hosts) on a single network, while class C addresses allow for more networks but fewer hosts per network, and class B addresses provide a median distribution between number of networks and number of hosts per network. Notice from Figure 2.11 that there can be only 127 class A networks, and the



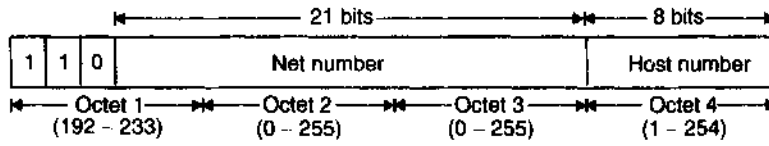
An example IP address is 78.3.50.8

(*a*)



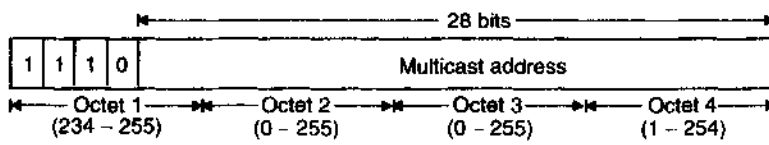
An example IP address is 130.150.10.28

(*b*)



An example IP address is 221.198.141.233

(*c*)



An example IP address is 236.8.26.54

(*d*)

Fig. 2.11 Internet address formats: Internet address for (*a*) class A, (*b*) class B, (*c*) class C networks, and (*d*) multicast communication facility.

highest host address in a class A network can be 255.255.254, thus accommodating a possible 16,777,214 hosts. On the other hand, a class C network can accommodate only 254 hosts.

Values from 0 to 255 can be assigned to each octet for the host number part of an address, with the restriction that the host number part cannot be all 0's or all 1's. That is, for example, on a network having a net address of 78, a host could have the

address 78.15.0.105 or 78.1.1.255 but it could not have the address 78.0.0.0 or 78.255.255.255. This is because Internet addresses with a host number part that is all 0's or all 1's are used for special purposes. Addresses with host number part set to all 0's are used to refer to "this host," and a host number set to all 1's is used to address a broadcast message to all of the hosts connected to the network specified in the net number part of the address.

A *multihomed host* is one that is connected to two or more networks. This implies that it must have two or more Internet addresses, one for each network to which it is connected. This means that every Internet address specifies a unique host but each host does not have a unique address.

Each packet received by the data-link layer from the network layer contains the IP addresses of the sender and receiver hosts. These IP addresses must be converted to physical network addresses for transmission across the network. Similarly, physical network addresses embedded in packets received from other networks must be converted to IP addresses before being passed on to the network layer. The job of translating IP addresses to physical network addresses and physical network addresses to IP addresses is performed by the data-link layer. Two important protocols that belong to this layer are ARP and RARP. *ARP (Address Resolution Protocol)* [Plummer 1982] is the Ethernet address resolution protocol that maps known IP addresses (32 bits long) to Ethernet addresses (48 bits long). On the other hand, *RARP (Reverse ARP)* [Finlayson et al. 1984] is the IP address resolution protocol that maps known Ethernet addresses (48 bits) to IP addresses (32 bits), the reverse of ARP.

Network Layer Protocols. The two network layer protocols are IP [Postel 1981a] and *ICMP (Internet Control Message Protocol)* [Postel 1981b]. IP performs the transmission of datagrams from one host to another. A *datagram* is a group of information transmitted as a unit to and from the upper layer protocols on sending and receiving hosts. It contains the IP addresses of sending and receiving hosts. For datagram transmission, the two main jobs of IP are fragmentation and reassembly of datagrams into IP packets and routing of IP packets by determining the actual path to be taken by each packet from its source to the destination. For packet routing, a dynamic routing algorithm that uses a minimal path selection policy is used.

On the sending host, when a datagram is received at the network layer from the transport layer, the IP attaches a 20-byte header. This header contains a number of parameters, most significantly the IP addresses of the sending and receiving hosts. Other parameters include datagram length and identify information if the datagram exceeds the allowable byte size for network packets [called *MTU (maximum transfer unit)* of the network] and must be fragmented. If a datagram is found to be larger than the allowable byte size for network packets, the IP breaks up the datagram into fragments and sends each fragment as an IP packet. When fragmentation does occur, the IP duplicates the source address and destination address into each IP packet, so that the resulting IP packets can be delivered independently of each other. The fragments are reassembled into the original datagram by the IP on the receiving host and then passed on to the higher protocol layers.

The data transmission service of the IP has *best-effort delivery* semantics. That is, a best effort is made to deliver the packets but delivery is not guaranteed. The IP computes and verifies a checksum that covers its own header, which is inexpensive to calculate. There is no data checksum, which avoids overheads when crossing routers, but leaves the higher level protocols to provide their own checksums. On the receiving host, if the checksum of the header is found to be in error, the packet is discarded, with the assumption that a higher layer protocol will retransmit the packet. Hence, the data transmission service of the IP is unreliable.

The ICMP of the network layer provides an elementary form of flow control. When IP packets arrive at a host or router so fast that they are discarded, the ICMP sends a message to the original source informing it that the data is arriving too fast, which then decreases the amount of data being sent on that connection link.

Transport Layer Protocols. Transport layer protocols enable communications between processes running on separate computers of a network. The two main protocols of this layer are *TCP* (*Transport Control Protocol*) [Postel 1981c] and *UDP* (*User Datagram Protocol*) [Postel 1980]. These two protocols are sometimes referred to as *TCP/IP* and *UDP/IP*, respectively, to indicate that they use the IP at the network layer.

The TCP provides a connection-oriented, reliable, byte-stream service to an application program. It is a reliable protocol because any data written to a TCP connection will be received by its peer or an error indication will be given. Since the IP of the network layer provides an unreliable, connectionless delivery service, it is the TCP that contains the logic necessary to provide a reliable, virtual circuit for a user process. Therefore, TCP handles the establishment and termination of connections between processes, the sequencing of data that might be received out of order, the end-to-end reliability (checksum, positive acknowledgments, timeouts), and the end-to-end flow control. Note that the ICMP of the network layer does not provide end-to-end flow control service between two communicating processes. TCP is used in most of the well-known Internet services that are defined at the application layer.

On the other hand, UDP provides a connectionless, unreliable datagram service. It is an unreliable protocol because no attempt is made to recover from failure or loss; packets may be lost, with no error indication given. Therefore, UDP is very similar to IP with two additional features that are not provided by IP: process addressing and an optional checksum to verify the contents of the UDP datagram. These two additional features are enough reason for a user process to use UDP instead of trying to use IP directly when a connectionless datagram protocol is required. UDP is often used for experimental or small-scale distributed applications in which either reliability of communication is not important or reliability of communication is taken care of at the application level. For example, *rwho* service, network monitoring, time service, Trivial File Transfer Protocol (TFTP), and so on, use UDP.

The IP layer provides host-to-host communication service, but the transport layer provides process-to-process communication service. For process addressing, both TCP and UDP use 16-bit integer *port numbers*. That is, both TCP and UDP attach a header to the data to be transmitted. Among other parameters, this header contains port numbers to specify sending and receiving processes. A port number uniquely identifies a process

within a particular computer and is valid only within that computer. Once an IP packet has been delivered to the destination computer, the transport layer protocols dispatch it to the specified port number at that computer. The process identified by the port number picks up the packet from the port. Further details of the port-based process-addressing mechanism is given in Chapter 3.

Application Layer Protocols. A variety of protocols exist at the application layer in the IP suite. These standard application protocols are available at almost all implementations of the IP suite. Some of the most widely used ones are briefly described next. Additional details of these protocols can be found in [Stallings 1992b]:

1. *File Transfer Protocol (FTP).* This protocol is used to transfer files to and from a remote host in a network. A file transfer takes place in the following manner:

   - A user executes the *ftp* command on its local host, specifying the remote host as a parameter.
   - The FTP client process of the user's machine establishes a connection with an FTP server process on the remote host using TCP.
   - The user is then prompted for login name and password to ensure that the user is allowed to access the remote host.
   - After successful login, the desired file(s) may be transferred in either direction by using *get* (for transfer from remote to local machine) and *put* (for transfer from local to remote machine) commands. Both binary and text files can be transferred. The user can also list directories or move between directories of the remote machine.

2. *Trivial File Transfer Protocol (TFTP).* This protocol also enables users to transfer files to and from a remote host in a network. However, unlike FTP, it uses UDP (not TCP), it cannot check the authenticity of the user, and it does not provide the facilities of listing directories and moving between directories.

3. *TELNET.* This protocol enables terminals and terminal-oriented processes to communicate with another host on the network. That is, a user can execute the *telnet* command on its local host to start a login session on a remote host. Once a login session is established, telnet enters the input mode. In this mode, anything typed on the keyboard by the user is sent to the remote host. The input mode entered will be either character or line mode depending on what the remote system supports. In the character mode, every character typed on the keyboard is immediately sent to the remote host for processing. On the other hand, in the line mode, all typed material is echoed locally, and (normally) only completed lines are sent to the remote host for processing. Like FTP, TELNET uses TCP.

4. *Simple Mail Transfer Protocol (SMTP).* This protocol enables two user processes on a network to exchange electronic mail using a TCP connection between them.

5. *Domain Name Service (DNS).* The client processes of application protocols, such as FTP, TFTP, TELNET, SMTP, can be designed to accept Internet addresses (in their

decimal form) from a user to identify the remote host with which the user wants to interact. However, as compared to numeric identifiers, symbolic names are easier for human beings to remember and use. Therefore, the Internet supports a scheme for the allocation and use of symbolic names for hosts and networks, such as *asuvax.eas.asu.edu* or *eas.asu.edu*. The named entities are called *domains* and the symbolic names are called *domain names*. The domain name space has a hierarchical structure that is entirely independent of the physical structure of the networks that constitute the Internet. A hierarchical naming scheme provides greater flexibility of name space management (described in detail in Chapter 10).

When domain names are accepted as parameters by application protocols such as FTP, TFTP, TELNET, SMTP, and so on, they must be translated to Internet addresses before making communication operation requests to lower level protocols. This job of mapping domain names to Internet addresses is performed by DNS. Further details of DNS are given in Chapter 10.

2.5.2 Protocols for Distributed Systems

Although the protocols mentioned previously provide adequate support for traditional network applications such as file transfer and remote login, they are not suitable for distributed systems and applications. This is mainly because of the following special requirements of distributed systems as compared to network systems [Kaashoek et al. 1993]:

- *Transparency.* Communication protocols for network systems use location-dependent process identifiers (such as port addresses that are unique only within a node). However, for efficient utilization of resources, distributed systems normally support process migration facility. With communication protocols for network systems, supporting process migration facility is difficult because when a process migrates, its identifier changes. Therefore, communication protocols for distributed systems must use location-independent process identifiers that do not change even when a process migrates from one node to another.
- *Client-server-based communication.* The communication protocols for network systems treat communication as an input/output device that is used to transport data between two nodes of a network. However, most communications in distributed systems are based on the client-server model in which a client requests a server to perform some work for it by sending the server a message and then waiting until the server sends back a reply. Therefore, communication protocols for distributed systems must have a simple, connectionless protocol having features to support request/response behavior.
- *Group communication.* Several distributed applications benefit from group communication facility that allows a message to be sent reliably from one sender to n receivers. Although many network systems provide mechanisms to do broadcast or multicast at the data-link layer, their communication protocols often hide these useful capabilities from the applications. Furthermore, although broadcast can be done by sending n point-to-point messages and waiting for n acknowledgments, this algorithm is inefficient and wastes bandwidth. Therefore,

communication protocols for distributed systems must support more flexible and efficient group communication facility in which a group address can be mapped on one or more data-link addresses and the routing protocol can use a data-link multicast address to send a message to all the receivers belonging to the group defined by the multicast address.

- *Security.* Security is a critical issue in networks, and encryption is the commonly used method to ensure security of message data transmitted across a network. However, encryption is expensive to use, and all nodes and all communication channels of a network are not untrustworthy for a particular user. Therefore, encryption should be used only when there is a possibility of a critical message to travel via an untrustworthy node/channel from its source node to the destination node. Hence a communication protocol is needed that can support a flexible and efficient encryption mechanism in which a message is encrypted only if the path it takes across the network cannot be trusted. Existing communication protocols for network systems do not provide such flexibility.
- *Network management.* Network management activities, such as adding/removing a node from a network, usually require manual intervention by a system administrator to update the configuration files that reflect the current configuration of the network. For example, the allocation of new addresses is often done manually. Ideally, network protocols should automatically handle network management activities to reflect dynamic changes in network configuration.
- *Scalability.* A communication protocol for distributed systems must scale well and allow efficient communication to take place in both LAN and WAN environments. A single communication protocol must be usable on both types of networks.

Two communication protocols that have been designed to achieve higher throughput and/or fast response in distributed systems and to address one or more of the issues stated above are *VMTP* (*Versatile Message Transport Protocol*) and *FLIP* (*Fast Local Internet Protocol*). VMPT provides group communication facility and implements a secure and efficient client-server-based communication protocol [Cheriton and Williamson 1989]. On the other hand, FLIP is designed to support transparency, efficient client-server-based communication, group communication, secure communication, and easy network management [Kaashoek et al. 1993]. These protocols are briefly described next.

The Versatile Message Transport Protocol

This is a transport protocol that has been especially designed for distributed operating systems and has been used in the V-System [Cheriton 1988]. It is a connectionless protocol that has special features to support request/response behavior between a client and one or more server processes. It is based on the concept of a message transaction that consists of a request message sent by a client to one or more servers followed by zero or more response messages sent back to the client by the servers, at most one per server. Most message transactions involve a single request message and a single response message.

For better performance, a response is used to serve as an acknowledgment for the corresponding request, and a response is usually acknowledged by the next request from the same client. Using special facilities, a client can request for an immediate acknowledgment for its request or a server can explicitly request an acknowledgment for its response.

To support transparency and to provide group communication facility, entities in VMTP are identified by 64-bit identifiers that are unique, stable, and independent of the host address. The latter property allows entities to be migrated and handled independent of network layer addressing. A portion of the entity identifier space is reserved for entity group identifiers that identify a group of zero or more entities. For example, each file server, as a separate entity, may belong to the group of file servers, identified by a single entity group identifier. To find out the location of a particular file directory, a client can send a request for this information to the entity group of file servers and receive a response from the server containing the directory. A group management protocol has been provided for creating new groups, adding new members or deleting members from an existing group, and querying information about existing groups.

Again for better performance, VMPT provides a *selective retransmission* mechanism. The packets of a message are divided into packet groups that contain up to a maximum of 16 kilobytes of segment data. The data segment is viewed as a sequence of segment blocks, each of 512 bytes (except for the last, which may be only partly full), allowing the portions of the segment in a packet group to be specified by a 32-bit mask. Each packet contains a delivery mask field that indicates the portions of the data segment the packet contains. The maximum number of blocks per packet is determined by the network maximum packet size. When a packet group is received, the delivery masks for the individual packets are ORed together to obtain a bitmap indicating which segment blocks are still outstanding. An acknowledgment packet contains this bitmap, and the sender selectively retransmits only the missing segment blocks.

The VMTP uses a rate-based flow control mechanism. In this mechanism, packets in a packet group are spaced out with interpacket gaps to reduce the arrival rate at the receiver. The mechanism allows clients and servers to explicitly communicate their desired interpacket gap times and to make adjustments based on selective retransmission requests described previously. For example, if the bitmap returned by the receiver indicates that every other packet needs to be retransmitted, the sender reasonably increases the interpacket gap. If the next acknowledgment bitmap indicates that every fourth packet is missing, the sender again increases the interpacket gap. When no packet loss occurs, the sender periodically reduces the interpacket gap to ensure that it is transmitting at the maximum rate the receiver can handle. Thus, selective retransmission provides feedback to indicate that the rate of transmission is too high and also minimizes the performance penalty arising from overflooding of packets from a fast sender to a slow receiver.

An optimization used in VMTP is to differentiate between idempotent and nonidempotent operations. An idempotent operation is one whose execution can be repeated any number of times without there being any side effects. For example, requesting the time of day is a typical idempotent operation, but transferring money from one bank account to another is a nonidempotent operation. In VMTP, a server can label a response to indicate that a message transaction was idempotent. By doing so,

arrangements need not be made for retransmitting the response when it is lost because the server can reproduce the response when the request is retransmitted. However, when a response is nonidempotent, VMPT prevents the server from executing a request more than once.

In addition to the aforementioned features, VMTP provides a rich collection of optional facilities that expand its functionality and efficiency in various situations. One such facility that is particularly useful for real-time communication is the facility of conditional message delivery. With this facility, a client can specify that its message should only be delivered if the server is able to process it immediately. The optional facilities are carefully designed to provide critical extensions to the basic facilities without imposing a significant performance penalty, especially on common-case processing.

Further details of the VMTP protocol can be found in [Cheriton 1986, Cheriton and Williamson 1989].

The Fast Local Internet Protocol

This is a connectionless protocol for distributed operating systems. It has been used in the Amoeba distributed system [Mullender et al. 1990]. Its main features include transparency, security, easy network management, group communication facility, and efficient client-server-based communication facility. The following description is based on the material presented in [Kaashoek et al. 1993].

For transparency, FLIP identifies entities, called *network service access points* (*NSAPs*), with location-independent 64-bit identifiers. Sites on an internetwork can have more than one NSAP, typically one or more for each entity (e.g., process). Each site is connected to the internetwork by a *FLIP box* that either can be a software layer in the operating system of the corresponding site or can be run on a separate communications processor. Each FLIP box maintains a routing table that is basically a dynamic hint cache mapping NSAP addresses on data-link addresses. Special primitives are provided to dynamically register and unregister NSAP addresses into the routing table of a FLIP box. An entity can register more than one address in a FLIP box (e.g., its own address to receive messages directed to the entity itself and the null address to receive broadcast messages). FLIP uses a one-way mapping between the private address used to register an entity and the public address used to advertise the entity. A one-way encryption function is used to ensure that one cannot deduce the private address from the public address. Therefore, entities that know the (public) address of an NSAP (because they have communicated with it) are not able to receive messages on that address, because they do not know the corresponding private address.

The FLIP messages are transmitted unreliably between NSAPs. A FLIP message may be of any size less than $2^{32} - 1$ bytes. If a message is too large for a particular network, it is fragmented into smaller chunks, called *fragments*. A fragment typically fits in a single network packet. The basic function of FLIP is to route an arbitrary-length message from the source NSAP to the destination NSAP. The path selection policy is based on the information stored in the routing tables of each FLIP box about the networks to which it is connected. The two main parameters used for this purpose are the *network weight* and a *security bit*. A low network weight means that the network is desirable on which to

forward a message. The network weight can be based, for example, on physical properties of the network, such as bandwidth and delay. On the other hand, the secure bit indicates whether sensitive data can be sent unencrypted over the network or not.

The three types of calls provided in FLIP for sending a message to a public address are *flip_unicast*, *flip_multicast*, and *flip_broadcast*. These calls provide both point-to-point and group communication facilities. The group communication protocols make heavy use of *flip_multicast*. This has the advantage that a group of *n* processes can be addressed using one FLIP address, even if they are located on multiple networks.

Although FLIP does not encrypt messages itself, it provides the following two mechanisms for secure delivery of messages. In the first mechanism, a sender can mark its message sensitive by using the *security* bit. Such messages are routed only over trusted networks. In the second mechanism, messages routed over an untrusted network by a FLIP are marked unsafe by setting the *unsafe* bit. When the receiver receives the message, by checking the unsafe bit, it can tell the sender whether or not there is a safe route between them. If a safe route exists, the sender tries to send sensitive messages in unencrypted form but with the *security* bit set. If at some stage during routing no further trusted path is found for the message (which can only happen due to network configuration changes), it is returned to the sender with the *unreachable* bit set. If this happens, the sender encrypts the message and retransmits it with the *security* bit cleared. Therefore, message encryption is done only when required.

The FLIP supports easy network management because dynamic changes in network configuration are automatically handled. The only network management job that requires human intervention is the specification of trusted and untrusted networks. That is, FLIP relies on the system administrator to mark a network interface as trusted or untrusted, because FLIP itself cannot determine if a network can be considered trusted.

One requirement for which FLIP does not provide full support is wide-area networking. Although FLIP has been used successfully in small WANs, it does not scale well enough to be used as the WAN communication protocol in a large WAN. FLIP designers traded scalability for functionality because they felt that wide-area communication should not be done at the network layer, but in higher layers.

Further details of the FLIP protocol can be found in [Kaashoek et al. 1993].

2.6 INTERNETWORKING

We saw in Chapter 1 that two desirable features of distributed systems are extensibility and openness. Both these features call for a need to integrate two or more networks (possibly supplied by different vendors and based on different networking standards) to form a single network so that computers that could not communicate because they were on different networks before interconnection can now communicate with each other. Interconnecting of two or more networks to form a single network is called *internetworking*, and the resulting network is called an *internetwork*. Therefore, a WAN of multiple LANs is an internetwork.

Internetworks are often heterogeneous networks composed of several network segments that may differ in topology and protocol. For example, an internetwork may

have multiple LANs, some of which may have multiaccess bus topology while others may have ring topology; some of these LANs may be using Ethernet technology while others may be using Token Ring technology; and some segments of the network may be using the IP suite while others may be using IBM's SNA (System Network Architecture) protocol suite. Internetworking allows these relatively unrelated networks to evolve into a single working system. That is, the goal of internetworking is to hide the details of different physical networks, so that the resulting internetwork functions as a single coordinated unit.

The three important internetworking issues are how to interconnect multiple (possibly heterogeneous) networks into a single network, which communication medium to use for connecting two networks, and how to manage the resulting internetwork. Some commonly used technologies to handle these issues are described below. An important point to remember here is that handling of internetworking issues becomes much easier if the network segments of the resulting internetwork were designed using widely accepted standards instead of proprietary topologies and protocols. If an organization has designed its networks using nonstandard technologies, interconnection to global networks may require tearing of the existing networks and starting over again. Therefore, adherence to standards is very important from an internetworking point of view.

2.6.1 Interconnection Technologies

Interconnection technologies enable interconnection of networks that may possibly have different topologies and protocols. Interconnecting two networks having the same topology and protocol is simple because the two networks can easily communicate with each other. However, interconnecting two dissimilar networks that have different topologies and protocols requires an internetworking scheme that provides some common point of reference for the two networks to communicate with each other. That point of reference might be a high-level protocol common to the two networks, a device that allows interconnection of different topologies with different physical and electrical characteristics, or a protocol that allows operating environment differences to be ignored. The most commonly used approach is to make use of common high-level protocols for moving data between common layers on a communications model such as the OSI or the IP suites. Internetworking tools, such as bridges, routers, brouters, and gateways, make extensive use of this approach. As described next, each of these tools has strengths, weaknesses, and specific applications in internetworking. These tools are "blackbox" internetworking technologies that enable interconnection of similar or dissimilar networks to form a single network system.

Bridges

Bridges operate at the bottom two layers of the OSI model (physical and data link). Therefore, they are used to connect networks that use the same communication protocols above the data-link layer but may or may not use the same protocols at the

physical and data-link layers. For example, bridges may be used to connect two networks, one of which uses fiber-optic communication medium and the other uses coaxial cable; or one of which uses Ethernet technology and the other uses Token Ring technology. But both networks must use the same high-level protocols (e.g., TCP/IP or XNS) to communicate.

The similarity of higher level protocols implies that bridges do not modify either the format or the contents of the frames when they transfer them from one network segment to another (they simply copy the frames). Hence bridges feature high-level protocol transparency. They can transfer data between two network segments over a third segment in the middle that cannot understand the data passing through it. As far as the bridge is concerned, the intermediate segment exists for routing purposes only.

Bridges are intelligent devices in the sense that they use a process of learning and filtering in data forwarding to keep network traffic within the segment of the network to which it belongs. Therefore, bridges are also useful in network partitioning. When the performance of a network segment degrades due to excessive network traffic, it can be broken into two network segments with a bridge interconnecting the two segments.

Routers

Routers operate at the network layer of the OSI model. Therefore, routers do not care what topologies or access-level protocols the interconnected network segments use. Since routers use the bottom three layers of the OSI model, they are usually used to interconnect those networks that use the same high-level protocols above the network layer. Note that the protocols of data-link and physical layers are transparent to routers. Therefore, if two network segments use different protocols at these two layers, a bridge must be used to connect them.

Unlike bridges, routers do not view an internetwork from end to end. That is, bridges know the ultimate destination of a data, but routers only know which is the next router for the data being transferred across the network. However, routers are smarter than bridges in the sense that they not only copy a data from one network segment to another, but they also choose the best route for the data by using information in a routing table to make this decision. That is, managing traffic congestion is a big plus of routers; they employ a flow control mechanism to direct traffic on to alternative, less congested paths.

Routers are commonly used to interconnect those network segments of large internetworks that use the same communication protocol. They are particularly useful in controlling traffic flow by making intelligent routing decisions.

An internetwork often uses both bridges and routers to handle both routing and multiprotocol issues. This requirement has resulted in the design of devices called *brouters*, which are a kind of hybrid of bridges and routers. They provide many of the advantages of both bridges and routers. They are complex, expensive, and difficult to install, but for very complex heterogeneous internetworks in which the network segments use the same high-level communication protocols, they often provide the best internetworking solution.

Gateways

Gateways operate at the top three layers of the OSI model (session, presentation, and application). They are the most sophisticated internetworking tools and are used for interconnecting dissimilar networks that use different communication protocols. That is, gateways are used to interconnect networks that are built on totally different communications architectures. For instance, a gateway may be used to interconnect two networks, one of which uses the IP suite and the other uses the SNA protocol suite.

Since networks interconnected by a gateway use dissimilar protocols, protocol conversion is the major job performed by gateways. Additionally, gateways sometimes also perform routing functions.

2.6.2 Which Communication Medium to Use?

Another important issue in internetworking is to decide the communication medium that should be used to connect two networks. This largely depends on the locations of the two networks and the throughput desired. For example, *FDDI* (*Fiber Distributed Data Interface*), specified by the American National Standards Institute (ANSI), operates at a speed of 100 Mbps and is an ideal high-bandwidth backbone for interconnecting LANs that are located within a multistory building or housed in several buildings in a campuslike environment.

If the two networks are located a little far from each other (such as on opposite sides of a town or in nearby cities), then they may be connected by leased telephone lines if the data traffic between the two networks is not heavy. If the data traffic between the two networks is heavy, dedicated lines may be used to interconnect the two networks. Use of dedicated lines is also suggested for security reasons when the data exchanged between the two networks often contains sensitive information.

Finally, if the two networks are located very far from each other (such as in different countries or in two distantly located cities of a country), then communication channels of public data networks, such as telephone lines or communication satellites, may be used to interconnect them. Long-haul communication channels are expensive. Moreover, if communication channels of a public data network are used to interconnect the two networks, inconsistencies of traffic and system reliability influence the data throughput between the two networks. Security is also a problem in this case. The data throughput problem may be solved to some extent by using one's own method of traffic routing. For example, if the two networks are located in New York and Los Angeles, the total traffic between the two networks may be routed through both Denver and Dallas for better throughput. Methods to handle the security problem are described in Chapter 11.

2.6.3 Network Management Technologies

Network management deals with the monitoring and analysis of network status and activities. Network monitoring tools watch network segments and provide information on data throughput, node and link failures, and other global occurrences on the network that may be useful in some manner to network managers. Simple network monitoring tools

report the existence, if not the cause, of a problem in any part of the network. On the other hand, network analysis tools analyze network activities from a wide variety of angles at a depth that includes packet-level protocol analysis. They add quantitative information to the monitor's qualitative data by providing a wide array of complete information about the operation of a network.

Management of an internetwork is more complex than management of a simple independent LAN because local problems become global problems when several LANs are interconnected to form an internetwork. Pinpointing the location of a problem in an internetwork is tricky because the problem may be on a local or remote LAN. Therefore, the tools available for managing a single independent LAN either work poorly or do not work at all when applied to internetworks. The heterogeneous nature of internetworks is also a major obstacle in the design of widely acceptable management tools for internetworks. Every manager of an internetwork dreams of a single tool that has the following features:

1. It can enfold all the protocols and devices found on a typical heterogeneous internetwork.
2. It should be easy to use. Highly graphical user interface that allows rapid user interaction and reduces the need for highly skilled network analysts at most levels is desirable.
3. It should be intelligent in the sense that it can learn and reason as it isolates network faults.
4. It should have no preferences regarding a device's vendor or protocol.

Unfortunately, no such tool exists at present. However, the future looks encouraging, as several network management frameworks and network management profiles are on the way. To provide for management of future interoperable multivendor networks, the ISO, the IETF, the OSF, and other organizations are currently developing management standards for communications networks based on several reference models and network management frameworks. Of the various emerging standards, three standards seem to be promising for future network management tools. These are *Simple Network Management Protocol* (*SNMP*), *Common Management Information Protocol* (*CMIP*), and *Distributed Management Environment* (*DME*).

The SNMP is a client-server protocol suitable for applications that operate in the client-server mode and do not require real-time notification of faults. It was introduced in the late 1980s to control and monitor networks that use the IP suite. Because of its simplicity for implementation and lower costs, SNMP-based tools are being implemented by most of the network management element vendors. To address speed, security, and manager-to-manager communication capability, IETF is working on version 2 of SNMP protocols. Further details of SNMP may be found in [Datapro 1990, Shevenell 1994, Janet 1993].

The CMIP is a network management standard developed by OSI (ISO/CCITT). It is designed to facilitate interoperability and true integration between large numbers of separate, isolated network management products and services in a multivendor

environment. It is based on a manager-agent model. The managing system invokes the management operations, and the managed system forwards the notifications to the manager. Communications between managing systems is also facilitated by the agent-manager role. Further details of CMIP may be found in [IT 1990, Janet 1993].

There are a lot more SNMP-based products available as network management tools as compared to CMIP-based products. In spite of providing richer functionality and more sophisticated features than SNMP-based products, CMIP-based products are not enjoying rapid growth because they are more expensive, more complex, and require more processing power to implement.

The OSF's DME is a set of specifications for distributed network management products. Its goal is to provide a framework to enable a consistent system and network management scheme across a global, multivendor distributed environment. To achieve this goal, the design of DME has been based on SNMP, CMIP, and other de facto standards. DME-based products are not yet available in the market. Further details of DME may be found in [Datapro 1993].

2.6.4 Internetwork Case Study: The Internet

The Internet is the best example of an internetwork. It is a single worldwide collection of interconnected heterogeneous networks that share a uniform scheme for addressing host computers and a suite of agreed protocols. Hosts and other resources on the Internet are named by using the DNS (Domain Name System) naming scheme described in Chapter 10.

The Internet has its roots in the ARPANET system of the Advanced Research Projects Agency of the U.S. Department of Defense. ARPANET was the first WAN and had only four sites in 1969. The Internet evolved from the basic ideas of ARPANET and was initially used by research organizations and universities to share and exchange information. Since restrictions for commercial use were lifted in 1989, the Internet has rapidly grown into an internetwork that now interconnects more than 10,000 networks, allowing more than 3 million computers and more than 40 million computer users in more than 150 countries around the world to communicate with each other. The Internet continues to grow at a rapid pace.

The Internet is a vast ocean of information that is of interest to a wide variety of users. Several user-friendly tools are available that allow users to successfully navigate the Internet and find useful information for one's own purposes. A few of the most popular of these tools are Gopher, Archie, WAIS (Wide-Area Information Servers), WWW (World-Wide Web), and Mosaic.

Gopher [Martin 1993] is a text-based tool that provides hierarchical collections of information of all sorts across the Internet. It is a seemingly endless maze of directory menus. Developed at the University of Minnesota in 1991, Gopher is currently the most commonly used tool to locate information on the Internet. For further details, ftp to *boombox.micro.umn.edu* and look in the directory */pub/gopher/docs*.

Archie is a collection of tools that allows searching of indexes of files available on public servers by anonymous *ftp* on the Internet. For further details, gopher to *gopher.gmu.edu*.

Wide-Area Information Servers (*WAIS*) is a group of freeware, shareware, and commercial software programs that help users locate information on the Internet. For further details, gopher to *gopher.gmu.edu*.

The World-Wide Web (*WWW*) is a hypermedia distribution system that uses hypertext links to other textual documents or files. With this facility, users can click on a highlighted word or words in a document to provide additional information about the selected word(s). With WWW, users can also access graphic pictures, images, audio clips, or even full-motion video that is set up at sites all over the world to provide a wealth of useful information. WWW was invented by the European Centre for Nuclear Research (CERN) in 1992 in an attempt to build a distributed hypermedia system. WWW traffic is the fastest growing part of the Internet and it is today's preferred vehicle for the Internet commerce. For further details, refer to [Vetter et al. 1994] or gopher to *info.cern.ch*.

Mosaic is a hypermedia-based browsing tool for finding and retrieving information from the Internet. Mosaic browsers are currently available for UNIX workstations running X Windows, PCs running Microsoft Windows, and the Apple Macintosh computers. Mosaic can access data in WWW servers, WAIS, Gopher servers, Archie servers, and several others. Its popularity is rapidly increasing because of its many useful features and capabilities. For further details, refer to [Vetter et al. 1994] or anonymous ftp to *ftp.NCSA.uiuc.edu* and look in the directory */PC/Mosaic*.

The worldwide scope of the Internet makes it perhaps the single most valuable tool for use in many significant ways by both non-profit and commercial organizations. Some of the important current strategic uses of the Internet are listed here. The following description is based on the material presented in [Nejmeh 1994]:

1. *On-line communication*. The electronic mail service (known as *e-mail*) on the Internet is extensively used today by computer users around the world to communicate with each other. With this facility, the Internet has proved to be a rapid and productive communication tool for millions of users. As compared to paper mail, telephone, and fax, e-mail is preferred by many because (a) it is faster than paper mail; (b) unlike the telephone, the persons communicating with each other need not be available at the same time; and (c) unlike fax documents, e-mail documents can be stored in a computer and be easily manipulated using editing programs.

2. *Software sharing*. The Internet provides access to a large number of shareware software development tools and utilities. A few examples of such available shareware tools are C++ compilers, code libraries, mail servers, and operating systems (all available via *ftp* from *sunsite.unc.edu*). The Free Software Foundation also provides a wealth of GNU software (for details anonymous ftp to *prep.ai.mit.edu* and look in the directory */pub/GNU*).

3. *Exchange of views on topics of common interest*. The Internet has a number of news groups. Each news group allows a group of users to exchange their views on some topic of common interest. For example, the news group *comp.os.os2.advocacy* contains candid dialog about the OS/2 operating system.

4. *Posting of information of general interest.* The Internet is also being extensively used as a large electronic bulletin board on which information of general interest can be posted to bring it to the attention of interested users around the world. Some commonly posted information include career opportunities, conference and event announcements, and calls for papers for conferences and journals.

5. *Product promotion.* Several commercial organizations are effectively using the Internet services for promoting their products. These organizations make use of corporate *ftp*, Gopher, or WWW server sites focused on disseminating timely information about corporate happenings, product announcements, recent strategic alliances, press releases, and other information of potential interest to existing and prospective customers. For example, *comp.sys.sun.announce* news group contains information about Sun Microsystem's latest product announcements.

6. *Feedback about products.* In addition to product promotion, commercial organizations are also using the Internet to gather information about user satisfaction of existing products, market opportunities of new products, and ideas for potential new products. This is usually accomplished by putting up an interactive survey application by the organization on a WWW or Gopher site on the Internet.

7. *Customer support service.* Many software organizations are also using the Internet to provide unprecedented levels of timely customer support. The combined electronic mail, *ftp*, and other services on the Internet provide all of the enabling tools necessary to provide such first-rate customer support. For example, bugs in fielded software products can be reported to an organization via electronic mail, and bug fixes, minor releases, workarounds, known problems and limitations, and general advice about a product can be made available by an organization to its customers via an *ftp* server.

8. *On-line journals and magazines.* The Internet now has literally hundreds of electronic subscriptions that can be found both for free and low cost. There are many Gopher and WWW sites on the Internet that deal with electronic versions of many journals and magazines. For example, Dow Jones News/Retrieval provides fee-based access to the electronic version of the *Wall Street Journal* on the Internet. Researchers are working in the direction to extend this idea to support full-fledged electronic libraries on the Internet.

9. *On-line shopping.* The Internet has also facilitated the introduction of a new market concept that consists of virtual shops. These shops remain open 24 hours all the year round and are accessible to purchasers all around the world. They provide information about products or services for sale through *ftp*, Gopher, or WWW servers. Using the Internet services, customers submit specific product queries and request specific sales quotes. Through a well-defined authorization and authentication scheme, the Internet services are then used to accept orders placed by the customers, to handle order payments, and to track orders to fulfillment. For example, the Internet Mall is a collection of shops, each providing several products or services for sale. For a list of the available products or services at the Internet Mall, ftp to *ftp.netcom.com* and look in the directory */pubs/Guides*.

10. *Worldwide video conferencing*. Worldwide video conferencing is an emerging service on the Internet that allows a group of users located around the globe to talk and interact with each other as if they were sitting and discussing in a single room. The CU-SeeMe system developed at Cornell University is an example of an Internet-based video-conferencing system. For information on CU-SeeMe, ftp to *gated.cornell.edu* and look in the directory */pub/video/CU-SeeMe.FAQ.7-6.txt*.

2.7 ATM TECHNOLOGY

Asynchronous Transfer Mode (*ATM*) is often described as the future computer networking paradigm. It is a high-speed, connection-oriented switching and multiplexing technology that uses short, fixed-length packets (called *cells*) to transmit different types of traffic simultaneously, including voice, video, and data. It is asynchronous in that information streams can be sent independently without a common clock. This emerging technology is briefly described next. For a more complete treatment of the state of the art in ATM technology, see [DePrycker 1993, Newman 1994, Fischer et al. 1994, Haendel et al. 1994, Vetter 1995, Kim and Wang 1995].

2.7.1 Main Features of ATM Technology

ATM technology is expected to have an enormous impact on future distributed systems because of its following attractive features:

1. It enables high-bandwidth distributed applications by providing data transmission speeds of 155 Mbps, 622 Mbps, and potentially 2.5 Gbps. This feature will make possible several new distributed applications, such as applications based on video-on-demand technique, video-conferencing applications, and applications that need to access remote databases of multimedia data.

2. It provides high transmission speeds for both local and wide-area networks and services, enabling high-powered distributed applications that previously had little hope of extending beyond LAN environments to be used in WAN environments as well.

3. It supports both the fundamental approaches to switching (circuit switching and packet switching) within a single integrated switching mechanism (called *cell switching*). This feature makes it suitable both for distributed applications that generate *constant-bit-rate* (*CBR*) traffic and distributed applications that generate *variable-bit-rate* (*VBR*) traffic. For instance, applications dealing with video and digitized voice generate CBR traffic. Constant-bit-rate traffic requires guaranteed throughput rates and service levels. On the other hand, most data applications generate VBR traffic. Variable-bit-rate traffic can tolerate delays and fluctuating throughput rates.

4. It uses the concept of virtual networking to pass traffic between two locations. This concept allows the available bandwidth of a physical channel to be shared by multiple applications, enabling multiple applications to simultaneously communicate at

different rates over the same path between two end points. That is, it allows the total bandwidth available to be dynamically distributed among a variety of user applications.

5. In addition to point-to-point communication in which there is a single sender and a single receiver, it can easily support multicasting facility in which there is a single sender but multiple receivers. Such a facility is needed for transmitting broadcast television to many houses at the same time. Many collaborative distributed applications also require frequent use of this kind of facility.

6. It enables the use of a single network to efficiently transport a wide range of multimedia data such as text, voice, video, broadcast television, and so on. Therefore, the use of separate networks such as a telephone network for voice, an X.25 network for data, and a cable television network for video can now be replaced by a single ATM network that provides a means for integrating voice, video, data, and other information. This integration will in turn lead to substantial cost savings and simplification in the design of communication networks.

7. It is flexible in the way it grants access to bandwidth. That is, it enables supply of bandwidth on demand and allows billing network users on a per-cell basis (more probably on a giga-cell basis, given the speed and transfer capabilities of ATM). A user can grab as big or as small a chunk of network bandwidth as he or she needs and pay only for as much as he or she uses.

8. It is a scalable technology. It enables increase or decrease of such things as bandwidths and data rates and still maintains the architecture of the signaling process. Moreover, the same switching technology (cell switching) and the same cell format is used for the transport of all types of information (voice, video, and data) in both LAN and WAN environments.

9. It has a fairly solid base of standards. It has been adopted by the International Telecommunications Union (ITU) (formerly CCITT) and internationally standardized as the basis for the Broadband Integrated Services Digital Network (B-ISDN).

2.7.2 Basic Concepts of ATM Technology

There are two fundamental types of network traffic—CBR and VBR. The CBR traffic (comprising of video and digitized voice information) is smooth, whereas VBR traffic (comprising of data information) is bursty. The CBR traffic requires a low but constant bandwidth and guaranteed throughput rates and service levels. On the other hand, VBR traffic requires large bandwidth for very short periods of time at random intervals and can tolerate delays and fluctuating throughput rates. Due to this basic difference in characteristics of the two types of traffic, circuit switching is the most suitable networking technology for handling CBR traffic, whereas packet switching is the most suitable networking technology for handling VBR traffic. However, neither circuit switching nor packet switching is suitable for handling both classes of network traffic. Therefore, when the standards bodies of the ITU were working on a universal multiplexing and switching mechanism that could support integrated transport of multiple-bit-rate traffic in an

efficient and cost-effective way, they came up with the idea of a hybrid form of switching technique called *cell switching*. ATM technology is based on this cell-switching technique.

Cell-switching technology is based on the digital packet-switching technology, which relays and routes traffic over virtual paths by means of an address contained within the packet (this is different from circuit-switching technology, which routes traffic not by address but over dedicated physical paths established before communication starts). However, unlike more familiar packet-switching technologies, such as X.25 or frame relay, cell-switching technology uses very short, fixed-length packets, called *cells*. In ATM, cells are 53 bytes long. They consist of a 5-byte header (containing the address) and a 48-byte information field.

The cell size of 53 bytes was chosen to make ATM useful for data as well as voice, video, and other real-time traffic that cannot tolerate randomly varying transmission intervals and delays. Pure data sources can produce very long messages—up to 64 kilobytes in many cases. By segmenting such messages into short cells, ATM ensures that CBR traffic such as voice and video can be given priority and need never wait more than one 53-byte cell transmission time (3 μs at a 155-Mbps transmission rate) before it can gain access to a communication channel. With frame-based packet-switching technology, the waiting time would be random, possibly several milliseconds in length. The use of short, fixed-size cells also eliminates the danger of a small packet being delayed because a big one is hogging a needed line. In case of cell switching, after each cell is transmitted, a new one (even one belonging to a different message) can be sent.

The proper size of a cell in ATM was the subject of much debate with the standards committees. This is because telephony people were in favor of a small cell to reduce delay for voice packets, whereas data communications people were in favor of a large cell to minimize the overhead involved in the segmentation and reassembly of message data. After much discussion, the cell size debate was narrowed into two choices—32-byte cells or 64-byte cells. As a compromise, the ITU set the cell size at 48 bytes plus the header.

Notice that with a fixed cell size of 53 bytes, ATM technology is not an ideal choice either for applications dealing with CBR traffic such as voice and video or for applications dealing with VBR traffic such as file transfers. It is, however, the best technology on the horizon for handling both types of traffic on a single integrated network. Because of its fast, hardware-based switching, it can emulate the dedicated circuits usually required for handling CBR traffic. And because it is packet based, it can efficiently handle VBR traffic as well.

The ATM is a connection-oriented technology because a sender first establishes a connection with the receiver. However, unlike circuit switching, in which a physical circuit is established between the sender and the receiver and reserved for them for the entire duration of their communication session, in ATM technology, a virtual circuit is established between the sender and the receiver. That is, ATM does not reserve the path for one user exclusively. Any time a given user is not occupying a channel, another user is free to use it. Connection establishment in ATM means that a route is determined from the sender to the receiver and routing information is stored in the switches along the route during connection establishment. All cells of messages from the sender to the receiver

follow this virtual path stored in the switches. When the connection is no longer needed, it is released and the routing information for this connection is purged from the switches.

The address information in the header of each cell is used by the routing protocol to determine the virtual path that the cell will traverse. Addresses in ATM are only of local significance, in that they matter only between two adjacent pieces of ATM equipment. When a virtual path is established, each switch is provided with a set of lookup tables that identify an incoming cell by header address, route it through the switch to the proper output port, and overwrite the incoming address with a new one that the next switch along the route will recognize as an entry in its routing table. A message is thus passed from switch to switch over a prescribed route, but the route is virtual since the facility carrying the message is dedicated to it only while a cell of the message traverses it.

In ATM, a virtual path is essentially a bundle of virtual channels that can be multiplexed together. Therefore, over a single virtual path, two hosts may multiplex cells of many individual applications. Cells are statistically multiplexed, allowing the total available bandwidth to be dynamically distributed among a variety of distributed applications. This is achieved by selecting virtual channel paths according to the anticipated traffic and allocating the network resources needed. For guaranteed bandwidth applications, users are required to specify the amount of bandwidth required. It is the virtual nature of ATM services that provides greater network efficiencies.

2.7.3 ATM Protocol Reference Model

The protocol reference model in ATM is divided into three layers—physical layer, ATM layer, and ATM adaptation layer (AAL) (Fig. 2.12). Applications involving data, voice, and video are built on top of these three layers. The functionalities of the three layers are described next.

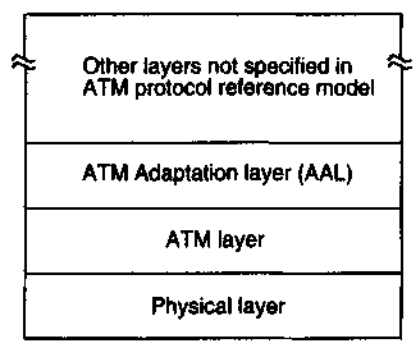


Fig. 2.12 ATM protocol reference model.

Physical Layer

The physical layer is the bottom-most layer of the ATM protocol suite. It is concerned with putting bits on the wire and taking them off again. It has two sublayers: the physical medium dependent (PMD) sublayer and the transmission convergence (TC) sublayer. The

PMD sublayer defines the actual speed for traffic transmission on the physical communication medium (electrical/optical) used. On the other hand, the TC sublayer defines a protocol for the encoding and decoding of cell data into suitable electrical/optical waveforms for transmission and reception on the physical communication medium defined by the PMD sublayer. The protocol of the TC sublayer differs according to the physical communication medium used.

The physical layer can transfer cells from one user to another in one of the following two ways:

1. *By carrying cells as a synchronous data stream.* In this case, the user-network interface (UNI), which takes the form of an ATM adaptor board plugged into a computer, puts out a stream of cells on to a wire or fiber. The transmission stream must be continuous, and when there is no data to be sent, empty cells are transmitted.

2. *By carrying cells in the payload portion of an externally framed transmission structure.* In this case, the UNI uses some standard transmission structure for framing and synchronization at the physical layer. *SONET (Synchronous Optical NETwork)* [Omidyar and Aldridge 1993], the most commonly used standard for this purpose, is briefly described next. The SONET format is currently supported by single-mode fiber, multimode fiber, and twisted pair.

SONET is an international suite of standards for transmitting digital information over optical interfaces. In SONET, the basic unit of data transmission is a frame whose structure is shown in Figure 2.13. As shown in the figure, a SONET frame consists of a total of 810 bytes (9×90), out of which 27 bytes (9×3) are overhead and the remaining 783 bytes (9×87) are payload. The overhead bytes are used for error monitoring, system maintenance functions, synchronization, and identification of payload type. The payload area can carry a variety of signals, such as several T1 signals, or a T3 signal, or several ATM virtual circuits. *T1* is a digital transmission service with a basic data rate of 1.544 Mbps, and *T3* is a digital transmission service with a basic data rate of 44.736 Mbps for transport of 28 T1 circuits.

The order of transmission of bytes is row by row, from left to right, with one entire frame transmitted every 125 μs. The basic time unit of one frame every 125 μs matches with the telephone system's standard sampling rate of 800 samples per second. Therefore, for the SONET frame format, the gross data rate is 51.840 Mbps (with the overhead bytes included), and the net data rate is 49.920 Mbps (with the overhead bytes excluded).

The basic unit of SONET with a bit rate of 51.840 Mbps is called *STS-1 (Synchronous Transport Signal Level 1)*. Higher rate SONET signals are obtained by byte-interleaving n frame-aligned STS-1's to form an STS-n signal [Vetter 1995].

STS uses an electrical rather than an optical signal. *Optical Carrier (OC)* levels are obtained from STS levels after scrambling (to avoid long strings of 1's and 0's and allow clock recovery at the receivers) and performing electrical-to-optical conversion [Vetter 1995]. Thus, OC-n level signals are obtained by scrambling and converting STS-n level signals. The most commonly used values of n are 1, 3, and 12, giving OC-1,

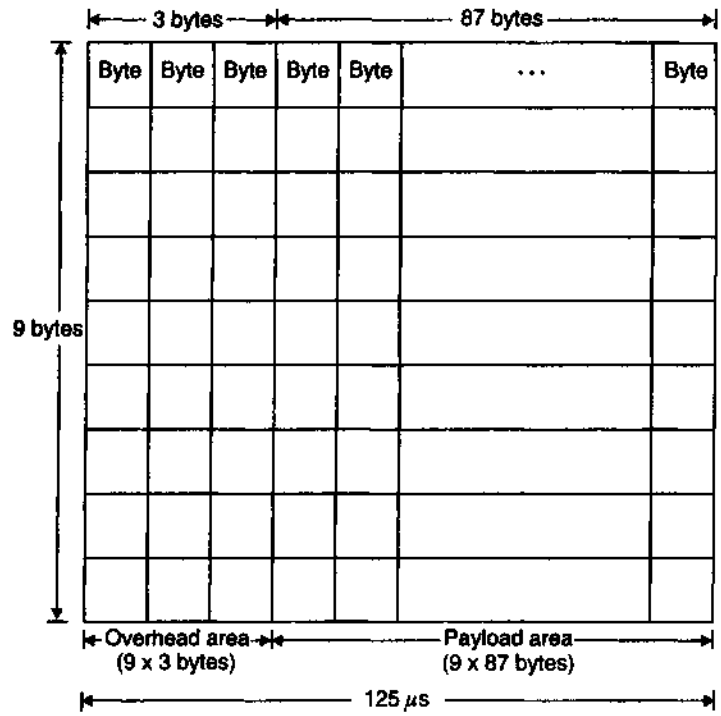


Fig. 2.13 SONET frame format.

OC-3, and OC-12 signal levels having data rates of 51.84, 155.52, and 622.08 Mbps, respectively.

In Europe, another standard for frame formatting called *SDH* (*Synchronous Digital Hierarchy*) [Omidyar and Aldridge 1993] is also available. For the SDH frame format, the gross data rate for the basic unit is 155.52 Mbps, instead of the 51.84 Mbps for SONET.

ATM Layer

The ATM layer handles most of the cell processing and routing activities. These include building the cell header, cell multiplexing of individual connections into composite flows of cells, cell demultiplexing of composite flows into individual connections, cell routing, cell payload type marking and differentiation, cell loss priority marking and reduction, cell reception and header validation, and generic flow control of cells. These functions are designed to be carried out in hardware at very high data rates. The ATM layer is independent of the physical medium used to transport the cells.

The functionality of the ATM layer is defined by the fields present in the ATM cell header (Fig. 2.14). These fields and their functions are as follows:

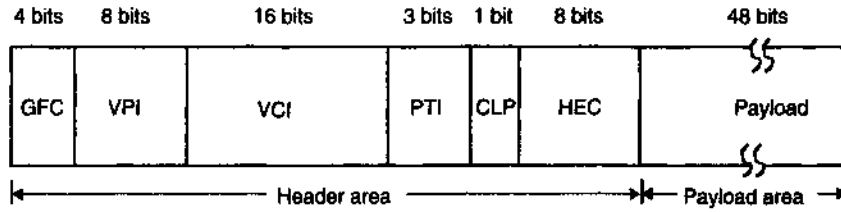


Fig. 2.14 ATM cell format: GFC, Generic Flow Control; VPI, Virtual Path Identifier; VCI, Virtual Channel Identifier; PTI, Payload Type Identifier; CLP, Cell Loss Priority; HEC, Header Error Control.

- *Generic flow control (GFC) field.* This field occupies 4 bits in an ATM cell header. The default setting of four 0's indicates that the cell is uncontrolled. An uncontrolled cell does not take precedence over another cell when contending for a virtual circuit. The bits in the GFC field can be suitably set to implement some form of prioritized congestion control. For example, the bits in the GFC field could be used to prioritize voice over video or to indicate that both voice and video take precedence over other types of data. The GFC field is also used by the UNI to control the amount of traffic entering the network, allowing the UNI to limit the amount of data entering the network during periods of congestion [Vetter 1995].
- *Virtual path identifier (VPI) and virtual channel identifier (VCI) fields.* The VPI field occupies 8 bits and the VCI field occupies 16 bits in an ATM cell header. These two fields are used by the routing protocol to determine the path(s) and channel(s) the cell will traverse. These fields are modified at each hop along the path. That is, when a cell arrives at an ATM switch, its VPI and VCI values are used to determine the new virtual identifier to be placed in the cell header and the outgoing link over which to transmit the cell. As implied by its name, the VPI field is used to establish virtual paths between network end-points. Recall that in ATM two hosts may multiplex cells of many individual applications over a single virtual path connection. This is achieved by the VCI fields of the cells that distinguish among cells of different applications and thus establish virtual links over a given virtual path.
- *Payload-type identifier (PTI) field.* This field occupies 3 bits in an ATM cell header. It is used to distinguish data cells from control cells so that user data and control data can be transmitted on different subchannels.
- *Cell loss priority (CLP) field.* This field occupies 1 bit in an ATM cell header. When set, it indicates that the cell can be discarded, if necessary, during periods of network congestion. For example, voice data may be able to suffer lost cells without the need for retransmission, whereas text data cannot. In this case, an application may set the CLP field of the cells for voice traffic.
- *Header error control (HEC) field.* This field occupies 8 bits in an ATM cell header. It is used to protect the header field from transmission errors. It contains a checksum of only the header (not the payload).

ATM Adaptation Layer

The functionality of the physical and the ATM layers of the ATM protocol suite is not tailored to any application. We saw that ATM can support various types of traffic, including voice, video, and data. The AAL is responsible for providing different types of services to different types of traffic according to their specific requirements. It packages various kinds of user traffic into 48-byte cells, together with the overhead needed to meet specific quality-of-service requirements of different types of traffic. To reflect the spectrum of applications, four service classes were defined by the ITU (Fig. 2.15):

1. *Class A.* Applications having delay-sensitive CBR traffic that require connection-oriented service belong to this class. Video and voice applications normally belong to this class.
2. *Class B.* Applications having delay-sensitive VBR traffic that require connection-oriented service belong to this class. Some video and audio applications belong to this class.
3. *Class C.* Applications having VBR traffic that are not delay-sensitive but that require connection-oriented service belong to this class. Connection-oriented file transfer is a typical example of an application that belongs to this class.
4. *Class D.* Applications having VBR traffic that are not delay-sensitive and does not require connection-oriented service belong to this class. LAN interconnection and electronic mail are typical examples of applications that belong to this class.

Service class	Class A	Class B	Class C	Class D
Bit rate type	CBR	VBR	VBR	VBR
Delay sensitive?	Yes	Yes	No	No
Connection oriented?	Yes	Yes	Yes	No
AAL protocol to be used	AAL1	AAL2	AAL3/4 or AAL5	AAL3/4 or AAL5

Fig. 2.15 Service classes for the ATM Adaptation Layer (AAL).

To support the four service classes, initially the ITU recommended four types of AAL protocols, called AAL1, AAL2, AAL3, and AAL4. It was soon discovered that the differences between AAL3 and AAL4 protocols are minor. Therefore, they were later merged into a single protocol, called AAL3/4.

It was later discovered that the mechanisms of the AAL3/4 protocol were fairly complex for computer data traffic. Therefore, a new protocol called AAL5 was later added

to the AAL layer for handling computer data traffic [Suzuki 1994]. The AAL5 protocol is also called *SEAL (Simple and Efficient Adaptation Layer)*.

As shown in Figure 2.15, class A traffic will use the AAL1 protocol, class B traffic the AAL2 protocol, and class C and D traffic either AAL3/4 or AAL5. Since both AAL3/4 and AAL5 protocols are meant for use by class C and D traffic, it is important to know the basic differences between the two protocols. AAL3/4 performs error detection on each cell and uses a sophisticated error-checking mechanism that consumes 4 bytes of each 48-byte payload. AAL3/4 allows ATM cells to be multiplexed. On the other hand, AAL5 uses a conventional 5-byte header (no extra byte from the payload of a cell) and it does not support cell multiplexing.

2.7.4 ATM Networks

In its simplest form, an ATM network has a mesh-star architecture with two or more ATM switches interconnected with copper or optical cables and the host computers connected to the ATM switches. Figure 2.16 shows such an ATM network with three ATM switches and nine host computers. Cells originating at any of the nine host computers can be switched to any of the other host computers attached to the system by traversing through one or more ATM switches. This simple form is normally suitable for local area ATM networks. In addition to ATM switches and host computers, a wide-area ATM network also contains internetworking devices, such as routers, gateways, and interfaces, to the public network.

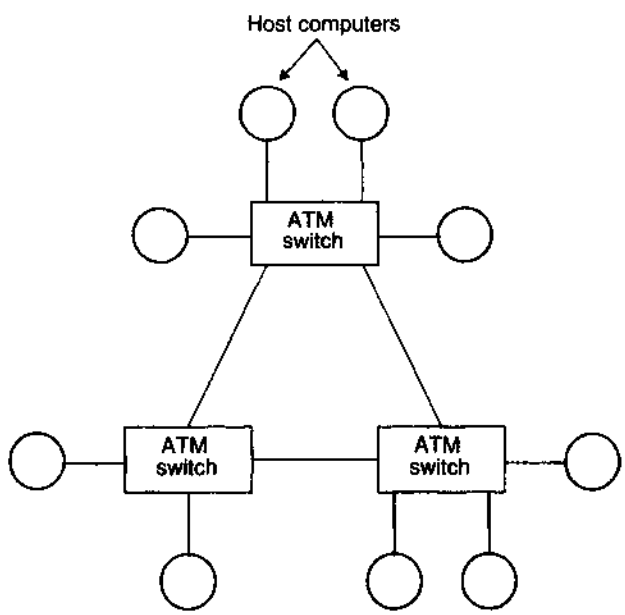


Fig. 2.16 An ATM network.

An ATM switch has several input and output ports. Each input port has an input port controller, and each output port has an output port controller. Each input port controller consists of a table, referred to as the VCI table, which maps the VPI and VCI of an incoming cell to an output VCI and an output port address. Before an incoming cell is released by an input port controller to the switching fabric of the switch, the VCI of the cell is replaced by the output VCI, and the output port address is appended for self-routing. Each ATM switch also has a switch controller that performs different switch management functions, including updating the tables of the input port controllers. Moreover, each ATM switch usually also has buffers to temporarily store data when cells arriving at different input ports contend for the same output port. Separate buffers may be associated either with the input ports or with the output ports, or the switch may have a pool of buffers that can be used for both input and output buffering.

The ATM switches that contain only a few ports are cheaper and easier to build than switches containing many ports. Local area ATM networks normally contain a small number of ATM switches with only a few ports per switch, whereas wide-area ATM networks normally contain a large number of ATM switches with many ports per switch.

Each host computer in an ATM network is assigned an ATM address that could be based either on a hierarchical 8-byte-long ISDN telephone number scheme E.164 or a 20-byte address proposed by the ATM Forum [ATM Forum 1993]. The latter is modeled after the address format of an OSI network service access point.

The ATM networks having protocol support for a mixture of high-level communication services (e.g., TCP/IP, UDP/IP, Berkeley Software Distributor [BSD] sockets, and RPC) may also be used as backbone networks to interconnect existing networks.

2.7.5 Problems and Challenges

As a networking technology, ATM possesses many attractive features, including enormously high bandwidth, scalability, traffic integration, statistical multiplexing, and network simplicity. With these features, ATM technology is certainly going to have a significant impact on the design of future distributed systems. However, for the success of ATM as a networking technology for future distributed systems, several problems have yet to be solved. These problems offer a new set of challenges to network designers and users. Some of the most important of these problems that are currently under investigation by researchers working in the area of ATM technology are briefly described next.

Interoperability

If ATM is to succeed as a networking technology, it must interwork with existing installed bases. This property is important to the acceptance of ATM since it will allow a huge number of existing distributed applications to be run over ATM networks. Two remarkable efforts being made in this direction are *LAN emulation over ATM* by the ATM Forum (the primary organization developing and defining ATM standards) [ATM Forum 1994] and *IP over ATM* by the IETF (Internet Engineering Task Force) [Chao et al. 1994, Laubach 1994, Brazdziunas 1994]. They are described next.

LAN Emulation over ATM. The LAN emulation over ATM (called *LAN emulation*) deals with enabling existing LAN-based distributed applications to be run over ATM networks. It also enables interconnection of ATM networks with traditional LANs.

Most existing LANs are based on shared-media interconnects and employ the IEEE 802 family of LAN protocols, which includes the Ethernet (802.3), the Token Bus (802.4), and the Token Ring (802.5) (see Section 2.5.1). Recall from Figure 2.9 that in the IEEE 802 model the data-link layer protocol of the ISO reference model is divided into two layers—the medium-access-control (MAC) layer, which defines the mechanisms that are used to access, share, and manage the communication medium, and the logical-link-control (LLC) layer, which defines a common interface for different network layer protocols to interwork with different MAC protocols. Each host attached to a LAN has a globally unique MAC address. MAC addresses are 48 bits long and form a flat address space. A host can send data to another host only if it knows the MAC address of the receiving host (if both the hosts are in the same LAN) or the MAC address of the next hop router (if both the hosts are in different LANs).

The key idea behind LAN emulation is to design a separate protocol layer, referred to as the ATM-MAC layer, above the AAL and below the LLC layer. Two key functions that must be supported by the ATM-MAC layer are as follows:

1. Emulation of the physical, broadcast, shared medium of LANs for supporting broadcast communication facility
2. Resolution of MAC addresses to ATM addresses for supporting point-to-point communication facility

The ATM emulates the physical, broadcast, shared medium of a conventional LAN by establishing an ATM multicast virtual connection between all of the hosts that are directly connected to the ATM network, referred to as the LAN emulation clients (LECs). This multicast connection is the broadcast channel of the ATM LAN segment. Any LEC may broadcast to all others on the ATM LAN segment by transmitting on the multicast virtual connection.

For point-to-point communication, an address resolution protocol is required to translate the 48-bit MAC address to an ATM address. Translating MAC addresses to ATM addresses can be done using either a broadcast-based approach or a server-based approach. The broadcast-based approach relies on the switch broadcast capability. On the other hand, in the server-based approach, a LAN emulation server (LES) is placed on a globally known virtual channel, and a set of query/response messages is implemented for interaction between the LES and the LECs. All MAC-to-ATM address resolution requests are sent to the LES, which responds to the requester using a predetermined virtual channel.

Once the ATM address of the receiving host has been obtained, a point-to-point ATM virtual connection may be established between the sending and receiving hosts by using the ATM signaling protocol. The result of the address resolution and the VCI of the established connection are cached in a table in the sending host on the assumption that further communication with the receiving host is likely. This mechanism operates entirely within

the ATM-MAC layer and is totally transparent to the LLC and higher layer protocols in the hosts. Further details of LAN emulation can be found in [Newman 1994].

IP over ATM. The IP over ATM deals with enabling existing distributed applications that have been designed for the IP suite to be run over ATM networks. Two cases that need to be considered for supporting IP over ATM are as follows:

1. Supporting IP over an ATM network that has LAN emulation functionality implemented over it
2. Supporting IP over a pure ATM network that consists of only ATM switches

Since LAN emulation defines the ATM-MAC layer above the AAL and below the LLC, supporting IP over an emulated LAN is the same as supporting IP over any IEEE 802 LAN. However, when IP is to be supported over a pure ATM network, it is possible to simplify the protocol stack and run IP directly over ATM. For implementing IP directly over ATM, an address resolution protocol is required to translate an IP address to an ATM address. Once again the server-based approach (as described for LAN emulation) can be used for this purpose. With IP over ATM, the address resolution server, referred to as the IP-ATM-ARP server, maintains tables that contain mapping information to map IP addresses to ATM addresses.

When a new host is added to the ATM network, it first goes through a registration process and obtains an ATM address for itself. It then sends a message to the IP-ATM-ARP server, requesting it to add its address resolution information in the mapping table. The message contains the IP and ATM addresses of the newly added host. The server updates the mapping table, allocates a new reserved VCI to the host, and returns the VCI to the host. The host uses this VCI to discriminate between messages received from the server and other hosts. The above process can also be adopted to allow port mobility of hosts. Further details of IP over ATM can be found in [Chao et al. 1994].

Bandwidth Management

The ATM networks are meant for carrying a mix of synchronous, asynchronous, and isochronous traffic. To achieve the desired quality of service, users are often required to specify the bandwidth requirement for their applications. Two important issues that need to be resolved in this connection are how users estimate and indicate the bandwidth requirements of their applications to the network and how the network allocates the bandwidth to the applications to make best use of the available bandwidth while satisfying the requests of the applications. Some proposals made by researchers working in this area are as follows [Turner 1992, Vetter 1995]:

1. *Peak-rate allocation method.* In this method, users only specify the maximum traffic rate for their applications. Based on user's requests, the network assigns virtual channels to the applications in such a manner that on every link of the network the sum of the rates on the virtual channels is no more than the link's maximum cell rate. If an application's traffic exceeds its specified peak rate, its cells are simply discarded.

2. *Minimum-throughput allocation method.* In this method, users specify the desired minimum throughputs for their applications, and the network guarantees the specified throughputs to the applications on a best-effort basis.

3. *Bursty-traffic specification method.* In this method, users specify the maximum and average traffic rates plus the maximum burst size for their applications. These parameters are used to properly configure the network to allocate the available bandwidth to the applications to meet their specified requirements.

None of these methods has been found to be satisfactory. For instance, Vetter [1995] points out that the first method offers a strong performance guarantee and is easy to implement, but it may make poor use of the available network bandwidth in case of bursty traffic; the second method can provide high efficiency, but its performance guarantee is weak; and the third method involves large computation overhead in computing when a new virtual channel can be safely multiplexed with other virtual channels. Moreover, all three methods suffer from two additional drawbacks. First, since end-to-end protocols normally operate on data units comprising several cells, for an application needing reliable transport service, the loss or discard of a single cell forces retransmission of the entire data, resulting in lower protocol throughput [Vetter 1995]. The problem becomes more acute in a wide-area ATM network because the retransmission may also result in discarding a large amount of data already in the connection pipe. This can have a serious performance implication. The second drawback is that none of the three methods adequately handles multicast virtual circuits [Vetter 1995]. Therefore, new bandwidth management mechanisms that can overcome the problems mentioned previously are needed.

Latency/Bandwidth Trade-off

Kleinrock [1992] pointed out that the latency/bandwidth trade-off in high-speed wide-area ATM networks will pose new challenges to the designers of these networks. This is primarily due to the fact that the propagation delay or speed of light over a wide area is several magnitudes greater than the time it takes to transmit an ATM cell. For instance, it takes roughly 15 ms for a bit to travel across the United States one way. At 1 Gbps, this time is more than 35,000 times greater than the time required to transmit a single ATM cell into the link [Vetter 1995]. This latency/bandwidth trade-off poses the following two main problems. These are the same problems described in [Tanenbaum 1995]:

1. Consider a WAN spanning across the United States. The round-trip propagation delay between two end sites of the WAN may be approximately 30 ms. Suppose the two sites are on a 622-Mbps ATM network. At 622 Mbps, it takes only about 1.6 ms (1/622 second) to pump all bits of a file of size 1 megabits on to the network. Since the round-trip propagation delay between the two sites is 30 ms, the receiver's reply can be received by the sender only after 31.6 ms. Therefore in this example, out of total 31.6 ms, the link was idle for 30 ms, or 95% of the total. The situation will be worse with higher speed networks and shorter messages. At higher speeds, the fraction of the available virtual circuit

bandwidth that can be effectively used will tend to zero. Hence, new protocols and system architectures are needed to deal with the latency problem in high-speed wide-area ATM networks.

2. Since the propagation delay is several magnitudes greater than the time it takes to transmit an ATM cell, a sender can send thousands of cells over the network before the first bit even arrives at the receiver's site. If the receiver does not possess a large amount of buffering capacity, most of the cells will be lost due to inadequate buffer space, and they will have to be retransmitted. This can have a serious performance implication. The use of the conventional sliding-window protocol for flow control to solve this problem does not work well in this case. This is because if it is decided that the sender must wait for an acknowledgment after sending every megabit of data, due to the latency problem already described above, the virtual circuit will be 95% idle. To solve this problem, areas that require particular attention include flow control, buffering, and congestion control. Some work performed in these areas may be found in [Hong and Suda 1991, Trajkovic and Golestani 1992, Eckberg 1992, Yazid and Mouftah 1992].

2.8 SUMMARY

A distributed system relies entirely on the underlying computer network for the communication of data and control information between the nodes of which they are composed. A computer network is a communication system that links the nodes by communication lines and software protocols to exchange data between two processes running on different nodes of the network.

Based on characteristics such as geographic distribution of nodes, data rate, error rate, and communication cost, networks are broadly classified into two types: LAN and WAN. Networks that share some of the characteristics of both LANs and WANs are sometimes referred to as MANs.

The two commonly used network topologies for constructing LANs are the multiaccess bus and ring. For both multiaccess bus and ring networks, a single channel is shared by all the sites of a network. Therefore, medium-access control protocols are needed to provide controlled access to the shared channel by all sites. Of many medium access control protocols, the CSMA/CD protocol is most commonly used for multiaccess bus networks, and the token ring and slotted ring protocols are used for ring networks.

A wide-area network of computers is constructed by interconnecting computers that are separated by large distances. Special hardware devices called packet-switching exchanges (PSEs) are used to connect the computers to the communication channels. The PSEs perform switching and routing activities to transmit data across the network. The two commonly used switching techniques are circuit switching and packet switching.

The selection of the actual path to be used to transmit a packet in a WAN is determined by the routing strategy used. The path used to transmit a packet from one site to another either may be fixed or may dynamically change based on network conditions. This depends on whether a static or a dynamic routing strategy is being used. Moreover, either the whole path may be decided at the source site or the subpath for each hop of the