IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re *Inter Partes* Review of:	)
U.S. Patent No. 9,146,777	)
Issued: Sep. 29, 2015	)
Application No.: 13/750,696	)
Filing Date: Jan. 25, 2013	)

For: Parallel Processing With Solidarity Cells By Proactively Retrieving From A Task Pool A Matching Task For The Solidarity Cell To Process

DECLARATION OF INGRID HSIEH-YEE, PH.D. IN SUPPORT OF PETITION FOR INTER PARTES REVIEW OF U.S. PATENT NO. 9,146,777

TABLE OF CONTENTS

I. Introduction....4

A. Qualifications and Professional Experience....4

B. Scope of This Declaration....13

C. Person of Ordinary Skills in the Arts....15

D. Evidence Considered in Forming My Opinions....16

II. Authentication and Public Availability of *DIX-Ethernet* (Ex. 1009)....19

A. Authentication....19

B. Internet Archive Copy and Records....21

C. ACM Copy of Ethernet Specification....24

D. University of Wisconsin Copy....24

E. Usage Record....26

F. Summary of My Opinion on *DIX-Ethernet* & *ACM-Ethernet*....26

III. AUTHENTICITY AND PUBLIC AVAILABILITY OF *Andrews* (Ex. 1011)....27

A. Authentication....27

B. Amazon Copy of Andrews....29

C. Library of Congress Records....30

D. Copyright Record....34

E. Internet Archive Records....35

F. Stanford University Records....38

Amazon
Exhibit 1005
Page 002

G. Usage Record 40
H. Summary of My Opinion on *Andrews* 40
IV. Authentication and Public Availability of *Batheja* (Ex. 1012) 41
A. Authentication 41
B. IEEE Database Record for *Batheja* 43
C. Copy of *Batheja* from the Linda Hall Library 46
D. Library Records from the Linda Hall Library 48
E. Usage Record 52
F. Summary of My Opinion on *Batheja* 52
V. Conclusion 54

Amazon
Exhibit 1005
Page 003

I. Introduction

I, Ingrid Hsieh-Yee, Ph.D., do hereby declare as follows:

1. I have personal knowledge of all of the matters about which I testify in this declaration.

2. I have been retained as an independent expert witness on behalf of Amazon.com, Inc. and Amazon Web Services, Inc. (collectively, "AMAZON") for proceedings before the United States Patent and Trademark Office ("PTO") regarding the authenticity and public availability of certain prior art references in relation to US Patent No. 9,146,777 ("the '777 patent").

3. I am being compensated for my work in this matter at my customary hourly rate. I am also being reimbursed for any reasonable expenses associated with my work and testimony in this investigation. My compensation is not contingent on the results of my study, the substance of my opinions, or the outcome of this matter.

A. Qualifications and Professional Experience

4. My complete qualifications and professional experience are described in my academic curriculum vitae (**Ex. 1006**). The following is a brief summary of my relevant qualifications and professional experience.

5. I was a professor in the Department of Library and Information Science at the Catholic University of America for 32 years and retired at the end of August

Amazon
Exhibit 1005
Page 004

2022. I have experience working in an academic library, a medical library, and a legislative library. I hold a Ph.D. in Library and Information Studies from the University of Wisconsin-Madison and a Masters in Library and Information Studies from the University of Wisconsin-Madison.

6. I am an expert on library cataloging and classification and have published two editions on this subject, *Organizing Audiovisual and Electronic Resources for Access*: *A Cataloging Guide* (2000 and 2006). I taught a variety of courses, including Cataloging and Classification, Advanced Cataloging and Classification, Organization of Internet Resources, Organization of Information, Digital Content Creation and Management, Metadata, Internet Searches and Web Design, Information Literacy Instruction, Advanced Information Retrieval and Analysis Strategies, and The Information Professions in Society. I am familiar with metadata schema design and implementation. In my teaching I covered the design and implementation of metadata in libraries, databases, search engines, digital repositories, digital libraries, and digital archives. I also covered how the ways we organize information resources affect the discovery and access to digital resources on the Internet. My research interests cover cataloging and classification, information organization, metadata, information retrieval, information architecture, digital collections, scholarly communication, social media, user interaction with information systems, and others.

Amazon
Exhibit 1005
Page 005

7. I am very familiar with a library cataloging encoding standard known as the "Machine-Readable Cataloging" standard, also known as "MARC," which became the national standard for sharing bibliographic data in the United States by 1971 and the international standard by 1973. MARC is the primary communications protocol for the transfer and storage of bibliographic metadata in libraries. Experts in my field reasonably rely upon MARC records when forming their opinions.

8. A MARC record consists of several fields, each of which contains specific data about the work. Each field is identified by a standardized, unique, three-digit code corresponding to the type of data that follows. **Appendix A** is a true and correct copy of Parts VII to X of "Understanding MARC Bibliographic: Machine-Readable Cataloging" (http://www.loc.gov/marc/umb/um07to10.html) from the Library of Congress. *Understanding MARC Bibliographic* was originally published by the Follett Software Co. and copyrighted in 1988. Editions in 1989, 1990, 1994, 1998, 2000, 2003 and 2009 include changes. The latest edition was copyrighted in 2009. This document explains commonly-used MARC fields. For example, the personal author of the work is recorded in Field 100, the title is recorded in Field 245, publisher information is recorded in Field 260, the physical volume and characteristics of a publication are recorded in Field 300, and topical subjects are recorded in the 650 fields.

Amazon
Exhibit 1005
Page 006

9. The Online Computer Library Center (OCLC) is the largest bibliographic network of the world, with more than 517 million records (as of July 2021) and thousands of member institutions (many of which are libraries of some type) in more than 100 countries. OCLC was founded in 1967 to promote and support library cooperation. According to the "Third Article, Amended Articles of Incorporation of OCLC Online Computer Library Center, Inc.," OCLC was created "to establish, maintain and operate a computerized library network and to promote the evolution of library use, of libraries themselves, and of librarianship, and to provide processes and products for the benefit of library users and libraries, including such objectives as increasing availability of library resources to individual library patrons and reducing the rate of rise of library per-unit costs, all for the fundamental public purpose of furthering ease of access to and use of the ever-expanding body of worldwide scientific, literary and educational knowledge and information" (source: https://www.oclc.org/content/dam/oclc/membership/articles-of-incorporation.pdf).

10. OCLC members can contribute original cataloging records in MARC to the system or derive cataloging records from existing records, an activity referred to as "copy cataloging." When an OCLC participating institution acquires a work, it can create an original MARC record for the work in OCLC's Connexion system (a system for catalogers to create and share MARC records), and the system will

Amazon
Exhibit 1005
Page 007

automatically generate a code for the date of record creation in the yymmdd format, and the creating library's OCLC symbol is recorded in subfield "a" of the 040 field. Once the MARC record is in Connexion, it becomes available to other OCLC members for adoption to their local online catalogs (*i.e.*, copy cataloging).

11. After a MARC record is created in Connexion, it also becomes searchable and viewable on WorldCat, which is a free web portal for users to explore the collections of more than 10,000 libraries worldwide. The record in WorldCat, however, is not presented in MARC fields. Instead, the data elements are labeled to help users interpret the record. Thus, the information stored in MARC records in Connexion is available to the interested public through the free and user-friendly WorldCat web portal.

12. WorldCat (http://www.worldcat.org) is "the world's largest network of library content and services" and its features are summarized in "What is WorldCat" (http://www.worldcat.org/whatis/default.jsp). WorldCat began operation in January 1998 as a free web portal to more than 10,000 library collections worldwide. Through WorldCat, users can search for information resources in their local libraries and libraries around the world. WorldCat allows users to search for books, CDs, videos, and many new types of digital content, such as audiobooks, in many languages. Users can also retrieve research materials and article citations with links to their full text. After an item is retrieved, WorldCat helps users identify a library

Amazon
Exhibit 1005
Page 008

nearby that holds the item or all the libraries that hold the item. WorldCat is an efficient way to explore library collections on the Internet. WorldCat is publicly accessible, and there is no fee to perform searches on WorldCat.

13. Library online catalogs gained acceptance in the early 1980s and many libraries migrated their systems to the World Wide Web in the mid-1990s. Library online catalogs are based on MARC records that represent their collections in order to help the public understand what materials are publicly accessible in those libraries. Most libraries with online catalogs have made their catalogs freely available on the Web. These online catalogs offer user-friendly search interfaces. Strong user interest in keyword searches and the popularity of Google have led to the "googlization" of library search systems. As a result, many library catalogs now provide a single search box for users to conduct keyword searches, with additional support for searches by author, title, subject terms, and other data elements such as ISBN (International Standard Book Number). Library catalogs these days also offer features for users to narrow their search results by language, year, format, and other elements. Many libraries display MARC records on their online catalogs with labels for the data elements to help the public interpret MARC records. Many libraries also offer the option to display MARC records in MARC fields.

14. Libraries create MARC records for works they acquire to facilitate the discovery and access to works in their collections, which may include books, serials,

Amazon
Exhibit 1005
Page 009

motion pictures, and publications in other formats. Monograph cataloging is fairly common in libraries, and most libraries make a newly cataloged monograph available to the public soon after the cataloging work is completed, usually within a week. Libraries can create original cataloging records or use an existing record in OCLC to create a copy cataloging record. As soon as the cataloging record is completed, it is added to the library's online catalog for users. If the record is an original record, it is also entered into the OCLC system. If it is a copy cataloging record, the library's holding symbol is attached to the existing original record in the OCLC system to facilitate searching and interlibrary loan.

15. Conference proceedings can be cataloged as monographs or serials, depending on a library's policy. Libraries usually choose to catalog conference proceedings as monographs because records for monographs are easier for users to discover and understand, and access to individual conference proceedings becomes easier. Once a record for a conference proceedings is found, users can see the title, conference name, subjects, and related contributors clearly identified in the record and the call number will enable users to access the desired title quickly. Conference proceedings cataloged as monographs are treated as regular monographs, and the newly cataloged monographs typically become publicly accessible soon after the cataloging work is completed, usually within a week. When conferences have the same conference name over time and are held on a regular basis, libraries may

Amazon
Exhibit 1005
Page 010

choose to catalog the proceedings of such conferences as serials. This means that only one serial record is created for all the proceedings of a named conference, and no individual records are created for newly received conference proceedings. This approach saves cataloging time, and newly received conference proceedings are made available for user access soon after serial check-in, usually within one week.

16. The cataloging of serials and the serial check-in process are discussed here to show how libraries usually provide access to newly received serial issues. According to the glossary of the *RDA: Resource Description and Access* cataloging standard, a serial is "a mode of issuance of a manifestation issued in successive parts, usually bearing numbering, that has no predetermined conclusion. A serial includes a periodical, monographic series, newspaper, etc." Because the publisher of a serial makes new issues of the serial available successively, a customary cataloging practice is to create one bibliographic record for the serial, and the serial record (encoded in MARC) typically provides information on the beginning date and frequency of the serial, not the dates of individual issues. In other words, libraries typically do not create MARC records for individual issues of a serial. Instead, they rely on a serial check-in system to track the receipt of new issues. A common check-in practice is to date stamp a new issue when it arrives. This practice has become automated since the late 1990s, and libraries now vary in how they share the receipt date of a new serial issue with the public. Some libraries use a date stamp, some affix

Amazon
Exhibit 1005
Page 011

a label to indicate the receipt date, some pencil in the receipt date, and some do not provide the information to the public.

17. The serial check-in process usually takes less than an hour, and one of the steps involves placing a date stamp on the new issue to document the date the issue is checked in. After that, the holdings information of the serial is updated in the library's catalog so that users know which issues are available for request or access. After serial check-in is completed, the new issue is placed on the shelf with the previous issues of the serial. Libraries with a public periodical room typically place new issues in the periodical room for easy user access. Because information presented in serials often reflects latest discovery, a general practice of libraries is to make new issues of serials available for user access soon after they are checked in, usually within a week.

18. I am personally familiar with many online catalogs, databases, and search engines. In preparing for this declaration, I used the following authoritative information systems to search for records:

- Google Scholar (https://scholar.google.com)
- Internet Archive (https://archive.org)
- Online catalog of the Library of Congress (https://catalog.loc.gov)
- Online catalog of the Linda Hall Library (https://catalog.lindahall.org/)

Amazon
Exhibit 1005
Page 012

- Online catalog of the Stanford University Libraries (https://searchworks.stanford.edu)
- Semantic Scholar (https://www.semanticscholar.org)
- WorldCat (free for public access on the Web) (https://www.worldcat.org)

B. Scope of This Declaration

19. I have been asked to offer an opinion on the authenticity and public availability dates of the following documents:

> Digital Equipment Corporation, Intel Corporation, Xerox Corporation, (1980, September), The Ethernet : A Local Area Network : Data Link Layer and Physical Layer Specifications, Version 1.0, September 30, 1980, available from https://ethernethistory.typepad.com/papers/EthernetSpec.pdf, accessed and obtained on November 17, 2021, **Ex. 1009** ("*DIX-Ethernet*");
>
> Andrews, G. R., (2000), *Foundations of Multithreaded, Parallel, and Distributed Programming*, Addison-Wesley, obtained from counsel, **Ex. 1011** ("*Andrews*");
>
> Batheja, J., & Parashar, M., (2001, October), Adaptive cluster computing using Javaspaces, in 2001 IEEE International Conference on Cluster Computing, pp. 323-330, obtained from IEEE, **Ex. 1012** ("*Batheja*").

Amazon
Exhibit 1005
Page 013

20. I am rendering my expert opinion on the authenticity of the documents referenced herein and on whether they were published. I will provide my opinion on when and how these documents were publicly disseminated or otherwise made publicly available to the extent that persons interested and ordinarily skilled in the subject matter or art, exercising reasonable diligence, could have located and retrieved these documents

21. I am informed by counsel that an item is considered authentic if there is sufficient evidence to support a finding that the item is what it is claimed to be. I am also informed that authenticity can be established based on the contents of the documents themselves, such as the appearance, content, substance, internal patterns, protocols followed or other distinctive characteristics of the item.

22. I am informed by counsel that a given reference is "published" if it is publicly accessible upon a satisfactory showing that such a document has been disseminated or otherwise made publicly available to the extent that persons interested and ordinarily skilled in the subject matter or art could locate it exercising reasonable diligence and obtain the document. I have also been informed by counsel that materials available in a library constitute "printed publications" if they are cataloged and indexed according to general library practices and protocols that make the references available and accessible to members of the interested public.

Amazon
Exhibit 1005
Page 014

23. I have been informed by counsel that the ’777 patent was filed in the United States on January 25, 2013, and that this is the earliest asserted priority date. I have assessed public accessibility of the documents referenced below as of this claimed priority date.

C. Person of Ordinary Skills in the Arts

24. I am told by counsel that the subject matter of this proceeding relates to parallel or distributed processing computing systems.

25. I have been informed by counsel that a “person of ordinary skill in the art at the time of the inventions” (POSA) is a hypothetical person who is presumed to be familiar with the relevant field and its literature at the time of the inventions. This hypothetical person is also a person of ordinary creativity, capable of understanding the scientific principles and literature applicable to the pertinent field.

26. I am told by counsel that persons of ordinary skill in this subject matter or art in the 2013 timeframe would have had at least a Bachelor’s Degree in computer science, computer engineering, or a related field, and about five or more years of industry or academic research experience working in the field of parallel or distributed computing. A higher level of education, such as a post-graduate degree in computer science, computer engineering, or a related field, could substitute for less work experience.

Amazon
Exhibit 1005
Page 015

27. It is my opinion that such a person would have been actively engaged in learning about the field, possibly through formal instruction through the bibliographic resources. By at least January 2013 such a person would have had access to a vast array of print and electronic resources, including at least the documents referenced below.

D. Evidence Considered in Forming My Opinions

28. In the preparation of this declaration, I have reviewed the documents referenced below and any other documents I reference herein, and each of these is a type of material that experts in my field would reasonably rely upon when forming their opinions:

(1) The document referenced above in Section I.B;

(2) Internet Archive archived copy of *DIX-Ethernet*, archived on July 25, 2011, and available at https://web.archive.org/web/20110725122201/https://ethernethistory.typepad.com/papers/EthernetSpec.pdf, accessed and obtained on November 18, 2021, **Appendix 1009A**;

(3) Internet Archive archived copy of "Ethernet History Timeline" of 1980, archived on Oct. 13, 2007, and available at https://web.archive.org/web/20071013131903/http://timeline.ethernethistory.com/, accessed and obtained on November 18, 2021, **Appendix 1009B**;

(4) Digital Equipment Corporation, Intel Corporation, Xerox Corporation, The Ethernet : A Local Area Network : Data Link

Amazon
Exhibit 1005
Page 016

Layer and Physical Layer Specifications, Version 1.0, September 30, 1980, in *Computer Communication Review*, vol. 11, no. 3 (July 1981), pp. 20–66, obtained from ACM Digital Library on November 19, 2021, **Appendix 1009C** ("*ACM-Ethernet*");

(5) *ACM-Ethernet* obtained from the University of Wisconsin-Madison Libraries, accessed and obtained on November 19, 2021, **Appendix 1009D**;

(6) Early citations to *DIX-Ethernet* and *ACM-Ethernet*, obtained from Google Scholar and Semantic Scholar, **Appendix 1009E**;

(7) Bibliographic and MARC records for *Andrews*, available from the online catalog of the Library of Congress at https://lccn.loc.gov/99045285 and https://catalog.loc.gov/vwebv/staffView?searchId=15695&recPointer=0&recCount=25&searchType=1&bibId=11770186, accessed and obtained November 17, 2021, **Appendix 1011A**;

(8) Copyright record for *Andrews*, available at the public catalog of the U. S. Copyright Registration Office at https://cocatalog.loc.gov/cgi-bin/Pwebrecon.cgi?Search_Arg=Foundations+of+multithreaded%2C+parallel%2C+and+distributed+programming&Search_Code=TALL&PID=C1x3fowBskBOyAz8j7OtmwWC50g2R&SEQ=20211117123052&CNT=25&HIST=1, accessed and obtained on November 17, 2021, **Appendix 1011B**;

(9) Author homepage for *Andrews*, archived on September 19, 2000, by the Internet Archive, available athttps://web.archive.org/web/20000919040130/http://www.cs.arizona.edu/people/greg/mpd

Amazon
Exhibit 1005
Page 017

book/, accessed and obtained on November 27, 2021, **Appendix 1011C**;

(10) Publisher's list of books including *Andrews*, archived on September 19, 2000, by the Internet Archive, available at https://web.archive.org/web/20000916001511/http://www.awlonline.com/aw/productList/0,1991,CS1105~CS,00.html; and publisher's webpage for *Andrews*, archived on January 19, 2001, by the Internet Archive, available at https://web.archive.org/web/20010119113900/http://www.awlonline.com/product/0,2627,0201357526,00.html, accessed and obtained on November 27, 2021, **Appendix 1011D**;

(11) Bibliographic and MARC records for *Andrews*, available from the online catalog of the Stanford University Libraries at https://searchworks.stanford.edu/view/4287207, accessed and obtained on November 17, 2021, **Appendix 1011E**;

(12) Early citations to *Andrews*, obtained from Google Scholar, **Appendix 1011F**;

(13) Copy of *Andrews*, scanned from a copy I obtained from Amazon, **Appendix 1011G**;

(14) Database record for *Batheja*, available from IEEE Xplore Digital Library at https://ieeexplore.ieee.org/document/959993, accessed and obtained on November 18, 2021, **Appendix 1012A**;

(15) Library copy of *Batheja*, obtained from the Linda Hall Library on November 19, 2021, **Appendix 1012B**;

Amazon
Exhibit 1005
Page 018

(16) Bibliographic and MARC records for *2001 IEEE International Conference on Cluster Computing* that contains *Batheja*, available from the online catalog of the Linda Hall Library at https://catalog.lindahall.org/permalink/01LINDAHALL_INST/19lda7s/alma993633983405961, accessed and obtained on November 18, 2021, **Appendix 1012C**;

(17) Citations to *Batheja*, obtained from Google Scholar, **Appendix 1012D**.

II. Authentication and Public Availability of *DIX-Ethernet* (Ex. 1009)

A. Authentication

29. **Ex. 1009** is a true and correct copy of The Ethernet : A Local Area Network : Data link Layer and Physical Layer Specification, Version 1.0, September 30, 1980, ("*DIX-Ethernet*"), by Digital Equipment Corporation, Intel Corporation, Xerox Corporation, that I obtained from https://ethernethistory.typepad.com/papers/EthernetSpec.pdf. When I was asked to prepare this declaration and provide an opinion about *DIX-Ethernet*, I searched Google by the title of this document and followed a link for "papers" in the Ethernet History Archives (https://ethernethistory.typepad.com/my_weblog/archives.html) to locate, identify and obtain this copy. It is presented as **Ex. 1009** in this declaration.

30. **Ex. 1009** is a true and correct copy of the PDF version of *DIX-Ethernet*. Page 1 is the title page that shows "The Ethernet : A Local Area Network, Data Link

Amazon
Exhibit 1005
Page 019

Layer and Physical Layer Specifications" as the title, and Digital Equipment Corporation, Intel Corporation, and Xerox Corporation as the authors. It also shows this is "Version 1.0, September 30, 1980." Page 2 presents "Important Information and Disclaimers" and pages 3 and 4 (internal pages i and ii) are the Preface that states "This document contains the specification of the Ethernet, a local area network developed jointly by Digital Equipment Corporation, Intel Corporation, and Xerox Corporation. The Ethernet specification arises from an extensive collaborative effort of the three corporations, and several years of work at Xerox on an earlier prototype Ethernet." Pages 5 to 8 (internal pages iii to vi) are the table of contents that shows this document has seven sections. Page 9 (internal page vii) shows the document includes five appendices, 22 figures and one table. The front matter ends on page 10 (internal page viii), the references are on page 13 (internal page 3), and the main text runs from internal page 1 to page 82.

31. This copy of *DIX-Ethernet* (**Ex. 1009**) is in a condition that creates no suspicion about its authenticity. Specifically, this copy is not missing any intermediate pages, the text on each page appears to flow seamlessly from one page to the next, and there are no visible alterations to the document. Moreover, **Ex. 1009** was found in the Ethernet History Archives (https://ethernethistory.typepad.com/my_weblog/archives.html), a place where, if authentic, it would likely be found.

Amazon
Exhibit 1005
Page 020

Accordingly, I see no reason to question the authenticity of this copy of *DIX-Ethernet* (**Ex. 1009**).

B. Internet Archive Copy and Records

32. To determine the public availability date of *DIX-Ethernet* I searched the Internet Archive for records. The Internet Archive is a digital library of archived websites, webpages, and other information resources, and it provides free access to researchers and the general public. **Appendix 1009A** is a true and correct copy of *Dix-Ethernet* that I personally located, identified and obtained from the Internet Archive at https://web.archive.org/web/20110725122201/https://ethernethistory.typepad.com/papers/EthernetSpec.pdf. This is the type of materials experts in my field would reasonably rely upon when forming their opinions.

33. Page 1 of **Appendix 1009A** is a screen capture by me of the first page of *DIX-Ethernet* with a Wayback Machine calendar to show this document (https://ethernethistory.typepad.com/papers/EthernetSpec.pdf) was archived 50 times from "25 Jul 2011 – 3 Nov 2021," and this copy was archived on July 25, 2011. Pages 2 to 93 show the archived copy of *DIX-Ethernet*. I have compared this Internet Archive copy (**Appendix 1009A**) with the copy in the Ethernet History Archives (**Ex. 1009**) and found them to have the same content.

34. I have used Internet Archive in the past and am familiar with their practice of assigning an extended URL to an archived file in the format of

Amazon
Exhibit 1005
Page 021

http://www.archive.org/web/[Year in yyyy][Month in mm][Day in dd][Time code in hh:mm:ss]/[Archived URL] to show the point of time the archived file was captured. The URL of **Appendix 1009A** is https://web.archive.org/web/20110725122201/https://ethernethistory.typepad.com/papers/EthernetSpec.pdf. This URL means that *DIX-Ethernet* was archived on "20110725122201" (i.e., July 25, 2011, 12 p.m., 22 minutes, 1 second). Because Internet Archive does not capture web resources as soon as they appear on the Internet, the archive date does not represent the first time a web resource appears on the Internet. Before a resource can be archived, it must exist on the Internet first, so the archive date of Internet Archive should be interpreted as the "no later than" date, meaning the archived resource was publicly accessible on the Internet no later than that date. Based on the Internet Archive copy of *DIX-Ethernet* and the archive history of this document, it is my opinion that *DIX-Ethernet* was publicly available on the Internet no later than July 25, 2011.

35. My further research discovered that *DIX-Ethernet* was actually publicly available earlier than 2011. In researching archived data on Ethernet history I discovered the 1980 timeline at https://web.archive.org/web/20071013131903/http://timeline.ethernethistory.com that shows *DIX-Ethernet* was linked to as early as 1980. **Appendix 1009B** is the Ethernet timeline of 1980 that I personally located,

Amazon
Exhibit 1005
Page 022

identified and captured. This is the type of materials experts in my field would reasonably rely upon when forming their opinions.

36. **Appendix 1009B** is a true and correct copy of an Ethernet timeline of 1980 archived by the Internet Archive. This is a screen capture by me of the timeline with a Wayback Machine calendar to show that this timeline was captured 81 times from "13 Oct 2007 – 2 Apr 2018" and this copy was archived on October 13, 2007. The timeline lists four major events in 1980, and the second item states "DIX group published 'Blue Book' Ethernet specification" and the hyperlink for "Ethernet specification" leads to the copy of *DIX-Ethernet* archived on July 25, 2011. Based on the Internet Archive convention for the extended URL, the archive date of this 1980 Ethernet time is "20071013131903" (i.e., October 13, 2007, 1 p.m., 19 minutes, 3 seconds). This archive date and the Ethernet specification inform my opinion that the publication of *DIX-Ethernet* in 1980 was marked as an important event in the history of Ethernet, and the link to *DIX-Ethernet* in this 1980 timeline means that users in 1980 would have access to *DIX-Ethernet*, or they would have access to this document at least by October 13, 2007 when the link to *DIX-Ethernet* in the 1980 Ethernet timeline was archived.

37. Based on the archived copy of *DIX-Ethernet* (**Appendix 1009A**) and the 1980 Ethernet timeline (**Appendix 1009B**), it is my opinion that *DIX-Ethernet*

Amazon
Exhibit 1005
Page 023

would have been publicly accessible as early as 1980 and no later than October 13, 2007.

C. ACM Copy of Ethernet Specification

38. In my research of the Ethernet Specification in Semantic Scholar, I learned that this specification was also published in volume 11, no. 3 (July 1981) of ACM's *Computer Communication Review*, so I searched the ACM Digital Library for this publication. After I located the database record for this publication, I purchased it from ACM. The copy of *ACM-Ethernet* is presented as **Appendix 1009C** in this declaration.

39. **Appendix 1009C** is a true and correct copy of Ethernet specification obtained from the ACM Digital Library. It shows that the Ethernet specification appears from page 20 to 66. I have closely compared *ACM-Ethernet* (**Appendix 1009C**) with *DIX-Ethernet* (**Ex. 1009**) and found them to have the same content. The difference is that *ACM-Ethernet* shows two pages per page and the page orientation is different from that of *DIX-Ethernet*, but the content is the same in these two copies.

D. University of Wisconsin Copy

40. To determine the public availability of *ACM-Ethernet* in libraries I searched WorldCat and identified the University of Wisconsin-Madison as a holding library of ACM's *Computer Communication Review*, so I requested a scanned copy

Amazon
Exhibit 1005
Page 024

from Wisconsin TechSearch (WTS), a document delivery service based at the University of Wisconsin-Madison. The Wisconsin copy of *ACM-Ethernet* is presented in this declaration as **Appendix 1009D**.

41. **Appendix 1009D** is a true and correct copy of *ACM-Ethernet*. Page 1 is the cover of volume 11, number 3 (July 1981) of *Computer Communication Review* that shows the journal is "a quarterly publication of the Special Interest Group on Data Communication" and the table of contents on the cover shows that "The Ethernet: Data Link Layer and Physical Layer Specifications" appears from pages 20 to 66 of this issue.

42. Page 2 of **Appendix 1009D** is the publisher information page and the bottom of this page displays a note that "this issue rec'd at ACM HQ for production on 8/13/1981." Pages 3 to 49 (internal pages 20 to 66) are the *ACM-Ethernet*. I have closely compared the Wisconsin copy (**Appendix 1009D**) with the copy purchased from ACM (**Appendix 1009C**) and found them to be identical.

43. The cover of **Appendix 1009D** carries a library date stamp of "Kurt F. Wendt Library College of Engineering OCT 4 1981 UW-Madison, WI 53706" that indicates this issue was received by the Engineering Library on October 4, 1981. As discussed earlier, a customary library processing practice of journals is to create a record for the entire journal and create no records for individual issues of a journal. When a new issue is received, it goes through the serial check-in process, then is

Amazon
Exhibit 1005
Page 025

processed and placed in the public periodical reading room, if a library has one, or on the shelf with earlier issues of the journal for public access. The serial check-in process usually takes less than an hour, and the newly received journal issue is processed quickly to make it available for user access, usually within a week after serial check-in. It is therefore my opinion that volume 11, number 3 of ACM's *Computer Communication Review* that contains *ACM-Ethernet* would have become publicly accessible no later than October 11, 1981.

E. Usage Record

44. Actual usage of a publication is reflected by the papers that make reference to it. I used Google Scholar and Semantic Scholar to identify citations to *DIX-Ethernet* and *ACM-Ethernet*. **Appendix 1009E** is a list I compiled that presents six citations from 1982 to 1989 to show early usage. Two earliest citations appeared in 1982, further supporting my opinion that the print copy of *ACM-Ethernet* was publicly accessible in October 1981.

F. Summary of My Opinion on *DIX-Ethernet* & *ACM-Ethernet*

45. The archived copy of *DIX-Ethernet* from the Internet Archive (**Appendix 1009A**) shows that this document was publicly accessible on the Internet no later than July 25, 2011. A record of the Ethernet timeline of 1980 (**Appendix 1009B**), archived on October 13, 2007, shows that *DIX-Ethernet* was referenced and linked to from the 1980 timeline of Ethernet History. It is therefore my opinion that

Amazon
Exhibit 1005
Page 026

the electronic copy of *DIX-Ethernet* would have been publicly available as early as 1980 and no later than October 13, 2007. *DIX-Ethernet* was also published in an ACM journal *Computer Communication Review*, and a print copy obtained from the University of Wisconsin-Madison shows that the ACM issue containing the Ethernet specification was received on October 4, 1981. It is therefore my opinion that the print copy of *ACM-Ethernet* would have been publicly accessible no later than October 11, 1981. Citation history shows usage began in 1982, further supporting my opinion of the public availability of the print copy in October 1981.

III. AUTHENTICITY AND PUBLIC AVAILABILITY OF *Andrews* (Ex. 1011)

A. Authentication

46. **Ex. 1011** is a true and correct copy of portions of *Foundations of Multithreaded, Parallel, and Distributed Programming*, ("*Andrews*"), by G. R. Andrews, published by Addison-Wesley in 2000, that I obtained from counsel. When I was asked to prepare this declaration, I searched WorldCat by the title of this book for records, and found a record (https://worldcat.org/title/42296220) that informed me the Library of Congress and Stanford University Libraries are two of the holding libraries. Because of international travel I was not able to visit the Library of Congress during this time. I therefore purchased a copy of *Andrews* from Amazon and used this copy, library records, copyright record, and webpages

Amazon
Exhibit 1005
Page 027

archived by the Internet Archive to authenticate a scanned copy provided by counsel and to determine the public availability date of this publication. The copy of *Andrews* obtained from counsel is presented as **Ex. 1011** in this declaration. If the patent owner raises questions about a library copy of *Andrews*, I reserve the right to submit a library copy when I am able to visit the Library of Congress after the winter holidays.

47. Page 1 of **Ex. 1011** is the cover that shows "Foundations of Multithreaded, Parallel, and Distributed Programming" as the title and "Gregory R. Andrews" as the author. Page 2 is the half title page, and page 3 is the title page that shows the same title and author information as the cover, and also shows Addison-Wesley of Reading, Massachusetts as the publisher. Page 4 is the copyright page that shows a "2000" copyright date, with Addison Wesley Longman, Inc. as the copyright holder. This page also shows a "Library of Congress Cataloging-in-Publication Data" that shows the ISBN of this work is "0-201-35752-6" and the Library of Congress control number assigned to this work is "99-045285." Pages 5 to 12 (internal pages v to xii) are the Preface, and page 11 (internal page xi) shows two online resources for this publication. The author states that "The 'home page' for the book is located at http://www.cs.arizona.edu/people/greg/mpdbook. The Web site contains the source for the larger programs in the text, links to software and information on the languages and libraries described in the case studies, and a

Amazon
Exhibit 1005
Page 028

variety of other useful links. It also contains extensive information from my Arizona class, including the syllabus, lecture notes, and copies of homework assignments, projects, and exams. Information on this book is also available at http://www.awlonline.com/cs." Pages 13 to 20 (internal pages xiii to xx) are the table of contents that shows *Andrews* has three parts, including 12 chapters, and each of them includes historical notes, references and exercises. The book also includes a glossary and an index. Pages 21 to 47 include internal pages 1-3, 31-32, 132-139, 337-340, 424-430, and 561-563. Page 48 shows the last page of *Andrews* is an index page numbered page 664. Page 49 is the back cover.

B. Amazon Copy of Andrews

48. To authenticate **Ex. 1011,** I obtained a used copy of *Andrews* from Amazon and scanned the pages corresponding to those included **Ex. 1011** from my Amazon copy. The pages from the Amazon copy are presented as **Appendix 1011G** in this declaration. I have closely compared **Ex. 1011** with **Appendix 1011G** and concluded that the two copies have the same content.

49. **Ex. 1011** is in a condition that creates no suspicion about its authenticity. Specifically, the pages in **Ex. 1011** are not missing any intermediate pages (within their sequences), the text on each page appears to flow seamlessly from one page to the next, and there are no visible alterations to the document. Furthermore, **Ex. 1011** has the same content as the Amazon copy I scanned

Amazon
Exhibit 1005
Page 029

(**Appendix 1011G**), confirming that the pages in **Ex. 1011** are from a published and publicly accessible work. I therefore see no reason to question the authenticity of **Ex. 1011**.

C. Library of Congress Records

50. As evidence of the public accessibility of *Andrews* in libraries, I obtained records for this book from the Library of Congress online catalog. I searched the Library of Congress because a WorldCat record I located (https://worldcat.org/title/42296220) informed me that the Library of Congress was a holding library. **Appendix 1011A** contains true and correct copies of the bibliographic and MARC records for *Foundations of Multithreaded, Parallel, and Distributed Programming* ("*Andrews*"). I personally identified, located, and obtained these records from the online catalog of the Library of Congress. They are the type of records experts in my field would reasonably rely upon when forming their opinion.

51. Pages 1 and 2 of **Appendix 1011A** are the bibliographic record that shows the author, title, publisher, and physical description of *Andrews*. It also shows the ISBN of Andrews is "0201357526" and the LCCN (Library of Congress control number) is "99045285." The "Item Available" area shows the book's call number is "QA76.58 .A57 2000 LANDOVR" and it can be requested at the "Jefferson or Adams Building Reading Rooms." The book is "stored offsite" and its status is "not

Amazon
Exhibit 1005
Page 030

charged" (meaning available for user request).

52. Page 3 is the MARC record that shows the date of record creation and revision. The first six digits of Field 008 show the MARC record was created on "990810" (*i.e.*, August 10, 1999), and the "s2000" code following this creation date indicates this publication has a single publication date of 2000. Field 955 is a field for recording local cataloging activities, and it shows the MARC record was created on "08-10-99" and "bk rec'd, to CIP ver. 04-06-00; CIP ver. 05-03-00; to BCCD 06-20-00." These notes indicate that after the MARC record was created, a copy was received for CIP verification on April 6, 2000, and a cataloger verified the record on May 3, 2000, then *Andrews* was sent to the Binding and Collections Care Division for processing on Jun 20, 2000. Field 040 shows that "DLC" created the original record. According to the *Directory of OCLC Members* (https://www.oclc.org/en/contacts/libraries.html), "DLC" is the OCLC library symbol for the Library of Congress. Fields 008, 040 and 955 inform my opinion that the original MARC record was a Cataloging-in-Publication record based on the information supplied by the publisher. After the CIP record was completed, it was sent to the publisher for inclusion in the physical volume. The Cataloging-in-Publication Data printed on the copyright page of **Ex. 1011** reflects this practice. After the published work was received by the Library of Congress, catalogers of the Cataloging-in-Publication Program used it to verify the CIP MARC record on May

Amazon
Exhibit 1005
Page 031

3, 2000, and then sent the book to processing on June 20, 2000. A customary practice of libraries is to process a newly cataloged monograph quickly to make it publicly accessible, usually within a week after the cataloging work is completed. Considering the large amount of works processed by the Library of Congress and my personal experience with their collections, it is my opinion that *Andrews* would have been publicly accessible one to three months after it was sent to the processing unit on June 20, 2000. This means that *Andrews* would have been publicly accessible at the Library of Congress as early as July 20, 2000, and no later than September 20, 2000.

53. The Cataloging in Publication Program (CIP) of the Library of Congress is responsible for cataloging books in advance of publication to alert the library community to forthcoming publications and facilitate acquisition. The initial CIP record is created based on information submitted by the publisher. When the CIP record is ready, it is sent to the publisher for inclusion in the publication. The Library of Congress distributes CIP records in MARC format to large libraries, bibliographic services, and book vendors worldwide. When a CIP record is ready, it is sent to the publisher, who then includes the CIP record in the publication.

54. CIP records are also entered into OCLC Connexion system, the cataloging system containing more than 517 million MARC records, for OCLC member libraries to use it for copy cataloging and acquisition purposes. After the

Amazon
Exhibit 1005
Page 032

book is published, the publisher is obligated to send a copy to the CIP Program, and Library of Congress catalogers use the newly received publication to verify and update data in the original CIP record.

55. Field 010 of the MARC record (**Appendix 1011A**) shows the Library of Congress control number assigned to *Andrews* and Field 020 shows the book's ISBN. This MARC record includes the author as the main access point in Field 100, and Field 245 shows the title and author statement. Field 246 shows a variant title as an additional access point to help users discover this work. Field 260 shows Addison-Wesley of Reading, Mass. published this work with a 2000 copyright date. Field 300 shows the front matter of *Andrews* ends on page xx and the last numbered page is page 664. Field 504 shows *Andrews* includes several bibliographical references and an index. The author, title, publisher, publication place, publication date, pagination, and the note on references and index match the information presented in **Ex. 1011** and confirm this MARC record represents *Andrews*.

56. Subjects of this book are represented in Field 050 by a Library of Congress Classification (LCC) number of "QA76.58" that represents "Parallel processing. Parallel computers" (source: Library of Congress Classification - Schedule Q at https://www.loc.gov/aba/publications/FreeLCC/LCC_Q2020TEXT.pdf), and in Field 082 by a Dewey Decimal Classification (DDC) number of "004.3" that represents "Processing modes" in computer science (source: WebDewey full

Amazon
Exhibit 1005
Page 033

manual at https://www.oclc.org/content/dam/oclc/webdewey/help/full_manual .pdf). Subjects are also represented in Fields 650 by three Library of Congress subject headings, "Parallel programming (Computer science)," "Electronic data processing |x Distributed processing," ("|x" indicates a topical subdivision), and "Threads (Computer programs)."

57. This MARC record (**Appendix 1011A**) shows that *Andrews* was published in 2000 and became searchable by at least May 2000 at the online catalog of the Library of Congress. Users would have been able to conduct subject searches by the LCC number, DDC number and the Library of Congress subject headings in the MARC record to discover this book. They would also have been able to search for *Andrews* by the author, title, variant title, ISBN, and LCCN.

D. Copyright Record

58. **Appendix 1011B** is a true and correct copy of the copyright registration record for *Andrews*. I obtained this record from the public catalog of the U.S. Copyright Office by first accessing their website at http://www.copyright.gov/, selecting the "Learn More" link under "Search Copyright Records," and following the link for "Official Public Catalog (1978 to present)" under "Online Records Collections" to perform a title search to retrieve the record. I personally located, identified and obtained this record. This is the type of record experts in my field would reasonably rely upon when forming their opinions.

Amazon
Exhibit 1005
Page 034

59. The "Type of Work" field identifies the publication as "Text." The title, imprint, description, and copyright claimant information match the information presented in **Ex. 1011**. The "Date of Creation" field shows this work was created in 1998, and the "Date of Publication" field shows "2000-02-09" (i.e., February 9, 2000). The "Registration Number / Date" field shows "TX0005144213 / 2000-02-22" indicating *Andrews* was registered with the Copyright Office on February 22, 2000. This record informs my opinion that *Andrews* was published on February 9, 2000, and was received by the U.S. Copyright Office no later than February 22, 2000 when the copyright registration number was assigned to *Andrews*.

E. Internet Archive Records

60. To determine when *Andrews* became publicly accessible, I searched the Internet Archive for records and found the homepage for the book (http://www.cs.arizona.edu/people/greg/mpdbook), described by the author in the Preface of his book, was first archived on September 19, 2000. **Appendix 1011C** is a true and correct copy of the homepage for *Andrews*. I personally located, identified, and obtained this archived copy. This is the type of materials experts in my field would reasonably rely upon when forming their opinions.

61. Page 1 of **Appendix 1011C** is a screen capture by me to show the homepage with a Wayback Machine calendar that shows this webpage was captured 223 times from "26 May 2000" to "24 Apr 2021" and this copy was captured on

Amazon
Exhibit 1005
Page 035

September 19, 2000. Pages 2 and 3 are the PDF copy of this home page that shows the title and author of *Andrews*, a quotation from the Preface, links to course outlines, syllabus, lecture notes, homework assignments, and tools, and ordering information that indicates the book's ISBN is "0-201-35752-6." The bottom of page 3 shows the homepage was "last updated August 1, 2000." The last update date of this home page informs my opinion that this homepage was publicly accessible on the Internet at least by August 1, 2000, when the author updated it. This means that by at least August 1, 2000, interested users would have been able to discover *Andrews* on the Internet and use the ISBN to search for it in libraries or order it from the publisher. Because this homepage was archived by the Internet Archive on September 19, 2000, it is my opinion that users would have discovered this book on the Internet no later than September 19, 2000.

62. My research on the Internet Archive discovered the publisher's book list, archived on September 19, 2000, that includes *Andrews*, and the publisher's webpage for *Andrews*, first archived on January 19, 2001. These two webpages are presented as **Appendix 1011D**. I personally located, identified, and obtained these archived copies. These are the type of materials experts in my field would reasonably rely on when forming their opinions.

63. Page 1 of **Appendix 1011D** is a screen capture by me to show the webpage for "Addison Wesley – Distributed Systems" with a Wayback Machine

Amazon
Exhibit 1005
Page 036

calendar that shows this webpage was captured six times from "7 Apr 2000 – 17 Jun 2001" and this copy was captured on September 16, 2000. This webpage is part of the Addison Wesley Catalog, and this section on "Distributed Systems" lists "Foundations of Multithreaded, Parallel, and Distributed Programming" as a book by Gregory Andrews, published in 2000 with an ISBN of "0-201-35752-6." Page 2 of **Appendix 1011D** is a screen capture by me of the publisher's webpage for *Andrews* with a Wayback Machine calendar that shows this homepage was archived 24 times from "19 Jan 2001 – 14 Apr 2021" and this copy was archived on January 19, 2001. The screen capture shows the title, author, copyright date of 2000, 664 pp, and the ISBN. It also includes a link to the author's homepage. Pages 3 to 9 are the PDF copy of this webpage that shows the book has 12 chapters. **Appendix 1011D** informs my opinion that at least by September 19, 2000, the publisher had already had a webpage listing Andrews, and at least by January 19, 2001, users would have discovered the publisher's webpage for *Andrews* on the Internet.

64. Taken together, the author's homepage and the publisher's webpages archived by the Internet Archive inform my opinion that the author had publicized his book at least by August 1, 2000, when he last updated his homepage for the book, so users would have been able to discover this book by August 1, 2000, or at least by September 19, 2000, when the author's home page and the publisher's listing of Andrews were archived by the Internet Archive.

Amazon
Exhibit 1005
Page 037

F. Stanford University Records

65. My research on WorldCat for *Andrews* (https://worldcat.org/title/42296220) also identified the Stanford University Libraries as a holding library, so I searched their online catalog for records, and the search results confirmed the holding information. **Appendix 1011E** present true and correct copies of the bibliographic record and MARC record for *Andrews*. I personally located, identified, and obtained these records from the online catalog of the Stanford University Libraries. These are the types of records experts in my field would reasonably rely upon when forming their opinions. I captured the images of the bibliographic record and the MARC record to produce **Appendix 1011E**.

66. Page 1 of **Appendix 1011E** is the bibliographic record for *Andrews* that shows the title, author, imprint (publication place, publisher and copyright date), and physical description of this book. These data match the information presented in **Ex. 1011** and confirm this record represents *Andrews*. The "At the library" area shows this book has a call number of "QA76.58 .A57 2000," with one copy stored at "SAL3 (off-campus storage)" and is available for user request.

67. Pages 1 to 3 shows the contents of *Andrews*, and page 4 shows the subjects, publication date, variant title and ISBNs of this book. Page 5 is the Stanford MARC record. The first six digits of Field 008 show the record was created on "990810" (*i.e.*, August 10, 1999), and the "s2000" code following this record

Amazon
Exhibit 1005
Page 038

creation date shows that this book has a "2000" publication date. Subfield "a" of Field 040 shows that "DLC" (*i.e.*, the Library of Congress) created this record. Filed 916 on page 8 of **Appendix 1011E** shows "Date cataloged 20000210" meaning the Stanford University MARC record was created on February 10, 2000. Data from Fields 008, 040, and 916 inform my opinion that the Stanford University Libraries used the Library of Congress MARC record, which is a Library of Congress CIP record created on August 10, 1999, to create their copy cataloging record on February 10, 2000, and their MARC record made *Andrews* discoverable from their online catalog from that day on. A common library practice is to process a newly cataloged book as soon as possible for user access, usually within a week after the cataloging work is completed. It is therefore my opinion that the physical copy of *Andrews* would have been publicly accessible by February 17, 2000, at the Stanford University Libraries.

68. Because the Stanford MARC record (**Appendix 1011E**) was based on the Library of Congress CIP MARC record (**Appendix 1011A**), it is nearly identical to the Library of Congress MARC record, except for the long contents notes and other local notes. As of February 10, 2000, the Stanford MARC record made *Andrews* searchable by ISBNs, author, title, variant title, LCC number, and three Library of Congress subject headings.

Amazon
Exhibit 1005
Page 039

G. Usage Record

69. Actual usage of a publication is reflected by the publications that make reference to it. The citation history on Google Scholar shows *Andrews* was cited 985 times, including 34 citations between 2000 and 2001. **Appendix 1011F** is a list I compiled that presents six citing publication from 2000 to show early usage. The earliest citation appeared in an IEEE publication in April 2000, supporting my opinion that the print copy of *Andrews* was publicly accessible in February 2000 when the Stanford University Libraries used the CIP record of the Library of Congress to create a copy cataloging record for *Andrews*.

H. Summary of My Opinion on *Andrews*

70. Based on the scanned copy of *Andrews* (**Ex. 1011**), the scanned copy of *Andrews* from the Amazon book I purchased (**Appendix 1011G**), and the Internet Archive records of the author's homepage for *Andrews* (**Appendix 1011C**) and the publisher's webpage for *Andrews* (**Appendix 1011D**), it is my opinion that the scanned copy obtained from counsel is authentic. Based on the library records from the Library of Congress and the Stanford University Libraries (**Appendices 1011A and 1011E**), the copyright registration record (**Appendix 1011B**), and my knowledge of library cataloging and processing procedures, it is my opinion that *Foundations of Multithreaded, Parallel, and Distributed Programming, to Mechanics* by Gregory Andrews was published in February 2000. Before the book

Amazon
Exhibit 1005
Page 040

was published, the Library of Congress created a CIP record for it in August 1999, and the CIP record was later used by the Stanford University Libraries to create their copy cataloging record on February 10, 2000. The record creation dates from these libraries inform my opinion that the print version of *Foundations of Multithreaded, Parallel, and Distributed Programming* by Gregory Andrews became publicly accessible at least by February 2000. Citation records show that the earliest citation was published in April 2000.

IV. Authentication and Public Availability of *Batheja* (Ex. 1012)

A. Authentication

71. **Ex. 1012** is a true and correct copy of "Adaptive cluster computing using Javaspaces," ("*Batheja*"), by J. Batheja and M. Parashar, in *2001 IEEE International Conference on Cluster Computing*, pp. 323-330, that I obtained from Institute of Electrical and Electronics. When I began preparing this declaration I searched WorldCat by the title of *Batheja* for records, and located a record (http://www.worldcat.org/oclc/8558313860) for *Batheja*. The "Find a copy online" area of the record showed a "View full text" link that led me to a database record for *Batheja* in the IEEE Xplore Digital Library. I then used the PDF link to purchase this article from the publisher. This IEEE copy is presented as **Ex. 1012** in this declaration.

72. **Ex. 1012** is a true and correct copy of *Batheja* that I obtained from IEEE

Amazon
Exhibit 1005
Page 041

Xplore, the digital library created and maintained by the Institute of Electrical and Electronics Engineers. Page 1 shows the title, authors, abstract, and five keywords for this article, including "Adaptive cluster computing," "Parallel/Distributed computing, "JavaSpaces," "Jini," and "SNMP." The bottom of this page shows a "2002" copyright date with IEEE as the copyright holder. It also shows *Batheja* appears in "Proceedings of the 2001 IEEE International Conference on Cluster Computing (CLUSTER'01)" and the ISBN (International Standard Book Number) of the proceedings is "0-7695-1116-3." **Ex. 1012** shows that *Batheja* has a total of eight pages, including five sections, 12 figures (some color, and Figures 9 and 10 have two figures), and 17 references. The references were published from 1992 to 1998 and resources available on the Internet had live hyperlink for easy access. This electronic copy from IEEE is not numbered.

73. The IEEE copy of *Batheja* (**Ex. 1012**) is in a condition that creates no suspicion about its authenticity. Specifically, this copy is not missing any intermediate pages, the text on each page appears to flow seamlessly from one page to the next, and there are no visible alterations to the document. Moreover, **Ex. 1012** was found in the custody of a digital library owned and managed by the publisher, a place where, if authentic, it would likely be found. Accordingly, I see no reason to question the authenticity of the IEEE copy of *Batheja* (**Ex. 1012**).

Amazon
Exhibit 1005
Page 042

B. IEEE Database Record for *Batheja*

74. **Appendix 1012A** is a true and correct copy of the database record for *Batheja* obtained from the IEEE Xplore Digital Library. I personally located, identified, and obtained this record. This is the type of record experts in my field would reasonably rely upon when forming their opinions.

75. Page 1 of **Appendix 1012A** shows the title and authors of the *Batheja* article. It also shows IEEE as the publisher, and indicates the full text of *Batheja* has nine paper citations and three patent citations, and has been viewed at the IEEE site 20 times. The abstract in this record is identical to the abstract in the IEEE copy (**Ex. 1012**), except the latter begins with "In this paper." The "Document Sections" shows the article has four sections, not including the section of references. The IEEE database record shows *Batheja* was published in "Proceedings 2001 IEEE International Conference on Cluster Computing." Page 2 of the database record shows that the conference took place in Las Vegas, NV, USA, from "14-17 Oct. 2001." The "Date added to IEEE Xplore" element shows that this database record was added to the IEEE Xplore system on "07 August 2002" and the article DOI (digital object identifier) is "10.1109/CLUSTR.2001.959993." The record also shows the conference proceedings has an ISBN of "0-7695-1116-3" and IEEE is the publisher. The article image on page 2 of this record shows the authors, abstract, keywords, ISBN of the volume in which *Batheja* appeared, copyright date, and the

Amazon
Exhibit 1005
Page 043

publisher. This page is the same as page 1 of the IEEE copy of *Batheja* (**Ex. 1012**).

76. The IEEE record for *Batheja* includes a direct link to "Proceedings 2001 IEEE International Conference on Cluster Computing" that contains *Batheja*, and the link leads to an IEEE record for this conference (available at https://ieeexplore.ieee.org/xpl/conhome/7601/proceeding). The front matter of the "2001 IEEE International Conference On Cluster Computing" is online and freely available to the public at https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=959944. The front matter is included in **Appendix 1012A** as pages 5 to 13.

77. Page 6 (internal page iii) of **Appendix 1012A** is the title page of the conference proceedings. It shows the title as "Proceedings, 2001 IEEE International Conference on Cluster Computing," and the conference dates and venue as "8-11 October 2001, Newport Beach, California, USA." It also shows the proceedings was "edited by Daniel S. Katz, Thomas Sterling, Mark Baker, Larry Bergman, Marcin Paprzqcki, and Rajkumar Buyya" and "IEEE Computer Society Task Force on Cluster Computing" was the conference sponsor. The page also shows IEEE Computer Society of Los Alamitos, California was the publisher. Page 7 is the copyright page that shows the print proceedings has a "2001" copyright date with The Institute of Electrical and Electronics Engineers, Inc. as the copyright holder. The copyright page also shows the ISBN of the proceedings is "0-7695-1116-3," the

Amazon
Exhibit 1005
Page 044

order number from IEEE Computer Society is "PR01116" and the Library of Congress Number for this proceedings is "2001093692." Pages 8 to 13 (internal pages v to x) are the table of contents of this proceedings, and page 11 (internal page viii) shows "Adaptive Cluster Computing Using JavaSpaces" by J. Batheja and M. Parashar is in the "Programming Models" section and appears on page 323 on. The data in the front matter confirm that *Batheja* is contained in the "Proceedings, 2001 IEEE International Conference on Cluster Computing." It should be noted that the IEEE record for *Batheja* and the front matter provided by IEEE show different conference dates and venue. Because the front matter displays the conference information clearly and the table of contents shows *Batheja* is included in this volume of proceedings, it is my opinion that the IEEE record for *Batheja* may have contained errors in the conference dates and venue. A library copy of the proceedings from the Linda Hall Library (**Appendix 1012B**) discussed below will confirm my opinion regarding the conference dates and venue of 2001 IEEE International Conference on Cluster Computing.

78. Data in the IEEE record for *Batheja* and the front matter of the print copy of the conference proceedings containing *Batheja* inform my opinion that *Batheja* was published in 2001, and IEEE Xplore indexed the article and added the database record into the IEEE Xplore Digital Library on August 7, 2002. The IEEE record made *Batheja* discoverable in the IEEE Xplore system from August 7, 2002

Amazon
Exhibit 1005
Page 045

on. Interested users in August 2007 would have been able to locate the electronic copy of *Batheja* and purchase the article from IEEE or obtain it from an institution that subscribed to IEEE Xplore Digital Library.

C. Copy of *Batheja* from the Linda Hall Library

79. To determine the availability of the print copy of *Batheja* in libraries, I searched for "2001 IEEE International Conference on Cluster Computing" that contains *Batheja* in WorldCat, and the retrieved record (http://www.worldcat.org/oclc/48478137) identified the Linda Hall Library as one of the holdings libraries of this proceedings, so I searched their online catalog for records, and the search results confirmed the holdings information. I then requested a scanned copy of *Batheja* from the Document Delivery Service of the Linda Hall Library and received the copy on November 19, 2021. This copy is presented as **Appendix 1012B**.

80. Pages 1 to 8 (internal pages 323-330) are the *Batheja* article obtained from the Linda Hall Library. I have closely compared this library copy of *Batheja* (**Appendix 1012B**) with the copy purchased from IEEE (**Ex. 1012**) and concluded that these two copies have the same content. The differences are that the IEEE copy is an electronic copy, does not have page numbers, includes some figures in color, and the hypertext links in the references are live, whereas the library copy is a print copy that carries page numbers, is black and white, and includes hypertext links that

Amazon
Exhibit 1005
Page 046

are not live (meaning users have to enter the URLs to access those resources).

81. Page 9 is the title page of "Proceedings, 2001 IEEE International Conference on Cluster Computing" that contains *Batheja*. I have closely compared this title page with the front matter of the conference proceedings that I obtained from IEEE (page 6 of **Appendix 1012A**) and found them to be identical. Page 10 of the Linda Hall copy (**Appendix 1012B**) is the cover of the conference proceedings that shows the conference name, editors, dates and venue of the conference, the publisher and conference sponsors. Page 11 is the copyright page that is identical to the copyright page in the front matter of the conference proceedings obtained from IEEE (page 7 of **Appendix 1012A**). The Linda Hall copy of the copyright page shows a hand-written call number of "QA76.58 .I132 2001." Pages 12-17 (internal pages v to x) are the table of contents, and page 12 shows a library stamp of "Linda Hall Library, Kansas City, MO" and a hand-written note of "6-1-02" in the upper left corner that indicates this publication was added to the library collections on June 1, 2002. The knowledge of this library practice is based on my experience with the Linda Hall Library collection and staff. I have closely compared the table of contents of the Linda Hall copy with the IEEE copy and found them to be identical, with both indicating that "Adaptive Cluster Computing Using JavaSpaces" by Batheja and Parashar is included in the "Programming Models" section and appears from page 323 on.

Amazon
Exhibit 1005
Page 047

82. The Linda Hall copy of *Batheja* (**Appendix 1012B**) is in a condition that creates no suspicion about its authenticity. Specifically, this copy is not missing any intermediate pages, the text on each page appears to flow seamlessly from one page to the next, and there are no visible alterations to the document. Moreover, **Appendix 1012B** was found in the custody of a library, a place where, if authentic, it would likely be found. Accordingly, I see no reason to question the authenticity of this library copy of *Batheja* (**Appendix 1012B**).

D. Library Records from the Linda Hall Library

83. **Appendix 1012C** contains true and correct copies of the bibliographic and MARC records for IEEE International Conference on Cluster Computing that contains *Batheja*. I personally located, identified, and obtained these records from the online catalog of the Linda Hall Library. Because the records could not be downloaded cleanly, I used screenshots of these record to produce **Appendix 1012C**. These are the type of records experts in my field would reasonably rely upon when forming their opinions.

84. **Appendix 1012C** contains true and correct copies of the bibliographic and MARC record for *2001 IEEE International Conference on Cluster Computing : 8-11 October 2001, Newport Beach, California, USA : proceedings*. The top of the record identifies this publication as "conference proceeding" and indicates it is "available at Linda Hall Library Books – 2nd Floor (QA76.58 .I132 2001)." The

Amazon
Exhibit 1005
Page 048

location item area shows the item is "loanable." Page 2 shows details of the bibliographic record, including the title, other titles, creator and contributors, subjects, identifiers, publisher, creation date, physical format, and general notes.

85. Page 3 of **Appendix 1012C** is the MARC record that shows the date of record creation. The first six digits of Field 008 show the MARC record was created on "001211" (i.e., December 11, 2000), and the "s2001" code after the first six digits shows the publication has a 2001 publication date. Subfields "a" and "c" of Field 040 show that "ZCU" is the cataloging agency that created the original MARC record, and subfield "d" shows "LHL" modified the ZCU record later. According to the *Directory of OCLC Members* (https://www.oclc.org/en/contacts/libraries.html), "ZCU" is the OCLC library symbol for Columbia University Libraries, and "LHL" is the OCLC library symbol for the Linda Hall Library. Data in Fields 008 and 040 inform my opinion that the Columbia University Libraries created the initial MARC record on December 11, 2000 for a book with a publication date of 2001. It is therefore my opinion that the original MARC record was probably created as an acquisition record at the Columbia University Libraries. What is significant is that the Linda Hall Library later used the Columbia University MARC record as the basis to create their copy cataloging record, and the Linda Hall copy of *Batheja* with the front matter of *2001 IEEE International Conference on Cluster Computing* shows that "6-1-02" (i.e. June 1, 2002) was the date when the conference proceedings was

Amazon
Exhibit 1005
Page 049

cataloged and added to the Linda Hall Library collections. The Linda Hall Library records show that the library cataloged *2001 IEEE International Conference on Cluster Computing* as a book. A common library practice is to make newly cataloged items available to users soon after the cataloging work is completed, usually within a week. It is therefore my opinion that this MARC record was created at the Linda Hall Library on June 1, 2002, making this publication discoverable, and the physical copy would have been available for public access no later than a week after June 1, meaning publicly accessible no later than June 8, 2002.

86. The second Field 020 in the MARC record (**Appendix 1012C**) shows the 10-digit ISBN of *2001 IEEE International Conference on Cluster Computing* that matches the ISBN on the copyright page of the proceedings (**Appendix 1012A**). The MARC record shows that the authorized form of the conference name is used as the author for this record in Field 111. Field 245 is the title and statement of responsibility field that shows the title, editors, and sponsor information from the title page. Three variant titles are included in Fields 246 of the MARC record to help users discover this publication. Field 260 shows IEEE Computer Society of Los Alamitos, Calif. published this work with a 2001 copyright date. A note in Field 500 shows that "IEEE Computer Society order number PR01116" appears on the verso of the title page (i.e., the copyright page). The ISBN, conference name, title, publisher, copyright date, and the order number match the information contained in

Amazon
Exhibit 1005
Page 050

the IEEE copy (**Ex. 1012**) and the Linda Hall copy (**Appendix 1012B**), confirming the MARC record represents *2001 IEEE International Conference on Cluster Computing*. The first editor and two IEEE organizations are included in Field 700 and Fields 710 as additional access points to help users discover this work.

87. Subjects of the conference are represented in Field 050 by a Library of Congress Classification (LCC) number of "QA76.58" that represents "parallel computers" and in Fields 650 by four Library of Congress subject headings: Beowulf clusters (Computer systems)," "High performance computing," "Parallel computers," and "Electronic data processing $x Distributed process" ("$x" indicates a topical subdivision), each followed by "Congresses" as a form subheading encoded in "$v" (subfield v) to indicate these topics are treated at a conference.

88. The MARC record (**Appendix 1012C**) and the Linda Hall copy (**Appendix 1012B**) inform my opinion that the Linda Hall Library cataloged *2001 IEEE International Conference on Cluster Computing* on June 1, 2002, and their MARC record has made this publication searchable in their online catalog from that day on. As of June 1, 2002 users interested in the subjects of this publication would have been able to conduct subject searches by the LCC number and the Library of Congress subject headings in this MARC record to discover this publication (and *Batheja* contained therein). They would also have been able to discover and access it by the ISBN, the name of the conference, the title, variant titles, the first editor,

Amazon
Exhibit 1005
Page 051

and the sponsors. The physical copy of this conference proceedings would have been publicly accessible no later than a week after the cataloging work was completed, meaning public access to this conference proceedings (and *Batheja*) would have been no later than June 8, 2002.

E. Usage Record

89. Actual usage of a publication is reflected by the papers that make reference to it. The citation history on Google Scholar shows that *Batheja* has been cited 27 times. **Appendix 1012D** is a list I compiled that presents four selected citations published from 2003 to 2006 to demonstrate early usage. The earliest citations were published in February 2003 in International Conference on Information Networking by Springer.

F. Summary of My Opinion on *Batheja*

90. The IEEE copy of *Batheja* (**Ex. 1012**) shows it was published in *2001 IEEE International Conference on Cluster Computing*, and the IEEE record for *Batheja* (**Appendix 1012A**) shows the index record was completed and entered into the IEEE Xplore system on August 8, 2002. It is therefore my opinion that the IEEE Xplore record made *Batheja* publicly searchable from August 7, 2002 on. Interested users would have been able to purchase *Batheja* from IEEE or obtain free access to the article through an institution with a subscription to IEEE Xplore Digital Library. The Linda Hall copy of *Batheja* (**Appendix 1012B**) shows that *2001 IEEE*

Amazon
Exhibit 1005
Page 052

International Conference on Cluster Computing that contains *Batheja* was cataloged and added to the collections of the Linda Hall Library on June 1, 2002, and the MARC record for *2001 IEEE International Conference on Cluster Computing* has made subject searches for the conference proceedings possible by LCC and Library of Congress subject headings. As of June 2002, interested users would also have been able to search for this conference proceedings by its ISBN, the conference name, title, variant titles, the first editor and two conference sponsors. Citation history shows the earliest citation to *Batheja* appeared in February 2003, further supporting my opinion *Batheja* was publicly accessible as early as June 1, 2002 when it was cataloged by the Linda Hall Library, and no later than August 7, 2002 when it was indexed by IEEE Xplore.

Amazon
Exhibit 1005
Page 053

V. Conclusion

91. In signing this declaration, I recognize that the declaration will be filed as evidence in a Petition for *Inter Partes* Review before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I also recognize that I may be subject to deposition in the case. If deposition is required of me, I will appear at a reasonable time and place to be agreed upon.

92. I hereby declare that all statements made herein on my own knowledge are true and that all statements made on information and belief are believed to be true, and further, that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Date: 12-18-2022 Executed: Ingrid Hsieh-Yee

Ingrid Hsieh-Yee, Ph.D.

Amazon
Exhibit 1005
Page 054

APPENDIX A

MARC 21 Reference Materials

Part VII: A Summary of Commonly Used MARC 21 Fields
Part VIII: A List of Other Fields Often Seen in MARC Records
Part IX: The Leader
Part X: Field 008 for Books

Part VII:

A Summary of Commonly Used MARC 21 Fields

This is a summary of the MARC 21 tags used most frequently by libraries in entering their own bibliographic records. For full listings of all MARC 21 tags, indicators, and subfield codes, see *MARC 21 Format for Bibliographic Data*.

In the explanations on these pages:

Tags -- The tags (3-digit numbers) are followed by the names of the fields they represent. In this summary, and in the *MARC 21 Format for Bibliographic Data*, if a tag can appear more than once in one bibliographic record, it is labeled repeatable (R). If it can only be used once, it is labeled non-repeatable (NR). For example, a catalog record can have several subjects, so the tags for subject added entries (6XX) are labeled repeatable (R).

Indicators -- The use of indicators is explained in fields where they are used. Indicators are one-digit numbers. Beginning with the 010 field, in every field -- following the tag -- are two character positions, one for Indicator 1 and one for Indicator 2. The indicators are not actually defined in all fields, however. And it is possible that a 2nd indicator will be used, while the 1st indicator remains undefined (or vice versa). When an indicator is undefined, the character position will be represented by the character # (for blank space).

Subfield codes -- All the data in each field (beginning with the 010 field) is divided into subfields, each of which is preceded by a delimiter-subfield code combination. The most common subfield codes used with each tag are shown. Each subfield code is preceded by the character $, signifying a delimiter. The name of the subfield follows the code.

In general, every field MUST have a subfield 'a' (**$a**). One exception that is often seen is in Field 020 (ISBN), when the ISBN information (subfield **$a**) is unavailable but the price (subfield **$c**) is known. Some subfields are repeatable. In this summary, repeatability is noted for only the more common repeatable subfields.

Examples: Examples follow the explanation for each field. For clarity, one space has

Amazon
Exhibit 1005
Page 056

been placed between the tag and the first indicator, one space has been placed between the second indicator and the first delimiter- subfield code, and one space has been inserted between the delimiter-subfield code and the subfield data.

010 Library of Congress Control Number -- (LCCN)
(NR, or Not Repeatable

Indicators undefined.
Subfield used most often:
$a -- Library of Congress control number

```
Example:    010 ##  $a ###86000988#
```

020 International Standard Book Number -- (ISBN)
(R, or Repeatable)

Indicators undefined.
Subfields used most often:
$a -- International Standard Book Number
$c -- Terms of availability (often a price)
$z -- Cancelled/invalid ISBN (R)

```
Example:    020 ##  $a 0877547637
```

040 Cataloging source -- (NR)

Indicators undefined.
Subfields used most often:
$a -- Original cataloging agency
$c -- Transcribing agency
$d -- Modifying agency (R)

```
Example:    040 ## $a DLC
                   $c DLC
                   $d gwhs
```

100 Main entry -- Personal name -- (primary author)
(NR; there can be only one main entry)

Amazon
Exhibit 1005
Page 057

Indicator 1: Type of personal name entry element
0 -- Forename
1 -- Surname (this is the most common form)
3 -- Family name
Indicator 2 undefined.
Indicator 2 became obsolete in 1990. Older records may display 0 or 1
Subfields used most often:
$a -- Personal name
$b -- Numeration
$c -- Titles and other words associated with a name (R)
$q -- Fuller form of name
$d -- Dates associated with a name (generally, year of birth)

```
Example:    100 1# $a Gregory, Ruth W.
                   $q (Ruth Wilhelme),
                   $d 1910-
```

130 Main entry -- Uniform title -- (NR)

Indicator 1: Nonfiling characters
0-9 -- Number of nonfiling characters present (for initial articles, including spaces)
Indicator 2 undefined.
Indicator 2 became obsolete in 1990. (See 100 above.)
Subfields used most often:
$a -- Uniform title
$p -- Name of part/section of a work (R)
$l -- Language of a work
$s -- Version
$f -- Date of a work

```
Example:    130 0# $a Bible.
                   $p O.T.
                   $p Psalms.
```

240 Uniform title (NR)

Indicator 1: Uniform title printed or displayed
0 -- Not printed or displayed
1 -- Printed or displayed (most common)
Indicator 2: Nonfiling characters

Amazon
Exhibit 1005
Page 058

0-9 -- Number of nonfiling characters present (for initial articles, including spaces)

Subfields used most often:

$a -- Uniform title
$l -- Language of a work
$f -- Date of a work

```
Example:    240 10 $a Ile mystérieuse.
                   $l English.
                   $f 1978
```

245 Title Statement (NR)

Indicator 1: Title added entry

(Should the title be indexed as a title added entry?)

0 -- No title added entry

(indicates a title main entry; i.e. no author is given)

1 -- Title added entry

(the proper indicator when an author given in 1XX; the most common situation)

Indicator 2: Nonfiling characters

0-9 -- Number of nonfiling characters present, including spaces; usually set at zero, except when the title begins with an article; e.g., for *The robe*, the second indicator would be set to 4. The letters *T*, *h*, *e*, and the space following them are then ignored in alphabetizing titles. The record will be automatically filed under "*r*" -- for *Robe*.

Subfields used most often:

$a -- Title proper
$h -- Medium (often used for non-book media)
$p -- Name of part/section of a work (R)
$b -- Reminder of title (subtitles, etc.)
$c -- Remainder of title page transcription/Statement of responsibility

```
Example:    245 14 $a The DNA story :
                   $b a documentary history of gene
                      cloning /
                   $c James D. Watson, John Tooze.
```

246 Varying form of title (R)

Amazon
Exhibit 1005
Page 059

Indicator 1: Note/title added entry controller
- 1 -- Note, title added entry
- 3 -- No note, title added entry

Indicator 2: Type of title
- # -- No information provided
- 0 -- Portion of title
- 1 -- Parallel title
- 4 -- Cover title
- 8 -- Spine title

Subfield used most often:
- **$a** -- Title proper

```
Example:    246 3# $a Four corners power review
```

250 Edition statement (NR)

Indicators undefined.
Subfield used most often:
- **$a** -- Edition statement

```
Example:    250 ## $a 6th ed.
```

260 Publication, distribution, etc. (Imprint) (R)

Indicator 1: Sequence of publishing statements
- # -- No information provided

Indicator 2: Undefined
Subfields used most often:
- **$a** -- Place of publication, distribution, etc. (R)
- **$b** -- Name of publisher, distributor, etc. (R)
- **$c** -- Date of publication, distribution, etc. (R)

```
Example:    260 ## $a New York :
                   $b Chelsea House,
                   $c 1986.
```

300 Physical description (R)

Amazon
Exhibit 1005
Page 060

Indicators undefined.
Subfields used most often:
$a -- Extent (number of pages) (R)
$b -- Other physical details (usually illustration information)
$c -- Dimensions (cm.) (R)
$e -- Accompanying material (for example, "teacher's guide" or "manual")

```
Example:    300 ## $a 139 p. :
                   $b ill. ;
                   $c 24 cm.
```

440 Series statement / Added entry--Title

This field was made obsolete in 2008 to simplify the series statement. See 490 and 830.

490 Series statement (No added entry is traced from field) (R)

Indicator 1: Specifies whether series is traced (whether an 8XX tag is also present)
0 -- Series not traced
1 -- Series traced (8XX is in record)
Indicator 2 undefined.
Subfield used most often:
$a -- Series statement (R)
$v -- Volume number (R)

```
Example:    490 1# $a Colonial American craftsmen
```

500 General note (R)

Indicators undefined.
Subfield used most often:
$a -- General note (Used when no specialized note field has been defined for the information. Examples: Notes regarding the index; the source of the title; variations in title; descriptions of the nature, form, or scope of the item.)

```
Example:    500 ##  $a Includes index.
```

504 Bibliography, etc. note (R)

Amazon
Exhibit 1005
Page 061

Indicators undefined.
Subfield used most often:
$a -- Bibliography, etc. note

```
Example:    504 ## $a Includes bibliographical
                      references.
```

505 Formatted contents note (R)

Indicator 1: Type of contents note
0 -- Complete contents
1 -- Incomplete contents (used with multivolume set when some volumes are not yet published)
2 -- Partial contents
Indicator 2: Level of content designation
-- Basic
Subfield used most often:
$a -- Formatted contents note

```
Example:    505 0# $a Pride and prejudice -- Emma
                      -- Northanger Abbey.
```

520 Summary, etc. note (R)

Indicator 1: Display constant controller
-- Summary
1 -- Review
2 -- Scope and content
3 -- Abstract
Indicator 2 undefined
Subfields used most often
$a -- Summary, abstract, or annotation
$b -- Expansion of summary note

```
Example:    520 ## $a This basic guide to parliamentary
                      procedure tells how to conduct
                      and participate in a meeting
                      properly.
```


Amazon
Exhibit 1005
Page 062

600 Subject added entry -- Personal name (R)

Indicator 1: Type of personal name entry element
- 0 -- Forename
- 1 -- Surname (this is the most common form)
- 3 -- Family name

Indicator 2: Subject heading system/thesaurus (identifies the specific list or file which was used)
- 0 -- Library of Congress Subject Headings
- 1 -- LC subject headings for children's literature
- 2 -- Medical Subject Headings
- 3 -- National Agricultural Library subject authority file
- 4 -- Source not specified
- 5 -- Canadian Subject Headings
- 6 -- Répertoire de vedettes-matière
- 7 -- Source specified in subfield $2

(Note regarding Sears subject headings: The MARC 21 format does not provide an assigned indicator for Sears subject headings. Therefore, an indicator of 7 is used, and the MARC defined code "sears" is placed in subfield $2.)

Subfields used most often:
- **$a** -- Personal name (surname and forename)
- **$b** -- Numeration
- **$c** -- Titles and other words associated with a name (R)
- **$q** -- Fuller form of name
- **$d** -- Dates associated with a name (generally, year of birth)
- **$t** -- Title of a work
- **$v** -- Form subdivision (R)
- **$x** -- General subdivision (R)
- **$y** -- Chronological subdivision (R)
- **$z** -- Geographic subdivision (R)
- **$2** -- Source of heading or term (used with 2nd indicator of 7)

```
Example:    600 10 $a Shakespeare, William,
                   $d 1564-1616
                   $x Comedies
                   $x Stage history.
```

```
Example:    600 10 $a Shakespeare, William,
                   $d 1564-1616
                   $x Knowledge
```


Amazon
Exhibit 1005
Page 063

```
                        $z Rome
                        $v Congresses.
```

Notice that subfields $v, $x, and $z in the 600 field are repeatable. Subfields $v, $x, $y, and $z do not have to be in alphabetical order. They will be in the order prescribed by the instructions given by the subject heading system.

610 Subject added entry -- Corporate name (R)

*Indicator 1:*Type of corporate name entry element
- 0 -- Inverted name (not used with AACR2)
- 1 -- Jurisdiction name
- 2 -- Name in direct order

Indicator 2: Subject heading system/thesaurus.
- See indicator 2 under 600

Subfields used most often:
- **$a** -- Corporate name or jurisdiction name as entry element
- **$b** -- Subordinate unit (R)
- **$v** -- Form subdivision (R)
- **$x** -- General subdivision (R)
- **$y** -- Chronological subdivision (R)
- **$z** -- Geographic subdivision (R)
- **$2** -- Source of heading or term (used with 2nd indicator of 7)

```
Example:    610 10 $a United States.
                   $b Army Air Forces
                   $v Biography.
```

650 Subject added entry -- Topical term (Most subject headings fit here.) (R)

Indicator 1: Level of subject
- # -- No information provided

Indicator 2: Subject heading system/thesaurus
(identifies the specific list or file which was used)
- 0 -- Library of Congress Subject Headings
- 1 -- LC subject headings for children's literature
- 2 -- Medical Subject Headings
- 3 -- National Agricultural Library subject authority file
- 4 -- Source not specified
- 5 -- Canadian Subject Headings
- 6 -- Répertoire de vedettes-matière
- 7 -- Source specified in subfield $2

Amazon
Exhibit 1005
Page 064

Note regarding Sears subject headings: The MARC 21 format does not provide an assigned indicator for Sears subject headings. Therefore, an indicator of 7 is used, and the MARC defined code "sears" is placed in subfield $2.)

Subfields used most often:
$a -- Topical term
$v -- Form subdivision (R)
$x -- General subdivision (R)
$y -- Chronological subdivision (R)
$z -- Geographic subdivision (R)
$2 -- Source of heading or term used with 2nd indicator of 7)

```
Example:    650 #0 $a Theater
                   $z United States
                   $v Biography
                   $v Dictionaries.
```

Notice that subfields $v, $x, and $z in the 650 field are repeatable. Subfields $v, $x, $y, and $z do not have to be in alphabetical order. They will be in the order prescribed by the instructions given by the subject heading system.

651 Subject added entry -- Geographic name (R)

Indicator 1: undefined.
Indicator 2: Subject heading system/thesaurus.
See indicator 2 under 600
Subfields used most often:
$a -- Geographic name
$v -- Form subdivision (R)
$x -- General subdivision (R)
$y -- Chronological subdivision (R)
$z -- Geographic subdivision (R)
$2 -- Source of heading or term (used with 2nd indicator of 7)

```
Example:    651 #0 $a United States
                   $x History
                   $v Chronology.
```

Notice that subfields $v, $x, and $z in the 651 field are repeatable. Subfields $v, $x, $y, and $z do not have to be in alphabetical order. They will be in the order prescribed by the instructions given by the subject heading system.

Amazon
Exhibit 1005
Page 065

700 Added entry -- Personal name (R)

Indicator 1: Type of personal name entry element
- 0 -- Forename
- 1 -- Surname (this is the most common form)
- 3 -- Family name

Indicator 2: Type of added entry
- # -- No information provided (most common; co-authors, editors, etc.)
- 2 -- Analytical entry (The values for Indicator 2 changed in 1994 with Format Integration, and older records may display additional values. An analytical entry involves an author/title of an item contained in a work.)

Subfields used most often:
- **$a** -- Personal name
- **$b** -- Numeration
- **$c** -- Titles and other words associated with a name (R)
- **$q** -- Fuller form of name
- **$d** -- Dates associated with a name (generally, year of birth)
- **$e** -- Relator term (such as ill.) (R)
- **$4** -- Relator code (R)

```
Example:    700 1# $a Baldridge, Letitia.
```

710 Added entry -- Corporate name (R)

Indicator 1: Type of corporate name entry element
- 0 -- Inverted name (not used with AACR2)
- 1 -- Jurisdiction name
- 2 -- Name in direct order

Indicator 2: Type of added entry.
See Indicator 2 under 700
- # -- No information provided
- 2 -- Analytical entry

Subfields used most often:
- **$a** -- Corporate name or jurisdiction name as entry element
- **$b** -- Subordinate unit (R)

```
Example:    710 2# $a Sunburst Communications (Firm)
```

740 Added entry -- Uncontrolled related/analytical title (R)

Indicator 1: Nonfiling characters

Amazon
Exhibit 1005
Page 066

0-9 -- Number of nonfiling characters present (for initial articles, including spaces)

*Indicator 2:*Type of added entry. See Indicator 2 under 700

-- No information provided

2 -- Analytical entry

(This field was redefined in 1994 with Format Integration. Prior to 1994, the field was also used for variant titles, such as a different wording on a spine title. In records created since Format Integration, those variant tiles appear in a 246 field.)

Subfield used most often:

$a -- Title

```
Example:    740 02 $a Uncle Vanya.
```

800 Series added entry -- Personal name (R)

Indicator 1: Type of personal name entry element

0 -- Forename

1 -- Surname

3 -- Family name

Indicator 2 undefined.

Subfields used most often:

$a -- Personal name

$b -- Numeration

$c -- Titles and other words associated with a name (R)

$q -- Fuller form of name

$d -- Dates associated with a name (generally, year of birth)

$t -- Title of a work (the series)

$v -- Volume number

```
Example:    800 1# $a Fisher, Leonard Everett.
                   $t Colonial American craftsmen.
```

830 Series added entry -- Uniform title (R)

Indicator 1 undefined.

Indicator 2: Nonfiling characters

0-9 -- Number of nonfiling characters present (for initial articles, including spaces)

Subfield used most often:

$a -- Uniform title

Amazon
Exhibit 1005
Page 067

$v -- Volume number

```
Example:    830 #0 $a Railroads of America (Macmillan)
```

[Back to Top of Page]

Part VIII:

A List of Other Fields Often Seen in MARC Records

001	Control number
003	Control number identifier
005	Date and time of latest transaction
006	Fixed-length data elements -- additional material characteristics
007	Physical description fixed field
008	Fixed length data elements (See Part X)
022	International Standard Serial Number (ISSN)
037	Source of acquisition
041	Language code
043	Geographic area code
050	Library of Congress call number
060	National Library of Medicine call number
082	Dewey Decimal classification number (the one recommended by the Library of Congress; locally-assigned call numbers may appear elsewhere)
110	Main entry -- Corporate name (less frequent under AACR2 rules)
256	Computer file characteristics
263	Projected publication date (indicates a CIP -- Cataloging in Publication -- record)
306	Playing time
508	Creation/production credits note
510	Citation/references note (review sources)
511	Participant or performer note
521	Target audience note (first indicator: 0 = reading grade level, 1 = interest age level, 2 = interest grade level, 3 = special audience characteristics, 4 = motivation interest level)
530	Additional physical form available note
538	System details note
586	Awards note
656	Index term -- Occupation
730	Added entry -- Uniform title
852	Location

Amazon
Exhibit 1005
Page 068

856 Electronic location and access
9XX Reserved for local use. (They are used by vendors, systems, or individual libraries to exchange additional data)

[Back to Top of Page]

Part IX:

The Leader

There are 24 positions in the Leader, numbered from 00 to 23. For fuller explanation, see the *MARC 21 Format for Bibliographic Data*.

00-04 Record length (calculated by the computer for each record)
05 Record status
- a = increase in encoding level
- c = corrected or revised
- d = deleted
- n = new
- p = increase in encoding from prepublication (previous CIP)

06 Type of record
- a = language material
- c = printed music
- d = manuscript music
- e = cartographic material
- f = manuscript cartographic material
- g = projected medium
- i = nonmusical sound recording
- j = musical sound recording
- k = 2-dimensional nonprojectable graphic
- m = computer file
- o = kit
- p = mixed materials
- r = 3-dimensional artifact or naturally occurring object
- t = manuscript language material

07 Bibliographic level
- a = monographic component part
- b = serial component part
- c = collection
- d = subunit
- i = integrating resource
- m = monograph/item
- s = serial

08 Type of control
- # = no specified type
- a = archival

Amazon
Exhibit 1005
Page 069

09 Character coding scheme
- # = MARC-8
- a = UCS/Unicode

10 Indicator count (always "2")

11 Subfield code count (always "2")

12-16 Base address of data (calculated by the computer for each record)

17 Encoding level
- # = full level
- 1 = full level, material not examined
- 2 = less-than-full level, material not examined
- 3 = abbreviated level
- 4 = core level
- 5 = partial (preliminary) level
- 7 = minimal level
- 8 = prepublication level (CIP)
- u = unknown
- z = not applicable

18 Descriptive cataloging form
- # = non-ISBD
- a = AACR2
- i = ISBD
- u = unknown

19 Multipart resource record level
- # = Not specified or not applicable
- a = Set
- b = Part with independent title
- c = Part with dependent title

20 Length of the length-of-field portion (always "4")

21 Length of the starting-character-position portion (always "5")

22 Length of the implementation-defined portion (always "0")

23 Undefined (always "0")

[Back to Top of Page]

Part X:

Field 008 for Books

Field 008 is used for Fixed Length Data Elements ("Fixed Field Codes"). There are 40 character positions in field 008, numbered from 00-39. Undefined positions must contain either a blank (#) or a fill character (|). Positions 00-17 and 35-39 are defined the same way for all media.

The information shown here for positions 18-34 applies only to books. For explanation of all the positions below and for positions 18-34 for other media, see the *MARC 21 Format*

Amazon
Exhibit 1005
Page 070

for Bibliographic Data.

Note that field 008 has no indicators or subfield codes.

00-05 Date entered on file (YYMMDD),
where Y=year, M=month, and D=day

06 Type of date/publication status:
- b = no dates given; B.C. date involved
- e = detailed date
- s = single known date/probable date
- m = multiple dates
- r = reprint/reissue date (Date 1) and original date (Date 2)
- n = dates unknown
- q = questionable date
- t = publication date and copyright date
- | = no attempt to code

07-10 Date 1/beginning date of publication

11-14 Date 2/ending date of publication

Date fields contain the year(s) of publication. The type of date(s) in these elements are specified in fixed field element 06: Type of date/publication status. (For further details, see the field 008 description in the *MARC 21 Format for Bibliographic Data*.)

15-17 Place of publication, production, or execution
For example:
- pk# = Pakistan
- cau = California (US)

(For a full list of codes used in these positions, see the *MARC Code List for Countries*.)

18-21 Illustrations (up to 4 codes):
- # = no illustrations
- a = illustrations
- b = maps
- c = portraits
- d = charts
- e = plans
- f = plates
- g = music
- h = facsimiles
- i = coats of arms
- j = genealogical tables
- k = forms
- l = samples
- m = phonodisc, phonowire, etc.
- o = photographs
- p = illuminations
- | = no attempt to code

Amazon
Exhibit 1005
Page 071

22 Target audience:
- # = unknown or not specified
- a = preschool
- b = primary
- c = pre-adolescent
- d = adolescent
- e = adult
- f = specialized
- g = general
- j = juvenile
- | = no attempt to code

23 Form of item:
- # = none of the following
- a = microfilm
- b = microfiche
- c = microopaque
- d = large print
- f = braille
- r = regular print reproduction
- s = electronic
- | = no attempt to code

24-27 Nature of contents (up to 4):
- # = no specified nature of contents
- a = abstracts/summaries
- b = bibliographies (is one or contains one)
- c = catalogs
- d = dictionaries
- e = encyclopedias
- f = handbooks
- g = legal articles
- i = indexes
- j = patent document
- k = discographies
- l = legislation
- m = theses
- n = surveys of literature
- o = reviews
- p = programmed texts
- q = filmographies
- r = directories
- s = statistics
- t = technical reports
- u = standards/specifications
- v = legal cases and notes
- w = law reports and digests
- z = treaties
- | = no attempt to code

Amazon
Exhibit 1005
Page 072

28 Government publication:
- # = not a government publication
- i = international intergovernmental
- f = federal/national
- a = autonomous or semi-autonomous component
- s = state, provincial, territorial, dependent, etc.
- m = multistate
- c = multilocal
- l = local
- z = other type of government publication
- o = government publication -- level undetermined
- u = unknown if item is government publication
- | = no attempt to code

29 Conference publication:
- 0 = not a conference publication
- 1 = conference publication
- | = no attempt to code

30 Festschrift:
- 0 = not a festschrift
- 1 = festschrift
- | = no attempt to code

31 Index:
- 0 = no index
- 1 = index present
- | = no attempt to code

32 Undefined (since 1990) (Earlier records may contain the values 0 or 1)
- # = Undefined
- | = no attempt to code

33 Literary form:
- 0 = not fiction (not further specified)
- 1 = fiction (not further specified)
- c = comic strips
- d = dramas
- e = essays
- f = novels
- h = humor, satires, etc.
- i = letters
- j = short stories
- m = mixed forms
- p = poetry
- s = speeches
- u = unknown
- | = no attempt to code

34 Biography:
- # = no biographical material
- a = autobiography
- b = individual biography

Amazon
Exhibit 1005
Page 073

c = collective biography
d = contains biographical information
| = no attempt to code

35-37 Language:
A three-letter code. For example: eng fre ger spa rus ita

(For a full list of codes used in these positions, see the *MARC Code List for Languages*.)

38 Modified record:
= not modified
x = missing characters (because of characters unavailable in MARC character set)
s = shortened
d = "dashed-on" information omitted
r = completely romanized/printed cards in script
o = completely romanized/printed cards romanized
| = no attempt to code

39 Cataloging source:
= national bibliographic agency
c = cooperative cataloging program
d = other sources
u = unknown
| = no attempt to code

[Back to Top of Page]

[Back to Table of Contents] -- [Continue to Part 11]



Library of Congress
Library of Congress Help Desk (10/27/2009)

Amazon
Exhibit 1005
Page 074

APPENDIX 1009A

https://ethernethistory.typepad.com/papers/EthernetSpec.pdf
Go
50 captures
25 Jul 2011 - 3 Nov 2021
JUN JUL MAR
25
2010 2011 2013
About this capture

The Ethernet

A Local Area Network

Data Link Layer
and
Physical Layer
Specifications

The Ethernet

A Local Area Network

Data Link Layer
and
Physical Layer
Specifications







Digital Equipment Corporation
Maynard, MA

Intel Corporation
Santa Clara, CA

Xerox Corporation
Stamford, CT

Version 1.0

September 30, 1980

Amazon
Exhibit 1005
Page 077

IMPORTANT INFORMATION AND DISCLAIMERS

1. This specification includes subject matter relating to a patent(s) of Xerox Corporation. No license under such patent(s) is granted by implication, estoppel or otherwise as a result of publication of this specification. Applicable licenses may be obtained from Xerox Corporation.

2. This specification is furnished for informational purposes only. Digital, Intel, and Xerox do not warrant or represent that this specification or any products made in conformance with it will work in the intended manner or be compatible with other products in a network system. Nor do they assume responsibility for any errors that the specification may contain, or have any liabilities or obligations for damages (including but not limited to special, indirect or consequential damages) arising out of or in connection with the use of this specification in any way. Digital, Intel and Xerox products may follow or deviate from the specification without notice at any time.

3. No representations or warranties are made that this specification or anything made from it is or will be free from infringements or patents of third persons.

Preface

This document contains the specification of the Ethernet, a local area network developed jointly by Digital Equipment Corporation, Intel Corporation, and Xerox Corporation. The Ethernet specification arises from an extensive collaborative effort of the three corporations, and several years of work at Xerox on an earlier prototype Ethernet.

This specification is intended as a design reference document, rather than an introduction or tutorial. Readers seeking introductory material are directed to the reference list in Section 2, which cites several papers describing the intent, theory, and history of the Ethernet.

This document contains 7 sections, falling into three main groups:

Sections 1, 2, and 3 provide an overall description of the Ethernet, including its goals, and the scope of the specification.

Sections 4 and 5 describe the architectural structure of the Ethernet in terms of a functional model consisting of two layers, the Data Link Layer and the Physical Layer.

Sections 6 and 7 specify the two layers in detail, providing the primary technical specification of the Ethernet.

Readers wishing to obtain an initial grasp of the organization and content of the specification will be best served by reading Sections 1, 3, and 4. Readers involved in actual implementation of the Ethernet will find Sections 5, 6, and 7 to contain the central material of the specification. Section 2 provides references, and the appendices provide supplementary material.

The approach taken in the specification of the Data Link Layer in Section 6 is a procedural one; in addition to describing the necessary algorithms in English and control flow charts, the specification presents these algorithms in the language Pascal. This approach makes clear the required behavior of Data Link Layer, while leaving individual implementations free to exploit any appropriate technology.

Because the procedural approach is not suitable for specifying the details of the Physical Layer, Section 7 uses carefully worded English prose and numerous figures and tables to specify the necessary parameters of this layer.

Some aspects of the Ethernet are necessarily discussed in more than one place in this specification. Whenever any doubt arises concerning the official definition in such a case, the reader should utilize the Pascal procedural specification of the Data Link Layer in Section 6.5, and the detailed prose specification of the Physical Layer in Sections 7.2 through 7.9.

Amazon
Exhibit 1005
Page 079

One aspect of an overall network architecture which is not addressed by this specification is network management. The network management facility performs operation, maintenence, and planning functions for the network:

- Operation functions include parameter setting, such as address selection.
- Maintenance functions provide for fault detection, isolation, and repair.
- Planning functions include collection of statisical and usage information, necessary for planned network growth.

While network management itself is properly performed outside the Ethernet Data Link and Physical Layers, it requires appropriate additional interfaces to those layers, which will be defined in a subsequent version of this specification.

Amazon
Exhibit 1005
Page 080

Table of Contents

Preface i

1. INTRODUCTION 1

2. REFERENCES 3

3. GOALS AND NON-GOALS 4

3.1 Goals 4

3.2 Non-Goals 5

4. FUNCTIONAL MODEL OF THE ETHERNET ARCHITECTURE 6

4.1 Layering 8

4.2 Data Link Layer 9

4.3 Physical Layer 10

4.4 Ethernet Operation and the Functional Model 12

4.4.1 Transmission Without Contention 12

4.4.2 Reception Without Contention 12

4.4.3 Collisions: Handling of Contention 13

5. INTER-LAYER INTERFACES 15

5.1 Client Layer to Data Link Layer 15

5.2 Data Link Layer to Physical Layer 16

6. ETHERNET DATA LINK LAYER SPECIFICATION 19

6.1 Data link Layer Overview and Model 19

6.2 Frame format 19

6.2.1 Address Fields 21

6.2.1.1 Destination Address Field 21

6.2.1.2 Source Address Field 21

6.2.2 Type Field 21

6.2.3 Data Field 22

6.2.4 Frame Check Sequence Field 22

6.2.5 Frame Size Limitations 22

6.3 Frame Transmission 23

6.3.1 Transmit Data Encapsulation 23

6.3.1.1 Frame Assembly 23

6.3.1.2 Frame Check Sequence Generation 23

6.3.2 Transmit Link Management 23

Amazon
Exhibit 1005
Page 081

6.3.2.1 Carrier Deference 23
6.3.2.2 Interframe Spacing 24
6.3.2.3 Collision Handling 24
6.3.2.3.1 Collision Detection and Enforcement 24
6.3.2.3.2 Collision Backoff and Retransmission 25
6.4 Frame Reception 25
6.4.1 Receive Data Decapsulation 25
6.4.1.1 Framing 25
6.4.1.1.1 Maximum Frame Size 26
6.4.1.1.2 Integral Number of Octets in Frame 26
6.4.1.2 Address Recognition 26
6.4.1.2.1 Physical Addresses 26
6.4.1.2.2 Multicast Addresses 26
6.4.1.3 Frame Check Sequence Validation 26
6.4.1.4 Frame Disassembly 27
6.4.2 Receive Link Management 27
6.4.2.1 Collision Filtering 27
6.5 The Data Link Layer Procedural Model 27
6.5.1 Overview of the Procedural Model 27
6.5.1.1 Ground Rules for the Procedural Model 27
6.5.1.2 Use of Pascal in the Procedural Model 29
6.5.2 The Procedural Model 31
6.5.2.1 Global Declarations 34
6.5.2.1.1 Common Constants and Types 34
6.5.2.1.2 Transmit State Variables 35
6.5.2.1.3 Receive State Variables 35
6.5.2.1.4 Summary of Interlayer Interfaces 36
6.5.2.1.5 State Variable Initialization 37
6.5.2.2 Frame Transmission 38
6.5.2.3 Frame Reception 42
6.5.2.4 Common Procedures 44
7. ETHERNET PHYSICAL LAYER SPECIFICATION 45
7.1 Physical Channel Overview and Model 45
7.1.1 Channel Goals and Non-Goals 45
7.1.1.1 Goals 45
7.1.1.2 Non-Goals 45
7.1.2 Characteristics of the Channel 46
7.1.3 Functions Provided by the Channel 46
7.1.4 Implementation of the Channel 46
7.1.4.1 General Overview of Channel Hardware 47
7.1.4.2 Compatibility Interfaces 48
7.1.5 Channel Configuration Model 49
7.1.6 Channel Interfaces 53

Amazon
Exhibit 1005
Page 082

7.2 Transceiver Cable Compatibility Interface Specifications 53

7.2.1 Transceiver Cable Signals 53
7.2.1.1 Transmit Signal 53
7.2.1.2 Receive Signal 54
7.2.1.3 Collision Presence Signal 54
7.2.1.4 Power 54

7.2.2 Transceiver Cable Parameters 54
7.2.2.1 Mechanical Configuration 54
7.2.2.2 Characteristic Impedance 55
7.2.2.3 Attenuation 55
7.2.2.4 Velocity of Propagation 55
7.2.2.5 Pulse Distortion 55
7.2.2.6 Resistance 55
7.2.2.7 Transfer Impedance 55

7.2.3 Transceiver Cable Connectors 56

7.2.4 Transceiver Cable Drive 56

7.2.5 Transceiver Cable Receive 57
7.2.5.1 Load Impedance and Termination 57
7.2.5.2 Common Mode and CMRR 57

7.3 Coaxial Cable Compatibility Interface Specifications 58

7.3.1 Coaxial Cable Component Specifications 58
7.3.1.1 Coaxial Cable Parameters 58
7.3.1.1.1 Characteristic Impedance 58
7.3.1.1.2 Attenuation 58
7.3.1.1.3 Velocity of Propagation 58
7.3.1.1.4 Mechanical Requirements 59
7.3.1.1.5 Pulse Distortion 59
7.3.1.1.6 Jacket Marking 59
7.3.1.1.7 Transfer Impedance 59
7.3.1.2 Coaxial Cable Connectors 60
7.3.1.3 Coaxial Cable Terminators 60
7.3.1.4 Transceiver-to-Coaxial Cable Connections 61

7.3.2 Coaxial Cable Signaling 61

7.4 Transceiver Specifications 62

7.4.1 Transceiver-to-Coaxial Cable Interface 62
7.4.1.1 Input Impedance 62
7.4.1.2 Bias Current 62
7.4.1.3 Transmit Output Levels 63

7.4.2 Transceiver-to-Transceiver Cable Interface 63
7.4.2.1 Transmit Pair 63
7.4.2.2 Receive Pair 63
7.4.2.3 Collision Presence Pair 64
7.4.2.4 Power Pair 64

7.4.3 Electrical Isolation 64

7.4.4 Reliability 64
7.5 Channel Logic 64
7.5.1 Channel Encoding 64
7.5.1.1 Encoder 65
7.5.1.2 Decoder 65
7.5.1.3 Preamble Generation 66
7.5.2 Collision Detect Signal 67
7.5.3 Carrier Sense Signal 67
7.5.4 Channel Framing 68
7.5.4.1 Beginning-of-Frame Sequence 68
7.5.4.2 End-of-Frame Sequence 69
7.6 Channel Configuration Requirements 69
7.6.1 Cable Sectioning 69
7.6.2 Transceiver Placement 70
7.6.3 System Grounding 70
7.6.4 Repeaters 71
7.6.4.1 Carrier Detect and Transmit Repeat 71
7.6.4.2 Collision Detect and Collision Repeat 71
7.6.4.3 Repeater Signal Generation 71
7.6.4.3.1 Signal Amplification 71
7.6.4.3.2 Signal Timing 71
7.7 Environment Specifications 72
7.7.1 Electromagnetic Environment 72
7.7.2 Temperature and Humidity 72

Appendices

APPENDIX A: GLOSSARY 73
APPENDIX B: ASSIGNMENT OF ADDRESS AND TYPE VALUES 76
APPENDIX C: CRC IMPLEMENTATION 77
APPENDIX D: IMPLEMENTATION OF TRANSCEIVER CABLE DRIVER AND RECEIVER 80
APPENDIX E: INTERFRAME RECOVERY 82

Figures and Tables

Figure 4-1: Ethernet Architecture and Typical Implementation 7
Figure 4-2: Architectural Layering 8
Figure 4-3: Data Link Layer Functions 10
Figure 4-4: Physical Layer Functions 11
Figure 6-1: Data Link Layer Frame Format 20
Figure 6-2: Structure of the Data Link Procedural Model 31
Figure 6-3: Control Flow Summary -- Client Layer Processes 32
Figure 6-4: Control Flow Summary -- Data Link Layer Processes 33
Figure 7-1: Physical Channel Configurations 50
Table 7-1: Physical Channel Propagation Delay Budget 52
Figure 7-2: Maximum Transceiver Cable Transfer Impedance 55
Figure 7-3: Typical Transceiver Cable Waveform 57
Figure 7-4: Maximum Coaxial Cable Transfer Impedance 60
Figure 7-5: Typical Coaxial Cable Waveform 62
Figure 7-6: Manchester Encoding 65
Figure 7-7: Preamble Encoding 66
Figure 7-8: Functional Logic of **collisionDetect** Signal 67
Figure 7-9: Functional Logic of **carrierSense** Signal 68
Figure C-1: CRC Implementation 78
Figure D-1: Typical Transceiver Cable Driver 80
Figure D-2: Typical Transceiver Cable Receiver 81

Amazon
Exhibit 1005
Page 085

Amazon
Exhibit 1005
Page 086

1. INTRODUCTION

The Ethernet local area network provides a communication facility for high speed data exchange among computers and other digital devices located within a moderate-sized geographic area. Its primary characteristics include:

Physical Layer:

Data rate: 10 Million bits/sec

Maximum station separation: 2.5 Kilometers

Maximum number of stations: 1024

Medium: Shielded coaxial cable, base-band signalling

Topology: Branching non-rooted tree

Data Link Layer:

Link control procedure: Fully distributed peer protocol, with statistical contention resolution (CSMA/CD)

Message protocol: Variable size frames, "best-effort" delivery

The Ethernet, like other local area networks, falls in a middle ground between long distance, low speed networks which carry data for hundreds or thousands of kilometers, and specialized, very high speed interconnections which are generally limited to tens of meters. The Ethernet is intended primarily for use in such areas as office automation, distributed data processing, terminal access, and other situations requiring economical connection to a local communication medium carrying bursty traffic at high peak data rates. Use in situations demanding resistance to hostile environments, real-time response guarantees, and so on, while not specifically excluded, do not constitute the primary environment for which the Ethernet is designed.

The precursor to the Ethernet specified in this document was the "Experimental Ethernet", designed and implemented by Xerox in 1975, and used continually since that time by thousands of stations. The Ethernet defined here builds on that experience, and on the larger base of the combined experience of Digital, Intel, and Xerox in many forms of networking and computer interconnection.

In specifying the Ethernet, this document provides precise detailed definitions of the lowest two layers of an overall network architecture. It thus defines what is generally

Amazon
Exhibit 1005
Page 087

referred to as a *link-level* facility. It does not specify the higher level protocols needed to provide a complete network architecture. Such higher level protocols would generally include such functions as internetwork communication, error recovery, flow control, security measures (e.g. encryption), and other higher level functions that increase the power of the communication facility and/or tailor it to specific applications. In particular, it should be noted that all error recovery functions have been relegated to higher level protocols, in keeping with the low error rates that characterize local networks.

One of the main objectives of this specification is *compatibility.* As stated in Section 3, it is intended that *every implementation of the Ethernet be able to exchange data with every other implementation.* It should be noted that higher level protocols raise their own issues of compatibility over and above those addressed by the Ethernet and other link-level facilities. This does not eliminate the importance of link-level compatibility, however. While the compatibility provided by the Ethernet does not guarantee solutions to higher level compatibility problems, it does provide a context within which such problems can be addressed, by avoiding low level incompatibilities that would make direct communication impossible.

Amazon
Exhibit 1005
Page 088

2. REFERENCES

The following three papers describe the Experimental Ethernet, and are reprinted in: "The Ethernet Local Network: Three Reports," Xerox Palo Alto Research Center Technical Report CSL-80-2. (February, 1980.)

[1] Metcalfe, R. M. and Boggs, D. R., "Ethernet: Distributed Packet Switching for Local Computer Networks," *Communications of the ACM 19* 7 (July 1976).

[2] Crane, R. C. and Taft, E. A. "Practical Considerations in Ethernet Local Network Design," Presented at *Hawaii International Conference on System Sciences* (January, 1980).

[3] Shoch, J. F. and Hupp, J. A. "Measured Performance of an Ethernet Local Network," Presented at *Local Area Communications Network Symposium* Boston (May 1979).

The following references describe the ISO Open Systems Model:

[4] Zimmermann, H., "OSI Reference Model -- The ISO Model of Architecture for Open Systems Interconnection," *IEEE Transactions on Communication COM-28* 4 (April 1980).

[5] International Organization for Standardization (ISO), "Reference Model of Open Systems Interconnection," *Document no. ISO/TC97/SC16 N227* (June 1979).

The following references describe the Pascal language (used in the Data Link Layer procedural model) and its derivative Concurrent Pascal:

[6] Jensen, K. and Wirth, N., *Pascal User Manual and Report, 2nd Edition.* Springer-Verlag (1974).

[7] Brinch Hansen, P., *Concurrent Pascal Report.* Technical Report CIT-IS-TR 17, California Institute of Technology (1975).

The following references discuss the CRC code used for the frame check sequence:

[8] Hammond, J. L., Brown, J. E. and Liu, S. S., "Development of a Transmission Error Model and an Error Control Model," Technical Report RADC-TR-75-138, Rome Air Development Center (1975).

[9] Bittel, R., "On Frame Check Sequence (FCS) Generation and Checking.," ANSI working paper X3-S34-77-43, (1977).

Amazon
Exhibit 1005
Page 089

3. GOALS AND NON-GOALS

This section states the assumptions underlying the design of the Ethernet.

3.1 Goals

The goals of the Ethernet design are:

Simplicity: Features which would complicate the design without substantially contributing to the meeting of the other goals have been excluded.

Low cost: Since technological improvements will continue to reduce the overall cost of stations wishing to connect to the Ethernet, the cost of the connection itself should be minimized.

Compatibility: All implementations of the Ethernet should be capable of exchanging data at the data link level. For this reason, the specification avoids optional features, to eliminate the possibility of incompatible variants of the Ethernet.

Addressing flexibility: The addressing mechanisms should provide the capability to target frames to a single node, a group of nodes, or to all nodes on the network.

Fairness: All nodes should have equal access to the network when averaged over time.

Progress: No single node operating in accordance with the protocol should be able to prevent the progress of other nodes.

High speed: The network should operate efficiently at a data rate of 10 Megabits per second.

Low delay: At any given level of offered traffic, the network should introduce as little delay as possible in the transfer of a frame.

Stability: The network should be stable under all load conditions, in the sense that the delivered traffic should be a monotonically non-decreasing function of the total offered traffic.

Maintainability: The Ethernet design should allow for network maintenance, operation, and planning.

Layered Architecture: The Ethernet design should be specified in layered terms to separate the logical aspects of the data link protocol from the physical details of the communication medium.

3.2 Non-Goals

The following are *not* goals of the Ethernet design:

Full duplex: At any given instant, the Ethernet can transfer data from one source station to one or more destination stations. Bi-directional communication is provided by rapid exhange of frames, rather than full duplex operation.

Error control: Error handling at the data link level is limited to detection of bit errors in the physical channel, and the detection and recovery from collisions. Provision of a complete error control facility to handle detected errors is relegated to higher layers of the network architecture.

Security: The data link protocol does not employ encryption or other mechanisms to provide security. Higher layers of the network architecture may provide such facilities as appropriate.

Speed flexibility: This specification defines a physical channel operating at a single fixed data rate of 10 Megabits per second.

Priority: The data link protocol provides no support of priority station operation.

Hostile user: There is no attempt to protect the network from a malicious user at the data link level.

Amazon
Exhibit 1005
Page 091

4. FUNCTIONAL MODEL OF THE ETHERNET ARCHITECTURE

There are two important ways to view the Ethernet design, corresponding to:

Architecture, emphasizing the logical divisions of the system, and how they fit together.

Implementation, emphasizing the actual components, and their packaging and interconnection.

Figure 4-1 illustrates these two views as they apply to a typical implementation, showing how each view groups the various functions.

This document is organized along *architectural* lines, emphasizing the large-scale separation of the Ethernet system into two parts: the *Data Link Layer* and the *Physical Layer.* These layers are intended to correspond closely to the lowest layers of the ISO Model for Open Systems Interconnection [4,5]. Architectural organization of the specification has two main advantages:

Clarity: A clean overall division of the design along architectural lines makes the specification clearer.

Flexibility: Segregation of medium-dependent aspects in the Physical Layer allows the Data Link Layer to apply to transmission media other than the specified coaxial cable.

As is evident in Figure 4-1, the architectural model is based on a set of interfaces different from those emphasized in the implementations. One crucial aspect of the design, however, must be addressed largely in terms of the implementation interfaces: *compatibility.* Two important compatibility interfaces are defined within what is architecturally the Physical Layer:

Coaxial cable interface: To communicate via the Ethernet, all stations must adhere rigidly to the exact specification of coaxial cable signals defined in this document, and to the procedures which define correct behavior of a station. The medium-independent aspects of the Data Link Layer should not be taken as detracting from this point: *communication via the Ethernet requires complete compatibility at the coaxial cable interface.*

Amazon
Exhibit 1005
Page 092

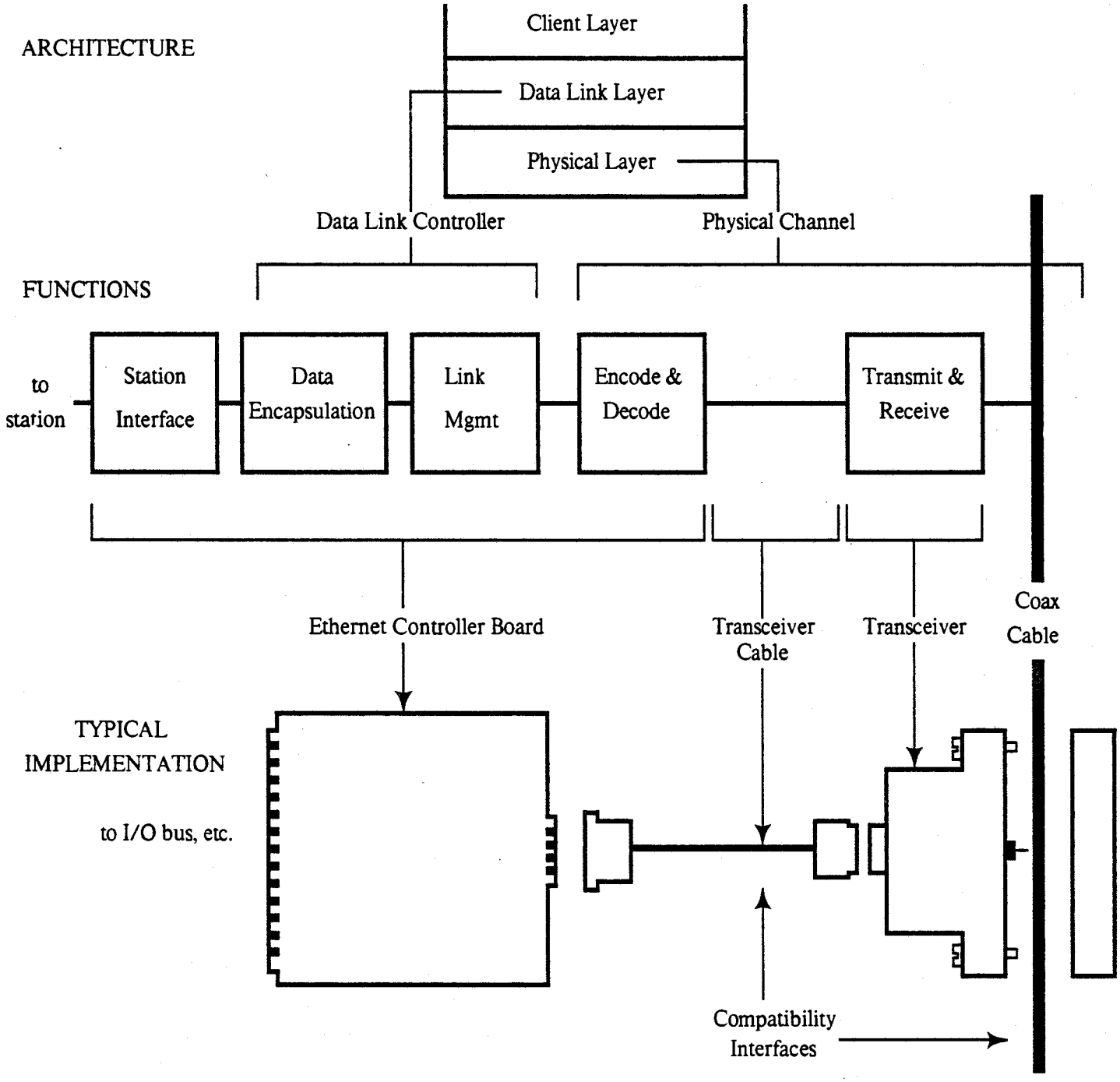


Figure 4-1: Ethernet Architecture and Typical Implementation

Transceiver cable interface: It is anticipated that most stations will be located some distance away from their connection to the coaxial cable. While it is necessary to place a small amount of circuitry (the *transceiver*) directly adjacent to the coaxial cable, the majority of the electronics (the *controller*) can and should be placed with the station. Since it is desirable for the same transceiver to be usable with a wide variety of stations, a second compatibility interface, the *transceiver cable interface*, is defined. While conformance with this interface is not strictly necessary to insure communication, it is highly recommended, since it allows maximum flexibility in intermixing transceivers and stations.

4.1 Layering

The major division in the Ethernet Architecture is between the Physical Layer and the Data Link Layer, corresponding to the lowest two levels in the ISO model. The higher levels of the overall network architecture, which use the Data Link Layer, will be collectively referred to in this document as the "Client Layer" since, strictly speaking, the identity and function of higher level facilities are outside the scope of this specification. The intent, however, is that the Ethernet Physical and Data Link Layers support the higher layers of the ISO model (Network Layer, Transport Layer, etc.).

The overall structure of the layered architecture is shown in Figure 4-2.

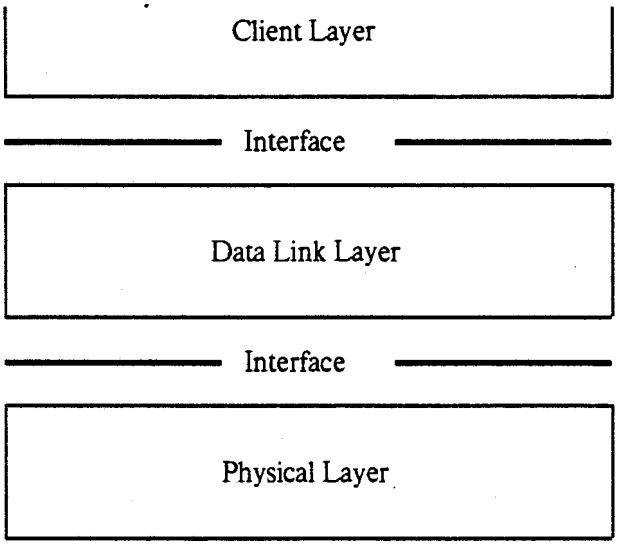


Figure 4-2: Architectural Layering

In the architectural model used here, the layers interact via well defined interfaces.

The interface between the Client Layer and the Data Link Layer includes facilities for transmitting and receiving frames, and provides per-operation status information for use by higher-level error recovery procedures.

The interface between the Data Link Layer and the Physical Layer includes signals for framing (carrier sense, transmit initiation) and contention resolution (collision detect), facilities for passing a pair of serial bit streams (transmit, receive) between the two layers, and a wait function for timing.

These interfaces are described more precisely in Section 5.

As mentioned in the preface, additional interfaces are necessary to allow a higher level network management facility to interact with the Data Link Layer and Physical Layer to perform operation, maintenance and planning functions.

4.2 Data Link Layer

The Data Link Layer defines a medium-independent link level communication facility, built on the medium-dependent physical channel provided by the Physical Layer. It is applicable to a general class of local area broadcast media suitable for use with the channel access discipline known as carrier-sense multiple-access with collision-detection (CSMA-CD). Compatibility with non-contention media (e.g., switched lines, token-passing rings, etc.), while a worthwhile topic for further research, is not addressed in this specification.

The Data Link Layer specified here is intended to be as similar as possible to that described in the ISO model. In a broadcast network like the Ethernet, the notion of a data link between two network entities does not correspond directly to a distinct physical connection. Nevertheless, the two main functions generally associated with a data link control procedure are present:

Data encapsulation

- framing (frame boundary delimitation)
- addressing (handling of source and destination addresses)
- error detection (detection of physical channel transmission errors)

Link management

- channel allocation (collision avoidance)
- contention resolution (collision handling)

This split is reflected in the division of the Data Link Layer into the Data Encapsulation sub-layer and the Link Management sub-layer, as shown in Figure 4-3.

Amazon
Exhibit 1005
Page 095

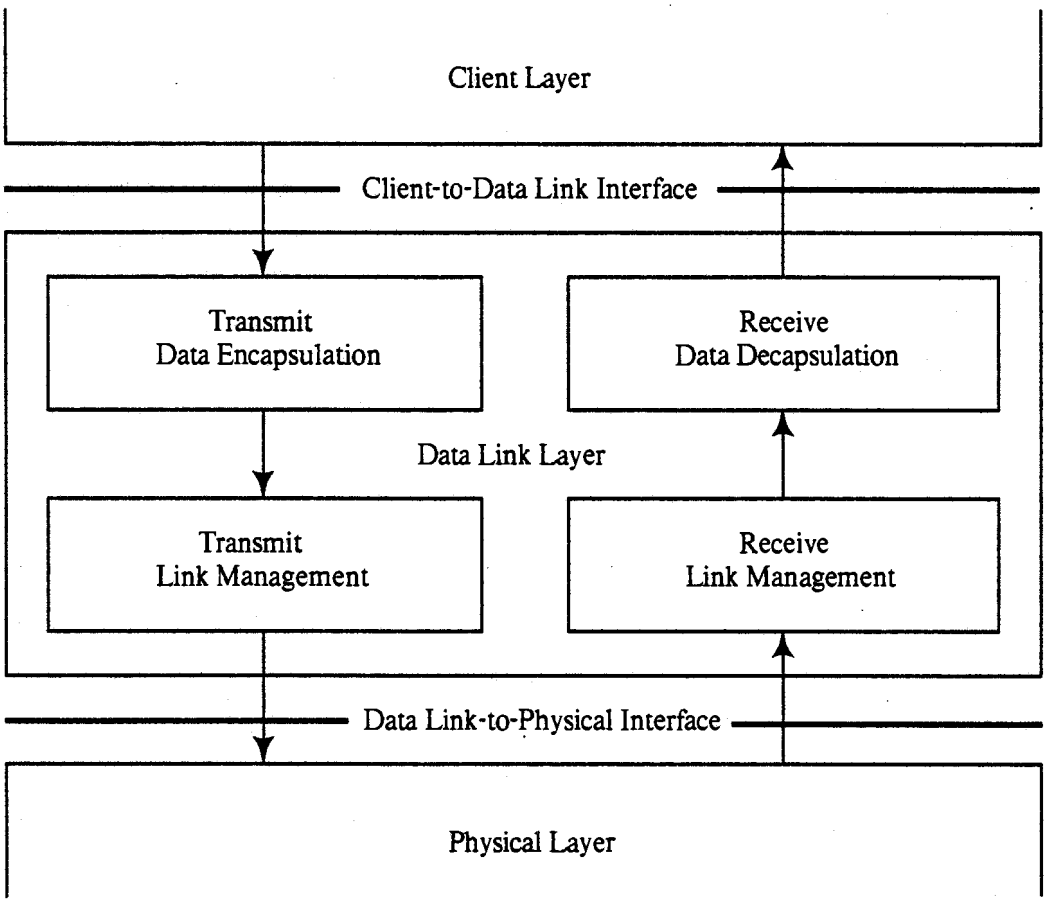


Figure 4-3: Data Link Layer Functions

In terms of the ISO model, the Ethernet Data Link Layer provides a multi-endpoint connection between higher-layer entities wishing to communicate. The connection provided is called a *data link*, and is implemented between two or more Data Link Layer entities called *data link controllers* via a Physical Layer connection called the *physical channel.*

4.3 Physical Layer

The Physical Layer specified in this document provides a 10 MBit/sec physical channel through a coaxial cable medium. Because one purpose of the layered architecture is to insulate the Data Link Layer from the medium-specific aspects of the channel, the Physical Layer completely specifies the essential physical characteristics of the Ethernet, such as data encoding, timing, voltage levels, etc. Implementation details are left unspecified, to retain maximum flexibility for the implementor. In all cases, the criterion applied in distinguishing between essential characteristics and implementation details is *guaranteed compatibility:* any two correct implementations of the Physical Layer specified here will be capable of exchanging data over the coaxial cable, enabling communication between their

Amazon
Exhibit 1005
Page 096

respective stations at the Data Link Layer.

The Physical Layer defined in this specification performs two main functions generally associated with physical channel control:

Data encoding

- preamble generation/removal (for synchronization)
- bit encoding/decoding (between binary and phase-encoded form)

Channel access

- bit transmission/reception (of encoded data)
- carrier sense (indicating traffic on the channel)
- collision detection (indicating contention on the channel)

This split is reflected in the division of the Physical Layer into the Data Encoding sub-layer and the Channel Access sub-layer, as shown in Figure 4-4.

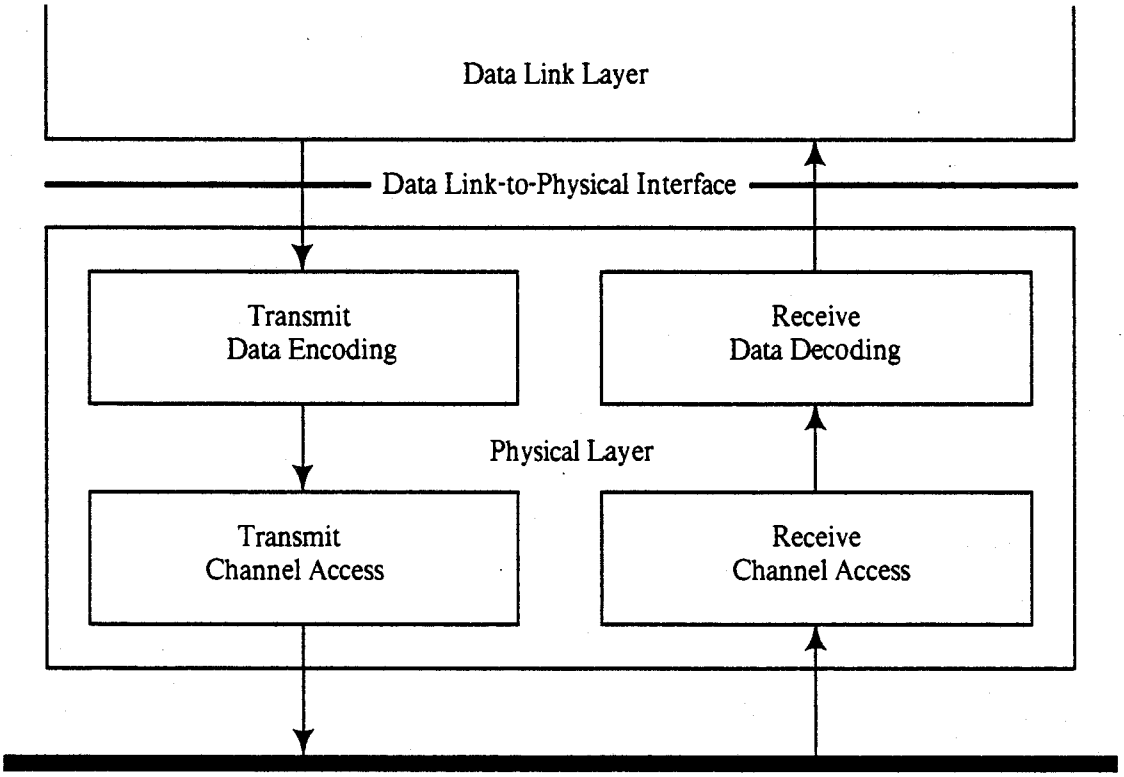


Figure 4-4: Physical Layer Functions

Amazon
Exhibit 1005
Page 097

4.4 Ethernet Operation and the Functional Model

This section provides an overview of frame transmission and reception in terms of the functional model of the architecture. This overview is descriptive, rather than definitional; the formal specifications of the operations described here are given in Sections 6 and 7.

4.4.1 Transmission Without Contention

When the Client Layer requests the transmission of a frame, the Transmit Data Encapsulation component of the Data Link Layer constructs the frame from the client-supplied data and appends a frame check sequence to provide for error detection. The frame is then handed to the Transmit Link Management component for transmission.

Transmit Link Management attempts to avoid contention with other traffic on the channel by monitoring the *carrier sense* signal and *deferring* to passing traffic. When the channel is clear, frame transmission is initiated (after a brief interframe delay to provide recovery time for other data link controllers and for the physical channel). The Data Link Layer then provides a serial stream of bits to the Physical Layer for transmission.

The Data Encoding component of the Physical Layer, before sending the actual bits of the frame, sends an encoded preamble to allow the receivers and repeaters along the channel to synchronize their clocks and other circuitry. It then begins translating the bits of the frame into encoded form and passes them to the Channel Access component for actual transmission over the medium.

The Channel Access component performs the task of actually generating the electrical signals on the medium which represent the bits of the frame. Simultaneously, it monitors the medium and generates the *collison detect* signal, which, in the contention-free case under discussion, remains off for the duration of the frame.

When transmission has completed without contention, the Data Link Layer so informs the Client Layer and awaits the next request for frame transmission.

4.4.2 Reception Without Contention

At the receiving station, the arrival of a frame is first detected by the Receive Channel Access component of the Physical Layer, which responds by synchronizing with the incoming preamble, and by turning on the carrier sense signal. As the encoded bits arrive from the medium, they are passed to the Receive Data Decoding component.

Amazon
Exhibit 1005
Page 098

Receive Data Decoding translates the encoded signal back into binary data and discards the leading bits, up to and including the end of the preamble. It then passes subsequent bits up to the Data Link Layer.

Meanwhile, the Receive Link Management component of the Data Link Layer, having seen carrier sense go on, has been waiting for the incoming bits to be delivered. Receive Link Management collects bits from the Physical Layer as long as the carrier sense signal remains on. When the carrier sense signal goes off, the frame is passed to Receive Data Decapsulation for processing.

Receive Data Decapsulation checks the frame's destination address field to decide whether the frame should be received by this station. If so, it passes the contents of the frame to the Client Layer along with an appropriate status code. The status code is generated by inspecting the frame check sequence to detect any damage to the frame enroute, and by checking for proper octet-boundary alignment of the end of the frame.

4.4.3 Collisions: Handling of Contention

If multiple stations attempt to transmit at the same time, it is possible for their transmitting data link controllers to interfere with each others' transmissions, in spite of their attempts to avoid this by deferring. When two stations' transmissions overlap, the resulting contention is called a *collision.* A given station can experience a collision during the initial part of its transmission (the "collision window"), before its transmitted signal has had time to propagate to all parts of the Ethernet channel. Once the collision window has passed, the station is said to have *acquired* the channel; subsequent collisions are avoided, since all other (properly functioning) stations can be assumed to have noticed the signal (via carrier sense) and to be deferring to it. The time to acquire the channel is thus based on the *round-trip propagation time* of the physical channel.

In the event of a collision, the Transmit Channel Access component of a transmitting station's Physical Layer first notices the interference on the channel and turns on the collision detect signal. This is noticed in turn by the Transmit Link Management component of the Data Link Layer, and collision handling begins. First, Transmit Link Management *enforces* the collision by transmitting a bit sequence called the *jam.* This insures that the duration of the collision is sufficient to be noticed by the other transmitting station(s) involved in the collision. After the jam is sent, Transmit Link Management terminates the transmission and schedules a retransmission attempt for a randomly selected time in the near future. Retransmission is attempted repeatedly in the face of repeated collisions. Since repeated collisions indicate a busy channel, however, Transmit Link Management attempts to adjust to the channel load by *backing off* (voluntarily delaying its own retransmissions to reduce its load on the channel). This is accomplished by

Amazon
Exhibit 1005
Page 099

expanding the interval from which the random retransmission time is selected on each retransmission attempt. Eventually, either the transmission succeeds, or the attempt is abandoned on the assumption that the channel has failed or has become overloaded.

At the receiving end, the bits resulting from a collision are received and decoded by the Physical Layer just as are the bits of a valid frame. In particular, collisions do *not* turn on the receiving station's collision detect signal, which is generated only during transmission. Instead, the fragmentary frames received during collisions are distinguished from valid frames by the Data Link's Receive Link Management component, by noting that a collision fragment is always smaller than the shortest valid frame. Such fragments are discarded by Receive Link Management.

Amazon
Exhibit 1005
Page 100

5. INTER-LAYER INTERFACES

The purpose of this section is to provide precise definitions of the interfaces between the architectural layers defined in Section 4. In order to provide such a definition, some precise notation must be adopted. The notation used here is the Pascal language, in keeping with the procedural nature of the formal Data Link Layer specification (see 6.5). Each interface is thus described as a set of procedures and/or shared variables which collectively provide the only valid interactions between layers. The accompanying text describes the meaning of each procedure or variable and points out any implicit interactions among them.

Note that the description of the interfaces in Pascal is a notational technique, and in no way implies that they can or should be implemented in software. This point is discussed more fully in 6.5, which provides complete Pascal declarations for the data types used in the remainder of this section. Note also that the "synchronous" (one frame at a time) nature of the frame transmission and reception operations is a property of the architectural interface between the Client Layer and the Data Link Layer, and need not be reflected in the implementation interface between a station and its controller.

5.1 Client Layer to Data Link Layer

The two primary services provided to the Client Layer by the Data Link Layer are transmission and reception of frames. The interface through which the Client Layer uses the facilities of the Data Link Layer therefore consists of a pair of functions.

Functions:
TransmitFrame
ReceiveFrame

Each of these functions has the components of a frame as its parameters (input or output), and returns a status code as its result.

The Client Layer transmits a frame by invoking *TransmitFrame*:

```
function TransmitFrame (
  destinationParam: AddressValue;
  sourceParam: AddressValue;
  typeParam: TypeValue;
  dataParam: DataValue): TransmitStatus;
```

The *TransmitFrame* operation is synchronous, in the sense that its duration is the entire attempt to transmit the frame, so that when the operation completes, transmission has either succeeded or failed, as indicated by the resulting status code:

```
type TransmitStatus = (transmitOK, excessiveCollisionError);
```

Successful transmission is indicated by the status code *transmitOK*; the code *excessiveCollisionError* indicates that the transmission attempt was aborted due to excessive collisions, because of heavy traffic or a network failure. Implementations may define additional implementation-dependent status codes if necessary.

The Client Layer accepts incoming frames by invoking *ReceiveFrame*:

```
function ReceiveFrame (
  var destinationParam: AddressValue;
  var sourceParam: AddressValue;
  var typeParam: TypeValue;
  var dataParam: DataValue): ReceiveStatus;
```

The *ReceiveFrame* operation is synchronous, in the sense that the operation does not complete until a frame has been received. The fields of the frame are delivered via the output parameters, along with a status code:

```
type ReceiveStatus = (receiveOK, frameCheckError, alignmentError);
```

Successful reception is indicated by the status code *receiveOK*. The code *frameCheckError* indicates that the frame received was damaged by a transmission error in the physical channel. The code *alignmentError* indicates that the frame received was damaged, and that in addition, its length was not an integral number of octets. Implementations may define additional implementation-dependent status codes if necessary.

5.2 Data Link Layer to Physical Layer

The interface through which the Data Link Layer uses the facilities of the Physical Layer consists of a function, a pair of procedures and three Boolean variables.

Function:	Variables:
ReceiveBit	*collisionDetect*
Procedures:	*carrierSense*
TransmitBit	*transmitting*
Wait	

During transmission, the contents of an outgoing frame are passed from the Data Link Layer to the Physical Layer via repeated use of the *TransmitBit* operation:

```
procedure TransmitBit (bitParam: Bit);
```

Each invocation of *TransmitBit* passes one new bit of the outgoing frame to the Physical Layer. The *TransmitBit* operation is synchronous, in the sense that the duration of the operation is the entire transmission of the bit, so that when the operation completes, the Physical Layer is ready to accept the next bit immediately. (Note: this does not imply that all invocations of *TransmitBit* are of exactly equal duration; for example, if the Physical Layer must perform some initial processing -- e.g., preamble generation -- before transmitting the first bit of a frame, the first

invocation of *TransmitBit* may take significantly longer.)

The overall event of data being transmitted is signaled to the Physical Layer via the variable *transmitting*:

```
var transmitting: Boolean;
```

Before sending the first bit of a frame, the Data Link Layer sets *transmitting* to *true*, to inform the Physical Link that a stream of bits will be presented via the *TransmitBit* operation. After the last bit of the frame has been presented, the Data Link Layer sets *transmitting* to *false* to indicate the end of the frame.

The presence of a collision in the physical channel is signaled to the Data Link Layer via the variable *collisionDetect*:

```
var collisionDetect: Boolean;
```

The *collisionDetect* signal remains *true* during the duration of the collision. (Note: Since an entire collision may occur during the first invocation of *TransmitBit* -- e.g., during preamble removal -- the Data Link Layer must handle this possibility by monitoring *collisionDetect* concurrently with its transmission of outgoing bits. See 6.5 for details.)

The *collisionDetect* signal is generated only during transmission and is never *true* at any other time; in particular, it cannot be used during frame reception to detect collisions between overlapping transmissions from two or more other stations.

During reception, the contents of an incoming frame are retrieved from the Physical Layer by the Data Link Layer via repeated use of the *ReceiveBit* operation:

```
function ReceiveBit: Bit;
```

Each invocation of *ReceiveBit* retrieves one new bit of the incoming frame (i.e., not including any preamble bits) from the Physical Layer. The *ReceiveBit* operation is synchronous, in the sense that its duration is the entire reception of a single bit. (As with *TransmitBit*, the first invocation of *ReceiveBit* make take significantly longer -- e.g., due to preamble removal). Upon receiving a bit, the Data Link Layer must immediately request the next bit until all bits of the frame have have been received. (See 6.5 for details.)

The overall event of data being received is signaled to the Data Link Layer via the variable *carrierSense*:

```
var carrierSense: Boolean;
```

When the Physical Layer sets *carrierSense* to *true*, the Data Link Layer must immediately begin retrieving the incoming bits via the *ReceiveBit* operation. When *carrierSense* subsequently becomes *false*, the Data Link Layer can begin processing the received bits as a completed frame. Note that the *true/false*

Amazon
Exhibit 1005
Page 103

transitions of *carrierSense* are not defined to be precisely synchronized with the beginning and end of the frame, but may precede the beginning and lag the end, respectively. If an invocation of *ReceiveBit* is pending when *carrierSense* becomes *false*, *ReceiveBit* returns an undefined value, which should be discarded by the Data Link Layer. (See 6.5 for details.)

The Data Link Layer must also monitor the value of *carrierSense* to defer its own transmissions when the channel is busy.

The Physical Layer also provides the procedure *Wait*:

```
procedure Wait (bitTimes: integer);
```

This procedure waits for the specified number of bit times. This allows the Data Link Layer to measure time intervals in units of the (physical-channel-dependent) bit time.

Another important property of the Physical Layer which is an implicit part of the interface presented to the Data Link Layer is the *round-trip propagation time* of the physical channel. This figure represents the maximum time required for a signal to propagate from one end of the network to the other, and for a collision to propagate back. The round-trip propagation time is primarily (but not entirely) a function of the physical size of the network. The round-trip propagation time of the Physical Layer is defined to be at most 450 bit times (see 7.1.2).

6. ETHERNET DATA LINK LAYER SPECIFICATION

6.1 Data Link Layer Overview and Model

As defined in Section 4, the Ethernet Architecture consists of the Data Link Layer, and below it, the Physical Layer. Furthermore, the Data Link Layer is divided into two sub-layers (see Figure 4-3).

Data encapsulation
- framing
- addressing
- error detection

Link management
- channel allocation
- contention resolution

This model is used throughout this section to structure the detailed specification of the Data Link Layer. An English description of the Data Link Layer is given in 6.2, 6.3, and 6.4. A more precise algorithmic definition is given in 6.5, which provides a procedural model for the Data Link Layer in the form of a program in the language Pascal. Note that whenever there is any apparent ambiguity concerning the definition of some aspect of the Data Link Layer, it is the Pascal procedural specification in 6.5 which should be consulted for the definitive statement.

6.2 Frame Format

The data encapsulation function of the Data Link Layer comprises the construction and processing of frames. The subfunctions of framing, addressing, and error detection are reflected in the frame format as follows:

Framing: No explicit framing information is needed, since the necessary framing cues (*carrierSense* and *transmitting*) are present in the interface to the Physical Layer.

Addressing: Two address fields are provided to identify the source and destination stations for the frame.

Error detection: A Frame Check Sequence field is provided for detection of transmission errors.

Figure 6-1 shows the five fields of a frame: the addresses of the frame's source and destination, a type field for use by higher layers (see 6.2.2), a data field containing the transmitted data, and the frame check sequence field containing a cyclic redundancy check value to detect transmission errors. Of these five fields, all are of fixed size except the data field, which may contain any integral number of octets between the minimum and maximum values specified below (see 6.2.5).

Amazon
Exhibit 1005
Page 105

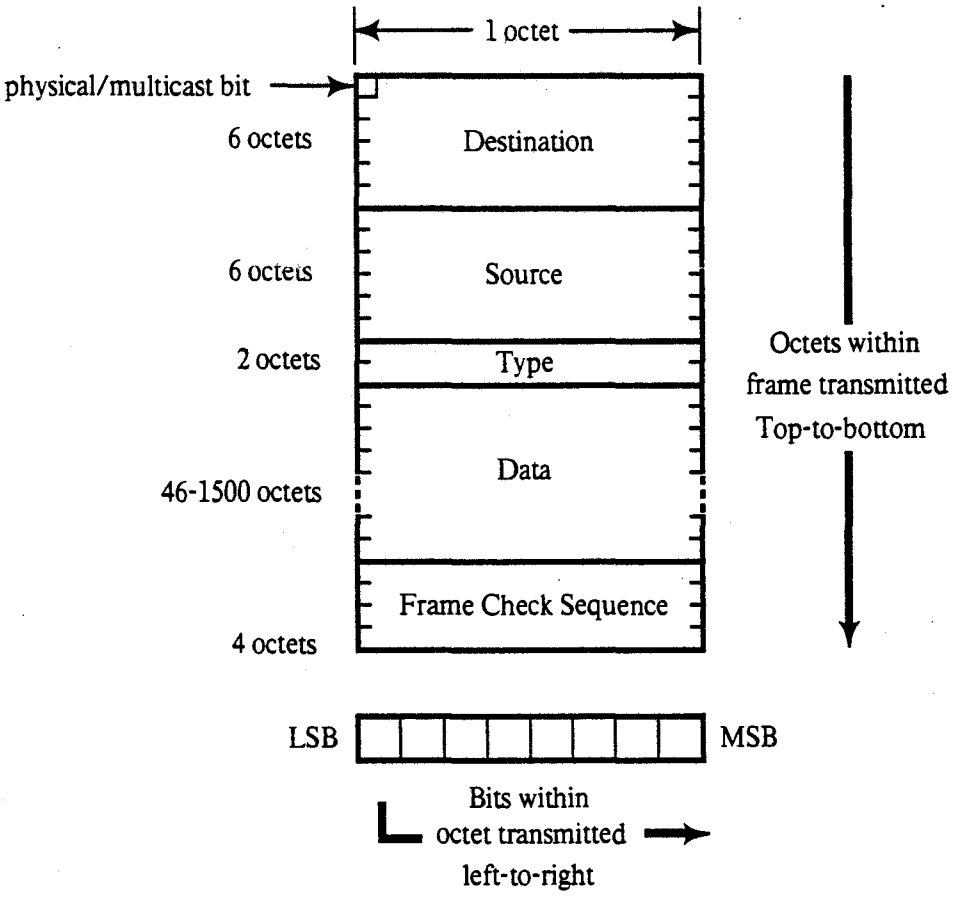


Figure 6-1: Data Link Layer Frame Format

Relative to Figure 6-1, the octets of a frame are transmitted from top to bottom, and the bits of each octet are transmitted from left to right.

NOTE

This document does *not* define an order of transmission for the octets of standard multi-octet data types (strings, integers, etc), since no values of such data types appear in the data link frame format. The order in which implementations of the Ethernet store the octets of a frame in computer memory, and the manner in which higher level protocols interpret the contents of the data field as values of various multi-octet data types, are beyond the scope of this specification.

The Ethernet itself is also totally insensitive to the interpretation of bits within a octet as constituting the digits of an 8-digit binary numeric value. Since some uniform convention is helpful, however, in avoiding needless incompatibility among different station types, the interpretation is arbitrarily defined to be that the left-most bit (first transmitted) is the low-order (2^0) digit and the right-most bit (last transmitted) is the high-order (2^7) digit.

Amazon
Exhibit 1005
Page 106

6.2.1 Address Fields

Data link addresses are 6 octets (48 bits) in length. A data link address is of one of two types:

Physical address: The unique address associated with a particular station on the Ethernet. A station's physical address should be distinct from the physical address of any other station on *any* Ethernet.

Multicast address: A multi-destination address, associated with one or more stations on a given Ethernet. There are two kinds of multicast address:

- *Multicast-group address:* An address associated by higher-level convention with a group of logically related stations.
- *Broadcast address:* A distinguished, predefined multicast address which always denotes the set of *all* stations on a given Ethernet.

The first bit of a data link address distinguishes physical from multicast addresses:

0 ⇒ physical address

1 ⇒ multicast address

In either case, the remainder of the first octet and all of the subsequent octets form a 47-bit pattern. In the case of the broadcast address, this pattern consists of 47 one-bits. There is no standard "null" address value.

The procedures for assigning suitably unique values for physical and multicast addresses are discussed in Appendix B.

6.2.1.1 Destination Address Field

The destination address field specifies the station(s) for which the frame is intended. It may be a physical or multicast (including broadcast) address. For details of address recognition by the receiving station(s), see 6.4.1.2.

6.2.1.2 Source Address Field

The source address field specifies the station sending the frame. The source address field is not interpreted at the Data Link Layer. It is specified at the data link level because a uniform convention for the placement of this field is crucial for most higher level protocols.

6.2.2 Type Field

The type field consists of a two-octet value reserved for use by higher levels (in particular, to identify the Client Layer protocol associated with the frame). The type field is uninterpreted at the Data Link Layer. It is specified at this level because a uniform convention for the placement and value assignment of this field is crucial if multiple higher level protocols are to be able to share the same Ethernet network without conflict. Appendix B discusses the assignment of type field values.

Amazon
Exhibit 1005
Page 107

6.2.3 Data Field

The data field contains a sequence of n octets, where $46 \leq n \leq 1500$ Within this range, full data transparency is provided, in the sense that any arbitrary sequence of octet values may appear in the data field.

6.2.4 Frame Check Sequence Field

The frame check sequence (FCS) field contains a 4-octet (32-bit) cyclic redundancy check (CRC) value. This value is computed as a function of the contents of the source, destination, type and data fields (i.e., all fields except the frame check sequence field itself). The encoding is defined by the generating polynomial:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

(This polynomial is also used in the Autodin-II network; its properties are investigated in [8].)

Mathematically, the CRC value corresponding to a given frame is defined by the following procedure:

1. The first 32 bits of the frame are complemented.
2. The n bits of the frame are then considered to be the coefficients of a polynomial M(x) of degree n-1. (The first bit of the destination address field corresponds to the x^{n-1} term and the last bit of the data field corresponds to the x^0 term.)
3. M(x) is multiplied by x^{32} and divided by G(x), producing a remainder R(x) of degree ≤ 31.
4. The coefficients of R(x) are considered to be a 32-bit sequence.
5. The bit sequence is complemented and the result is the CRC.

The 32 bits of the CRC value are placed in the frame check sequence field so that the x^{31} term is the leftmost bit of the first octet, and the x^0 term is the rightmost bit of the last octet. (The bits of the CRC are thus transmitted in the order x^{31}, x^{30},...,x^1, x^0)

Appendix C discusses CRC implementation issues.

6.2.5 Frame Size Limitations

Given the limitations on the size of the data field specified in 6.2.3 and the 18 octet total size for the other four fields, the smallest valid frame contains 64 octets and the largest valid frame contains 1518 octets.

Amazon
Exhibit 1005
Page 108

6.3 Frame Transmission

The Data Link frame transmission and reception are as follows:

Frame transmission includes data encapsulation and link management aspects:

Transmit Data Encapsulation includes the assembly of the outgoing frame (from the values provided by the Client Layer) and frame check sequence generation.

Transmit Link Management includes carrier deference, interframe spacing, collision detection and enforcement, and collision backoff and retransmission.

The performance of these functions by a transmitting data link controller interacts with corresponding actions by other data link controllers to jointly implement the Ethernet data link protocol.

6.3.1 Transmit Data Encapsulation

6.3.1.1 Frame Assembly

The fields of the data link frame are set to the values provided by the Client Layer as arguments to the *TransmitFrame* operation (see 5.1), with the exception of the frame check sequence, which is set to the CRC value generated by the data link controller.

6.3.1.2 Frame Check Sequence Generation

The CRC value defined in 6.2.4 is generated and inserted in the frame check sequence field, following the fields supplied by the Client Layer. Appendix C discusses CRC implementation.

6.3.2 Transmit Link Management

6.3.2.1 Carrier Deference

Even when it has nothing to transmit, the data link controller monitors the physical channel for traffic by watching the *carrierSense* signal provided by the Physical Layer. Whenever the channel is busy, the data link controller *defers* to the passing frame by delaying any pending transmission of its own. After the last bit of the passing frame (i.e., when *carrierSense* changes from true to false), the data link controller continues to defer for 9.6 μsec to provide proper interframe spacing (see 6.3.2.2). At the end of that time, if it has a frame waiting to be transmitted, transmission is initiated independent of the value of *carrierSense*. When transmission has completed (or immediately, if there was nothing to transmit) the data link controller resumes its original monitoring of *carrierSense*.

When a frame is submitted by the Client Layer for transmission, the transmission is initiated as soon as possible, but in conformance with the rules of deference stated above.

Amazon
Exhibit 1005
Page 109

6.3.2.2 Interframe Spacing

As defined in 6.3.2.1, the rules for deferring to passing frames insure a minimum interframe spacing of 9.6 μsec. This is intended to provide interframe recovery time for other data link controllers and for the physical channel.

Note that 9.6 μsec is the minimum value of the interframe spacing. If necessary for implementation reasons, a transmitting controller may use a larger value with a resulting decrease in its throughput. The value should not exceed 10.6 μsec.

6.3.2.3 Collision Handling

Once a data link controller has finished deferring and has started transmission, it is still possible for it to experience contention for the channel. As discussed in 4.4.3, collisions can occur until acquisition of the network has been accomplished through the deference of all other stations' data link controllers.

The dynamics of collision handling are largely determined by a single parameter called the *slot time*. This single parameter describes three important aspects of collision handling:

- It is an upper bound on the acquisition time of the network.
- It is an upper bound on the length of a frame fragment generated by a collision. (See 6.4.2.1)
- It is the scheduling quantum for retransmission. (See 6.3.2.3.2)

In order to fulfill all three functions, the slot time must be larger than the sum of the Physical Layer round-trip propagation time (450 bit times; see 7.1.2) and the Data Link Layer maximum jam time (48 bit times, see 6.3.2.3.1). The slot time is defined to be 512 bit times.

6.3.2.3.1 Collision Detection and Enforcement

Collisions are detected by monitoring the *collisionDetect* signal provided by the Physical Layer. When a collision is detected during a frame transmission, the transmission is not terminated immediately. Instead, the transmission continues until at least 32 (but not more than 48) additional bits have been transmitted (counting from the time *collisionDetect* went on). This collision enforcement or "jam" guarantees that the duration of the collision is sufficient to insure its detection by all transmitting stations on the network. The content of the jam is unspecified; it may be any fixed or variable pattern convenient to the data link controller implementation, but should not be the 32-bit CRC value corresponding to the (partial) frame transmitted prior to the jam.

6.3.2.3.2 Collision Backoff and Retransmission

When a transmission attempt has terminated due to a collision, it is retried by the transmitting data link controller until either it is successful, or 16 attempts (the original attempt plus 15 retries) have been made and all have terminated due to collisions. Note that all attempts to transmit a given frame are completed before any subsequent outgoing frames are transmitted. The scheduling of the retransmissions is determined by a controlled randomization process called "truncated binary exponential backoff". At the end of enforcing a collision (jamming), the data link controller delays before attempting to retransmit the frame. The delay is an integral multiple of the slot time. (See 6.3.2.3). The number of slot times to delay before the n^{th} retransmission attempt is chosen as a uniformly distributed random integer r in the range $0 \leq r < 2^k$ where $k = \min(n, 10)$. If all 16 attempts fail, this event is reported as an error.

Note that the values given above define the most aggressive behavior that a station may exhibit in attempting to retransmit after a collision. In the course of implementing the retransmission scheduling procedure, a station may introduce extra delays which will degrade its own throughput, but in no case may a station's retransmission scheduling result in a lower average delay between retransmission attempts than the procedure defined above.

6.4 Frame Reception

Frame reception includes both data decapsulation and link management aspects:

Receive Data Decapsulation comprises framing, address recognition, frame check sequence validation, and frame disassembly to pass the fields of the received frame to the Client Layer.

Receive Link Management's main function is the filtering of collision fragments from complete incoming frames.

The performance of these functions by a receiving data link controller interacts with corresponding actions by other data link controllers to jointly implement the Ethernet data link protocol.

6.4.1 Receive Data Decapsulation

6.4.1.1 Framing

The data link controller recognizes the boundaries of an incoming frame by monitoring the *carrierSense* signal provided by the Physical Layer. There are two possible length errors that can occur, which indicate ill-framed data: the frame may be too long, or its length may not be an integral number of octets.

Amazon
Exhibit 1005
Page 111

6.4.1.1.1 Maximum Frame Size

The receiving data link controller is not required to enforce the frame size limit specified in 6.2.5, but it is allowed to truncate frames longer than 1518 octets and report this event as an (implementation-dependent) error.

6.4.1.1.2 Integral Number of Octets in Frame

Since the format of a valid frame specifies an integral number of octets, only a collision or an error can produce a frame with a length that is not an integral multiple of 8. Complete frames (i.e., not rejected as collision fragments; see 6.4.2.1) that do not contain an integral number of octets are truncated to the nearest octet boundary. If frame check sequence validation (see 6.4.1.3) detects an error in such a frame, the status code *alignmentError* is reported.

6.4.1.2 Address Recognition

The Ethernet data link controller is capable of recognizing physical and multicast addresses, as defined in 6.2.1.

6.4.1.2.1 Physical Addresses

The data link controller recognizes and accepts any frame whose destination field contains the physical address of the station.

The physical address of each station is set by network management to a unique value associated with the station, and distinct from the address of any other station on any Ethernet. The setting of the station's physical address by network management allows multiple data link controllers connected to single station all to respond to the same physical address. The procedures for allocating unique addresses are discussed in Appendix B.

6.4.1.2.2 Multicast Addresses

The data link controller recognizes and accepts any frame whose destination field contains the broadcast address.

The data link controller is capable of activating some number of multicast-group addresses as specified by higher layers. The data link controller recognizes and accepts any frame whose destination field contains an active multicast-group address. An active multicast-group address may be deactivated.

6.4.1.3 Frame Check Sequence Validation

FCS validation is essentially identical to FCS generation. If the bits of the incoming frame (exclusive of the FCS field itself) do not generate a CRC value identical to the one received, an error has occurred and is reported as such. Implementation issues are discussed in Appendix C.

6.4.1.4 Frame Disassembly

The frame is disassembled and the fields are passed to the Client Layer via the output parameters of the *ReceiveFrame* operation (see 5.1).

6.4.2 Receive Link Management

6.4.2.1 Collision Filtering

As specified in 6.2.5, the smallest valid frame must contain at least 64 octets. Any frame containing less than 64 octets is presumed to be a fragment resulting from a collision and is discarded by the receiving data link controller. Since occasional collisions are a normal part of the link management procedure, the discarding of such a fragment is not reported as an error to the Client Layer.

6.5 The Data Link Layer Procedural Model

6.5.1 Overview of the Procedural Model

The functions of the Ethernet Data Link Layer are presented below, modeled as a program written in the language Pascal [6]. This procedural model is intended as the primary specification of the functions to be provided in any Ethernet Data Link Layer implementation. It is important to distinguish, however, between the model and a real implementation. The model is optimized for simplicity and clarity of presentation, while any realistic implementation must place heavier emphasis on such constraints as efficiency and suitability to a particular implementation technology or computer architecture. In this context, several important properties of the procedural model must be considered.

6.5.1.1 Ground Rules for the Procedural Model

a) First, it must be emphasized that the description of the Data Link Layer in a programming language is in no way intended to imply that a data link controller must be implemented as a program executed by a computer. The implementation may consist of any appropriate technology including hardware, firmware, software, or any combination.

b) Similarly, it must be emphasized that it is the *behavior* of Data Link Layer implementations that must match the specification, *not* their internal structure. The internal details of the procedural model are useful only to the extent that they help specify that behavior clearly and precisely.

Amazon
Exhibit 1005
Page 113

c) The handling of incoming and outgoing frames is rather stylized in the procedural model, in the sense that frames are handled as single entities by most of the Data Link Layer and are only serialized for presentation to the Physical Layer. In reality, many data link controller implementations will instead handle frames serially on a bit, octet or word basis. A serial implementation would typically perform the required functions (address recognition, frame check sequence generation/validation, etc.) in an overlapped, pipelined fashion. This approach has not been reflected in the procedural model, since this would only complicate the description of the functions without changing them in any way.

d) The model consists of algorithms designed to be executed by a number of concurrent processes; these algorithms collectively implement the Ethernet data link control procedure. The timing dependencies introduced by the need for concurrent activity are resolved in two ways:

- *Processes vs. External events:* It is assumed that the algorithms are executed "very fast" relative to external events, in the sense that a process never falls behind in its work and fails to respond to an external event in a timely manner. For example, when a frame is to be received, it is assumed that the data link procedure *ReceiveFrame* is always called well before the frame in question has started to arrive.

- *Processes vs. Processes:* Among processes, no assumptions are made about relative speeds of execution. This means that each interaction between two processes must be structured to work correctly independent of their respective speeds. Note, however, that the timing of interactions among processes is often, in part, an indirect reflection of the timing of external events, in which case appropriate timing assumptions may still be made.

It is intended that the concurrency in the model reflect the parallelism intrinsic to the task of implementing the Ethernet data link, although the actual parallel structure of the implementations is likely to vary.

Amazon
Exhibit 1005
Page 114

6.5.1.2 Use of Pascal in the Procedural Model

Pascal was chosen for the procedural model because of its relative simplicity and clarity, and its general acceptance.

Several observations need to be made about the way in which Pascal is used for the model, including:

a) Some limitations of the language have been circumvented in order to simplify the specification:

1) The elements of the program (variables, procedures, etc) are presented in logical groupings, in top-down order. Certain Pascal ordering restrictions have thus been circumvented to improve readability.

2) The **process** and **cycle** constructs of the Pascal derivative Concurrent Pascal [7] have been introduced to indicate the sites of autonomous concurrent activity. As used here, a process is simply a parameterless procedure that begins execution at "the beginning of time" rather than being invoked by a procedure call. A cycle statement represents the main body of a process and is executed repeatedly forever.

3) The lack of variable array bounds in the language has been circumvented by treating frames as if they are always of a single fixed size (which is never actually specified). In fact, of course, the size of a frame depends on the size of its data field, hence the value of the "pseudo-constant" *frameSize* should be thought of as varying in the long-term, even though it is fixed for any given frame.

4) The use of a variant record to represent a frame (both as fields and as bits) follows the letter but not the spirit of the Pascal Report, since it allows the underlying representation to be viewed as two different data types. (It also assumes that this representation is as shown in Figure 6-1.)

b) The model makes no use of any explicit interprocess synchronization primitives . Instead, all interprocess interaction is done via carefully stylized manipulation of shared variables. For example, some variables are set by only one process and inspected by another process in such a manner that the net result is independent of their execution speeds. While such techniques are not generally suitable for the construction of large concurrent programs, they simplify the model and more nearly resemble the methods appropriate to the most likely implementation technologies (e.g. microcode, hardware state-machines, etc.)

Amazon
Exhibit 1005
Page 115

6.5.2 Procedural Model

The procedural model used here is based on five cooperating concurrent processes. Of these, three are actually defined in the Data Link Layer. The remaining two processes are provided by the Client Layer and utilize the interface operations provided by the Data Link Layer. The five processes are thus:

Client Layer:

Frame Transmitter Process **Frame Receiver Process**

Data Link Layer:

Bit Transmitter Process **Bit Receiver Process**

Deference Process

This organization of the model is illustrated in Figure 6-2, and reflects the fact that the communication of entire frames is initiated by the Client Layer, while the timing of collision backoff and of individual bit transfers is based on interactions between the Data Link Layer and the Physical-Layer-dependent bit-time.

Figure 6-2 depicts the static structure of the procedural model, showing how the various processes and procedures interact by invoking each other. Figures 6-3 and 6-4 summarize the dynamic behavior of the model during transmission and reception, focusing on the steps that must be performed, rather than the procedural structure which performs them. The usage of the shared state variables is not depicted in the figures, but is described in the comments in 6.5.2.1.

Amazon
Exhibit 1005
Page 116

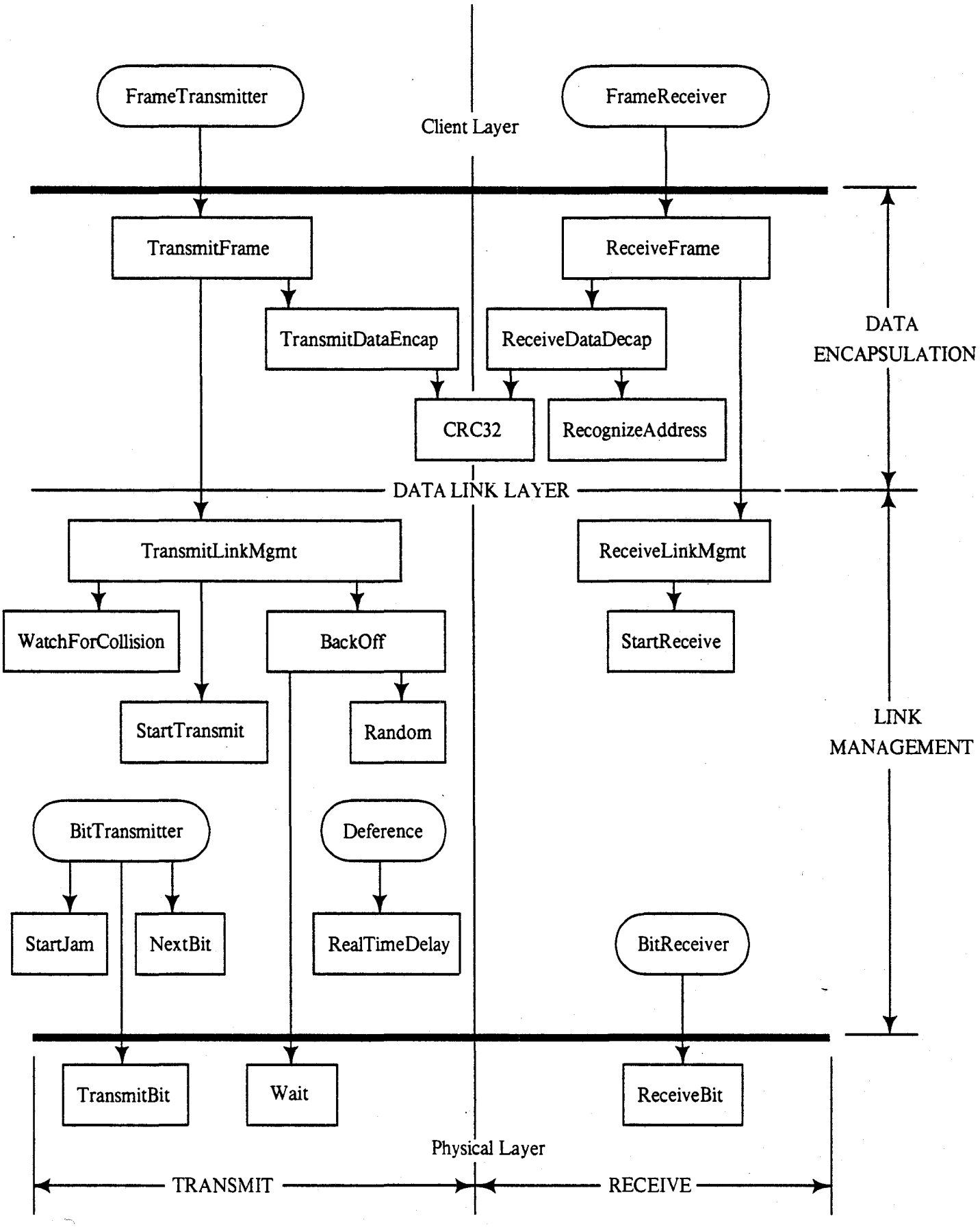


Figure 6-2: Structure of the Data Link Procedural Model

Amazon
Exhibit 1005
Page 117

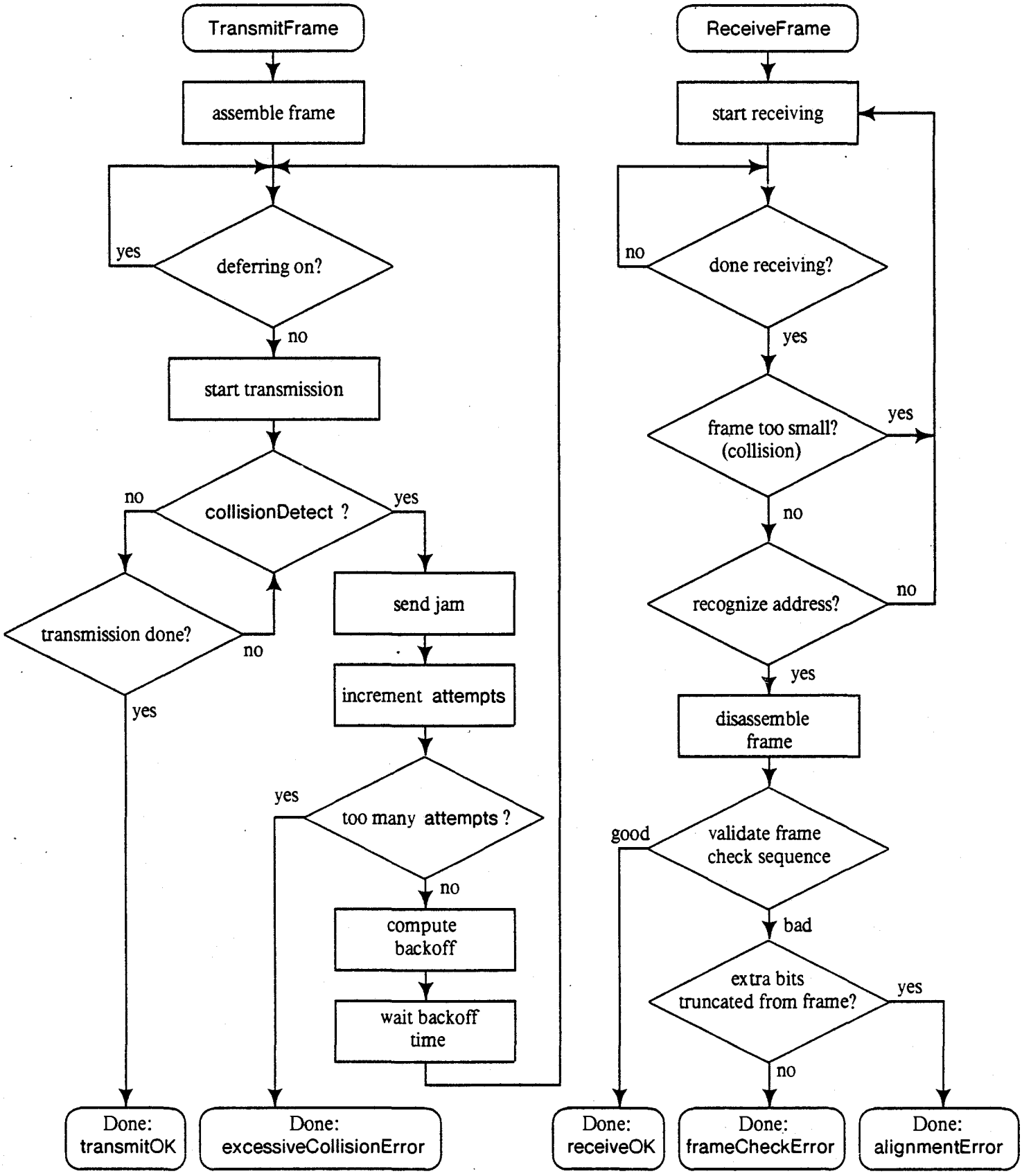
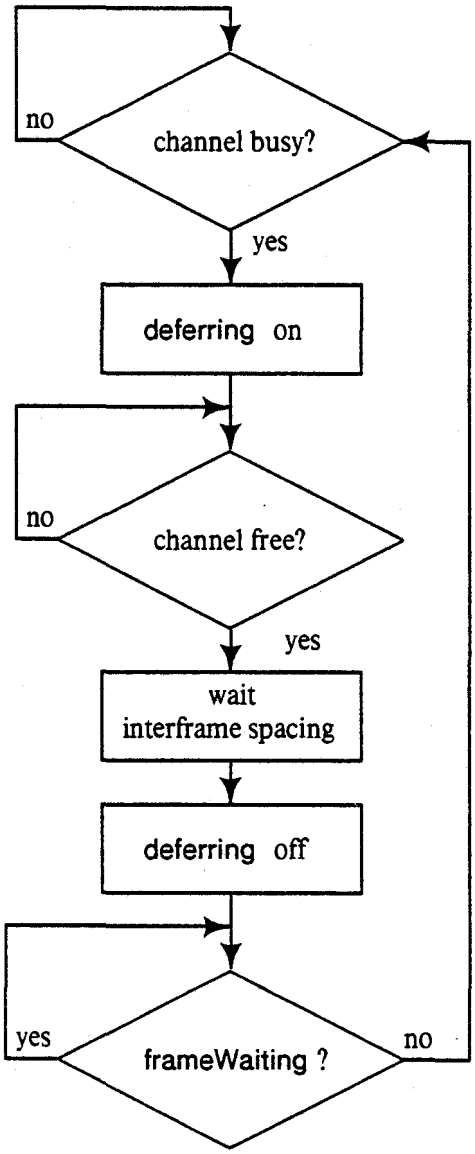
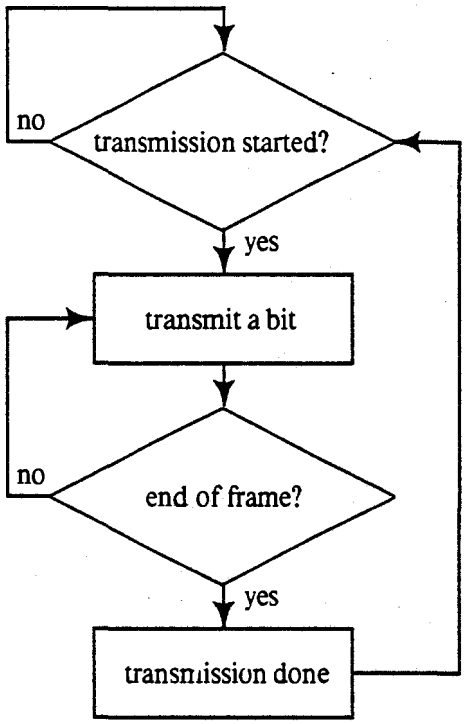


Figure 6-3: Control Flow Summary -- Client Layer Processes

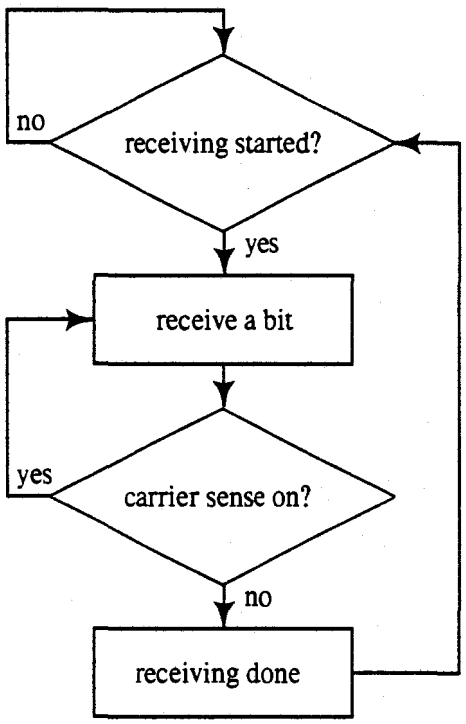
Amazon
Exhibit 1005
Page 118



Deference process



Bit Transmitter process



Bit Receiver process

Figure 6-4: Control Flow Summary -- Data Link Layer Processes

Amazon
Exhibit 1005
Page 119

6.5.2.1 Global Declarations

6.5.2.1.1 Common Constants and Types

The following declarations of constants and types are used by the frame transmission and reception sections of each data link controller:

```
const
    addressSize = 48; {48 bit address = 6 octets}
    typeSize = 16; {16 bit protocol type = 2 octets}
    dataSize = ...; {see 6.5.1.2, note 3}
    crcSize = 32; {32 bit CRC = 4 octets}
    frameSize = ...; { = 2*addressSize + typeSize + dataSize + crcSize...see 6.5.1.2,
    note 3}

    slotTime = 512; {unit of time for collision handling}

type
    Bit = 0..1;
    AddressValue = array [1..addressSize] of Bit;
    TypeValue = array [1..typeSize] of Bit;
    DataValue = array [1..dataSize] of Bit;
    CRCValue = array [1..crcSize] of Bit;

    ViewPoint = (fields, bits); {Two ways to view the contents of a frame}

    Frame = record {Format of data link frame}
      case view: ViewPoint of
        fields: (
            destinationField: AddressValue;
            sourceField: AddressValue;
            typeField: TypeValue;
            dataField: DataValue;
            fcsField: CRCValue);
        bits: (
            contents: array [1..frameSize] of Bit)
      end; {Frame}
```

6.5.2.1.2 Transmit State Variables

The following items are specific to frame transmission. (See also 6.5.2.1.4 on interfaces)

```
const
    interFrameSpacing = 9.6; {minimum time between frames, in microseconds}
    attemptLimit = 16; {Max number of times attempt transmission}
    backOffLimit = 10; {Limit on number of times to back off}
    jamSize = 32; {jam may be 32 to 48 bits long}
var
    outgoingFrame: Frame; {The frame to be transmitted}
    currentTransmitBit, lastTransmitBit: 1..frameSize; {Positions of current and
      last outgoing bits in outgoingFrame}
    deferring: Boolean; {True implies any pending transmission must wait for the
      channel to clear}
    frameWaiting: Boolean; {Indicates that outgoingFrame is deferring}
    attempts: 0..attemptLimit; {Number of transmission attempts on outgoingFrame}
    newCollision: Boolean; {Indicates that a collision has occurred but has not yet
      been jammed}
    transmitSucceeding: Boolean; {Running indicator of whether transmission is
      succeeding}
```

6.5.2.1.3 Receive State Variables

The following items are specific to frame reception. (See also 6.5.2.1.4 on interfaces)

```
var
    incomingFrame: Frame; {The frame being received}
    currentReceiveBit: 1..frameSize; {Position of current bit in incomingFrame}
    receiving: Boolean; {Indicates that a frame reception is in progress}
    excessBits: 0..7; {Count of excess trailing bits beyond octet boundary}
    receiveSucceeding: Boolean; {Running indicator of whether reception is
      succeeding}
```

6.5.2.1.4 Summary of Interlayer Interfaces

The interface to the Client Layer, defined in 5.1, is summarized below:

```
type
  TransmitStatus = (transmitOK, excessiveCollisionError); {Result of
    TransmitFrame operation}
  ReceiveStatus = (receiveOK, frameCheckError, alignmentError); {Result
    of ReceiveFrame operation}

function TransmitFrame (
  destinationParam: AddressValue;
  sourceParam: AddressValue;
  typeParam: typeValue;
  dataParam: DataValue): TransmitStatus; {Transmits one frame}

function ReceiveFrame (
  var destinationParam: AddressValue;
  var sourceParam: AddressValue;
  var typeParam: TypeValue;
  var dataParam: DataValue): ReceiveStatus; {Receives one frame}
```

The interface to the Physical Layer, defined in 5.2, is summarized below:

```
var
    carrierSense: Boolean; {Indicates incoming bits}
    transmitting: Boolean; {Indicates outgoing bits}
    collisionDetect: Boolean; {Indicates channel contention}
procedure TransmitBit (bitParam: Bit); {Transmits one bit}
function ReceiveBit: Bit; {Receives one bit}
procedure Wait (bitTimes: integer); {Waits for indicated number of bit-times}
```

Amazon
Exhibit 1005
Page 122

6.5.2.1.5 State Variable Initialization

The procedure *Initialize* must be run when the Data Link Layer begins operation, before any of the processes begin execution. *Initialize* sets certain crucial shared state variables to their initial values. (All other global variables are appropriately reinitialized before each use.) *Initialize* then waits for the channel to be idle, and starts operation of the various processes.

```
procedure Initialize;
begin
  frameWaiting := false;
  deferring := false;
  newCollision := false;
  transmitting := false; {In interface to Physical Layer; see below}
  receiving := false;
  while carrierSense do nothing;
  {Start execution of all processes}
end; {Initialize}
```

6.5.2.2 Frame Transmission

The algorithms in this section define data link frame transmission.

The function *TransmitFrame* implements the frame transmission operation provided to the Client Layer:

```
function TransmitFrame (
    destinationParam: AddressValue;
    sourceParam: AddressValue;
    typeParam: typeValue;
    dataParam: DataValue): TransmitStatus;
  procedure TransmitDataEncap; ... {nested procedure; see body below}
begin
  TransmitDataEncap;
  TransmitFrame : = TransmitLinkMgmt
end; {TransmitFrame}
```

First, *TransmitFrame* calls the internal procedure *TransmitDataEncap* to construct the frame. It then calls *TransmitLinkMgmt* to perform the actual transmission. The *TransmitStatus* returned indicates the success or failure of the transmission attempt.

TransmitDataEncap builds the frame and places the 32-bit CRC in the frame check sequence field:

```
procedure TransmitDataEncap;
begin
  with outgoingFrame do
  begin {assemble frame}
    view : = fields;
    destinationField : = destinationParam;
    sourceField : = sourceParam;
    typeField : = typeParam;
    dataField : = dataParam;
    fcsField : = CRC32(outgoingFrame);
    view : = bits
  end {assemble frame}
end; {TransmitDataEncap}
```

TransmitLinkMgmt attempts to transmit the frame, deferring first to any passing traffic. If a collision occurs, transmission is terminated properly and retransmission is scheduled following a suitable backoff interval:

```
function TransmitLinkMgmt: TransmitStatus;
begin
  attempts := 0; transmitSucceeding := false;
  while attempts < attemptLimit and not transmitSucceeding do
  begin {loop}
    if attempts > 0 then BackOff;
    frameWaiting := true;
    while deferring do nothing;  {defer to passing frame, if any}
    frameWaiting := false;
    StartTransmit;
    while transmitting do WatchForCollision;
    attempts := attempts + 1
  end; {loop}
  if transmitSucceeding then TransmitLinkMgmt := transmitOK
  else TransmitLinkMgmt := excessiveCollisionError
end; {TransmitLinkMgmt}
```

Each time a frame transmission attempt is initiated, *StartTransmit* is called to alert the *BitTransmitter* process that bit transmission should begin:

```
procedure StartTransmit;
begin
  currentTransmitBit := 1;
  lastTransmitBit := frameSize;
  transmitSucceeding := true;
  transmitting := true
end; {StartTransmit}
```

Once frame transmission has been initiated, *TransmitLinkMgmt* monitors the channel for contention by repeatedly calling *WatchForCollision:*

```
procedure WatchForCollision;
begin
  if transmitSucceeding and collisionDetect then
  begin
    newCollision := true;
    transmitSucceeding := false
  end
end; {WatchForCollision}
```

WatchForCollision, upon detecting a collision, updates *newCollision* to insure proper jamming by the *BitTransmitter* process.

Amazon
Exhibit 1005
Page 125

After transmission of the jam has completed, if *TransmitLinkMgmt* determines that another attempt should be made, *BackOff* is called to schedule the next attempt to retransmit the frame.

```
var maxBackOff: 2..1024; {Working variable of BackOff}

procedure BackOff;
begin
  if attempts = 1 then maxBackOff := 2 else if attempts ≤ backOffLimit
    then maxBackOff := maxBackOff*2;
  Wait(slotTime*Random(0, maxBackOff))
end; {BackOff}

function Random (low, high: integer): integer;
begin
  Random := ...{uniformly distributed random integer r such that low ≤ r <
  high}
end; {Random}
```

BackOff performs the truncated binary exponential backoff computation and then waits for the selected multiple of the slot time.

The *Deference* process runs asynchronously to continuously compute the proper value for the variable *deferring*.

```
process Deference;
begin
  cycle {main loop}
    while not carrierSense do nothing; {watch for carrier to appear}
    deferring := true; {delay start of new transmissions}
    while carrierSense do nothing; {wait for carrier to disappear}
    RealTimeDelay(interFrameSpacing);
    deferring := false; {allow new transmissions to proceed}
    while frameWaiting do nothing {allow waiting transmission (if any)}
  end {main loop}
end; {Deference}

procedure RealTimeDelay (usec: real);
begin
  {Wait for the specified number of microseconds}
end; {RealTimeDelay}
```

The *BitTransmitter* process runs asynchronously, transmitting bits at a rate determined by the Physical Layer's *TransmitBit* operation:

```
process BitTransmitter;
begin
  cycle {outer loop}
    while transmitting do
    begin {inner loop}
      TransmitBit(outgoingFrame[currentTransmitBit]); {send next bit to
      Physical Layer}
      if newCollision then StartJam else NextBit
    end {inner loop}
  end {outer loop}
end; {BitTransmitter}

procedure NextBit;
begin
  currentTransmitBit := currentTransmitBit + 1;
  transmitting := (currentTransmitBit ≤ lastTransmitBit)
end; {NextBit}

procedure StartJam;
begin
  currentTransmitBit := 1;
  lastTransmitBit := jamSize;
  newCollision := false
end; {StartJam}
```

BitTransmitter, upon detecting a new collision, immediately enforces it by calling *StartJam* to initiate the transmission of the jam. The jam may contain 32 to 48 bits of arbitrary data. (*StartJam* uses the first 32 bits of the frame, merely to simplify this program).

Amazon
Exhibit 1005
Page 127

6.5.2.3 Frame Reception

The algorithms in this section define data link frame reception:

The procedure *ReceiveFrame* implements the frame reception operation provided to the Client Layer:

```
function ReceiveFrame (
    var destinationParam: AddressValue;
    var sourceParam: AddressValue;
    var typeParam: TypeValue;
    var dataParam: DataValue): ReceiveStatus;
  function ReceiveDataDecap: ReceiveStatus; ... {nested function; see body
  below}
begin
  repeat
    ReceiveLinkMgmt;
    ReceiveFrame : = ReceiveDataDecap;
  until receiveSucceeding
end; {ReceiveFrame}
```

ReceiveFrame calls *ReceiveLinkMgmt* to receive the next valid frame, and then calls the internal procedure *ReceiveDataDecap* to return the frame's fields to the Client Layer if the frame's address indicates that it should do so. The returned *ReceiveStatus* indicates the presence or absence of detected transmission errors in the frame.

```
function ReceiveDataDecap: ReceiveStatus;
begin
  receiveSucceeding : = RecognizeAddress
    (incomingFrame.destinationField);
  if receiveSucceeding then with incomingFrame do
  begin {disassemble frame}
    view : = fields;
    destinationParam : = destinationField;
    sourceParam : = sourceField;
    typeParam : = typeField;
    dataParam : = dataField;
    if fcsField = CRC32(incomingFrame) then ReceiveDataDecap : = receiveOK
    else if excessBits = 0 then ReceiveDataDecap : = frameCheckError
    else ReceiveDataDecap : = alignmentError;
    view : = bits
  end {disassemble frame}
end; {ReceiveDataDecap}
```

```
function RecognizeAddress (address: AddressValue): Boolean;
begin
  RecognizeAddress := ... {Returns true for the set of physical, broadcast, and
      multicast-group addresses corresponding to this station}
end; {RecognizeAddress}
```

ReceiveLinkMgmt attempts repeatedly to receive the bits of a frame, discarding any fragments from collisions by comparing them to the minimum valid frame size:

```
procedure ReceiveLinkMgmt;
begin
  repeat
    StartReceive;
    while receiving do nothing; {wait for frame to finish arriving}
    excessBits := frameSize mod 8;
    frameSize := frameSize - excessBits; {truncate to octet boundary}
    receiveSucceeding := (frameSize ≥ slotTime); {reject collision fragments}
  until receiveSucceeding
end; {ReceiveLinkMgmt}

procedure StartReceive;
begin
  currentReceiveBit := 1;
  receiving := true
end; {StartReceive}
```

The *BitReceiver* process run asynchronously, receiving bits from the channel at the rate determined by the Physical Layer's *ReceiveBit* operation:

```
process BitReceiver;
  var b: Bit;
begin
  cycle {outer loop}
    while receiving do
    begin {inner loop}
      b := ReceiveBit; {Get next bit from physical link}
      if carrierSense then
      begin{append bit to packet}
        incomingFrame[currentReceiveBit] := b;
        currentReceiveBit := currentReceiveBit + 1
      end; {append bit to packet}
      receiving := carrierSense
    end {inner loop}
  end {outer loop}
end; {BitReceiver}
```

6.5.2.4 Common procedures

The function *CRC32* is used by both the transmit and receive algorithms to generate a 32 bit CRC value:

```
function CRC32 (f: Frame): CRCValue;
begin
  CRC32 := {The 32-bit CRC as defined in 6.2.4}
end; {CRC32}
```

Purely to enhance readability, the following procedure is also defined:

```
procedure nothing; begin end;
```

The idle state of a process (i.e., while waiting for some event) is cast as repeated calls on this procedure.

7. ETHERNET PHYSICAL LAYER SPECIFICATION: Baseband Coaxial System

7.1 Physical Channel Overview and Model

The Ethernet physical channel (henceforth referred to as the channel) provides the lowest layer in the Ethernet architecture. It performs all the functions needed to transmit and receive data at the physical level, while supporting the Data Link to Physical Layer Interface described in 5.2.

This section describes the requirements for interface and compatibility with a baseband coaxial implementation of the channel.

7.1.1 Channel Goals and Non-goals

This section states the objectives underlying the design of the channel.

7.1.1.1 Goals

The following are the goals of the channel:

1. Provide a means for communication between Ethernet Data Link Entities.
2. Define physical interfaces which can be implemented compatibly among different manufacturers of hardware.
3. Provide all clocks, synchronization, and timing required for both itself and the Ethernet Data Link.
4. Provide high bandwidth and low bit error rates.
5. Provide for ease of installability and serviceability.
6. Provide for high network availability.
7. Support the Ethernet Data Link to Physical Link interface.
8. Low cost.

7.1.1.2 Non-Goals

The following are not goals of the baseband coaxial channel design:

1. Operation at data rates other than 10 megabits per second.
2. Operation with media other than the specified coaxial cable.
3. Simultaneous use of the channel by transmitters using signals not specified in this document.
4. Protection against a malicious user or a malfunctioning Data Link Entity is not provided by the channel as specified. However, higher layers (above the Data Link) and/or physical security means may be employed to acheive this.

Amazon
Exhibit 1005
Page 131

7.1.2 Characteristics of the Channel

The channel provides (and the data link assumes) the following characteristics:

1. The ability to send and receive information (non-simultaneously) between any two or more data link entities on the same network.
2. The ability to detect the presence of another station's transmission while not transmitting (carrier sense).
3. The ability to detect the presence of another station's transmission while transmitting (collision detect).
4. A total worst-case round trip signal propagation delay (including actual propagation time, synchronization time for all intervening electronics, and signal rise time degradation) of 450 bit times (equal to 45 μs for this 10 Mbit channel).

7.1.3 Functions Provided by the Channel

The channel hardware provides the following functions in the performance of its role:

1. Means for transmitting and receiving serial bit streams between the data link layer and the media.
2. Generation of clock for synchronization and timing.
3. Means for detecting carrier (non-idle channel).
4. Means for detecting collisions (simultaneous transmission attempts by multiple stations).
5. Coding and decoding of the data link bit stream into a self-synchronizable sequence of electrical signals suitable for transmission on the media provided by the channel.
6. Generation and removal of coding-specific preamble information (a synchronizing header sequence inserted before the first bit of the frame) to ensure that all channel electronics are brought to a known steady-state before the data link frame is transmitted.

7.1.4 Implementation of the Channel

The physical channel specification is implementation dependent; most of the channel hardware is fully specified, and little leeway is given to the individual designer. This is done in the interest of compatibility; any system which allows different implementors to use different channel cables, connectors, clock speeds and the like will not be compatible across manufacturer boundaries. Only the design of channel components which are not critical to system compatibility is left to the implementor.

Amazon
Exhibit 1005
Page 132

7.1.4.1 General Overview of Channel Hardware

The channel minimally consists of the following functional blocks:

1. The passive broadcast medium (coaxial cable),
2. The transceiver (transmitter-receiver for the coaxial cable),
3. The means for connecting transceivers to a coaxial cable segment and for connecting coaxial cable segments together,
4. The channel clock,
5. The channel data encoder and decoder,
6. The preamble generator and remover,
7. The carrier and collision detect circuits.

The coaxial medium is the only element common to the entire network. A transceiver is required for each station connected to the medium. The transceiver must be located adjacent to the coaxial cable. The latter four components are generally located within, and tightly coupled to, the station hardware implementing the data link function.

It may be useful to be able to physically separate the transceiver from the rest of the channel hardware. This allows topological flexibility, packaging advantages, and improved system availability, as well as allowing for independent manufacture of station hardware and transceivers. To ensure that compatibility is maintained, a physical interface (known as the *transceiver cable*) is identified and specified to connect the transceiver to the station.

Finally, it may be necessary to add *repeaters* to the system, to reach the maximum allowable distance between stations, and to provide additional topological flexibility. Repeaters are implemented using standard transceivers, plus a simple, non-buffered finite state machine.

Amazon
Exhibit 1005
Page 133

7.1.4.2 Compatibility Interfaces

There are a number of possibilities for implementing systems or subsystems compatible in whole or in part with this specification. It is important that all implementations be compatible at some point, so that heterogenous systems from different manufacturers' implementations can be interconnected on the same medium. It is not necessary in every case to implement all of the components described herein; e.g., it is possible to design an integrated station/transceiver (without requiring the transceiver cable). The implementor must make the required trade-offs between topological flexibility, system availability, configurability, user needs, and cost when designing the system.

For a device to be considered compatible, it must meet the applicable requirements at either the transceiver cable or the coaxial cable interface, as appropriate, in addition to the Data Link compatibility required for all stations connected to the network.

All Ethernets must be compatible at the coaxial cable.

If a transceiver cable is used, it should be the one specified in this document. This allows device manufacturers to build hardware compatible with the Ethernet at the transceiver cable level, without concerning themselves with the details of transceiver implementation. Devices implementing transceiver cable compatibility should be capable of using transceivers designed and built by another manufacturer, on the specified coaxial cable.

Equipment designed for connection to the specified coaxial cable either without a physically separate transceiver or with a non-standard transceiver cable interface will be capable of communication. However, a sacrifice may have been made with respect to interchangeability with other stations.

This scheme of multiple compatibility interfaces allows individual designers some flexibility in making system tradeoffs, yet allows cable manufacturers, transceiver manufacturers and systems manufacturers to use standard commodity parts to produce a compatible communications system.

Amazon
Exhibit 1005
Page 134

7.1.5 Channel Configuration Model

Certain physical limits have been placed on the physical channel. These revolve mostly around maximum cable lengths (or maximum propagation times), as these affect the slot time as defined in the data link. While the precise specification (in later sections) specify these maxima in terms of propagation times, they were derived from the physical configuration model described here.

The maximum configuration is as follows:

1. A coaxial cable, terminated in its characteristic impedance at each end, constitutes a cable *segment*. A segment may contain a maximum of 500 meters of coaxial cable.
2. A maximum of 2 repeaters in the path between any two stations. Repeaters do not have to be located at the ends of segments, nor is the user limited to one repeater per segment. In fact, repeaters can be used not only to extend the length of the channel, but to extend the topology from one to three-dimensional. Repeaters occupy transceiver positions on each cable segment and count towards the maximum number of transceivers on a segment just as do the logically distinguishable stations.
3. A maximum total coaxial cable length along the longest path between any two transceivers of 1500 meters. The propagation velocity of the coaxial cable is assumed to be 0.77 c worst-case. (c is the velocity of light *in vacuo*; 300,000 kilometers per second.) The total round-trip delay for all the coaxial cable in the system is therefore 13 μs worst-case.
4. A maximum of 50 meters of transceiver cable between any station and its associated transceiver. Note that in the worst case the signal must pass through six 50 meter transceiver cables, one at the transmitting station, one at the receiving station, and 2 at each repeater (two repeaters possible). The propagation velocity of the transceiver cable is assumed to be .65 c worst-case. The total round-trip delay for all the transceiver cables is therefore 3.08 μs worst-case.
5. A maximum of 1000 meters of point-to-point link anywhere in the system. This will typically be used as a way of linking cable segments in different buildings. Note that a repeater with this internal point-to-point link can be used to repeat signals between segments many hundreds of meters apart. The worst-case propagation velocity of the link cable is assumed to be .65 c; the round-trip propagation delay for 1000 meters is 10.26 μs.

Table 7-1 summarizes the allocation of the round-trip propagation delay to the individual components in the channel. Figure 7-1 shows a minimum, typical, and large-scale channel configuration.

Amazon
Exhibit 1005
Page 135

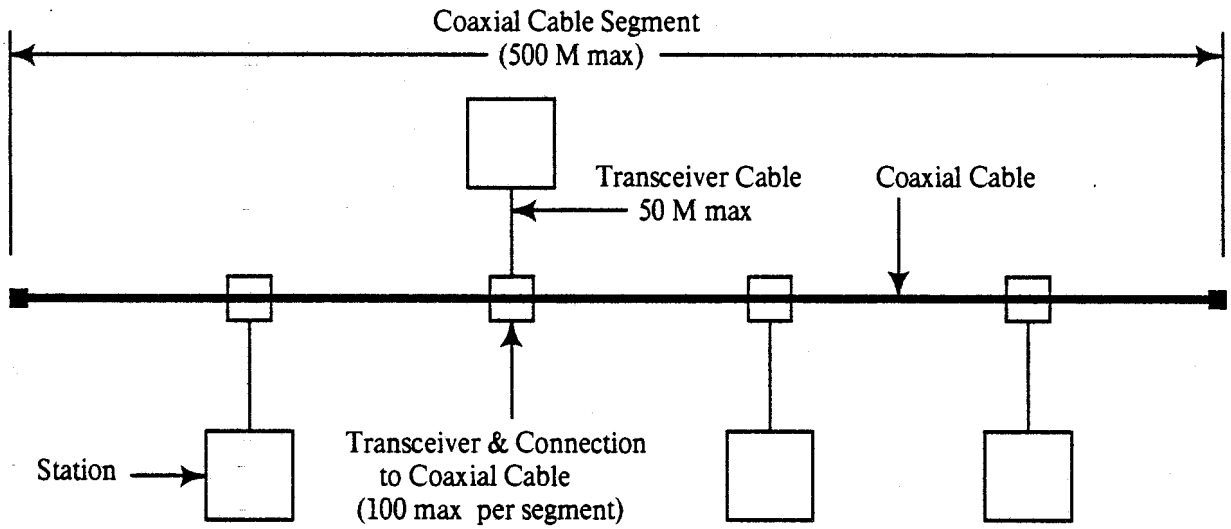


Figure 7-1a: Minimal Configuration

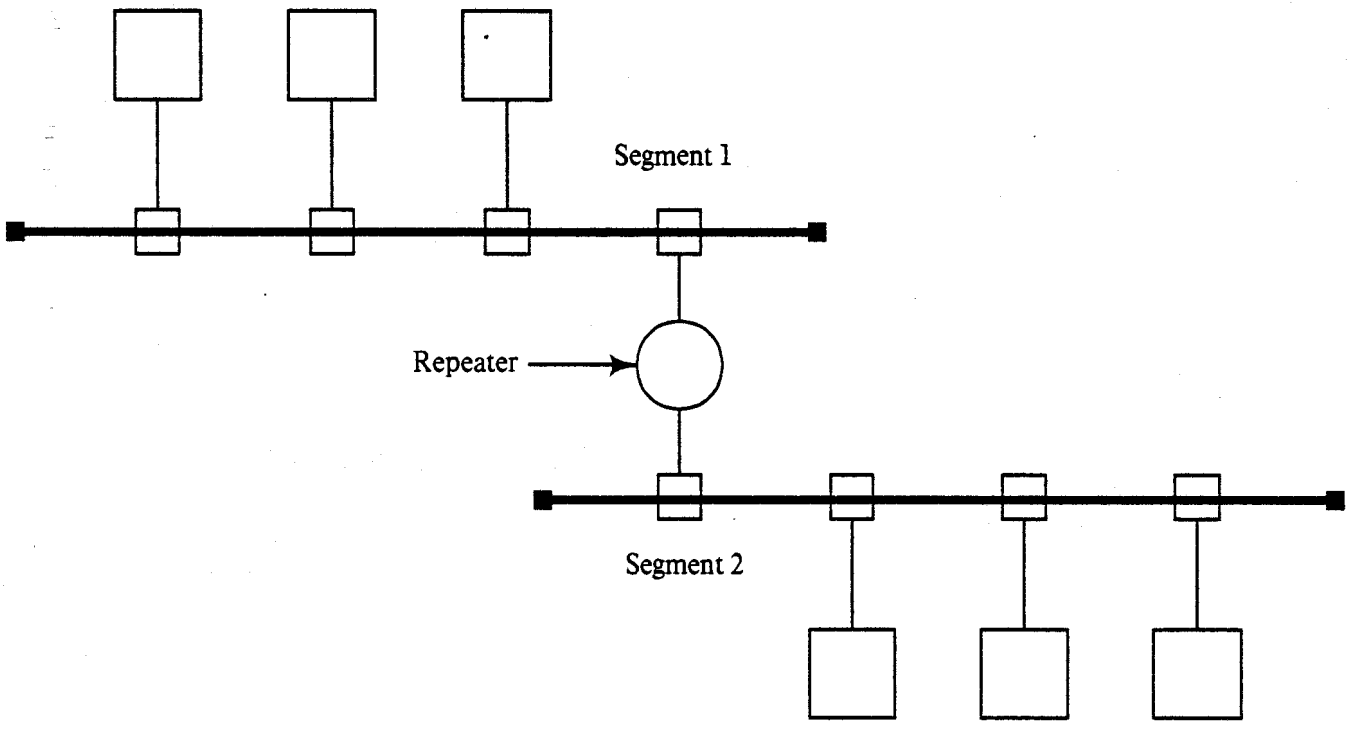


Figure 7-1b: A Typical Medium-scale Configuration

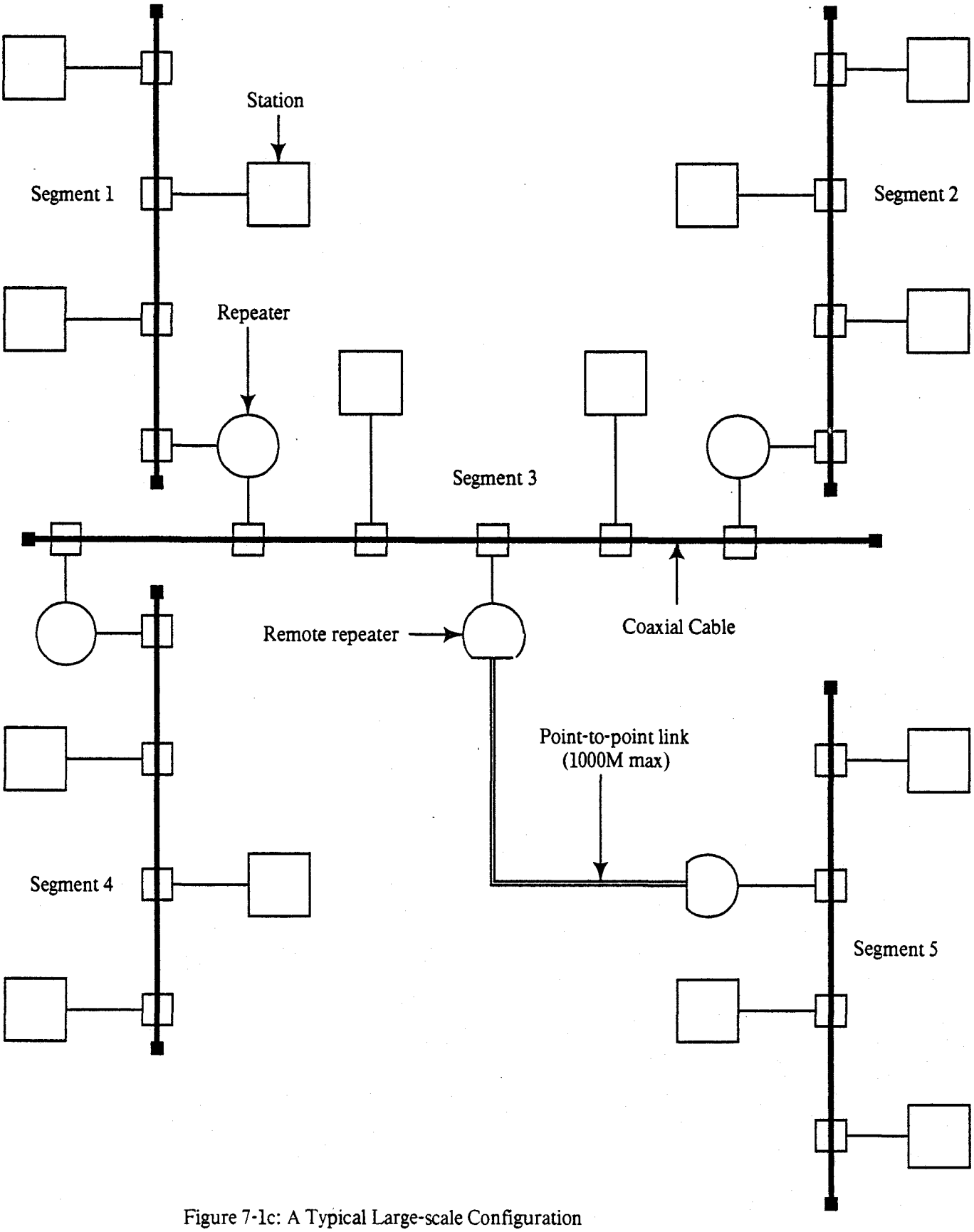


Figure 7-1c: A Typical Large-scale Configuration

Amazon
Exhibit 1005
Page 137

Table 7-1: Physical Channel Propagation Delay Budget (Note 1)

Element	Unit Steady-State Delay	Unit Startup Delay	# Units Forward Path (Note 2)	# Units Return Path	Total Delay
Encoder	0.1 μS	0	3	3	0.60 μS
Transceiver Cable	5.13 nS/M	0	300 M	300 M	3.08 μS
Transceiver (transmit path)	0.50 μS	0.2 μS	3	3	1.50 μS
Transceiver (receive path)	0.50 μS	0.5 μS	3	0	1.65 μS
Transceiver (collision path)	0	0.5 μS	0	3	1.50 μS
Coaxial Cable	4.33 nS/M	0	1500 M	1500 M	13.00 μS
Point-to-Point Link Cable	5.13 nS/M	0	1000 M	1000 M	10.26 μS
Repeater (repeat path)	0.8 μS	0	2	0	1.60 μS
Repeater (collision path)	0.2 μS	0	0	2	0.40 μS
Decoder	0.1 μS	0.8 μS	2	0	1.80 μS
Carrier Sense	0	0.2 μS	3	0	0.60 μS
Collision Detect	0	0.2 μS	0	3	0.60 μS
Signal Rise Time (to 70% in 500 M) (Note 3)	0	0.1 μS	3	0	0.30 μS
Signal Rise Time (50% to 94% in 500 M) (Note 4)	0	2.7 μS	0	3	8.10 μS
		Total Worst-Case Round-Trip Delay			44.99 μS

Note 1: All quantities given are worst-case (both number of units and unit delays per unit).

Note 2: The propagation delay has been separated into "forward-path" and "return path" delay. This is because in one direction it is carrier sense which is being propagated through the channel, and in the return direction it is collision detect which is being propagated. The two signals have different propagation delays.

Note 3: In the worst-case, the propagated signal must reach 70% of its final value to be detected as valid carrier at the end of 500 meters of coaxial cable. This rise time must be included in the propagation delay budget.

Note 4: In the worst-case the propagated collision on the return path must reach 94% of its final value to be detected as a collision at the end of 500 meters of coaxial cable.

Amazon
Exhibit 1005
Page 138

7.1.6 Channel Interfaces

The channel specification hinges around three well-defined entities; the transceiver and coaxial cables (shown as compatibility interfaces in Figure 4-1), and the logical interface between the physical channel and the data link controller (shown in Figure 4-4). Note that the former two are physical interfaces specific to the channel, and are specified in the interest of compatibility. The latter is provided as a means by which the data link controller can interact with the physical channel.

The channel access component of the logical interface (discussed in 4.4.1) comprises the collision and carrier detect functions described in 7.5.2 and 7.5.3, as well as the actual transmission of signals on the media. The data encoding and decoding functions described in 4.4.1 comprise the generation and decomposition of encoded signals suitable for transmission (described in 7.5.1), the generation and removal of code-specific preamble (described in 7.5.1.3 and 7.5.4.1), and the serial bit stream interface between the layers.

Section 5 describes the interface between the data link and physical layers as a series of Pascal procedures, functions, and shared variables. The data link specification in section 6 shows how the data link uses this interface to communicate between client layers. However, this specification will not attempt to model the operation of the physical channel in Pascal. The interface between layers is supported by the physical hardware which provides the ability to send and receive bit streams, provide timing, and signal carrier sense and collision detect to the data link.

The remainder of this section specifies the requirements for compatibility at both the transceiver cable and the coaxial cable. In addition, the specifications for the transceiver, which interfaces the transceiver cable to the coaxial cable is given, as well as the specification for the logic required between the transceiver cable and the interface to the data link.

7.2 Transceiver Cable Compatibility Interface Specifications

The transceiver cable is the means by which a physically separate transceiver is connected to a station. It provides one of the compatibility interfaces described in 7.1.4.2.

7.2.1 Transceiver Cable Signals

The transceiver cable carries four signals: Transmit, Receive, Collision Presence, and Power. Each signal is carried on a twisted pair of conductors in the cable.

7.2.1.1 Transmit Signal

The transmit pair carries encoded data for which the data link is requesting transmission on the channel. This signal is generated by the data encoder, with the transceiver cable drive characteristics specified in 7.2.4.

Amazon
Exhibit 1005
Page 139

7.2.1.2 Receive Signal

The receive pair carries encoded data from the transceiver to the station. It typically goes to the data decoder and the carrier sense circuitry. In the steady-state, all transitions and lack of transitions on the coaxial cable become transitions and lack of transitions on the receive pair, with the transceiver cable drive characteristics specified in 7.2.4. (During start-up, the first few bits may be absorbed by the transceiver to attain steady-state.)

In the case of a station transmitting without collision interference, the station's own transmit transitions on the coaxial cable will also appear on the receive pair, after a delay due to propagation through the transceiver. During collisions (whether or not that transceiver is involved in the collision) transitions on the receive lead are undefined; they may or may not meet decoder phase requirements, or they may not be present at all for extended periods. Thus the receive signal on the transceiver cable cannot be used alone to deterministically generate the carrier sense signal. This is described in more detail in 7.5.3.

7.2.1.3 Collision Presence Signal

The collision presence pair is used by the transceiver to indicate the presence of multiple transmission attempts on the coaxial cable. This is done by transmitting a square wave with a 10MHz fundamental frequency through the standard transceiver cable driver (described in 7.2.4). An oscillator is used instead of a simple level shift to allow AC coupling at the transceiver. Transceivers use the collision presence signal to indicate one of two conditions: the transceiver is transmitting and there is an attempt by another station to transmit at the same time, or there is a simultaneous transmission attempt by three or more stations regardless of whether the transceiver in question is transmitting.

7.2.1.4 Power

A pair of wires is designated for providing power to the transceiver. When the transceiver cable is implemented, the station end of the cable must supply a voltage between +12 and +15 Vdc ± 5% with at least 0.5 Amperes available to the cable for remotely powering the transceiver. The power source must meet applicable requirements for UL Class 2 wiring devices.

7.2.2 Transceiver Cable Parameters

7.2.2.1 Mechanical Configuration

The transceiver cable consists of four stranded, twisted pair conductors, plus an overall shield and insulating jacket. The conductor and jacket insulating material may be polyethylene or other suitable material. The flammability characteristics of the insulating material must be suitable for the installed environment.

Amazon
Exhibit 1005
Page 140

7.2.2.2 Characteristic Impedance

The differential mode characteristic impedance of all pairs shall be 78 Ω, ± 5 Ω, in the configuration.

7.2.2.3 Attenuation

The signal attenuation of any pair shall not exceed 3 dB (measured at 10 MHz) for the total length between the transceiver and the station.

7.2.2.4 Velocity of Propagation

The minimum velocity of propagation of the transceiver cable shall be 0.65 c.

7.2.2.5 Pulse Distortion

Pulse distortion shall not exceed ± 1 nS at the end of 50 meters of cable when driven with random 10 Mbit data encoded in accordance with 7.5.1.

7.2.2.6 Resistance

The resistance of the conductors used for the power pair shall not exceed 40 milliohms per meter.

7.2.2.7 Transfer Impedance

The common mode transfer impedance of the transceiver cable shall not exceed the values shown in Figure 7-2 as a function of frequency. The differential mode transfer impedance of the cable with respect to any pair shall be 20 dB lower than the specified common mode transfer impedance.

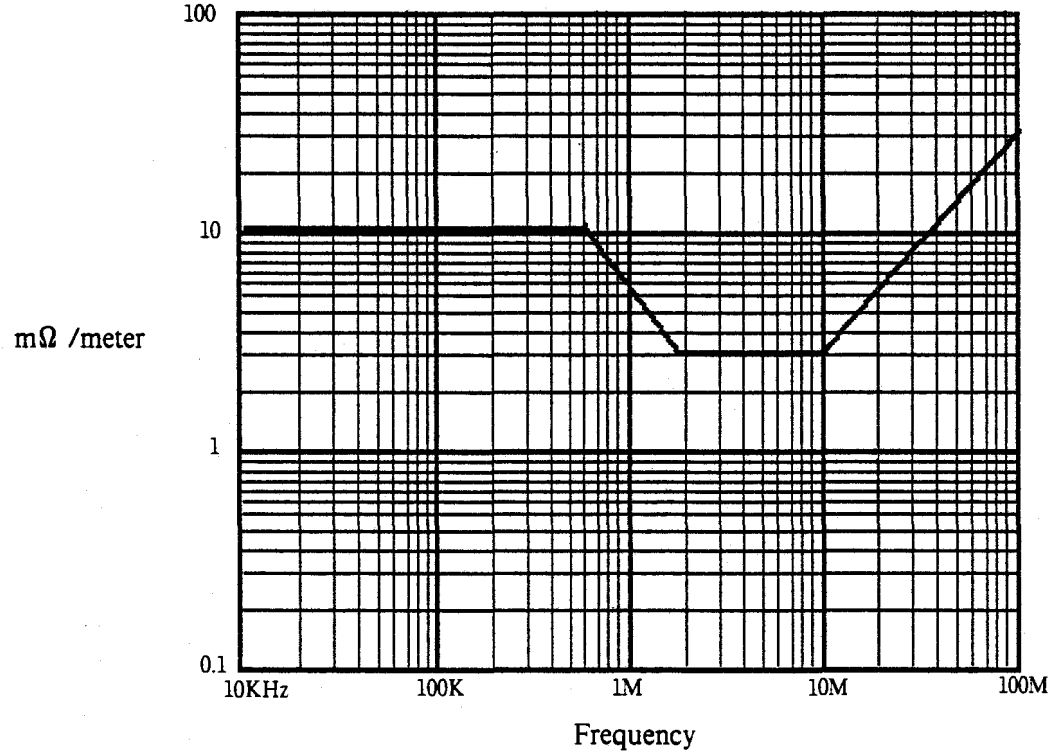


Figure 7-2: Transceiver Cable Transfer Impedance

Amazon
Exhibit 1005
Page 141

7.2.3 Transceiver Cable Connectors

The connectors used at the ends of the transceiver cable shall be 15 conductor 'D' subminiature types (Cinch type DASM-15 or equivalent). The end of the cable that mates with the transceiver must use a female connector with a slide lock assembly (Cinch type DA 51220-1 or equivalent). The transceiver must provide a mating male connector with locking posts. The other end of the transceiver cable (which mates with a female connector at the station) must use a male connector with locking posts (Cinch type D 53018 or equivalent). The station must provide a female connector with the slide lock assembly.

Because of the end-to-end matching of the connectors, transceiver cables may be extended by concatenating transceiver cable sections. (The transceiver cable sections function as 'extension cords'.) A cable with multiple sections must still meet the cable loss characteristics of 7.3.1.1.2.

The pin assignment is given in the following table:

Transceiver Cable Connector Pin Assignment

1.	Shield (See note)		
2.	Collision Presence +	9.	Collision Presence -
3.	Transmit +	10.	Transmit -
4.	Reserved	11.	Reserved
5.	Receive +	12.	Receive -
6.	Power return	13.	Power
7.	Reserved	14.	Reserved
8.	Reserved	15.	Reserved

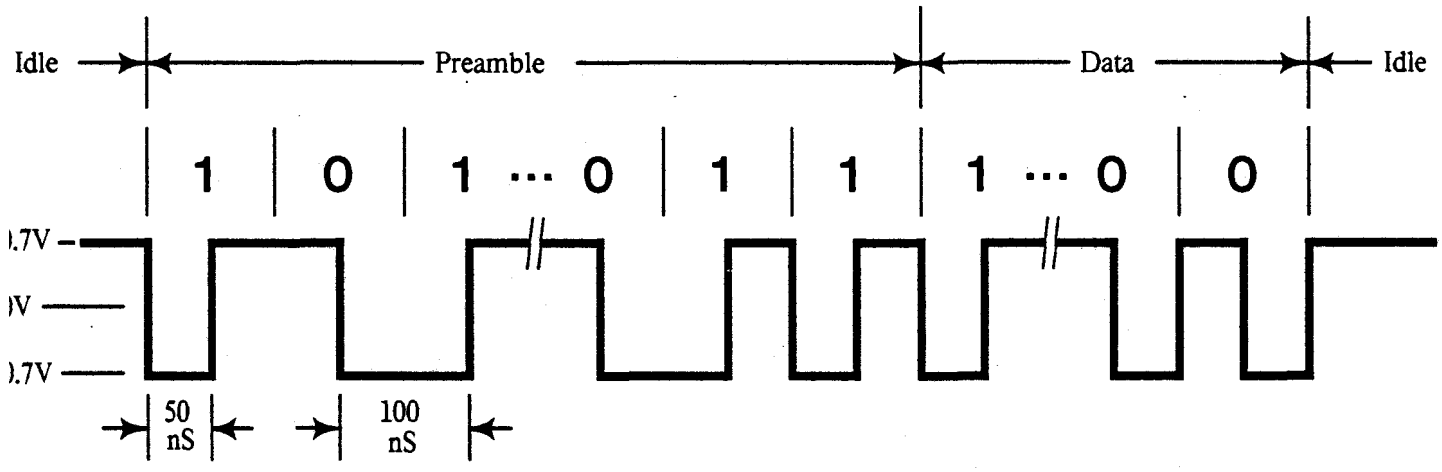
Note: Shield must be terminated to connector shell as well as pin 1.

Metal, metallized plastic, or otherwise shielded connector backshells must be used to ensure shield integrity.

7.2.4 Transceiver Cable Drive

This section describes the requirements for driving any of the signal pairs in the transceiver cable: transmit, receive, and collision presence.

The AC signal levels presented to the transceiver cable shall be ± 700 mV nominal, balanced differential drive into 78 ± 5 Ω. The common mode voltage presented to the transceiver cable shall not exceed that allowed at the receiver, as specified in 7.2.5.2. Signal waveform shall be as shown in Figure 7-3.



1. Voltages are measured differentially at output of transceiver cable driver.
2. Rise and fall times meet 10,000 series ECL requirements.

Figure 7-3: Typical Transceiver Cable Waveform

The transceiver cable driver must be capable of maintaining the specified minimum differential signal into the worst case low cable impedance (73 Ω differential, 18.5 Ω common mode) in the environment specified in section 7-7.

The idle state of the output shall be high (+ 700 mV nominal); the first transition presented is negative-going, the last transition must be positive-going. Note that the presence of AC coupling may cause the voltage as specified at the output of the transceiver cable drive circuit not to appear on the transceiver cable in the idle state.

A typical transceiver cable drive circuit is given in Appendix D.

7.2.5 Transceiver Cable Receive

The following sections specify the requirements for receiving signals from any signal pair in the transceiver cable: transmit, receive, and collision presence. The circuit must be capable of receiving the signals from the transceiver cable driver specified in 7.2.4 through the cable specified in 7.2.2 in the worst case. A typical receive circuit is given in Appendix D.

7.2.5.1 Load Impedance and Termination

The termination impedance shall be 78 Ω ± 1% differential mode, and 18.5 Ω minimum common-mode, over the frequency range of 3-20 MHz.

7.2.5.2 Common Mode and CMRR

The common mode range and the common mode rejection ratio shall be sufficient to maintain a 5:1 signal to noise ratio in the environment specified in 7.7, measured at the input to the transceiver cable receiver. The common mode DC voltage at the input of the receiver shall be in the range of zero to +5 Vdc..

Amazon
Exhibit 1005
Page 143

7.3 Coaxial Cable Compatibility Interface Specifications

The coaxial cable is the common, shared broadcast medium through which stations communicate. It provides one of the compatibility interface points described in 7.1.4.2.

7.3.1 Coaxial Cable Component Specifications

The cable is of constant impedance, coaxial construction. It is terminated at each end by a *terminator* (specified in 7.3.1.3), and connection provided for each transceiver. Coaxial cable connectors are used to make the connection from the cable to the terminators, and between cable sections (if needed). The cable has various electrical and mechanical requirements which must be met to ensure proper operation.

7.3.1.1 Coaxial Cable Parameters

7.3.1.1.1 Characteristic Impedance

The average characteristic impedance of the cable shall be 50 ± 2 Ω, measured according to Mil. Std. C17-E. Periodic variations in impedance along a single piece of cable may be up to ± 3 Ω sinusoidal, centered around the average value, with a period < 2 meters. Note that the proper operation of the network is dependent upon the cable characteristic impedance; its value and tolerance are critical.

7.3.1.1.2 Attenuation

The attenuation of a cable segment shall not exceed 8.5 dB measured at 10 MHz, nor 6.0 dB measured at 5 MHz.

7.3.1.1.3 Velocity of Propagation

The minimum acceptable velocity of propagation is 0.77 c..

Amazon
Exhibit 1005
Page 144

7.3.1.1.4 Mechanical Requirements

The cable used should be suitable for routing in various environments, including but not limited to, dropped ceilings, raised floors, and cable troughs. The jacket must provide insulation between the cable sheath and any building structural metal. Also, the cable must be capable of accepting coaxial cable connectors, described in 7.3.1.2. The cable must in addition conform to the following requirements:

1. The center conductor must be 0.0855" ± .0005" diameter solid, tinned copper,
2. The core dielectric material must be foamed,
3. The inside diameter of the innermost shield must be .242" minimum,
4. The outside diameter of the outermost shield must be .326" ± .007",
5. The outermost shield must be greater than 90% coverage tinned copper braid,
6. The jacket O.D. must be 0.405" nominal,
7. The cable concentricity must be 90% minimum.

The cable must also meet applicable flammability criteria and local codes for the installed environment. Different (e.g., polyethylene and Teflon dielectric) types of cable sections may be interconnected, while meeting the sectioning requirements of 7.6.1.

7.3.1.1.5 Pulse Distortion

Pulse distortion shall not exceed ± 7 nS at the end of 500 meters of cable when driven with random 10 Mbit data encoded in accordance with 7.5.1.

7.3.1.1.6 Jacket Marking

The cable jacket must be marked with annular rings in a color contrasting with the background color of the jacket. The rings must be spaced at 2.5 meter ± 5 cm regularly along the entire length of the cable. It is permissible for the 2.5 meter spacing to be interrupted at discontinuities between cable sections joined by connectors. (See 7.6.2 for transceiver placement rules which mandate cable markings.)

7.3.1.1.7 Transfer Impedance

The transfer impedance of the cable shall not exceed the values shown in Figure 7-4 as a function of frequency.

Amazon
Exhibit 1005
Page 145

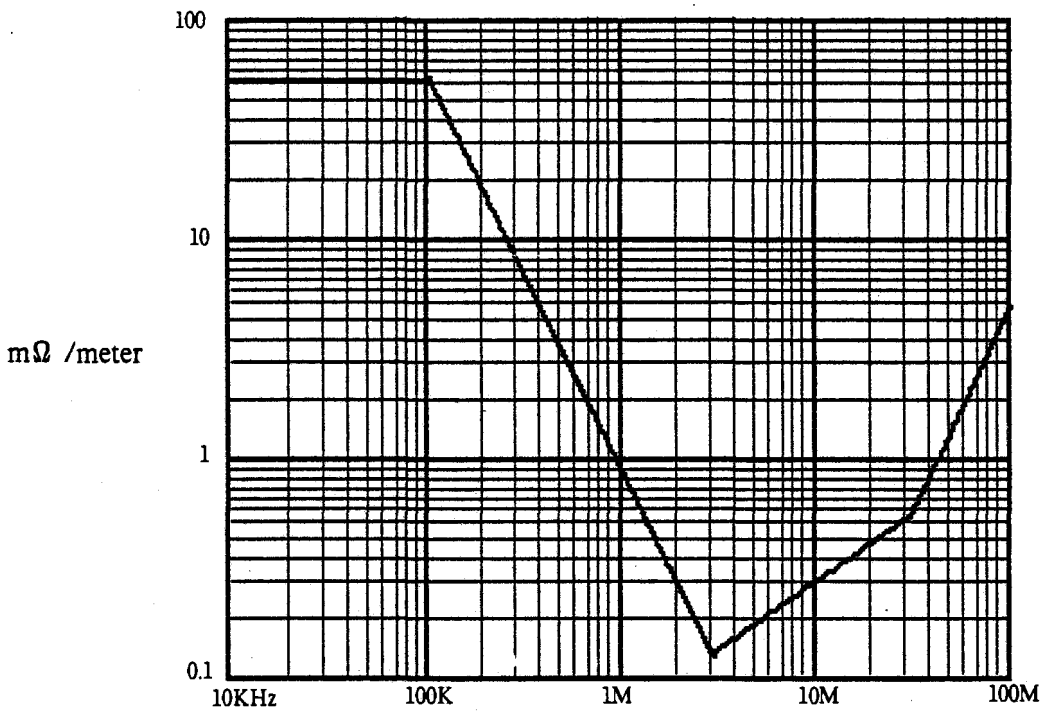


Figure 7-4: Maximum Coaxial Cable Transfer Impedance

7.3.1.2 Coaxial Cable Connectors

Coaxial cable connectors are used to join cable sections and attach terminators. Three types of connectors may be necessary; male plugs, female jacks, and female-to-female barrels. Plugs are used exclusively at the ends of all cable sections. Jacks are used to house cable terminators. Barrels are used to join cable sections.

All connectors are N series, 50 Ω constant impedance types. Since the frequencies present in the transmitted data are well below UHF range (being band-limited to approximately 20 MHz), military versions of the connectors are not required (but are acceptable).

Means must be provided to ensure that the connector shell (which connects to the cable sheath) does not make contact with any building metal, or other unintended conductor. A sleeve or boot to be slid over the connector at installation time is suitable.

7.3.1.3 Coaxial Cable Terminators

Coaxial cable terminators are used to provide a termination impedance for the cable equal in value to its characteristic impedance, thereby eliminating any reflection from the ends of the cables. Terminators shall be packaged within an inline female jack connector. The termination impedance shall be 50 Ω ± 1% measured from 0-50 MHz, with the magnitude of the phase angle of the impedance not to exceed 5 degrees. The terminator power rating shall be 1 watt or greater.

7.3.1.4 Transceiver-to-Coaxial Cable Connections

A means must be provided to allow for attaching a transceiver to the coaxial cable. The connection must disturb the transmission line characteristics of the cable as little as possible; it must present a predictably low shunt capacitance, and therefore a negligibly short stub length. For this reason, the transceiver must be located as close to its cable connection as possible; they are normally considered to be one assembly. Long (greater than 3 cm) connections between the coaxial cable and the input of the transceiver are not acceptable.

The transceiver-to-coaxial cable connection shall present less than 2 picofarads shunt capacitance to the coaxial cable, not including any transceiver electronics. If the design of the connection is such that the coaxial cable must be severed to install the transceiver, the coaxial cable segment must still meet the sectioning requirements of 7.6.1. Any coaxial connectors used on a severed cable must be type N, as specified in 7.3.1.2.

7.3.2 Coaxial Cable Signaling

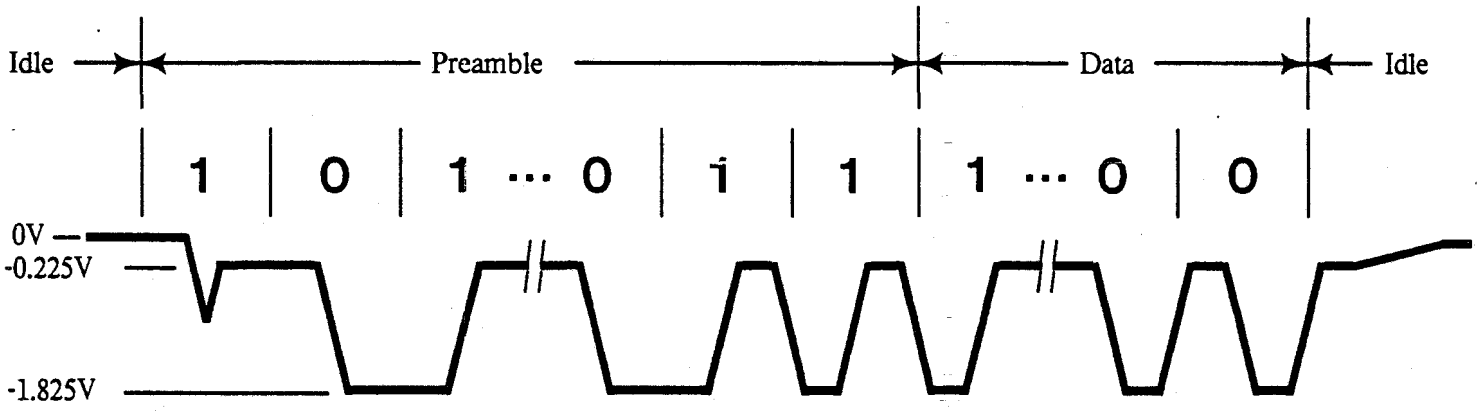
The AC component of the signal on the coaxial cable due to a single transceiver as measured on the coaxial cable immediately adjacent to the transceiver connection shall be ± 16 mA nominal (14 mA min, 19 mA max). The DC component shall be one-half the AC component, plus 4.5 mA (4 mA min, 5 mA max). The actual current measured at a given point on the cable is a function of the transmitted current and the cable loss to the point of measurement. Positive current is defined as current out of the center conductor of the cable (into the transceiver). Cable loss is specified in 7.3.1.1.2.

The 10%-90% rise and fall times shall be 25 ± 5 nsec. Figure 7-5 shows typical waveforms present on the cable. Harmonic content generated from a 10 MHz fundamental periodic input shall meet the following requirements:

Second and Third Harmonics:	-20 dB min
Fourth and Fifth Harmonics:	-30 dB min
Sixth and Seventh Harmonics:	-40 dB min
All Higher Harmonics:	-50 dB min

The signals as generated from the encoder (described in 7.5.1.1) shall appear on the coaxial cable without any inversions.

Amazon
Exhibit 1005
Page 147



1. Voltages given are nominal; worst case is given in text.
2. Rise time is 25 nS nominal.
3. Voltages are measured on coaxial cable adjacent to transceiver.

Figure 7-5: Typical Coaxial Cable Waveform

7.4 Transceiver Specifications

The following sections specify the requirements for a transceiver.

7.4.1 Transceiver-to-Coaxial Cable Interface

The following sections describe the interface between the transceiver and the coaxial cable. Positive current is defined as current into the transceiver (out of the center conductor of the cable).

7.4.1.1 Input Impedance

The shunt capacitance presented to the coaxial cable by the transceiver circuitry (not including the means of attachment to the coaxial cable) shall not exceed 2 picofarads. The shunt resistance presented to the coaxial cable shall be greater than 50 KΩ.

These conditions must be met in both the power off and the power on, not transmitting states.

7.4.1.2 Bias Current

The transceiver must draw between -2 and +50 uA in the power-off and the power-on, not transmitting states.

7.4.1.3 Transmit Output Levels

Signals received from the transceiver cable transmit pair must be transmitted onto the coaxial cable with the characteristics specified in 7.3.2. Note that 7.3.2 specifies the current level on the coaxial cable. Since the coaxial cable proceeds in two directions away from the transceiver, the current into the transceiver is actually twice the current measured on the coaxial cable.

Transmitted output asymmetry shall not exceed 2 ns for a 50/50 duty cycle input on the transceiver cable transmit pair.

7.4.2 Transceiver-to-Transceiver Cable Interface

7.4.2.1 Transmit Pair

The transceiver must present the transceiver cable receive characteristics specified in 7.2.5 to the transmit pair. At the start of a frame transmission, no more than 2 bits (two 100 ns bit cells) of information may be received from the transmit pair and not transmitted onto the coaxial cable. The steady-state propagation delay between the transmit pair input and the coaxial cable output shall not exceed 50 ns. There are no signal inversion between the transceiver cable transmit pair and the coaxial cable.

7.4.2.2 Receive Pair

The transceiver must present the transceiver cable transmit characteristics specified in 7.2.4 to the receive pair. Asymmetry as seen on the receive pair shall not exceed ± 2 nsec for a ± 200 mV peak sinusoidal input from the coaxial cable.

The signal from the coaxial cable shall pass through AC coupling with an appropriate time constant before proceeding to the receive pair. The time constant should compensate for the coaxial cable pulse distortion.

At the start of a frame reception from the coaxial cable, no more than 5 bits (five 100 ns bit cells) of information may be received from the coaxial cable and not transmitted onto the receive pair. In addition, it is permissible for the first bit sent over the receive pair to contain encoding phase violations or invalid data, however all successive bits of the frame shall be valid and meet encoding rules. The steady-state propagation delay between the coaxial cable and the receive pair output shall not exceed 50 ns. There are no signal inversions between the coaxial cable and the transceiver cable receive pair.

Amazon
Exhibit 1005
Page 149

7.4.2.3 Collision Presence Pair

The transceiver must present the transmitter characteristics specified in 7.2.4 to the collision presence pair. The signal presented to the collision presence pair shall be a periodic waveform with a 10 MHz ± 15% frequency. This signal shall be presented to the collision presence pair no more than 5 bit times (500 nS) after the average signal on the coaxial cable at the transceiver exceeds either that which could be produced by two transceiver outputs in the worst case (if the transceiver in question is not transmitting), or that which could be produced by that transceiver alone in the worst case (if that transceiver is transmitting).

7.4.2.4 Power Pair

The transceiver cable provides power which may be used for operation of the transceiver electronics. The power available shall be as described in 7.2.1.4. The distribution impedance of the transceiver cable is 4 Ω maximum, for a 50 meter cable with the resistance specified in 7.2.2.6. In order for the transceiver to derive its operating power from the power pair, circuitry must be employed to provide the required electrical isolation specified in 7.4.3.

7.4.3 Electrical Isolation

The transceiver must provide electrical isolation between the transceiver cable and the coaxial cable. The isolation impedance shall be greater than 250 KΩ, measured between any conductor (including shield) of the transceiver cable and either the center conductor or shield of the coaxial cable, at 60 Hz. The breakdown voltage of the isolation means provided shall be at least 250 VAC, rms.

7.4.4 Reliability

No single nor double component failure within the transceiver electronics shall impede communication among other transceivers on the coaxial cable. Connectors and other passive components comprising the means of connecting the transceiver to the coaxial cable shall be designed to minimize the probability of total network failure.

7.5 Channel Logic

The following sections describe the functions that must be performed to properly interface between the data link and the transceiver cable. They are normally implemented as logic, typically within the same device implementing the data link layer.

7.5.1 Channel Encoding

The channel shall use Manchester phase encoding, with a data rate of 10 Mbps, ± .01%, measured at the encoder clock. Thus, each bit cell is 100 ns long.

The following section describes the requirements for encoding and decoding signals to be transmitted on, or received from the coaxial or transceiver cables.

Amazon
Exhibit 1005
Page 150

7.5.1.1 Encoder

The encoder is used to translate physically separate signals of clock (synchronization) and data into a single, self-synchronizable serial bit stream, suitable for transmission on the coaxial cable by the transceiver.

During the first half of the bit cell time, the serial signal transmitted is the logical complement of the bit value being encoded during that cell. During the second half of the bit cell time, the uncomplemented value of the bit being encoded is transmitted. Therefore, there is always a signal transition (either positive-going or negative-going, depending on the bit being encoded) in the center of each bit cell. A timing diagram for a typical bit stream is given in Figure 7-6.

The encoder output drives the transmit pair of the transceiver cable, and ultimately, the coaxial cable through the transceiver. The encoder output asymmetry must not exceed 0.5 ns. The encoder shall provide the defined output for the first (and all subsequent) bits presented to its input. All information submitted for encoding shall appear at the output of the encoder.

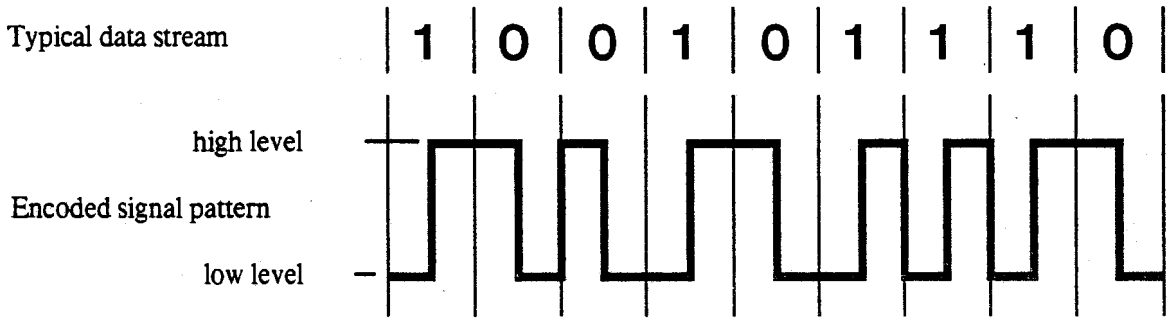


Figure 7-6: Manchester Encoding

7.5.1.2 Decoder

The decoder is used to separate the incoming phase encoded bit stream into a data stream and a clock signal. The decoder must be able to provide data and clock signals usable by the data link under the asymmetry imposed by the worst case system configuration. The decoder must provide usable output (clock and data) after no more than 8 bit cell times after reception of an encoded signal. The first signals received from the transceiver at the beginning of frame reception may not constitute a valid, properly encoded bit; it is possible for the time from the first transition seen to the first true mid-bit cell transition to assume any value from zero to 100 nS.

The decoder input is normally derived from the coaxial cable, through the transceiver cable receive pair. It is not necessary for the decoder to provide usable output when there is a collision on the coaxial cable, regardless of whether the station using that decoder is involved in the collision.

Amazon
Exhibit 1005
Page 151

7.5.1.3 Preamble Generation

Because most of the channel circuitry is allowed to provide valid output some number of bit times after being presented valid input, it is necessary for a preamble to be sent before the start of data link information, to allow the channel circuitry to reach its steady-state, with valid outputs throughout the system. Upon request by the data link to transmit the first bit of a new frame, the channel shall first transmit the preamble; a predetermined bit sequence used for channel stabilization and synchronization. If, while transmitting the preamble, the channel logic asserts the collision detect signal as specified in 7.5.2, any remaining preamble bits shall not be sent. The channel should immediately proceed with the transmission of the bit submitted by the data link.

The preamble is a 64 bit pattern to be presented to the channel encoder in the same manner as data link information. The pattern is:

10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101011.

The bits are transmitted in order, from left to right. The nature of the pattern is such that when encoded, it appears as a periodic waveform on the cable, with a 5 MHz frequency. Excepting the final two bits, the only transitions present in the waveform are in the center of the bit cells. This is depicted in Figure 7-7. The last two bits of the preamble contain transitions at both the bit cell centers and the edges, and are used to indicate the end of the preamble, and the beginning of the data link encapsulation portion of the frame. The next bit transmitted is the bit originally submitted by transmission by the data link.

Preamble removal on reception is discussed in 7.5.4.1.

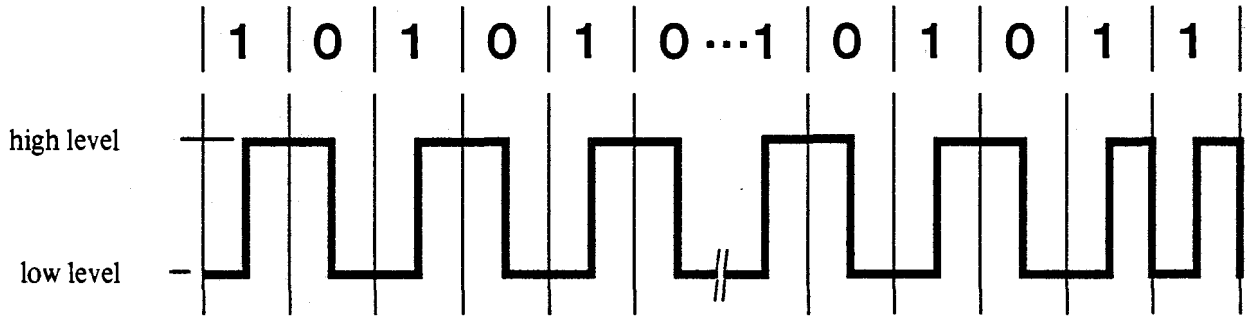


Figure 7-7: Preamble Encoding

7.5.2 Collision Detect Signal

The channel must indicate to the data link when the signals on the coaxial cable imply simultaneous transmission attempts by more than one station. This is normally indicated through the collision presence pair in the transceiver cable, described in 7.2.1.3.

The channel logic must assert the collision detect signal within 2 bit times (200 ns), following the onset of collision presence. This collision detect signal shall be asserted only when the data link is transmitting. A functional logic description of the collision detect signal is shown in Figure 7-8.

Following the loss of collision presence information, the channel must deassert the collision detect signal within 1.6 bit cell times (160 ns).

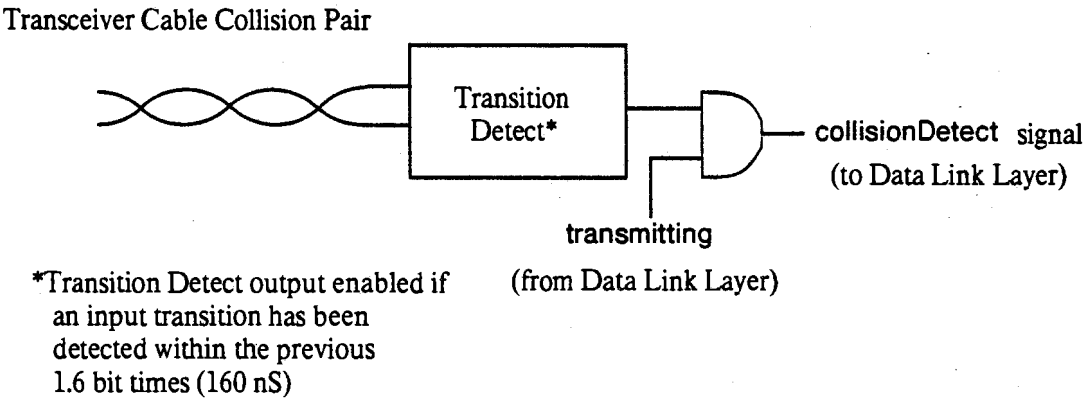


Figure 7-8: Functional Logic of collisionDetect Signal

7.5.3 Carrier Sense Signal

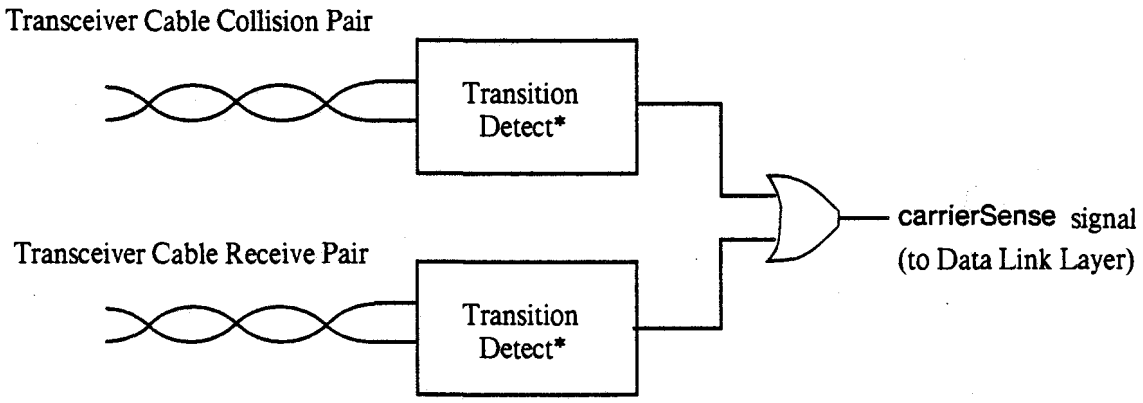
The channel must indicate to the data link the presence of carrier, a signal transmission attempt on the coaxial cable by a station. This is normally indicated through both the receive and collision presence pairs in the transceiver cable, described in 7.2.1.

The carrier sense signal shall be asserted when one or more station is attempting transmission on the cable, regardless of whether the station sensing carrier is transmitting at that time. The channel logic must assert the carrier sense signal within 2 bit times (200 ns) following the onset of carrier presence information. A

Amazon
Exhibit 1005
Page 153

functional logic description of these signals is shown in Figure 7-9.

Following the loss of carrier presence information (receive transitions and collision presence information) the channel must deassert the carrier sense signal within 1.6 bit cell times (160 ns).



*Transition Detect output enabled if an input transition has been detected within the previous 1.6 bit times (160 nS)

Figure 7-9: Functional Logic of carrierSense Signal

7.5.4 Channel Framing

During reception, the channel must provide the data link with signals to indicate beginning and end of frame.

7.5.4.1 Beginning-of-Frame Sequence

The channel logic recognizes the presence of activity on the medium through the carrier sense signal. This is the first indication that the frame reception process should begin. However, dependent upon the physical configuration of the system, there are some number of preamble bits to be received by the channel before the start of the data link frame as indicated by the double-1 at the end of preamble. In addition, the first signals received from the decoder may be invalid due to the first bit allowance of the transceiver (see 7.4.2.2). The channel must wait no less than 8 bit times (800 nS) before monitoring the output of the decoder for the 'double-1' indicating end of preamble, and beginning of data link frame. Upon reception of the double-1, the channel shall begin passing successive bits to the data link through the defined receive bit stream interface. If, after waiting the required 8 bit times, a 'double-0' is encountered, the physical channel shall not pass any bits of

the current frame to the data link. Normal operation of the data link and channel shall resume on the subsequent frame.

7.5.4.2 End-of-Frame Sequence

As specified in 7.5.3, the carrier sense signal must be deasserted no later than 1.6 bit times (160 ns) after the cessation of activity on the coaxial cable as seen by the channel logic. The channel ensures that no extraneous bits will appear at the end of a frame following the last valid bit.

7.6 Channel Configuration Requirements

7.6.1 Cable Sectioning

The 500 meter maximum length coaxial cable segment need not be made from a single, homogeneous length of cable. The boundary between two cable sections (joined by coaxial connectors; two male plugs and a barrel) represents a signal reflection point due to the impedance discontinuity caused by the batch-to-batch impedance tolerance of the cable. Since the worst-case variation from 50 Ω is 2 Ω (see 7.3.1.1.1), a possible worst-case reflection of 4% may result from the joining of two cable sections. The configuration of long cable segments (up to 500 meters) from smaller sections must be made with care. The following recommendations apply, and are given in order of preference:

1. If possible, the total segment should be made from one homogeneous (no breaks) cable. This is feasible for short segments, and results in minimal reflections from cable impedance discontinuities.
2. If cable segments must be built up from smaller sections, it is highly desirable to ensure that all the sections are from the same manufacturer and lot. This is equivalent to using a single cable, since the cable discontinuities are due to extruder limitations, and not extruder-to-extruder tolerances. There are no restrictions in cable sectioning if this method is used. However, if a cable section in such a system is later replaced, it must be replaced either with another cable from the same manufacturer and lot, or with one of the standard lengths described below.
3. If uncontrolled cable sections must be used in building up a longer segment, the lengths should be chosen such that reflections, when they occur, do not have a high probability of adding in phase. This can be accomplished by using lengths which are odd integral multiples of a half-wavelength in the cable at 5 MHz; this corresponds to using lengths of 23.4, 70.2, and 117 meters (± 0.5 meters) for all sections. These are considered to be the standard lengths for all cable sections. Using these lengths exclusively, any mix or match of cable sections may be used to build up a 500 meter segment without incurring excessive reflections.

Amazon
Exhibit 1005
Page 155

4. As a last resort, an arbitrary configuration of cable sections may be employed, if it has been confirmed by analysis or measurement that the worst-case signal reflection due to the impedance discontinuities at any point on the cable does not exceed 7% of the incident wave when driven by a transceiver meeting the specifications of 7.4.

7.6.2 Transceiver Placement

Transceivers and their associated connections to the cable cause signal reflections due to their non-infinite bridging impedance. While this impedance must be implemented as specified in 7.3.1.4 and 7.4.1, the placement of transceivers along the coaxial cable must also be controlled to insure that reflections from transceiver do not add in phase to a significant degree.

Coaxial cables marked as specified in 7.3.1.1.6 have marks at regular 2.5 meters spacing; a transceiver may be placed at any mark on the cable. This guarantees both a minimum spacing between transceivers of 2.5 meters, as well as controlling the relative spacing of transceivers to insure non-alignment on fractional wavelength boundaries.

The total number of transceivers on a cable segment shall not exceed 100.

7.6.3 System Grounding

The sheath conductor of the coaxial cable shall not make electrical contact with any earth reference, building structural metal, ducting, plumbing fixture, or other unintentioned conductor. Insulators may be used to cover any coaxial connectors used to join cable sections and terminators, to insure that this requirement is met. A sleeve or boot attached at installation time is acceptable.

The sheath conductor of the transceiver cable shall be connected to the earth reference or chassis of the device housing the station logic.

Amazon
Exhibit 1005
Page 156

7.6.4 Repeaters

Repeaters are used to extend the channel length and topology beyond that which could be achieved by a single coaxial cable segment. (See the channel configuration model in 7.1.5.) A repeater requires a transceiver on each of the segments between which it is repeating signals. These transceivers must be as specified in 7.4, and must be counted towards the maximum specified in 7.6.2.

A maximum of two repeaters may be in the signal path between any two transceivers on the channel.

7.6.4.1 Carrier Detect and Transmit Repeat

Repeaters must implement the carrier sense function as specified in 7.5.3 for both segments between which it is connected. Upon detection of carrier from one segment, the repeater must retransmit all received signals from that segment onto the other segment. Signals shall be retimed and amplified as specified in 7.6.4.3. The maximum steady-state propagation delay through the repeater for the repeated signal (not including startup delays, carrier sense delay or retiming delays) shall not exceed 800 nS.

7.6.4.2 Collision Detect and Collision Repeat

Repeaters must implement the collision detect function as specified in 7.5.2 for both segments between which it is connected. If, while repeating signals as specified in 7.6.4.1, collision is detected on either side, the repeater must ensure that all stations involved in the collision recognize the event as a collision, regardless of which side of the repeater the station is on. The maximum time between the recongnition of the collision and the repeating of the collision indication (not including carrier sense of retiming delays) shall not exceed 200 nS.

7.6.4.3 Repeater Signal Regeneration

7.6.4.3.1 Signal Amplification

The repeater (with its associated transceivers) shall ensure that any signals repeated between segments shall have the same amplitude characteristics at the transceiver output of the repeated-to segment as they did at the output of the transmitter on the repeated-from segment, allowing for transceiver output tolerances as specified in 7.4.1.3. Any loss of signal-to-noise ratio due to cable loss and noise pickup is thus regained at the output of the repeater.

7.6.4.3.2 Signal Timing

The repeater must ensure that the symmetry characteristics of the signals at the transceiver output of the repeated-to segment are the same as those at the output of the transmitter on the repeated-from segment, allowing for transceiver and transceiver cable tolerances. Any loss of symmetry due to transceivers and cable distortion is thus regained at the output of the repeater.

Amazon
Exhibit 1005
Page 157

7.7 Environment Specifications

The following sections specify the physical environment in which all channel components must operate to be considered compatible.

7.7.1 Electromagnetic Environment

The physical channel hardware shall meet its specifications when operating in the following ambient plane-wave fields:

2 Volts/Meter from 10 KHz through 30 MHz

5 Volts/Meter from 30 MHz through 1 GHz

7.7.2 Temperature and Humidity

All physical channel hardware, with the possible exception of the channel logic components shall operate over the ambient temperature range of 5 to 50 degrees Celsius, and humidity range of 10% to 95% non-condensing. The channel logic components are normally part of the station hardware, and are thus subject to individual station product requirements. Hardware which does not meet the temperature and humidity requirements specified must state so in its published product specification.

Amazon
Exhibit 1005
Page 158

APPENDIX A: GLOSSARY

This section defines some of the essential terminology associated with the Ethernet.

baseband coaxial system: A system whereby information is directly encoded and impressed on the coaxial transmission medium. One information signal at a time can be present on the medium without disruption (see collision).

binary exponential backoff: The algorithm used to schedule retransmissions after a collision. So called because the interval from which the retransmission time is selected is expanded exponentially with repeated collisions.

broadcast: Describes the class of media for which the Ethernet is designed, in which all stations are capable of receiving a signal transmitted by any other station. Also, describes the mode of usage of such a medium by the Data Link Layer in which all stations are instructed to receive a given frame.

carrier sense: A signal provided by the Physical Layer to the Data Link Layer to indicate that one or more stations are currently transmitting on the channel.

channel logic: The logical functions provided between the transceiver cable and the Data Link, which support the defined interface between the data link and the physical layers.

Client Layer: Collective term used to describe any layer of a network architecture, which use the Ethernet Data Link and Client interface.

coaxial cable: A two-conductor, concentric, constant impedance transmission line.

coaxial cable interface: The electrical, mechanical, and logical interface to the shared coaxial cable medium. This is a mandatory compatibility interface, which must be correctly implemented by every Ethernet implementation.

coaxial cable section: An unbroken piece of coaxial cable, fitted with coaxial connectors at its ends, used to build up coaxial cable segments.

coaxial cable segment: A length of coaxial cable made up from one or more coaxial cable sections and coaxial connectors, terminated at each end in its characteristic impedance. A 500 meter segment is the longest configuration possible without repeaters.

collision: The result of multiple transmissions overlapping in the physical channel, resulting in garbled data and necessitating retransmission.

collision detect: A signal provided by the Physical Layer to the Data Link Layer to indicate that one or more other stations are contending with the local station's

Amazon
Exhibit 1005
Page 159

transmission. It can be true only during transmission.

collision enforcement: Transmission of extra, encoded "jam" bits after a collision is detected, to insure that the duration of the collision is sufficient to guarantee its detection by all transmitting stations.

compatibility interfaces: The coaxial cable interface, and the transceiver cable interface, the two points at which hardware compatibility is defined to allow connection of independently designed and manufactured components to the Ethernet.

contention: Interference between colliding transmissions (see collision). Resolution of occasional contention is a normal part of the Ethernet's distributed link management procedure (see CSMA-CD).

controller: The implementation unit which connects a station to the Ethernet, typically comprising part of the Physical Layer, much or all of the Data Link Layer, and appropriate electronics for interfacing to the station.

CSMA-CD: Carrier Sense Multiple Access with Collision Detection, the generic term for the class of link management procedure used by the Ethernet. So called because it a) allows multiple stations to access the broadcast channel at will, b) avoids contention via carrier sense and deference, and c) resolves contention via collision detection and retransmission.

Data Link Layer: The higher of the two layers in the Ethernet design, which implements a medium-independent link level communication facility on top of the physical channel provided by the Physical Layer.

deference: A process by which a data link controller delays its transmission when the channel is busy to avoid contention with ongoing transmissions.

frame check sequence: An encoded value appended to each frame by the Data Link Layer to allow detection of transmission errors in the physical channel.

interframe spacing: An enforced idle time between transmission of successive frames to allow receiving data link controllers and the physical channel to recover.

jam: An encoded bit sequence used for collision enforcement.

Manchester encoding: A means by which separate data and clock signals can be combined into a single, self-synchronizable data stream, suitable for transmission on a serial channel.

multicast: An addressing mode in which a given frame is targeted to a group of logically related stations.

physical address: The unique address value associated with a given station on the network. An Ethernet physical address is defined to be distinct from all other physical addresses on all Ethernets.

Amazon
Exhibit 1005
Page 160

Physical Channel: The implementation of the physical layer.

Physical Layer: The lower of the two layers of the Ethernet design, implemented by the physical channel using the specified coaxial cable medium. The Physical Layer insulates the Data Link Layer from medium-dependent physical characteristics.

preamble: A sequence of 64 encoded bits which the Physical Layer transmits before each frame to allow synchronization of clocks and other Physical Layer circuitry at other sites on the channel.

repeater: A device used to extend the length and topology of the physical channel beyond that imposed by a single segment, up to the maximum allowable end-to-end channel length.

round-trip propagation time: In bit times, the time required in the worst-case for a transmitting station's collision detect signal to be asserted due to normal contention for the channel. This delay is the primary component of the slot time.

slot time: A multi-purpose parameter which describes the contention behavior of the Data Link Layer. It serves as a) an upper bound on the collison vulnerability of a given transmission, b) an upper bound on the size of the frame fragment produced by a collision, and c) the scheduling quantum for collision retransmission.

station: A single addressable site on the Ethernet, generally implemented as a computer and appropriate peripherals, and connected to the Ethernet via a controller and a transceiver.

transceiver: The portion of the Physical Layer implementation that connects directly to the coaxial cable and provides both the electronics which send and receive the encoded signals on the cable and the required electrical isolation.

transceiver cable: A four pair, shielded cable used for the transceiver cable interface.

transceiver cable interface: The electrical, mechanical and logical interface which connects the transceiver to the controller. The standard transceiver cable is a recommended compatibility interface.

Amazon
Exhibit 1005
Page 161

APPENDIX B: NOTES ON ADDRESS AND TYPE ASSIGNMENT, AND LICENSING

Address and Type Assignment

The address and type fields will be administered by Xerox Corporation.

A block of addresses will be assigned to each licensee of Ethernet patents (see below). Others may obtain an address block or type field assignment by request. A nominal fee to cover administrative costs will be charged.

Submit written requests to:

Xerox Corporation
Ethernet Address Administration Office
3333 Coyote Hill Road
Palo Alto, CA 94304

Licensing

Ethernet incorporates features that are protected by one or more patents assigned to Xerox Corporation. Questions on the need for licensing particular uses of this specification should be directed to:

Xerox Corporation
Director of Licensing
Long Ridge Road
Stamford, CT 06904

APPENDIX C: CRC IMPLEMENTATION

Every frame contains, in its frame check sequence field, a 32-bit cyclic redundancy check (CRC) code. Because the formal mathematical definition of this code (see 6.2.4) is not suggestive of an appropriate implementation, this appendix outlines one possible implementation in terms of a feedback shift register. This type of implementation is likely to be common in practice, but is not a mandatory part of the specification.

The feedback shift register (see Figure C-1) is used to represent division of the pre-scaled message by the generating polynomial. The 32-bit register is accessed via the three signals Input, Output, and Control. When Control = 1, Input bits are shifted into the feedback shift register and also fed directly back to Output. When Control = 0, the feedback paths are disabled and the shift register shifts the complement of its contents to Output.

Before CRC generation at the transmitting end, initialization logic (not shown in Figure C-1) preloads the shift register to all 1's. Control is then held at 1 while the address, type and data fields of the outgoing frame are shifted into Input and the CRC is generated. Meanwhile, the same bits emerging at Output are transmitted over the network. When the last bit of the data field has been processed, Control is set to 0 and the complemented CRC is shifted out for transmission, starting with the x^{31} term (see 6.2.4).

CRC checking at the receiving end also begins with the shift register preloaded to all 1's. Control is then held at 1 while the incoming bits are shifted into Input to regenerate the CRC. When the last bit of the data field has been processed, the shift register should contain the CRC whose binary complement is about to arrive on the network. Since this field boundary cannot be recognized by the receiver, however, Control remains at 1 and the bits of the CRC continue to feed into the the shift register until the end of the entire frame is reached. If the two CRCs match, the final contents of the shift register is the value:

11000111 00000100 11011101 01111011

(where the leftmost bit corresponds to the x^{31} term of the polynomial and the rightmost to the x^0 term). Any other final value indicates a detected error. (The extra logic to test for this value is not shown in Figure C-1).

Amazon
Exhibit 1005
Page 163

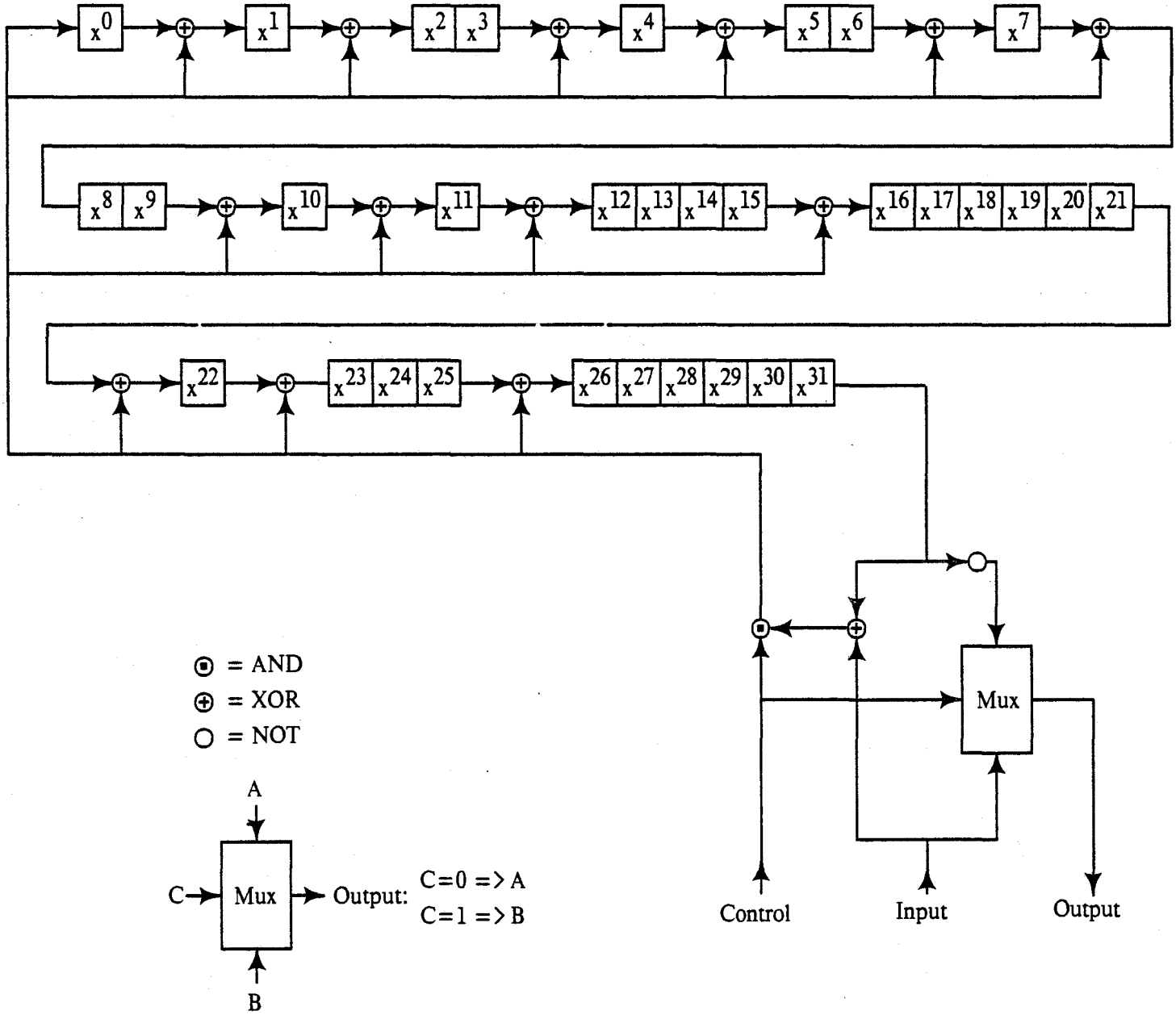


Figure C-1: CRC Implmentation

One potential problem which is avoided in this implementation is insensitivity of the shift register to incoming zero-bits when it is in the all-zero state. Following standard practice, this state is avoided at the beginning and end of the frame by preloading the shift register with all 1-bits, and by inverting each bit of the final CRC. Logically, these correspond, respectively, to the complementing of the first 32 bits of the frame and to the final complementing of the remainder, as specified in the mathematical definition in 6.2.4. See also [9] for further discussion.

Amazon
Exhibit 1005
Page 165

APPENDIX D: IMPLEMENTATION OF TRANSCEIVER CABLE DRIVER AND RECEIVER

This appendix presents circuit digrams for typical implementations of the transceiver cable drivers and receivers. The use of these exact circuits is not necessary for conformance to the specification; equivalent circuits may be used as long as the relevant specifications are met.

Figure D-1 depicts an implementation of the transceiver cable driver specified in 7.2.4. It is suitable for use at either end of the transceiver cable, as necessary; i.e., it would be located at the station end to drive the transmit pair, and at the transceiver end to drive the receive and collision presence pairs. In addition, it is capable of driving suitable isolation circuits required to be located within the transceiver.

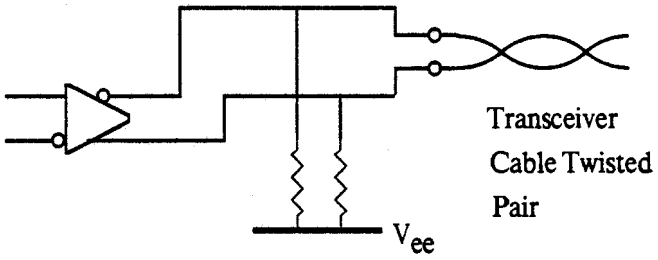


Figure D-1: Typical Transceiver Cable Driver

Figure D-2 depicts an implementation of the transceiver cable receiver specified in 7.2.5. It is suitable for use at either end of the transceiver cable, as necessary; i.e., it would be located at the station end to receive from the receive and collision presence pairs, and at the transceiver to receive from the transmit pair. It is capable of operating through suitable isolation circuits required to be located within the transceiver.

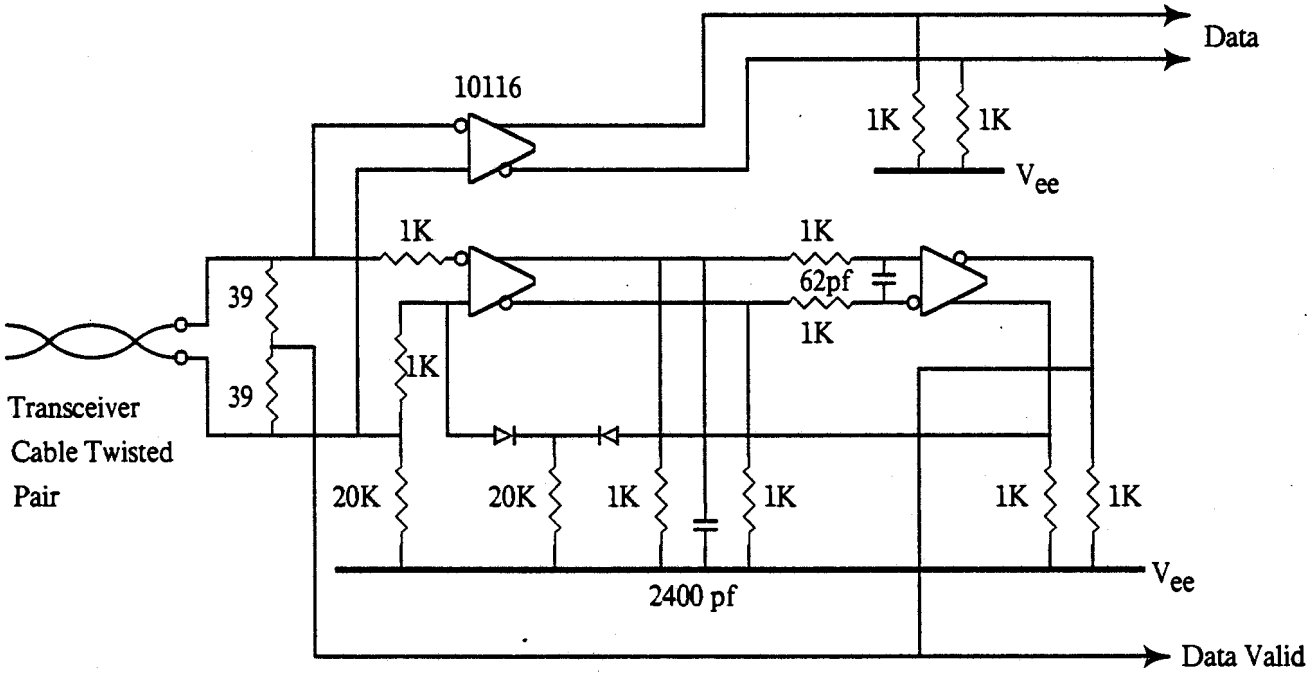


Figure D-2: Typical Transceiver Cable Receiver

APPENDIX E: INTERFRAME RECOVERY

It is important that data link controller implementations be able to receive a frame that arrives immediately after another frame has been transmitted or received. Here, "immediately" means 9.6 μsec, based on the minimum interframe spacing provided as recovery time for the data link. (See 6.3.2.2) It is important that the data link controller be able to resume reception within that time.

Reception of multiple closely spaced incoming frames is a very desirable capability, and is crucial for stations which tend to communicate with several other stations concurrently. There is one important case in which a data link controller implementation cannot reasonably be expected to receive closely spaced incoming frames: if the station hardware (e.g. I/O bus) is intrinsically unable to accept the bits of a frame at the rate at which they arrive over the network, each incoming frame must be buffered to allow the station to accept it at some lower rate. Assuming limited buffering resources (e.g. a one frame buffer), reception of subsequent frames cannot occur until sufficient buffer space is available. This mode of operation is allowed for low performance stations.

Reception of an incoming frame immediately after transmission of an outgoing frame is a very important capability, even for stations which do not tend to communicate with several other stations concurrently. All stations, low performance to high performance, should allow reception of an incoming frame immediately after transmission of an outgoing frame.

Amazon
Exhibit 1005
Page 168

APPENDIX 1009B

http://timeline.ethernethistory.com/ Go
81 captures
13 Oct 2007 - 2 Apr 2018
SEP OCT JUL
2006 13 2007 2009
About this capture

1980

- John Shoch published his Stanford doctorate thesis, "Design and Performance of Local Computer Networks."
- DIX group published "Blue Book" Ethernet specification.
- IEEE formed an 802 committee to develop local area networking standards.
- Vitalink was founded to develop satellite data communication systems. Vitalink then moved into Ethernet-wide area bridges.

APPENDIX 1009C

The Ethernet

A Local Area Network

Data Link Layer
and
Physical Layer
Specifications







Digital Equipment Corporation
Maynard, MA

Intel Corporation
Santa Clara, CA

Xerox Corporation
Stamford, CT

Version 1.0

September 30, 1980

Amazon
Exhibit 1005
Page 172

IMPORTANT INFORMATION AND DISCLAIMERS

1. This specification includes subject matter relating to a patent(s) of Xerox Corporation. No license under such patent(s) is granted by implication, estoppel or otherwise as a result of publication of this specification. Applicable licenses may be obtained from Xerox Corporation.

2. This specification is furnished for informational purposes only. Digital, Intel, and Xerox do not warrant or represent that this specification or any products made in conformance with it will work in the intended manner or be compatible with other products in a network system. Nor do they assume responsibility for any errors that the specification may contain, or have any liabilities or obligations for damages (including but not limited to special, indirect or consequential damages) arising out of or in connection with the use of this specification in any way. Digital, Intel and Xerox products may follow or deviate from the specification without notice at any time.

3. No representations or warranties are made that this specification or anything made from it is or will be free from infringements or patents of third persons.

Amazon
Exhibit 1005
Page 173

Preface

This document contains the specification of the Ethernet, a local area network developed jointly by Digital Equipment Corporation, Intel Corporation, and Xerox Corporation. The Ethernet specification arises from an extensive collaborative effort of the three corporations, and several years of work at Xerox on an earlier prototype Ethernet.

This specification is intended as a design reference document, rather than an introduction or tutorial. Readers seeking introductory material are directed to the reference list in Section 2, which cites several papers describing the intent, theory, and history of the Ethernet.

This document contains 7 sections, falling into three main groups:

Sections 1, 2, and 3 provide an overall description of the Ethernet, including its goals, and the scope of the specification.

Sections 4 and 5 describe the architectural structure of the Ethernet in terms of a functional model consisting of two layers, the Data Link Layer and the Physical Layer.

Sections 6 and 7 specify the two layers in detail, providing the primary technical specification of the Ethernet.

Readers wishing to obtain an initial grasp of the organization and content of the specification will be best served by reading Sections 1, 3, and 4. Readers involved in actual implementation of the Ethernet will find Sections 5, 6, and 7 to contain the central material of the specification. Section 2 provides references, and the appendices provide supplementary material.

The approach taken in the specification of the Data Link Layer in Section 6 is a procedural one; in addition to describing the necessary algorithms in English and control flow charts, the specification presents these algorithms in the language Pascal. This approach makes clear the required behavior of Data Link Layer, while leaving individual implementations free to exploit any appropriate technology.

Because the procedural approach is not suitable for specifying the details of the Physical Layer, Section 7 uses carefully worded English prose and numerous figures and tables to specify the necessary parameters of this layer.

Some aspects of the Ethernet are necessarily discussed in more than one place in this specification. Whenever any doubt arises concerning the official definition in such a case, the reader should utilize the Pascal procedural specification of the Data Link Layer in Section 6.5, and the detailed prose specification of the Physical Layer in Sections 7.2 through 7.9.

One aspect of an overall network architecture which is not addressed by this specification is network management. The network management facility performs operation, maintenence, and planning functions for the network:

- Operation functions include parameter setting, such as address selection.
- Maintenance functions provide for fault detection, isolation, and repair.
- Planning functions include collection of statisical and usage information, necessary for planned network growth.

While network management itself is properly performed outside the Ethernet Data Link and Physical Layers, it requires appropriate additional interfaces to those layers, which will be defined in a subsequent version of this specification.

Amazon
Exhibit 1005
Page 174

Table of Contents

Preface i
1. INTRODUCTION 1
2. REFERENCES 3
3. GOALS AND NON-GOALS 4
3.1 Goals 4
3.2 Non-Goals 5
4. FUNCTIONAL MODEL OF THE ETHERNET ARCHITECTURE 6
4.1 Layering 8
4.2 Data Link Layer 9
4.3 Physical Layer 10
4.4 Ethernet Operation and the Functional Model 12
4.4.1 Transmission Without Contention 12
4.4.2 Reception Without Contention 12
4.4.3 Collisions: Handling of Contention 13
5. INTER-LAYER INTERFACES 15
5.1 Client Layer to Data Link Layer 15
5.2 Data Link Layer to Physical Layer 16
6. ETHERNET DATA LINK LAYER SPECIFICATION 19
6.1 Data link Layer Overview and Model 19
6.2 Frame format 19
6.2.1 Address Fields 21
6.2.1.1 Destination Address Field 21
6.2.1.2 Source Address Field 21
6.2.2 Type Field 21
6.2.3 Data Field 22
6.2.4 Frame Check Sequence Field 22
6.2.5 Frame Size Limitations 22
6.3 Frame Transmission 23
6.3.1 Transmit Data Encapsulation 23
6.3.1.1 Frame Assembly 23
6.3.1.2 Frame Check Sequence Generation 23
6.3.2 Transmit Link Management 23

6.3.2.1 Carrier Deference 23
6.3.2.2 Interframe Spacing 24
6.3.2.3 Collision Handling 24
6.3.2.3.1 Collision Detection and Enforcement 24
6.3.2.3.2 Collision Backoff and Retransmission 25
6.4 Frame Reception 25
6.4.1 Receive Data Decapsulation 25
6.4.1.1 Framing 25
6.4.1.1.1 Maximum Frame Size 26
6.4.1.1.2 Integral Number of Octets in Frame 26
6.4.1.2 Address Recognition 26
6.4.1.2.1 Physical Addresses 26
6.4.1.2.2 Multicast Addresses 26
6.4.1.3 Frame Check Sequence Validation 26
6.4.1.4 Frame Disassembly 27
6.4.2 Receive Link Management 27
6.4.2.1 Collision Filtering 27
6.5 The Data Link Layer Procedural Model 27
6.5.1 Overview of the Procedural Model 27
6.5.1.1 Ground Rules for the Procedural Model 27
6.5.1.2 Use of Pascal in the Procedural Model 29
6.5.2 The Procedural Model 31
6.5.2.1 Global Declarations 34
6.5.2.1.1 Common Constants and Types 34
6.5.2.1.2 Transmit State Variables 35
6.5.2.1.3 Receive State Variables 35
6.5.2.1.4 Summary of Interlayer Interfaces 36
6.5.2.1.5 State Variable Initialization 37
6.5.2.2 Frame Transmission 38
6.5.2.3 Frame Reception 42
6.5.2.4 Common Procedures 44
7. ETHERNET PHYSICAL LAYER SPECIFICATION 45
7.1 Physical Channel Overview and Model 45
7.1.1 Channel Goals and Non-Goals 45
7.1.1.1 Goals 45
7.1.1.2 Non-Goals 45
7.1.2 Characteristics of the Channel 46
7.1.3 Functions Provided by the Channel 46
7.1.4 Implementation of the Channel 46
7.1.4.1 General Overview of Channel Hardware 47
7.1.4.2 Compatibility Interfaces 48
7.1.5 Channel Configuration Model 49
7.1.6 Channel Interfaces 53

7.2 Transceiver Cable Compatibility Interface Specifications 53
7.2.1 Transceiver Cable Signals 53
7.2.1.1 Transmit Signal 53
7.2.1.2 Receive Signal 54
7.2.1.3 Collision Presence Signal 54
7.2.1.4 Power 54
7.2.2 Transceiver Cable Parameters 54
7.2.2.1 Mechanical Configuration 54
7.2.2.2 Characteristic Impedance 55
7.2.2.3 Attenuation 55
7.2.2.4 Velocity of Propagation 55
7.2.2.5 Pulse Distortion 55
7.2.2.6 Resistance 55
7.2.2.7 Transfer Impedance 55
7.2.3 Transceiver Cable Connectors 56
7.2.4 Transceiver Cable Drive 56
7.2.5 Transceiver Cable Receive 57
7.2.5.1 Load Impedance and Termination 57
7.2.5.2 Common Mode and CMRR 57
7.3 Coaxial Cable Compatibility Interface Specifications 58
7.3.1 Coaxial Cable Component Specifications 58
7.3.1.1 Coaxial Cable Parameters 58
7.3.1.1.1 Characteristic Impedance 58
7.3.1.1.2 Attenuation 58
7.3.1.1.3 Velocity of Propagation 58
7.3.1.1.4 Mechanical Requirements 59
7.3.1.1.5 Pulse Distortion 59
7.3.1.1.6 Jacket Marking 59
7.3.1.1.7 Transfer Impedance 59
7.3.1.2 Coaxial Cable Connectors 60
7.3.1.3 Coaxial Cable Terminators 60
7.3.1.4 Transceiver-to-Coaxial Cable Connections 61
7.3.2 Coaxial Cable Signaling 61
7.4 Transceiver Specifications 62
7.4.1 Transceiver-to-Coaxial Cable Interface 62
7.4.1.1 Input Impedance 62
7.4.1.2 Bias Current 62
7.4.1.3 Transmit Output Levels 63
7.4.2 Transceiver-to-Transceiver Cable Interface 63
7.4.2.1 Transmit Pair 63
7.4.2.2 Receive Pair 63
7.4.2.3 Collision Presence Pair 64
7.4.2.4 Power Pair 64
7.4.3 Electrical Isolation 64

Amazon
Exhibit 1005
Page 175

7.4.4 Reliability 64
7.5 Channel Logic 64
7.5.1 Channel Encoding 64
7.5.1.1 Encoder 65
7.5.1.2 Decoder 65
7.5.1.3 Preamble Generation 66
7.5.2 Collision Detect Signal 67
7.5.3 Carrier Sense Signal 67
7.5.4 Channel Framing 68
7.5.4.1 Beginning-of-Frame Sequence 68
7.5.4.2 End-of-Frame Sequence 69
7.6 Channel Configuration Requirements 69
7.6.1 Cable Sectioning 69
7.6.2 Transceiver Placement 70
7.6.3 System Grounding 70
7.6.4 Repeaters 71
7.6.4.1 Carrier Detect and Transmit Repeat 71
7.6.4.2 Collision Detect and Collision Repeat 71
7.6.4.3 Repeater Signal Generation 71
7.6.4.3.1 Signal Amplification 71
7.6.4.3.2 Signal Timing 71
7.7 Environment Specifications 72
7.7.1 Electromagnetic Environment 72
7.7.2 Temperature and Humidity 72

Appendices

APPENDIX A: GLOSSARY 73
APPENDIX B: ASSIGNMENT OF ADDRESS AND TYPE VALUES 76
APPENDIX C: CRC IMPLEMENTATION 77
APPENDIX D: IMPLEMENTATION OF TRANSCEIVER CABLE DRIVER AND RECEIVER 80
APPENDIX E: INTERFRAME RECOVERY 82

Figures and Tables

Figure 4-1: Ethernet Architecture and Typical Implementation 7
Figure 4-2: Architectural Layering 8
Figure 4-3: Data Link Layer Functions 10
Figure 4-4: Physical Layer Functions 11
Figure 6-1: Data Link Layer Frame Format 20
Figure 6-2: Structure of the Data Link Procedural Model 31
Figure 6-3: Control Flow Summary -- Client Layer Processes 32
Figure 6-4: Control Flow Summary -- Data Link Layer Processes 33
Figure 7-1: Physical Channel Configurations 50
Table 7-1: Physical Channel Propagation Delay Budget 52
Figure 7-2: Maximum Transceiver Cable Transfer Impedance 55
Figure 7-3: Typical Transceiver Cable Waveform 57
Figure 7-4: Maximum Coaxial Cable Transfer Impedance 60
Figure 7-5: Typical Coaxial Cable Waveform 62
Figure 7-6: Manchester Encoding 65
Figure 7-7: Preamble Encoding 66
Figure 7-8: Functional Logic of **collisionDetect** Signal 67
Figure 7-9: Functional Logic of **carrierSense** Signal 68
Figure C-1: CRC Implementation 78
Figure D-1: Typical Transceiver Cable Driver 80
Figure D-2: Typical Transceiver Cable Receiver 81

Amazon
Exhibit 1005
Page 176

Amazon
Exhibit 1005
Page 177

1. INTRODUCTION

The Ethernet local area network provides a communication facility for high speed data exchange among computers and other digital devices located within a moderate-sized geographic area. Its primary characteristics include:

Physical Layer:

Data rate: 10 Million bits/sec

Maximum station separation: 2.5 Kilometers

Maximum number of stations: 1024

Medium: Shielded coaxial cable, base-band signalling

Topology: Branching non-rooted tree

Data Link Layer:

Link control procedure: Fully distributed peer protocol, with statistical contention resolution (CSMA/CD)

Message protocol: Variable size frames, "best-effort" delivery

The Ethernet, like other local area networks, falls in a middle ground between long distance, low speed networks which carry data for hundreds or thousands of kilometers, and specialized, very high speed interconnections which are generally limited to tens of meters. The Ethernet is intended primarily for use in such areas as office automation, distributed data processing, terminal access, and other situations requiring economical connection to a local communication medium carrying bursty traffic at high peak data rates. Use in situations demanding resistance to hostile environments, real-time response guarantees, and so on, while not specifically excluded, do not constitute the primary environment for which the Ethernet is designed.

The precursor to the Ethernet specified in this document was the "Experimental Ethernet", designed and implemented by Xerox in 1975, and used continually since that time by thousands of stations. The Ethernet defined here builds on that experience, and on the larger base of the combined experience of Digital, Intel, and Xerox in many forms of networking and computer interconnection.

In specifying the Ethernet, this document provides precise detailed definitions of the lowest two layers of an overall network architecture. It thus defines what is generally

referred to as a *link-level* facility. It does not specify the higher level protocols needed to provide a complete network architecture. Such higher level protocols would generally include such functions as internetwork communication, error recovery, flow control, security measures (e.g. encryption), and other higher level functions that increase the power of the communication facility and/or tailor it to specific applications. In particular, it should be noted that all error recovery functions have been relegated to higher level protocols, in keeping with the low error rates that characterize local networks.

One of the main objectives of this specification is *compatibility*. As stated in Section 3, it is intended that *every implementation of the Ethernet be able to exchange data with every other implementation*. It should be noted that higher level protocols raise their own issues of compatibility over and above those addressed by the Ethernet and other link-level facilities. This does not eliminate the importance of link-level compatibility, however. While the compatibility provided by the Ethernet does not guarantee solutions to higher level compatibility problems, it does provide a context within which such problems can be addressed, by avoiding low level incompatibilities that would make direct communication impossible.

2. REFERENCES

The following three papers describe the Experimental Ethernet, and are reprinted in: "The Ethernet Local Network: Three Reports," Xerox Palo Alto Research Center Technical Report CSL-80-2. (February, 1980.)

[1] Metcalfe, R. M. and Boggs, D. R., "Ethernet: Distributed Packet Switching for Local Computer Networks," *Communications of the ACM 19* 7 (July 1976).

[2] Crane, R. C. and Taft, E. A. "Practical Considerations in Ethernet Local Network Design," Presented at *Hawaii International Conference on System Sciences* (January, 1980).

[3] Shoch, J. F. and Hupp, J. A. "Measured Performance of an Ethernet Local Network," Presented at *Local Area Communications Network Symposium* Boston (May 1979).

The following references describe the ISO Open Systems Model:

[4] Zimmermann, H., "OSI Reference Model – The ISO Model of Architecture for Open Systems Interconnection," *IEEE Transactions on Communication COM-28* 4 (April 1980).

[5] International Organization for Standardization (ISO), "Reference Model of Open Systems Interconnection," *Document no. ISO/TC97/SC16 N227* (June 1979).

The following references describe the Pascal language (used in the Data Link Layer procedural model) and its derivative Concurrent Pascal:

[6] Jensen, K. and Wirth, N., *Pascal User Manual and Report, 2nd Edition.* Springer-Verlag (1974).

[7] Brinch Hansen, P., *Concurrent Pascal Report.* Technical Report CIT-IS-TR 17, California Institute of Technology (1975).

The following references discuss the CRC code used for the frame check sequence:

[8] Hammond, J. L., Brown, J. E. and Liu, S. S., "Development of a Transmission Error Model and an Error Control Model," Technical Report RADC-TR-75-138, Rome Air Development Center (1975).

[9] Bittel, R., "On Frame Check Sequence (FCS) Generation and Checking.," ANSI working paper X3-S34-77-43, (1977).

Amazon
Exhibit 1005
Page 178

3. GOALS AND NON-GOALS

This section states the assumptions underlying the design of the Ethernet.

3.1 Goals

The goals of the Ethernet design are:

Simplicity: Features which would complicate the design without substantially contributing to the meeting of the other goals have been excluded.

Low cost: Since technological improvements will continue to reduce the overall cost of stations wishing to connect to the Ethernet, the cost of the connection itself should be minimized.

Compatibility: All implementations of the Ethernet should be capable of exchanging data at the data link level. For this reason, the specification avoids optional features, to eliminate the possibility of incompatible variants of the Ethernet.

Addressing flexibility: The addressing mechanisms should provide the capability to target frames to a single node, a group of nodes, or to all nodes on the network.

Fairness: All nodes should have equal access to the network when averaged over time.

Progress: No single node operating in accordance with the protocol should be able to prevent the progress of other nodes.

High speed: The network should operate efficiently at a data rate of 10 Megabits per second.

Low delay: At any given level of offered traffic, the network should introduce as little delay as possible in the transfer of a frame.

Stability: The network should be stable under all load conditions, in the sense that the delivered traffic should be a monotonically non-decreasing function of the total offered traffic.

Maintainability: The Ethernet design should allow for network maintenance, operation, and planning.

Layered architecture: The Ethernet design should be specified in layered terms to separate the logical aspects of the data link protocol from the physical details of the communication medium.

3.2 Non-Goals

The following are *not* goals of the Ethernet design:

Full duplex: At any given instant, the Ethernet can transfer data from one source station to one or more destination stations. Bi-directional communication is provided by rapid exhange of frames, rather than full duplex operation.

Error control: Error handling at the data link level is limited to detection of bit errors in the physical channel, and the detection and recovery from collisions. Provision of a complete error control facility to handle detected errors is relegated to higher layers of the network architecture.

Security: The data link protocol does not employ encryption or other mechanisms to provide security. Higher layers of the network architecture may provide such facilities as appropriate.

Speed flexibility: This specification defines a physical channel operating at a single fixed data rate of 10 Megabits per second.

Priority: The data link protocol provides no support of priority station operation.

Hostile user: There is no attempt to protect the network from a malicious user at the data link level.

Amazon
Exhibit 1005
Page 179

4. FUNCTIONAL MODEL OF THE ETHERNET ARCHITECTURE

There are two important ways to view the Ethernet design, corresponding to:

Architecture, emphasizing the logical divisions of the system, and how they fit together.

Implementation, emphasizing the actual components, and their packaging and interconnection.

Figure 4-1 illustrates these two views as they apply to a typical implementation, showing how each view groups the various functions.

This document is organized along *architectural* lines, emphasizing the large-scale separation of the Ethernet system into two parts: the *Data Link Layer* and the *Physical Layer*. These layers are intended to correspond closely to the lowest layers of the ISO Model for Open Systems Interconnection [4,5]. Architectural organization of the specification has two main advantages:

Clarity: A clean overall division of the design along architectural lines makes the specification clearer.

Flexibility: Segregation of medium-dependent aspects in the Physical Layer allows the Data Link Layer to apply to transmission media other than the specified coaxial cable.

As is evident in Figure 4-1, the architectural model is based on a set of interfaces different from those emphasized in the implementations. One crucial aspect of the design, however, must be addressed largely in terms of the implementation interfaces: *compatibility*. Two important compatibility interfaces are defined within what is architecturally the Physical Layer:

Coaxial cable interface: To communicate via the Ethernet, all stations must adhere rigidly to the exact specification of coaxial cable signals defined in this document, and to the procedures which define correct behavior of a station. The medium-independent aspects of the Data Link Layer should not be taken as detracting from this point: *communication via the Ethernet requires complete compatibility at the coaxial cable interface.*

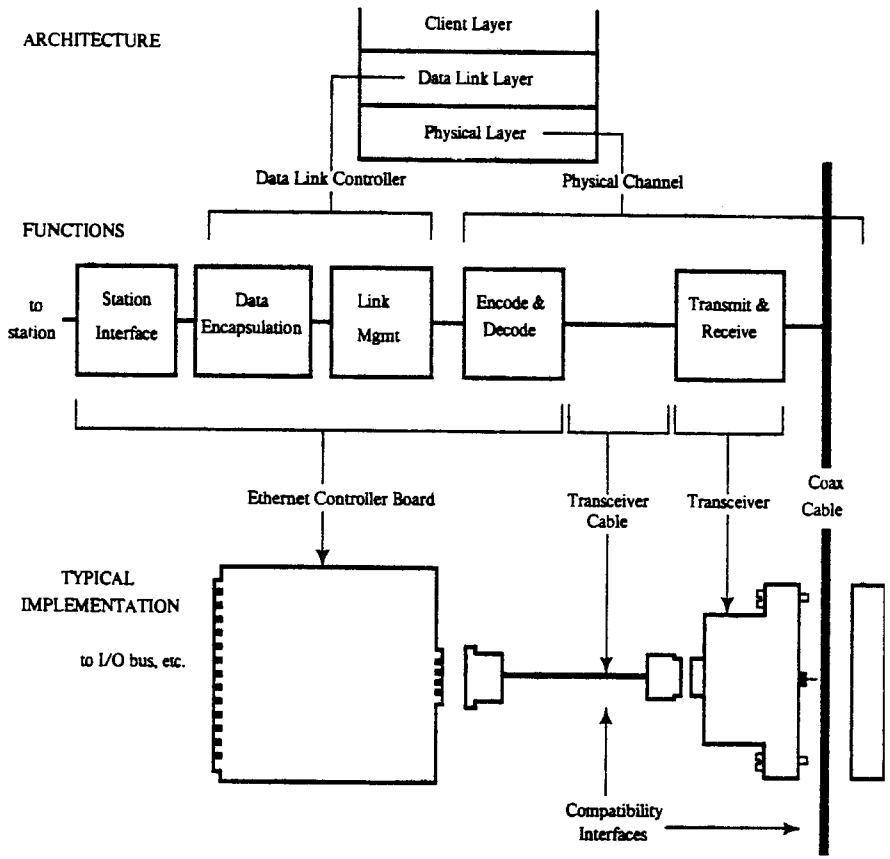


Figure 4-1: Ethernet Architecture and Typical Implementation

Amazon
Exhibit 1005
Page 180

Transceiver cable interface: It is anticipated that most stations will be located some distance away from their connection to the coaxial cable. While it is necessary to place a small amount of circuitry (the *transceiver*) directly adjacent to the coaxial cable, the majority of the electronics (the *controller*) can and should be placed with the station. Since it is desirable for the same transceiver to be usable with a wide variety of stations, a second compatibility interface, the *transceiver cable interface*, is defined. While conformance with this interface is not strictly necessary to insure communication, it is highly recommended, since it allows maximum flexibility in intermixing transceivers and stations.

4.1 Layering

The major division in the Ethernet Architecture is between the Physical Layer and the Data Link Layer, corresponding to the lowest two levels in the ISO model. The higher levels of the overall network architecture, which use the Data Link Layer, will be collectively referred to in this document as the "Client Layer" since, strictly speaking, the identity and function of higher level facilities are outside the scope of this specification. The intent, however, is that the Ethernet Physical and Data Link Layers support the higher layers of the ISO model (Network Layer, Transport Layer, etc.).

The overall structure of the layered architecture is shown in Figure 4-2.

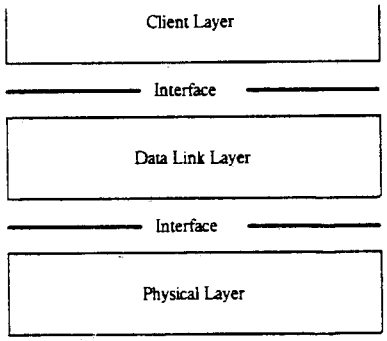


Figure 4-2: Architectural Layering

Amazon
Exhibit 1005
Page 181

In the architectural model used here, the layers interact via well defined interfaces.

The interface between the Client Layer and the Data Link Layer includes facilities for transmitting and receiving frames, and provides per-operation status information for use by higher-level error recovery procedures.

The interface between the Data Link Layer and the Physical Layer includes signals for framing (carrier sense, transmit initiation) and contention resolution (collision detect), facilities for passing a pair of serial bit streams (transmit, receive) between the two layers, and a wait function for timing.

These interfaces are described more precisely in Section 5.

As mentioned in the preface, additional interfaces are necessary to allow a higher level network management facility to interact with the Data Link Layer and Physical Layer to perform operation, maintenance and planning functions.

4.2 Data Link Layer

The Data Link Layer defines a medium-independent link level communication facility, built on the medium-dependent physical channel provided by the Physical Layer. It is applicable to a general class of local area broadcast media suitable for use with the channel access discipline known as carrier-sense multiple-access with collision-detection (CSMA-CD). Compatibility with non-contention media (e.g., switched lines, token-passing rings, etc.), while a worthwhile topic for further research, is not addressed in this specification.

The Data Link Layer specified here is intended to be as similar as possible to that described in the ISO model. In a broadcast network like the Ethernet, the notion of a data link between two network entities does not correspond directly to a distinct physical connection. Nevertheless, the two main functions generally associated with a data link control procedure are present:

Data encapsulation
- framing (frame boundary delimitation)
- addressing (handling of source and destination addresses)
- error detection (detection of physical channel transmission errors)

Link management
- channel allocation (collision avoidance)
- contention resolution (collision handling)

This split is reflected in the division of the Data Link Layer into the Data Encapsulation sub-layer and the Link Management sub-layer, as shown in Figure 4-3.

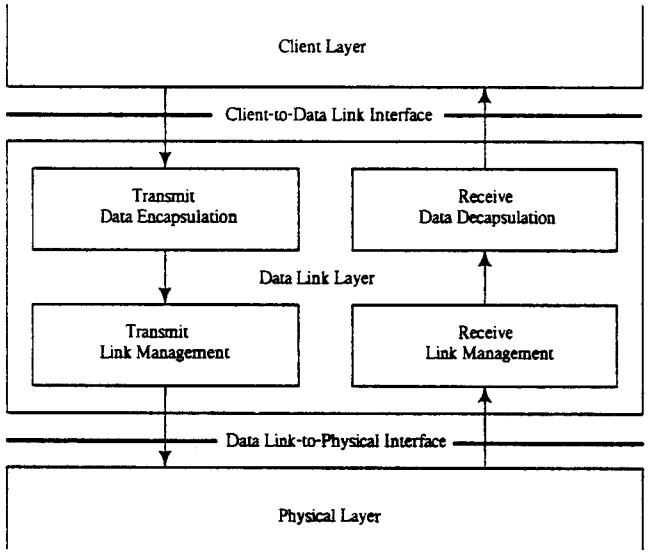


Figure 4-3: Data Link Layer Functions

In terms of the ISO model, the Ethernet Data Link Layer provides a multi-endpoint connection between higher-layer entities wishing to communicate. The connection provided is called a *data link*, and is implemented between two or more Data Link Layer entities called *data link controllers* via a Physical Layer connection called the *physical channel.*

4.3 Physical Layer

The Physical Layer specified in this document provides a 10 MBit/sec physical channel through a coaxial cable medium. Because one purpose of the layered architecture is to insulate the Data Link Layer from the medium-specific aspects of the channel, the Physical Layer completely specifies the essential physical characteristics of the Ethernet, such as data encoding, timing, voltage levels, etc. Implementation details are left unspecified, to retain maximum flexibility for the implementor. In all cases, the criterion applied in distinguishing between essential characteristics and implementation details is *guaranteed compatibility:* any two correct implementations of the Physical Layer specified here will be capable of exchanging data over the coaxial cable, enabling communication between their

respective stations at the Data Link Layer.

The Physical Layer defined in this specification performs two main functions generally associated with physical channel control:

Data encoding

- preamble generation/removal (for synchronization)
- bit encoding/decoding (between binary and phase-encoded form)

Channel access

- bit transmission/reception (of encoded data)
- carrier sense (indicating traffic on the channel)
- collision detection (indicating contention on the channel)

This split is reflected in the division of the Physical Layer into the Data Encoding sub-layer and the Channel Access sub-layer, as shown in Figure 4-4.

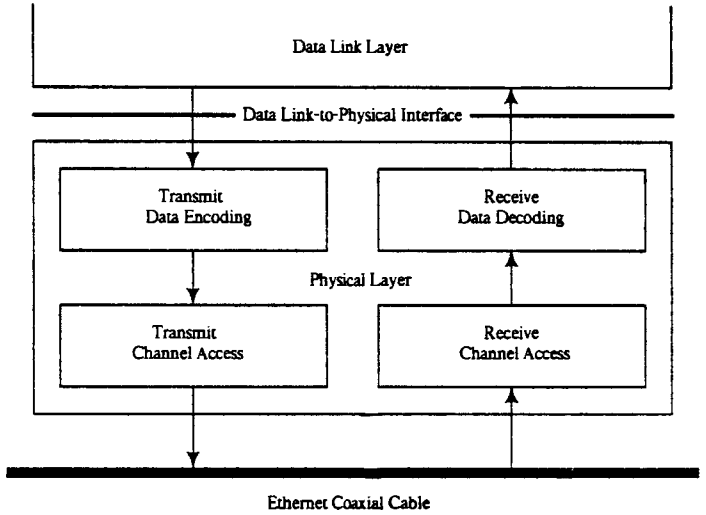


Figure 4-4: Physical Layer Functions

Amazon
Exhibit 1005
Page 182

4.4 Ethernet Operation and the Functional Model

This section provides an overview of frame transmission and reception in terms of the functional model of the architecture. This overview is descriptive, rather than definitional; the formal specifications of the operations described here are given in Sections 6 and 7.

4.4.1 Transmission Without Contention

When the Client Layer requests the transmission of a frame, the Transmit Data Encapsulation component of the Data Link Layer constructs the frame from the client-supplied data and appends a frame check sequence to provide for error detection. The frame is then handed to the Transmit Link Management component for transmission.

Transmit Link Management attempts to avoid contention with other traffic on the channel by monitoring the *carrier sense* signal and *deferring* to passing traffic. When the channel is clear, frame transmission is initiated (after a brief interframe delay to provide recovery time for other data link controllers and for the physical channel). The Data Link Layer then provides a serial stream of bits to the Physical Layer for transmission.

The Data Encoding component of the Physical Layer, before sending the actual bits of the frame, sends an encoded preamble to allow the receivers and repeaters along the channel to synchronize their clocks and other circuitry. It then begins translating the bits of the frame into encoded form and passes them to the Channel Access component for actual transmission over the medium.

The Channel Access component performs the task of actually generating the electrical signals on the medium which represent the bits of the frame. Simultaneously, it monitors the medium and generates the *collison detect* signal, which, in the contention-free case under discussion, remains off for the duration of the frame.

When transmission has completed without contention, the Data Link Layer so informs the Client Layer and awaits the next request for frame transmission.

4.4.2 Reception Without Contention

At the receiving station, the arrival of a frame is first detected by the Receive Channel Access component of the Physical Layer, which responds by synchronizing with the incoming preamble, and by turning on the carrier sense signal. As the encoded bits arrive from the medium, they are passed to the Receive Data Decoding component.

Amazon
Exhibit 1005
Page 183

Receive Data Decoding translates the encoded signal back into binary data and discards the leading bits, up to and including the end of the preamble. It then passes subsequent bits up to the Data Link Layer.

Meanwhile, the Receive Link Management component of the Data Link Layer, having seen carrier sense go on, has been waiting for the incoming bits to be delivered. Receive Link Management collects bits from the Physical Layer as long as the carrier sense signal remains on. When the carrier sense signal goes off, the frame is passed to Receive Data Decapsulation for processing.

Receive Data Decapsulation checks the frame's destination address field to decide whether the frame should be received by this station. If so, it passes the contents of the frame to the Client Layer along with an appropriate status code. The status code is generated by inspecting the frame check sequence to detect any damage to the frame enroute, and by checking for proper octet-boundary alignment of the end of the frame.

4.4.3 Collisions: Handling of Contention

If multiple stations attempt to transmit at the same time, it is possible for their transmitting data link controllers to interfere with each others' transmissions, in spite of their attempts to avoid this by deferring. When two stations' transmissions overlap, the resulting contention is called a *collision*. A given station can experience a collision during the initial part of its transmission (the "collision window"), before its transmitted signal has had time to propagate to all parts of the Ethernet channel. Once the collision window has passed, the station is said to have *acquired* the channel; subsequent collisions are avoided, since all other (properly functioning) stations can be assumed to have noticed the signal (via carrier sense) and to be deferring to it. The time to acquire the channel is thus based on the *round-trip propagation time* of the physical channel.

In the event of a collision, the Transmit Channel Access component of a transmitting station's Physical Layer first notices the interference on the channel and turns on the collision detect signal. This is noticed in turn by the Transmit Link Management component of the Data Link Layer, and collision handling begins. First, Transmit Link Management *enforces* the collision by transmitting a bit sequence called the *jam*. This insures that the duration of the collision is sufficient to be noticed by the other transmitting station(s) involved in the collision. After the jam is sent, Transmit Link Management terminates the transmission and schedules a retransmission attempt for a randomly selected time in the near future. Retransmission is attempted repeatedly in the face of repeated collisions. Since repeated collisions indicate a busy channel, however, Transmit Link Management attempts to adjust to the channel load by *backing off* (voluntarily delaying its own retransmissions to reduce its load on the channel). This is accomplished by

expanding the interval from which the random retransmission time is selected on each retransmission attempt. Eventually, either the transmission succeeds, or the attempt is abandoned on the assumption that the channel has failed or has become overloaded.

At the receiving end, the bits resulting from a collision are received and decoded by the Physical Layer just as are the bits of a valid frame. In particular, collisions do *not* turn on the receiving station's collision detect signal, which is generated only during transmission. Instead, the fragmentary frames received during collisions are distinguished from valid frames by the Data Link's Receive Link Management component, by noting that a collision fragment is always smaller than the shortest valid frame. Such fragments are discarded by Receive Link Management.

5. INTER-LAYER INTERFACES

The purpose of this section is to provide precise definitions of the interfaces between the architectural layers defined in Section 4. In order to provide such a definition, some precise notation must be adopted. The notation used here is the Pascal language, in keeping with the procedural nature of the formal Data Link Layer specification (see 6.5). Each interface is thus described as a set of procedures and/or shared variables which collectively provide the only valid interactions between layers. The accompanying text describes the meaning of each procedure or variable and points out any implicit interactions among them.

Note that the description of the interfaces in Pascal is a notational technique, and in no way implies that they can or should be implemented in software. This point is discussed more fully in 6.5, which provides complete Pascal declarations for the data types used in the remainder of this section. Note also that the "synchronous" (one frame at a time) nature of the frame transmission and reception operations is a property of the architectural interface between the Client Layer and the Data Link Layer, and need not be reflected in the implementation interface between a station and its controller.

5.1 Client Layer to Data Link Layer

The two primary services provided to the Client Layer by the Data Link Layer are transmission and reception of frames. The interface through which the Client Layer uses the facilities of the Data Link Layer therefore consists of a pair of functions.

Functions:
TransmitFrame
ReceiveFrame

Each of these functions has the components of a frame as its parameters (input or output), and returns a status code as its result.

The Client Layer transmits a frame by invoking *TransmitFrame*:

```
function TransmitFrame (
  destinationParam: AddressValue;
  sourceParam: AddressValue;
  typeParam: TypeValue;
  dataParam: DataValue): TransmitStatus;
```

The *TransmitFrame* operation is synchronous, in the sense that its duration is the entire attempt to transmit the frame, so that when the operation completes, transmission has either succeeded or failed, as indicated by the resulting status code:

```
type TransmitStatus = (transmitOK, excessiveCollisionError);
```


Amazon
Exhibit 1005
Page 184

Successful transmission is indicated by the status code *transmitOK*; the code *excessiveCollisionError* indicates that the transmission attempt was aborted due to excessive collisions, because of heavy traffic or a network failure. Implementations may define additional implementation-dependent status codes if necessary.

The Client Layer accepts incoming frames by invoking *ReceiveFrame*:

```
function ReceiveFrame (
  var destinationParam: AddressValue;
  var sourceParam: AddressValue;
  var typeParam: TypeValue;
  var dataParam: DataValue): ReceiveStatus;
```

The *ReceiveFrame* operation is synchronous, in the sense that the operation does not complete until a frame has been received. The fields of the frame are delivered via the output parameters, along with a status code:

```
type ReceiveStatus = (receiveOK, frameCheckError, alignmentError);
```

Successful reception is indicated by the status code *receiveOK*. The code *frameCheckError* indicates that the frame received was damaged by a transmission error in the physical channel. The code *alignmentError* indicates that the frame received was damaged, and that in addition, its length was not an integral number of octets. Implementations may define additional implementation-dependent status codes if necessary.

5.2 Data Link Layer to Physical Layer

The interface through which the Data Link Layer uses the facilities of the Physical Layer consists of a function, a pair of procedures and three Boolean variables.

Function:	Variables:
ReceiveBit	*collisionDetect*
Procedures:	*carrierSense*
TransmitBit	*transmitting*
Wait	

During transmission, the contents of an outgoing frame are passed from the Data Link Layer to the Physical Layer via repeated use of the *TransmitBit* operation:

```
procedure TransmitBit (bitParam: Bit);
```

Each invocation of *TransmitBit* passes one new bit of the outgoing frame to the Physical Layer. The *TransmitBit* operation is synchronous, in the sense that the duration of the operation is the entire transmission of the bit, so that when the operation completes, the Physical Layer is ready to accept the next bit immediately. (Note: this does not imply that all invocations of *TransmitBit* are of exactly equal duration; for example, if the Physical Layer must perform some initial processing -- e.g., preamble generation -- before transmitting the first bit of a frame, the first

Amazon
Exhibit 1005
Page 185

invocation of *TransmitBit* may take significantly longer.)

The overall event of data being transmitted is signaled to the Physical Layer via the variable *transmitting*:

```
var transmitting: Boolean;
```

Before sending the first bit of a frame, the Data Link Layer sets *transmitting* to *true*, to inform the Physical Link that a stream of bits will be presented via the *TransmitBit* operation. After the last bit of the frame has been presented, the Data Link Layer sets *transmitting* to *false* to indicate the end of the frame.

The presence of a collision in the physical channel is signaled to the Data Link Layer via the variable *collisionDetect*:

```
var collisionDetect: Boolean;
```

The *collisionDetect* signal remains *true* during the duration of the collision. (Note: Since an entire collision may occur during the first invocation of *TransmitBit* -- e.g., during preamble removal -- the Data Link Layer must handle this possibility by monitoring *collisionDetect* concurrently with its transmission of outgoing bits. See 6.5 for details.)

The *collisionDetect* signal is generated only during transmission and is never *true* at any other time; in particular, it cannot be used during frame reception to detect collisions between overlapping transmissions from two or more other stations.

During reception, the contents of an incoming frame are retrieved from the Physical Layer by the Data Link Layer via repeated use of the *ReceiveBit* operation:

```
function ReceiveBit: Bit;
```

Each invocation of *ReceiveBit* retrieves one new bit of the incoming frame (i.e., not including any preamble bits) from the Physical Layer. The *ReceiveBit* operation is synchronous, in the sense that its duration is the entire reception of a single bit. (As with *TransmitBit*, the first invocation of *ReceiveBit* make take significantly longer -- e.g., due to preamble removal). Upon receiving a bit, the Data Link Layer must immediately request the next bit until all bits of the frame have have been received. (See 6.5 for details.)

The overall event of data being received is signaled to the Data Link Layer via the variable *carrierSense*:

```
var carrierSense: Boolean;
```

When the Physical Layer sets *carrierSense* to *true*, the Data Link Layer must immediately begin retrieving the incoming bits via the *ReceiveBit* operation. When *carrierSense* subsequently becomes *false*, the Data Link Layer can begin processing the received bits as a completed frame. Note that the *true/false*

transitions of *carrierSense* are not defined to be precisely synchronized with the beginning and end of the frame, but may precede the beginning and lag the end, respectively. If an invocation of *ReceiveBit* is pending when *carrierSense* becomes *false*, *ReceiveBit* returns an undefined value, which should be discarded by the Data Link Layer. (See 6.5 for details.)

The Data Link Layer must also monitor the value of *carrierSense* to defer its own transmissions when the channel is busy.

The Physical Layer also provides the procedure *Wait*:

```
procedure Wait (bitTimes: integer);
```

This procedure waits for the specified number of bit times. This allows the Data Link Layer to measure time intervals in units of the (physical-channel-dependent) bit time.

Another important property of the Physical Layer which is an implicit part of the interface presented to the Data Link Layer is the *round-trip propagation time* of the physical channel. This figure represents the maximum time required for a signal to propagate from one end of the network to the other, and for a collision to propagate back. The round-trip propagation time is primarily (but not entirely) a function of the physical size of the network. The round-trip propagation time of the Physical Layer is defined to be at most 450 bit times (see 7.1.2).

Amazon
Exhibit 1005
Page 186

6. ETHERNET DATA LINK LAYER SPECIFICATION

6.1 Data Link Layer Overview and Model

As defined in Section 4, the Ethernet Architecture consists of the Data Link Layer, and below it, the Physical Layer. Furthermore, the Data Link Layer is divided into two sub-layers (see Figure 4-3).

Data encapsulation
- framing
- addressing
- error detection

Link management
- channel allocation
- contention resolution

This model is used throughout this section to structure the detailed specification of the Data Link Layer. An English description of the Data Link Layer is given in 6.2, 6.3, and 6.4. A more precise algorithmic definition is given in 6.5, which provides a procedural model for the Data Link Layer in the form of a program in the language Pascal. Note that whenever there is any apparent ambiguity concerning the definition of some aspect of the Data Link Layer, it is the Pascal procedural specification in 6.5 which should be consulted for the definitive statement.

6.2 Frame Format

The data encapsulation function of the Data Link Layer comprises the construction and processing of frames. The subfunctions of framing, addressing, and error detection are reflected in the frame format as follows:

Framing: No explicit framing information is needed, since the necessary framing cues (*carrierSense* and *transmitting*) are present in the interface to the Physical Layer.

Addressing: Two address fields are provided to identify the source and destination stations for the frame.

Error detection: A Frame Check Sequence field is provided for detection of transmission errors.

Figure 6-1 shows the five fields of a frame: the addresses of the frame's source and destination, a type field for use by higher layers (see 6.2.2), a data field containing the transmitted data, and the frame check sequence field containing a cyclic redundancy check value to detect transmission errors. Of these five fields, all are of fixed size except the data field, which may contain any integral number of octets between the minimum and maximum values specified below (see 6.2.5).

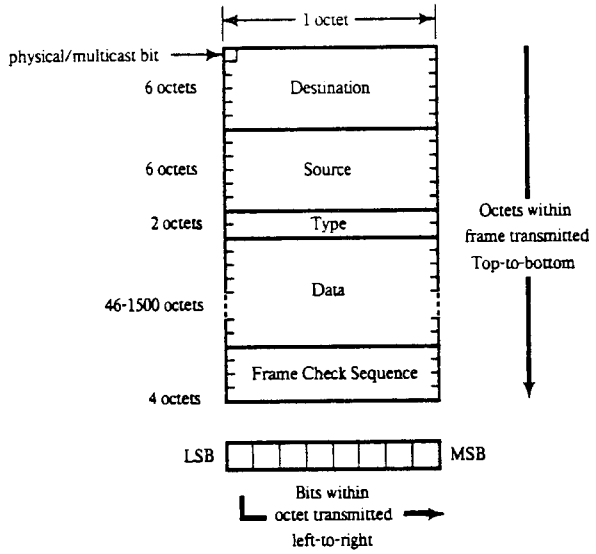


Figure 6-1: Data Link Layer Frame Format

Relative to Figure 6-1, the octets of a frame are transmitted from top to bottom, and the bits of each octet are transmitted from left to right.

NOTE

This document does *not* define an order of transmission for the octets of standard multi-octet data types (strings, integers, etc), since no values of such data types appear in the data link frame format. The order in which implementations of the Ethernet store the octets of a frame in computer memory, and the manner in which higher level protocols interpret the contents of the data field as values of various multi-octet data types, are beyond the scope of this specification.

The Ethernet itself is also totally insensitive to the interpretation of bits within a octet as constituting the digits of an 8-digit binary numeric value. Since some uniform convention is helpful, however, in avoiding needless incompatibility among different station types, the interpretation is arbitrarily defined to be that the left-most bit (first transmitted) is the low-order (2^0) digit and the right-most bit (last transmitted) is the high-order (2^7) digit.

Amazon
Exhibit 1005
Page 187

6.2.1 Address Fields

Data link addresses are 6 octets (48 bits) in length. A data link address is of one of two types:

Physical address: The unique address associated with a particular station on the Ethernet. A station's physical address should be distinct from the physical address of any other station on *any* Ethernet.

Multicast address: A multi-destination address, associated with one or more stations on a given Ethernet. There are two kinds of multicast address:

- *Multicast-group address:* An address associated by higher-level convention with a group of logically related stations.
- *Broadcast address:* A distinguished, predefined multicast address which always denotes the set of *all* stations on a given Ethernet.

The first bit of a data link address distinguishes physical from multicast addresses:

0 ⇒ physical address

1 ⇒ multicast address

In either case, the remainder of the first octet and all of the subsequent octets form a 47-bit pattern. In the case of the broadcast address, this pattern consists of 47 one-bits. There is no standard "null" address value.

The procedures for assigning suitably unique values for physical and multicast addresses are discussed in Appendix B.

6.2.1.1 Destination Address Field

The destination address field specifies the station(s) for which the frame is intended. It may be a physical or multicast (including broadcast) address. For details of address recognition by the receiving station(s), see 6.4.1.2.

6.2.1.2 Source Address Field

The source address field specifies the station sending the frame. The source address field is not interpreted at the Data Link Layer. It is specified at the data link level because a uniform convention for the placement of this field is crucial for most higher level protocols.

6.2.2 Type Field

The type field consists of a two-octet value reserved for use by higher levels (in particular, to identify the Client Layer protocol associated with the frame). The type field is uninterpreted at the Data Link Layer. It is specified at this level because a uniform convention for the placement and value assignment of this field is crucial if multiple higher level protocols are to be able to share the same Ethernet network without conflict. Appendix B discusses the assignment of type field values.

6.2.3 Data Field

The data field contains a sequence of n octets, where $46 \leq n \leq 1500$ Within this range, full data transparency is provided, in the sense that any arbitrary sequence of octet values may appear in the data field.

6.2.4 Frame Check Sequence Field

The frame check sequence (FCS) field contains a 4-octet (32-bit) cyclic redundancy check (CRC) value. This value is computed as a function of the contents of the source, destination, type and data fields (i.e., all fields except the frame check sequence field itself). The encoding is defined by the generating polynomial:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

(This polynomial is also used in the Autodin-II network; its properties are investigated in [8].)

Mathematically, the CRC value corresponding to a given frame is defined by the following procedure:

1. The first 32 bits of the frame are complemented.
2. The n bits of the frame are then considered to be the coefficients of a polynomial M(x) of degree n-1. (The first bit of the destination address field corresponds to the x^{n-1} term and the last bit of the data field corresponds to the x^0 term.)
3. M(x) is multiplied by x^{32} and divided by G(x), producing a remainder R(x) of degree ≤ 31.
4. The coefficients of R(x) are considered to be a 32-bit sequence.
5. The bit sequence is complemented and the result is the CRC.

The 32 bits of the CRC value are placed in the frame check sequence field so that the x^{31} term is the leftmost bit of the first octet, and the x^0 term is the rightmost bit of the last octet. (The bits of the CRC are thus transmitted in the order x^{31}, x^{30},...,x^1, x^0)

Appendix C discusses CRC implementation issues.

6.2.5 Frame Size Limitations

Given the limitations on the size of the data field specified in 6.2.3 and the 18 octet total size for the other four fields, the smallest valid frame contains 64 octets and the largest valid frame contains 1518 octets.

Amazon
Exhibit 1005
Page 188

6.3 Frame Transmission

The Data Link frame transmission and reception are as follows:

Frame transmission includes data encapsulation and link management aspects:

Transmit Data Encapsulation includes the assembly of the outgoing frame (from the values provided by the Client Layer) and frame check sequence generation.

Transmit Link Management includes carrier deference, interframe spacing, collision detection and enforcement, and collision backoff and retransmission.

The performance of these functions by a transmitting data link controller interacts with corresponding actions by other data link controllers to jointly implement the Ethernet data link protocol.

6.3.1 Transmit Data Encapsulation

6.3.1.1 Frame Assembly

The fields of the data link frame are set to the values provided by the Client Layer as arguments to the *TransmitFrame* operation (see 5.1), with the exception of the frame check sequence, which is set to the CRC value generated by the data link controller.

6.3.1.2 Frame Check Sequence Generation

The CRC value defined in 6.2.4 is generated and inserted in the frame check sequence field, following the fields supplied by the Client Layer. Appendix C discusses CRC implementation.

6.3.2 Transmit Link Management

6.3.2.1 Carrier Deference

Even when it has nothing to transmit, the data link controller monitors the physical channel for traffic by watching the *carrierSense* signal provided by the Physical Layer. Whenever the channel is busy, the data link controller *defers* to the passing frame by delaying any pending transmission of its own. After the last bit of the passing frame (i.e., when *carrierSense* changes from true to false), the data link controller continues to defer for 9.6 μsec to provide proper interframe spacing (see 6.3.2.2). At the end of that time, if it has a frame waiting to be transmitted, transmission is initiated independent of the value of *carrierSense*. When transmission has completed (or immediately, if there was nothing to transmit) the data link controller resumes its original monitoring of *carrierSense*.

When a frame is submitted by the Client Layer for transmission, the transmission is initiated as soon as possible, but in conformance with the rules of deference stated above.

6.3.2.2 Interframe Spacing

As defined in 6.3.2.1, the rules for deferring to passing frames insure a minimum interframe spacing of 9.6 μsec. This is intended to provide interframe recovery time for other data link controllers and for the physical channel.

Note that 9.6 μsec is the minimum value of the interframe spacing. If necessary for implementation reasons, a transmitting controller may use a larger value with a resulting decrease in its throughput. The value should not exceed 10.6 μsec.

6.3.2.3 Collision Handling

Once a data link controller has finished deferring and has started transmission, it is still possible for it to experience contention for the channel. As discussed in 4.4.3, collisions can occur until acquisition of the network has been accomplished through the deference of all other stations' data link controllers.

The dynamics of collision handling are largely determined by a single parameter called the *slot time*. This single parameter describes three important aspects of collision handling:

- It is an upper bound on the acquisition time of the network.
- It is an upper bound on the length of a frame fragment generated by a collision. (See 6.4.2.1)
- It is the scheduling quantum for retransmission. (See 6.3.2.3.2)

In order to fulfill all three functions, the slot time must be larger than the sum of the Physical Layer round-trip propagation time (450 bit times; see 7.1.2) and the Data Link Layer maximum jam time (48 bit times, see 6.3.2.3.1). The slot time is defined to be 512 bit times.

6.3.2.3.1 Collision Detection and Enforcement

Collisions are detected by monitoring the *collisionDetect* signal provided by the Physical Layer. When a collision is detected during a frame transmission, the transmission is not terminated immediately. Instead, the transmission continues until at least 32 (but not more than 48) additional bits have been transmitted (counting from the time *collisionDetect* went on). This collision enforcement or "jam" guarantees that the duration of the collision is sufficient to insure its detection by all transmitting stations on the network. The content of the jam is unspecified; it may be any fixed or variable pattern convenient to the data link controller implementation, but should not be the 32-bit CRC value corresponding to the (partial) frame transmitted prior to the jam.

Amazon
Exhibit 1005
Page 189

6.3.2.3.2 Collision Backoff and Retransmission

When a transmission attempt has terminated due to a collision, it is retried by the transmitting data link controller until either it is successful, or 16 attempts (the original attempt plus 15 retries) have been made and all have terminated due to collisions. Note that all attempts to transmit a given frame are completed before any subsequent outgoing frames are transmitted. The scheduling of the retransmissions is determined by a controlled randomization process called "truncated binary exponential backoff". At the end of enforcing a collision (jamming), the data link controller delays before attempting to retransmit the frame. The delay is an integral multiple of the slot time. (See 6.3.2.3). The number of slot times to delay before the n^{th} retransmission attempt is chosen as a uniformly distributed random integer r in the range $0 \leq r < 2^k$ where $k = \min(n, 10)$. If all 16 attempts fail, this event is reported as an error.

Note that the values given above define the most aggressive behavior that a station may exhibit in attempting to retransmit after a collision. In the course of implementing the retransmission scheduling procedure, a station may introduce extra delays which will degrade its own throughput, but in no case may a station's retransmission scheduling result in a lower average delay between retransmission attempts than the procedure defined above.

6.4 Frame Reception

Frame reception includes both data decapsulation and link management aspects:

Receive Data Decapsulation comprises framing, address recognition, frame check sequence validation, and frame disassembly to pass the fields of the received frame to the Client Layer.

Receive Link Management's main function is the filtering of collision fragments from complete incoming frames.

The performance of these functions by a receiving data link controller interacts with corresponding actions by other data link controllers to jointly implement the Ethernet data link protocol.

6.4.1 Receive Data Decapsulation

6.4.1.1 Framing

The data link controller recognizes the boundaries of an incoming frame by monitoring the *carrierSense* signal provided by the Physical Layer. There are two possible length errors that can occur, which indicate ill-framed data: the frame may be too long, or its length may not be an integral number of octets.

6.4.1.1.1 Maximum Frame Size

The receiving data link controller is not required to enforce the frame size limit specified in 6.2.5, but it is allowed to truncate frames longer than 1518 octets and report this event as an (implementation-dependent) error.

6.4.1.1.2 Integral Number of Octets in Frame

Since the format of a valid frame specifies an integral number of octets, only a collision or an error can produce a frame with a length that is not an integral multiple of 8. Complete frames (i.e., not rejected as collision fragments; see 6.4.2.1) that do not contain an integral number of octets are truncated to the nearest octet boundary. If frame check sequence validation (see 6.4.1.3) detects an error in such a frame, the status code *alignmentError* is reported.

6.4.1.2 Address Recognition

The Ethernet data link controller is capable of recognizing physical and multicast addresses, as defined in 6.2.1.

6.4.1.2.1 Physical Addresses

The data link controller recognizes and accepts any frame whose destination field contains the physical address of the station.

The physical address of each station is set by network management to a unique value associated with the station, and distinct from the address of any other station on any Ethernet. The setting of the station's physical address by network management allows multiple data link controllers connected to single station all to respond to the same physical address. The procedures for allocating unique addresses are discussed in Appendix B.

6.4.1.2.2 Multicast Addresses

The data link controller recognizes and accepts any frame whose destination field contains the broadcast address.

The data link controller is capable of activating some number of multicast-group addresses as specified by higher layers. The data link controller recognizes and accepts any frame whose destination field contains an active multicast-group address. An active multicast-group address may be deactivated.

6.4.1.3 Frame Check Sequence Validation

FCS validation is essentially identical to FCS generation. If the bits of the incoming frame (exclusive of the FCS field itself) do not generate a CRC value identical to the one received, an error has occurred and is reported as such. Implementation issues are discussed in Appendix C.

Amazon
Exhibit 1005
Page 190

6.4.1.4 Frame Disassembly

The frame is disassembled and the fields are passed to the Client Layer via the output parameters of the *ReceiveFrame* operation (see 5.1).

6.4.2 Receive Link Management

6.4.2.1 Collision Filtering

As specified in 6.2.5, the smallest valid frame must contain at least 64 octets. Any frame containing less than 64 octets is presumed to be a fragment resulting from a collision and is discarded by the receiving data link controller. Since occasional collisions are a normal part of the link management procedure, the discarding of such a fragment is not reported as an error to the Client Layer.

6.5 The Data Link Layer Procedural Model

6.5.1 Overview of the Procedural Model

The functions of the Ethernet Data Link Layer are presented below, modeled as a program written in the language Pascal [6]. This procedural model is intended as the primary specification of the functions to be provided in any Ethernet Data Link Layer implementation. It is important to distinguish, however, between the model and a real implementation. The model is optimized for simplicity and clarity of presentation, while any realistic implementation must place heavier emphasis on such constraints as efficiency and suitability to a particular implementation technology or computer architecture. In this context, several important properties of the procedural model must be considered.

6.5.1.1 Ground Rules for the Procedural Model

a) First, it must be emphasized that the description of the Data Link Layer in a programming language is in no way intended to imply that a data link controller must be implemented as a program executed by a computer. The implementation may consist of any appropriate technology including hardware, firmware, software, or any combination.

b) Similarly, it must be emphasized that it is the *behavior* of Data Link Layer implementations that must match the specification, *not* their internal structure. The internal details of the procedural model are useful only to the extent that they help specify that behavior clearly and precisely.

c) The handling of incoming and outgoing frames is rather stylized in the procedural model, in the sense that frames are handled as single entities by most of the Data Link Layer and are only serialized for presentation to the Physical Layer. In reality, many data link controller implementations will instead handle frames serially on a bit, octet or word basis. A serial implementation would typically perform the required functions (address recognition, frame check sequence generation/validation, etc.) in an overlapped, pipelined fashion. This approach has not been reflected in the procedural model, since this would only complicate the description of the functions without changing them in any way.

d) The model consists of algorithms designed to be executed by a number of concurrent processes; these algorithms collectively implement the Ethernet data link control procedure. The timing dependencies introduced by the need for concurrent activity are resolved in two ways:

- *Processes vs. External events:* It is assumed that the algorithms are executed "very fast" relative to external events, in the sense that a process never falls behind in its work and fails to respond to an external event in a timely manner. For example, when a frame is to be received, it is assumed that the data link procedure *ReceiveFrame* is always called well before the frame in question has started to arrive.

- *Processes vs. Processes:* Among processes, no assumptions are made about relative speeds of execution. This means that each interaction between two processes must be structured to work correctly independent of their respective speeds. Note, however, that the timing of interactions among processes is often, in part, an indirect reflection of the timing of external events, in which case appropriate timing assumptions may still be made.

It is intended that the concurrency in the model reflect the parallelism intrinsic to the task of implementing the Ethernet data link, although the actual parallel structure of the implementations is likely to vary.

6.5.1.2 Use of Pascal in the Procedural Model

Pascal was chosen for the procedural model because of its relative simplicity and clarity, and its general acceptance.

Several observations need to be made about the way in which Pascal is used for the model, including:

a) Some limitations of the language have been circumvented in order to simplify the specification:

1) The elements of the program (variables, procedures, etc) are presented in logical groupings, in top-down order. Certain Pascal ordering restrictions have thus been circumvented to improve readability.

2) The **process** and **cycle** constructs of the Pascal derivative Concurrent Pascal [7] have been introduced to indicate the sites of autonomous concurrent activity. As used here, a process is simply a parameterless procedure that begins execution at "the beginning of time" rather than being invoked by a procedure call. A cycle statement represents the main body of a process and is executed repeatedly forever.

3) The lack of variable array bounds in the language has been circumvented by treating frames as if they are always of a single fixed size (which is never actually specified). In fact, of course, the size of a frame depends on the size of its data field, hence the value of the "pseudo-constant" *frameSize* should be thought of as varying in the long-term, even though it is fixed for any given frame.

4) The use of a variant record to represent a frame (both as fields and as bits) follows the letter but not the spirit of the Pascal Report, since it allows the underlying representation to be viewed as two different data types. (It also assumes that this representation is as shown in Figure 6-1.)

b) The model makes no use of any explicit interprocess synchronization primitives. Instead, all interprocess interaction is done via carefully stylized manipulation of shared variables. For example, some variables are set by only one process and inspected by another process in such a manner that the net result is independent of their execution speeds. While such techniques are not generally suitable for the construction of large concurrent programs, they simplify the model and more nearly resemble the methods appropriate to the most likely implementation technologies (e.g. microcode, hardware state-machines, etc.)

Amazon
Exhibit 1005
Page 191

6.5.2 Procedural Model

The procedural model used here is based on five cooperating concurrent processes. Of these, three are actually defined in the Data Link Layer. The remaining two processes are provided by the Client Layer and utilize the interface operations provided by the Data Link Layer. The five processes are thus:

Client Layer:

Frame Transmitter Process Frame Receiver Process

Data Link Layer:

Bit Transmitter Process Bit Receiver Process

Deference Process

This organization of the model is illustrated in Figure 6-2, and reflects the fact that the communication of entire frames is initiated by the Client Layer, while the timing of collision backoff and of individual bit transfers is based on interactions between the Data Link Layer and the Physical-Layer-dependent bit-time.

Figure 6-2 depicts the static structure of the procedural model, showing how the various processes and procedures interact by invoking each other. Figures 6-3 and 6-4 summarize the dynamic behavior of the model during transmission and reception, focusing on the steps that must be performed, rather than the procedural structure which performs them. The usage of the shared state variables is not depicted in the figures, but is described in the comments in 6.5.2.1.

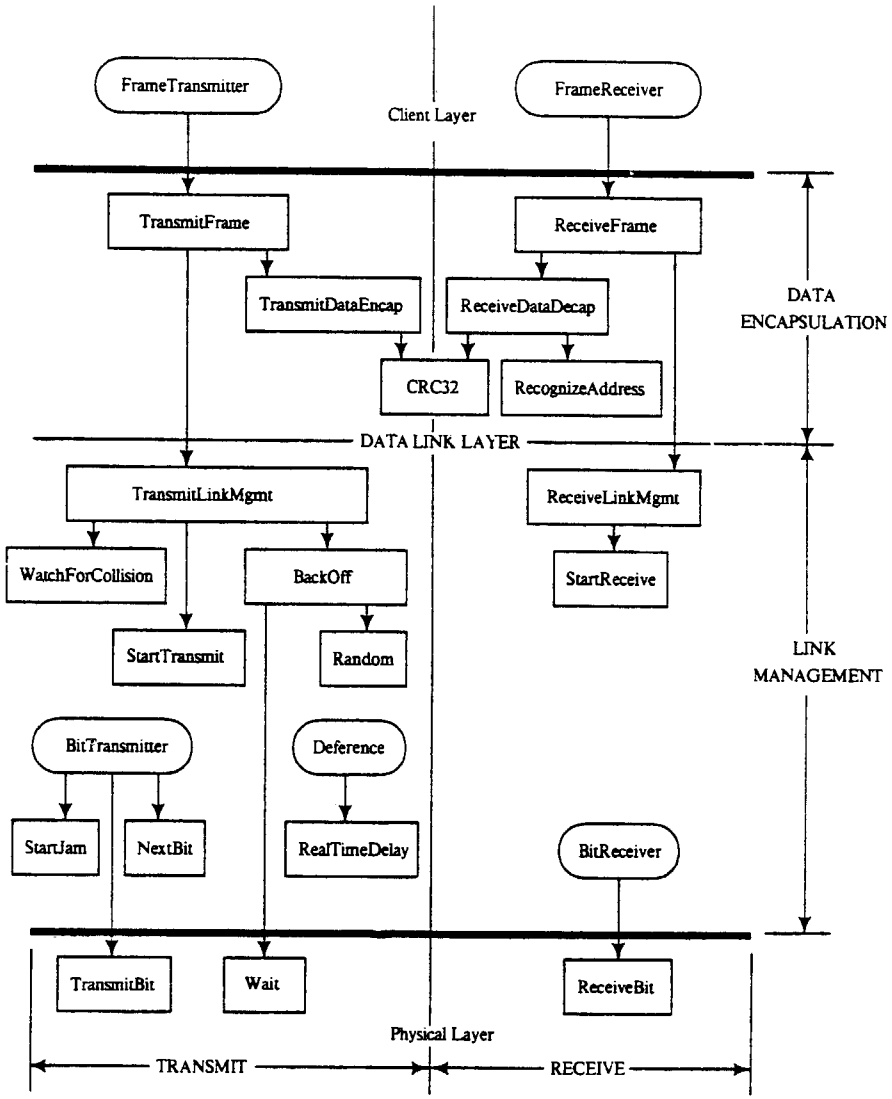


Figure 6-2: Structure of the Data Link Procedural Model

Amazon
Exhibit 1005
Page 192

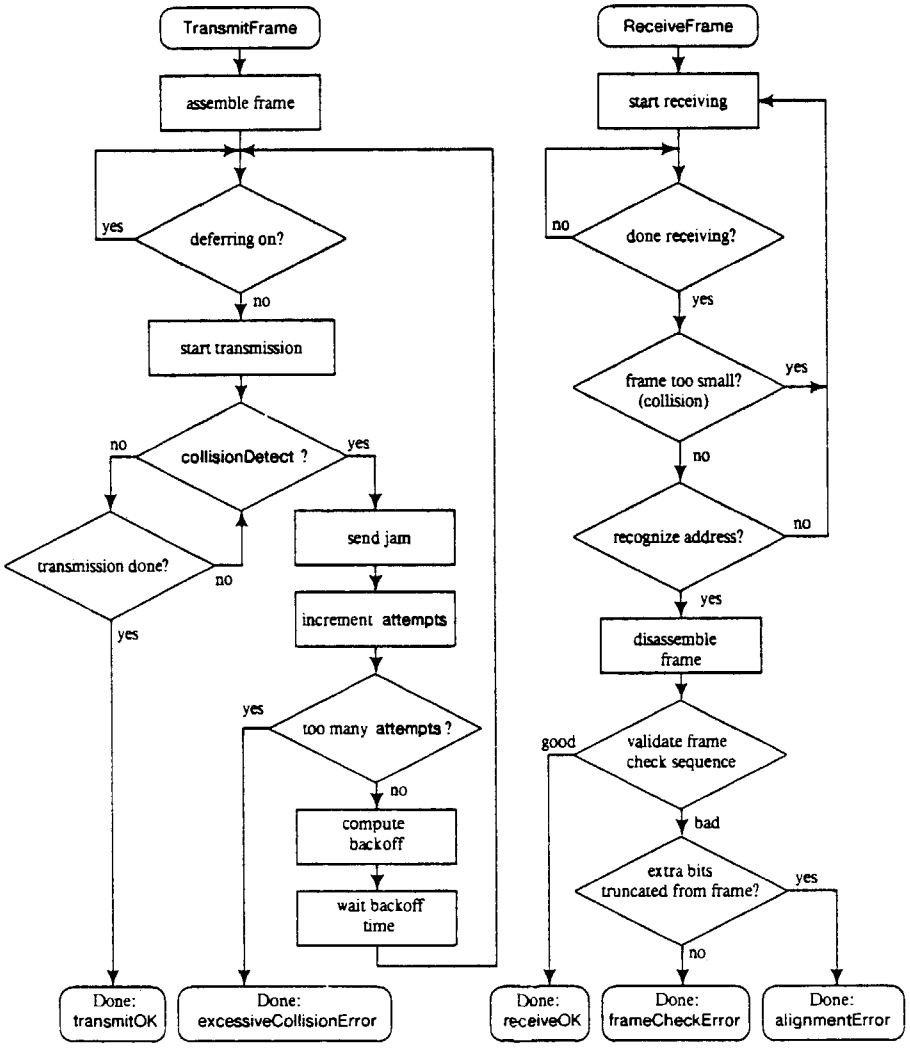


Figure 6-3: Control Flow Summary -- Client Layer Processes

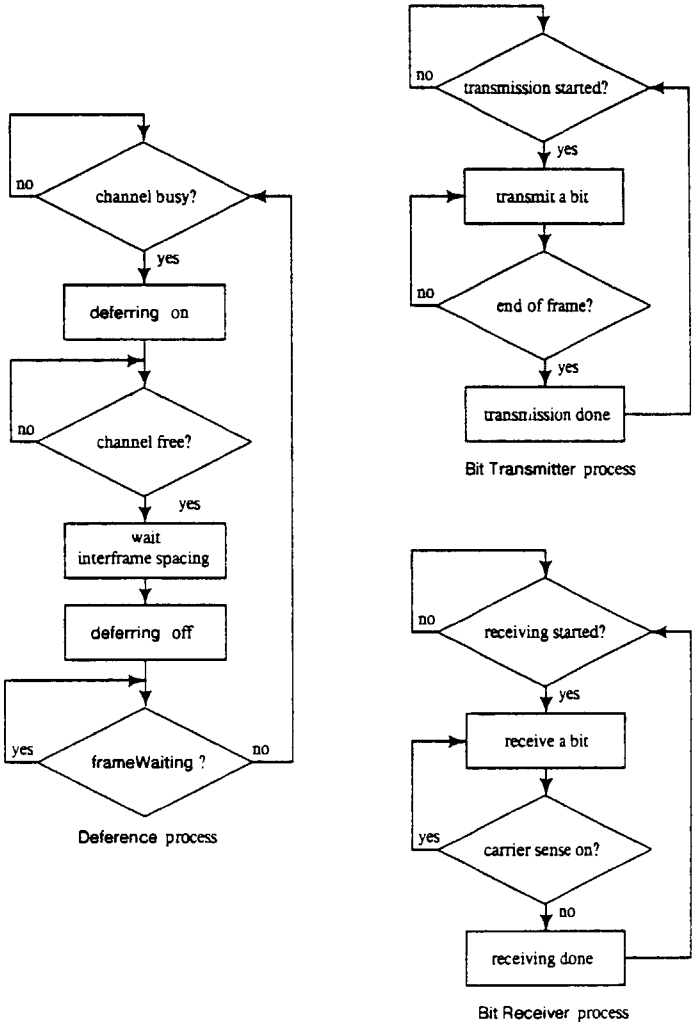


Figure 6-4: Control Flow Summary -- Data Link Layer Processes

Amazon
Exhibit 1005
Page 193

6.5.2.1 Global Declarations

6.5.2.1.1 Common Constants and Types

The following declarations of constants and types are used by the frame transmission and reception sections of each data link controller:

```
const
    addressSize = 48;  {48 bit address = 6 octets}
    typeSize = 16;  {16 bit protocol type = 2 octets}
    dataSize = ...;  {see 6.5.1.2, note 3}
    crcSize = 32;  {32 bit CRC = 4 octets}
    frameSize = ...;  { = 2*addressSize + typeSize + dataSize + crcSize...see 6.5.1.2,
    note 3}

    slotTime = 512;  {unit of time for collision handling}

type
    Bit = 0..1;
    AddressValue = array [1..addressSize] of Bit;
    TypeValue = array [1..typeSize] of Bit;
    DataValue = array [1..dataSize] of Bit;
    CRCValue = array [1..crcSize] of Bit;

    ViewPoint = (fields, bits);  {Two ways to view the contents of a frame}

    Frame = record  {Format of data link frame}
      case view: ViewPoint of
        fields: (
            destinationField: AddressValue;
            sourceField: AddressValue;
            typeField: TypeValue;
            dataField: DataValue;
            fcsField: CRCValue);
        bits: (
            contents: array [1..frameSize] of Bit)
      end; {Frame}
```


6.5.2.1.2 Transmit State Variables

The following items are specific to frame transmission. (See also 6.5.2.1.4 on interfaces)

```
const
    interFrameSpacing = 9.6;  {minimum time between frames, in microseconds}
    attemptLimit = 16;  {Max number of times attempt transmission}
    backOffLimit = 10;  {Limit on number of times to back off}
    jamSize = 32;  {jam may be 32 to 48 bits long}

var
    outgoingFrame: Frame;  {The frame to be transmitted}
    currentTransmitBit, lastTransmitBit: 1..frameSize;  {Positions of current and
      last outgoing bits in outgoingFrame}
    deferring: Boolean;  {True implies any pending transmission must wait for the
      channel to clear}
    frameWaiting: Boolean;  {Indicates that outgoingFrame is deferring}
    attempts: 0..attemptLimit;  {Number of transmission attempts on outgoingFrame}
    newCollision: Boolean;  {Indicates that a collision has occurred but has not yet
      been jammed}
    transmitSucceeding: Boolean;  {Running indicator of whether transmission is
      succeeding}
```

6.5.2.1.3 Receive State Variables

The following items are specific to frame reception. (See also 6.5.2.1.4 on interfaces)

```
var
    incomingFrame: Frame;  {The frame being received}
    currentReceiveBit: 1..frameSize;  {Position of current bit in incomingFrame}
    receiving: Boolean;  {Indicates that a frame reception is in progress}
    excessBits: 0..7;  {Count of excess trailing bits beyond octet boundary}
    receiveSucceeding: Boolean;  {Running indicator of whether reception is
      succeeding}
```

Amazon
Exhibit 1005
Page 194

6.5.2.1.4 Summary of Interlayer Interfaces

The interface to the Client Layer, defined in 5.1, is summarized below:

```
type
  TransmitStatus = (transmitOK, excessiveCollisionError); {Result of
    TransmitFrame operation}
  ReceiveStatus = (receiveOK, frameCheckError, alignmentError); {Result
    of ReceiveFrame operation}

function TransmitFrame (
  destinationParam: AddressValue;
  sourceParam: AddressValue;
  typeParam: typeValue;
  dataParam: DataValue): TransmitStatus; {Transmits one frame}

function ReceiveFrame (
  var destinationParam: AddressValue;
  var sourceParam: AddressValue;
  var typeParam: TypeValue;
  var dataParam: DataValue): ReceiveStatus; {Receives one frame}
```

The interface to the Physical Layer, defined in 5.2, is summarized below:

```
var
    carrierSense: Boolean; {Indicates incoming bits}
    transmitting: Boolean; {Indicates outgoing bits}
    collisionDetect: Boolean; {Indicates channel contention}
procedure TransmitBit (bitParam: Bit); {Transmits one bit}
function ReceiveBit: Bit; {Receives one bit}
procedure Wait (bitTimes: integer); {Waits for indicated number of bit-times}
```


6.5.2.1.5 State Variable Initialization

The procedure *Initialize* must be run when the Data Link Layer begins operation, before any of the processes begin execution. *Initialize* sets certain crucial shared state variables to their initial values. (All other global variables are appropriately reinitialized before each use.) *Initialize* then waits for the channel to be idle, and starts operation of the various processes.

```
procedure Initialize;
begin
  frameWaiting := false;
  deferring := false;
  newCollision := false;
  transmitting := false; {In interface to Physical Layer; see below}
  receiving := false;
  while carrierSense do nothing;
  {Start execution of all processes}
end; {Initialize}
```

Amazon
Exhibit 1005
Page 195

6.5.2.2 Frame Transmission

The algorithms in this section define data link frame transmission.

The function *TransmitFrame* implements the frame transmission operation provided to the Client Layer:

```
function TransmitFrame (
    destinationParam: AddressValue;
    sourceParam: AddressValue;
    typeParam: typeValue;
    dataParam: DataValue): TransmitStatus;
  procedure TransmitDataEncap; ... {nested procedure; see body below}
begin
  TransmitDataEncap;
  TransmitFrame := TransmitLinkMgmt
end; {TransmitFrame}
```

First, *TransmitFrame* calls the internal procedure *TransmitDataEncap* to construct the frame. It then calls *TransmitLinkMgmt* to perform the actual transmission. The *TransmitStatus* returned indicates the success or failure of the transmission attempt.

TransmitDataEncap builds the frame and places the 32-bit CRC in the frame check sequence field:

```
procedure TransmitDataEncap;
begin
  with outgoingFrame do
  begin {assemble frame}
    view := fields;
    destinationField := destinationParam;
    sourceField := sourceParam;
    typeField := typeParam;
    dataField := dataParam;
    fcsField := CRC32(outgoingFrame);
    view := bits
  end {assemble frame}
end; {TransmitDataEncap}
```


TransmitLinkMgmt attempts to transmit the frame, deferring first to any passing traffic. If a collision occurs, transmission is terminated properly and retransmission is scheduled following a suitable backoff interval:

```
function TransmitLinkMgmt: TransmitStatus;
begin
  attempts := 0; transmitSucceeding := false;
  while attempts < attemptLimit and not transmitSucceeding do
  begin {loop}
    if attempts > 0 then BackOff;
    frameWaiting := true;
    while deferring do nothing; {defer to passing frame, if any}
    frameWaiting := false;
    StartTransmit;
    while transmitting do WatchForCollision;
    attempts := attempts + 1
  end; {loop}
  if transmitSucceeding then TransmitLinkMgmt := transmitOK
  else TransmitLinkMgmt := excessiveCollisionError
end; {TransmitLinkMgmt}
```

Each time a frame transmission attempt is initiated, *StartTransmit* is called to alert the *BitTransmitter* process that bit transmission should begin:

```
procedure StartTransmit;
begin
  currentTransmitBit := 1;
  lastTransmitBit := frameSize;
  transmitSucceeding := true;
  transmitting := true
end; {StartTransmit}
```

Once frame transmission has been initiated, *TransmitLinkMgmt* monitors the channel for contention by repeatedly calling *WatchForCollision*:

```
procedure WatchForCollision;
begin
  if transmitSucceeding and collisionDetect then
  begin
    newCollision := true;
    transmitSucceeding := false
  end
end; {WatchForCollision}
```

WatchForCollision, upon detecting a collision, updates *newCollision* to insure proper jamming by the *BitTransmitter* process.

Amazon
Exhibit 1005
Page 196

After transmission of the jam has completed, if *TransmitLinkMgmt* determines that another attempt should be made, *BackOff* is called to schedule the next attempt to retransmit the frame.

```
var maxBackOff: 2..1024; {Working variable of BackOff}

procedure BackOff;
begin
  if attempts = 1 then maxBackOff := 2 else if attempts ≤ backOffLimit
    then maxBackOff := maxBackOff*2;
  Wait(slotTime*Random(0, maxBackOff))
end; {BackOff}

function Random (low, high: integer): integer;
begin
  Random := ...{uniformly distributed random integer r such that low ≤ r <
  high}
end; {Random}
```

BackOff performs the truncated binary exponential backoff computation and then waits for the selected multiple of the slot time.

The *Deference* process runs asynchronously to continuously compute the proper value for the variable *deferring*.

```
process Deference;
begin
  cycle {main loop}
    while not carrierSense do nothing; {watch for carrier to appear}
    deferring := true; {delay start of new transmissions}
    while carrierSense do nothing; {wait for carrier to disappear}
    RealTimeDelay(interFrameSpacing);
    deferring := false; {allow new transmissions to proceed}
    while frameWaiting do nothing {allow waiting transmission (if any)}
  end {main loop}
end; {Deference}

procedure RealTimeDelay (usec: real);
begin
  {Wait for the specified number of microseconds}
end; {RealTimeDelay}
```


The *BitTransmitter* process runs asynchronously, transmitting bits at a rate determined by the Physical Layer's *TransmitBit* operation:

```
process BitTransmitter;
begin
  cycle {outer loop}
    while transmitting do
    begin {inner loop}
      TransmitBit(outgoingFrame[currentTransmitBit]); {send next bit to
      Physical Layer}
      if newCollision then StartJam else NextBit
    end {inner loop}
  end {outer loop}
end; {BitTransmitter}

procedure NextBit;
begin
  currentTransmitBit := currentTransmitBit + 1;
  transmitting := (currentTransmitBit ≤ lastTransmitBit)
end; {NextBit}

procedure StartJam;
begin
  currentTransmitBit := 1;
  lastTransmitBit := jamSize;
  newCollision := false
end; {StartJam}
```

BitTransmitter, upon detecting a new collision, immediately enforces it by calling *StartJam* to initiate the transmission of the jam. The jam may contain 32 to 48 bits of arbitrary data. (*StartJam* uses the first 32 bits of the frame, merely to simplify this program).

Amazon
Exhibit 1005
Page 197

6.5.2.3 Frame Reception

The algorithms in this section define data link frame reception:

The procedure *ReceiveFrame* implements the frame reception operation provided to the Client Layer:

```
function ReceiveFrame (
    var destinationParam: AddressValue;
    var sourceParam: AddressValue;
    var typeParam: TypeValue;
    var dataParam: DataValue): ReceiveStatus;
  function ReceiveDataDecap: ReceiveStatus; ... {nested function; see body
  below}
begin
  repeat
    ReceiveLinkMgmt;
    ReceiveFrame := ReceiveDataDecap;
  until receiveSucceeding
end; {ReceiveFrame}
```

ReceiveFrame calls *ReceiveLinkMgmt* to receive the next valid frame, and then calls the internal procedure *ReceiveDataDecap* to return the frame's fields to the Client Layer if the frame's address indicates that it should do so. The returned *ReceiveStatus* indicates the presence or absence of detected transmission errors in the frame.

```
function ReceiveDataDecap: ReceiveStatus;
begin
  receiveSucceeding := RecognizeAddress
    (incomingFrame.destinationField);
  if receiveSucceeding then with incomingFrame do
  begin {disassemble frame}
    view := fields;
    destinationParam := destinationField;
    sourceParam := sourceField;
    typeParam := typeField;
    dataParam := dataField;
    if fcsField = CRC32(incomingFrame) then ReceiveDataDecap := receiveOK
    else if excessBits = 0 then ReceiveDataDecap := frameCheckError
    else ReceiveDataDecap := alignmentError;
    view := bits
  end {disassemble frame}
end; {ReceiveDataDecap}
```



```
function RecognizeAddress (address: AddressValue): Boolean;
begin
  RecognizeAddress := ... {Returns true for the set of physical, broadcast, and
      multicast-group addresses corresponding to this station}
end; {RecognizeAddress}
```

ReceiveLinkMgmt attempts repeatedly to receive the bits of a frame, discarding any fragments from collisions by comparing them to the minimum valid frame size:

```
procedure ReceiveLinkMgmt;
begin
  repeat
    StartReceive;
    while receiving do nothing; {wait for frame to finish arriving}
    excessBits := frameSize mod 8;
    frameSize := frameSize - excessBits; {truncate to octet boundary}
    receiveSucceeding := (frameSize ≥ slotTime); {reject collision fragments}
  until receiveSucceeding
end; {ReceiveLinkMgmt}

procedure StartReceive;
begin
  currentReceiveBit := 1;
  receiving := true
end; {StartReceive}
```

The *BitReceiver* process run asynchronously, receiving bits from the channel at the rate determined by the Physical Layer's *ReceiveBit* operation:

```
process BitReceiver;
  var b: Bit;
begin
  cycle {outer loop}
    while receiving do
    begin {inner loop}
      b := ReceiveBit; {Get next bit from physical link}
      if carrierSense then
      begin {append bit to packet}
        incomingFrame[currentReceiveBit] := b;
        currentReceiveBit := currentReceiveBit + 1
      end; {append bit to packet}
      receiving := carrierSense
    end {inner loop}
  end {outer loop}
end; {BitReceiver}
```

Amazon
Exhibit 1005
Page 198

6.5.2.4 Common procedures

The function *CRC32* is used by both the transmit and receive algorithms to generate a 32 bit CRC value:

```
function CRC32 (f: Frame): CRCValue;
begin
  CRC32 := {The 32-bit CRC as defined in 6.2.4}
end; {CRC32}
```

Purely to enhance readability, the following procedure is also defined:

```
procedure nothing; begin end;
```

The idle state of a process (i.e., while waiting for some event) is cast as repeated calls on this procedure.

Amazon
Exhibit 1005
Page 199

7. ETHERNET PHYSICAL LAYER SPECIFICATION: Baseband Coaxial System

7.1 Physical Channel Overview and Model

The Ethernet physical channel (henceforth referred to as the channel) provides the lowest layer in the Ethernet architecture. It performs all the functions needed to transmit and receive data at the physical level, while supporting the Data Link to Physical Layer Interface described in 5.2.

This section describes the requirements for interface and compatibility with a baseband coaxial implementation of the channel.

7.1.1 Channel Goals and Non-goals

This section states the objectives underlying the design of the channel.

7.1.1.1 Goals

The following are the goals of the channel:

1. Provide a means for communication between Ethernet Data Link Entities.
2. Define physical interfaces which can be implemented compatibly among different manufacturers of hardware.
3. Provide all clocks, synchronization, and timing required for both itself and the Ethernet Data Link.
4. Provide high bandwidth and low bit error rates.
5. Provide for ease of installability and serviceability.
6. Provide for high network availability.
7. Support the Ethernet Data Link to Physical Link interface.
8. Low cost.

7.1.1.2 Non-Goals

The following are not goals of the baseband coaxial channel design:

1. Operation at data rates other than 10 megabits per second.
2. Operation with media other than the specified coaxial cable.
3. Simultaneous use of the channel by transmitters using signals not specified in this document.
4. Protection against a malicious user or a malfunctioning Data Link Entity is not provided by the channel as specified. However, higher layers (above the Data Link) and/or physical security means may be employed to acheive this.

7.1.2 Characteristics of the Channel

The channel provides (and the data link assumes) the following characteristics:

1. The ability to send and receive information (non-simultaneously) between any two or more data link entities on the same network.
2. The ability to detect the presence of another station's transmission while not transmitting (carrier sense).
3. The ability to detect the presence of another station's transmission while transmitting (collision detect).
4. A total worst-case round trip signal propagation delay (including actual propagation time, synchronization time for all intervening electronics, and signal rise time degradation) of 450 bit times (equal to 45 μs for this 10 Mbit channel).

7.1.3 Functions Provided by the Channel

The channel hardware provides the following functions in the performance of its role:

1. Means for transmitting and receiving serial bit streams between the data link layer and the media.
2. Generation of clock for synchronization and timing.
3. Means for detecting carrier (non-idle channel).
4. Means for detecting collisions (simultaneous transmission attempts by multiple stations).
5. Coding and decoding of the data link bit stream into a self-synchronizable sequence of electrical signals suitable for transmission on the media provided by the channel.
6. Generation and removal of coding-specific preamble information (a synchronizing header sequence inserted before the first bit of the frame) to ensure that all channel electronics are brought to a known steady-state before the data link frame is transmitted.

7.1.4 Implementation of the Channel

The physical channel specification is implementation dependent; most of the channel hardware is fully specified, and little leeway is given to the individual designer. This is done in the interest of compatibility; any system which allows different implementors to use different channel cables, connectors, clock speeds and the like will not be compatible across manufacturer boundaries. Only the design of channel components which are not critical to system compatibility is left to the implementor.

Amazon
Exhibit 1005
Page 200

7.1.4.1 General Overview of Channel Hardware

The channel minimally consists of the following functional blocks:

1. The passive broadcast medium (coaxial cable),
2. The transceiver (transmitter-receiver for the coaxial cable),
3. The means for connecting transceivers to a coaxial cable segment and for connecting coaxial cable segments together,
4. The channel clock,
5. The channel data encoder and decoder,
6. The preamble generator and remover,
7. The carrier and collision detect circuits.

The coaxial medium is the only element common to the entire network. A transceiver is required for each station connected to the medium. The transceiver must be located adjacent to the coaxial cable. The latter four components are generally located within, and tightly coupled to, the station hardware implementing the data link function.

It may be useful to be able to physically separate the transceiver from the rest of the channel hardware. This allows topological flexibility, packaging advantages, and improved system availability, as well as allowing for independent manufacture of station hardware and transceivers. To ensure that compatibility is maintained, a physical interface (known as the *transceiver cable*) is identified and specified to connect the transceiver to the station.

Finally, it may be necessary to add *repeaters* to the system, to reach the maximum allowable distance between stations, and to provide additional topological flexibility. Repeaters are implemented using standard transceivers, plus a simple, non-buffered finite state machine.

7.1.4.2 Compatibility Interfaces

There are a number of possibilities for implementing systems or subsystems compatible in whole or in part with this specification. It is important that all implementations be compatible at some point, so that heterogenous systems from different manufacturers' implementations can be interconnected on the same medium. It is not necessary in every case to implement all of the components described herein; e.g., it is possible to design an integrated station/transceiver (without requiring the transceiver cable). The implementor must make the required trade-offs between topological flexibility, system availability, configurability, user needs, and cost when designing the system.

For a device to be considered compatible, it must meet the applicable requirements at either the transceiver cable or the coaxial cable interface, as appropriate, in addition to the Data Link compatibility required for all stations connected to the network.

All Ethernets must be compatible at the coaxial cable.

If a transceiver cable is used, it should be the one specified in this document. This allows device manufacturers to build hardware compatible with the Ethernet at the transceiver cable level, without concerning themselves with the details of transceiver implementation. Devices implementing transceiver cable compatibility should be capable of using transceivers designed and built by another manufacturer, on the specified coaxial cable.

Equipment designed for connection to the specified coaxial cable either without a physically separate transceiver or with a non-standard transceiver cable interface will be capable of communication. However, a sacrifice may have been made with respect to interchangeability with other stations.

This scheme of multiple compatibility interfaces allows individual designers some flexibility in making system tradeoffs, yet allows cable manufacturers, transceiver manufacturers and systems manufacturers to use standard commodity parts to produce a compatible communications system.

Amazon
Exhibit 1005
Page 201

7.1.5 Channel Configuration Model

Certain physical limits have been placed on the physical channel. These revolve mostly around maximum cable lengths (or maximum propagation times), as these affect the slot time as defined in the data link. While the precise specification (in later sections) specify these maxima in terms of propagation times, they were derived from the physical configuration model described here.

The maximum configuration is as follows:

1. A coaxial cable, terminated in its characteristic impedance at each end, constitutes a cable *segment*. A segment may contain a maximum of 500 meters of coaxial cable.
2. A maximum of 2 repeaters in the path between any two stations. Repeaters do not have to be located at the ends of segments, nor is the user limited to one repeater per segment. In fact, repeaters can be used not only to extend the length of the channel, but to extend the topology from one to three-dimensional. Repeaters occupy transceiver positions on each cable segment and count towards the maximum number of transceivers on a segment just as do the logically distinguishable stations.
3. A maximum total coaxial cable length along the longest path between any two transceivers of 1500 meters. The propagation velocity of the coaxial cable is assumed to be 0.77 c worst-case. (c is the velocity of light *in vacuo*; 300,000 kilometers per second.) The total round-trip delay for all the coaxial cable in the system is therefore 13 μs worst-case.
4. A maximum of 50 meters of transceiver cable between any station and its associated transceiver. Note that in the worst case the signal must pass through six 50 meter transceiver cables, one at the transmitting station, one at the receiving station, and 2 at each repeater (two repeaters possible). The propagation velocity of the transceiver cable is assumed to be .65 c worst-case. The total round-trip delay for all the transceiver cables is therefore 3.08 μs worst-case.
5. A maximum of 1000 meters of point-to-point link anywhere in the system. This will typically be used as a way of linking cable segments in different buildings. Note that a repeater with this internal point-to-point link can be used to repeat signals between segments many hundreds of meters apart. The worst-case propagation velocity of the link cable is assumed to be .65 c; the round-trip propagation delay for 1000 meters is 10.26 μs.

Table 7-1 summarizes the allocation of the round-trip propagation delay to the individual components in the channel. Figure 7-1 shows a minimum, typical, and large-scale channel configuration.

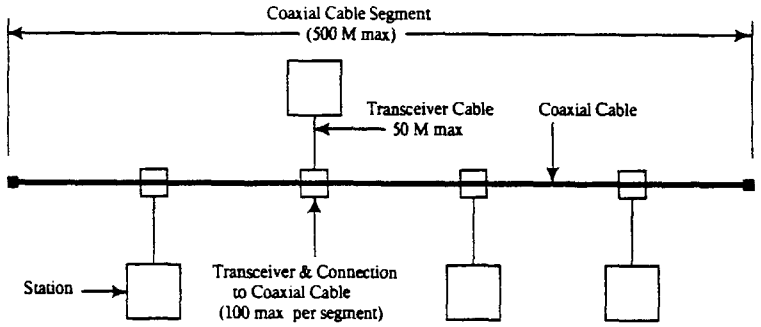


Figure 7-1a: Minimal Configuration

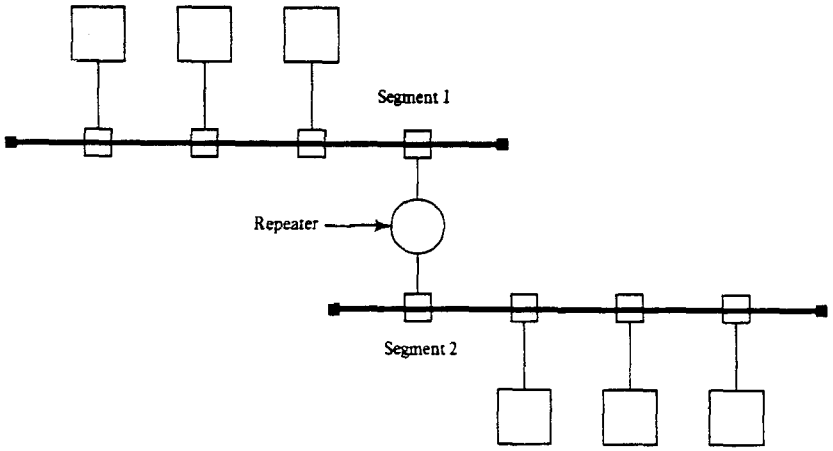


Figure 7-1b: A Typical Medium-scale Configuration

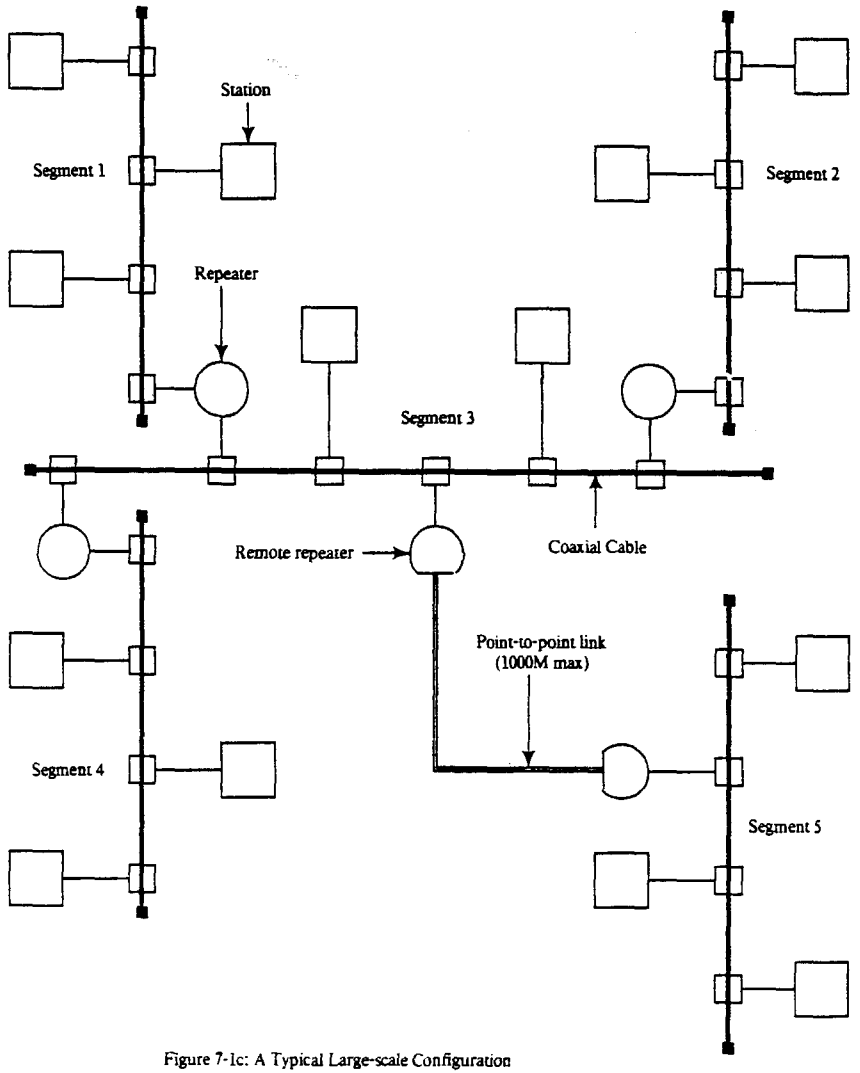


Figure 7-1c: A Typical Large-scale Configuration

Amazon
Exhibit 1005
Page 202

Table 7-1: Physical Channel Propagation Delay Budget (Note 1)

Element	Unit Steady-State Delay	Unit Startup Delay	# Units Forward Path (Note 2)	# Units Return Path	Total Delay
Encoder	0.1 μS	0	3	3	0.60 μS
Transceiver Cable	5.13 nS/M	0	300 M	300 M	3.08 μS
Transceiver (transmit path)	0.50 μS	0.2 μS	3	3	1.50 μS
Transceiver (receive path)	0.50 μS	0.5 μS	3	0	1.65 μS
Transceiver (collision path)	0	0.5 μS	0	3	1.50 μS
Coaxial Cable	4.33 nS/M	0	1500 M	1500 M	13.00 μS
Point-to-Point Link Cable	5.13 nS/M	0	1000 M	1000 M	10.26 μS
Repeater (repeat path)	0.8 μS	0	2	0	1.60 μS
Repeater (collision path)	0.2 μS	0	0	2	0.40 μS
Decoder	0.1 μS	0.8 μS	2	0	1.80 μS
Carrier Sense	0	0.2 μS	3	0	0.60 μS
Collision Detect	0	0.2 μS	0	3	0.60 μS
Signal Rise Time (to 70% in 500 M) (Note 3)	0	0.1 μS	3	0	0.30 μS
Signal Rise Time (50% to 94% in 500 M) (Note 4)	0	2.7 μS	0	3	8.10 μS
		Total Worst-Case Round-Trip Delay			44.99 μS

Note 1: All quantities given are worst-case (both number of units and unit delays per unit).

Note 2: The propagation delay has been separated into "forward-path" and "return path" delay. This is because in one direction it is carrier sense which is being propagated through the channel, and in the return direction it is collision detect which is being propagated. The two signals have different propagation delays.

Note 3: In the worst-case, the propagated signal must reach 70% of its final value to be detected as valid carrier at the end of 500 meters of coaxial cable. This rise time must be included in the propagation delay budget.

Note 4: In the worst-case the propagated collision on the return path must reach 94% of its final value to be detected as a collision at the end of 500 meters of coaxial cable.

7.1.6 Channel Interfaces

The channel specification hinges around three well-defined entities; the transceiver and coaxial cables (shown as compatibility interfaces in Figure 4-1), and the logical interface between the physical channel and the data link controller (shown in Figure 4-4). Note that the former two are physical interfaces specific to the channel, and are specified in the interest of compatibility. The latter is provided as a means by which the data link controller can interact with the physical channel.

The channel access component of the logical interface (discussed in 4.4.1) comprises the collision and carrier detect functions described in 7.5.2 and 7.5.3, as well as the actual transmission of signals on the media. The data encoding and decoding functions described in 4.4.1 comprise the generation and decomposition of encoded signals suitable for transmission (described in 7.5.1), the generation and removal of code-specific preamble (described in 7.5.1.3 and 7.5.4.1), and the serial bit stream interface between the layers.

Section 5 describes the interface between the data link and physical layers as a series of Pascal procedures, functions, and shared variables. The data link specification in section 6 shows how the data link uses this interface to communicate between client layers. However, this specification will not attempt to model the operation of the physical channel in Pascal. The interface between layers is supported by the physical hardware which provides the ability to send and receive bit streams, provide timing, and signal carrier sense and collision detect to the data link.

The remainder of this section specifies the requirements for compatibility at both the transceiver cable and the coaxial cable. In addition, the specifications for the transceiver, which interfaces the transceiver cable to the coaxial cable is given, as well as the specification for the logic required between the transceiver cable and the interface to the data link.

7.2 Transceiver Cable Compatibility Interface Specifications

The transceiver cable is the means by which a physically separate transceiver is connected to a station. It provides one of the compatibility interfaces described in 7.1.4.2.

7.2.1 Transceiver Cable Signals

The transceiver cable carries four signals: Transmit, Receive, Collision Presence, and Power. Each signal is carried on a twisted pair of conductors in the cable.

7.2.1.1 Transmit Signal

The transmit pair carries encoded data for which the data link is requesting transmission on the channel. This signal is generated by the data encoder, with the transceiver cable drive characteristics specified in 7.2.4.

Amazon
Exhibit 1005
Page 203

7.2.1.2 Receive Signal

The receive pair carries encoded data from the transceiver to the station. It typically goes to the data decoder and the carrier sense circuitry. In the steady-state, all transitions and lack of transitions on the coaxial cable become transitions and lack of transitions on the receive pair, with the transceiver cable drive characteristics specified in 7.2.4. (During start-up, the first few bits may be absorbed by the transceiver to attain steady-state.)

In the case of a station transmitting without collision interference, the station's own transmit transitions on the coaxial cable will also appear on the receive pair, after a delay due to propagation through the transceiver. During collisions (whether or not that transceiver is involved in the collision) transitions on the receive lead are undefined; they may or may not meet decoder phase requirements, or they may not be present at all for extended periods. Thus the receive signal on the transceiver cable cannot be used alone to deterministically generate the carrier sense signal. This is described in more detail in 7.5.3.

7.2.1.3 Collision Presence Signal

The collision presence pair is used by the transceiver to indicate the presence of multiple transmission attempts on the coaxial cable. This is done by transmitting a square wave with a 10MHz fundamental frequency through the standard transceiver cable driver (described in 7.2.4). An oscillator is used instead of a simple level shift to allow AC coupling at the transceiver. Transceivers use the collision presence signal to indicate one of two conditions: the transceiver is transmitting and there is an attempt by another station to transmit at the same time, or there is a simultaneous transmission attempt by three or more stations regardless of whether the transceiver in question is transmitting.

7.2.1.4 Power

A pair of wires is designated for providing power to the transceiver. When the transceiver cable is implemented, the station end of the cable must supply a voltage between +12 and +15 Vdc ± 5% with at least 0.5 Amperes available to the cable for remotely powering the transceiver. The power source must meet applicable requirements for UL Class 2 wiring devices.

7.2.2 Transceiver Cable Parameters

7.2.2.1 Mechanical Configuration

The transceiver cable consists of four stranded, twisted pair conductors, plus an overall shield and insulating jacket. The conductor and jacket insulating material may be polyethylene or other suitable material. The flammability characteristics of the insulating material must be suitable for the installed environment.

7.2.2.2 Characteristic Impedance

The differential mode characteristic impedance of all pairs shall be 78 Ω, ± 5 Ω, in the configuration.

7.2.2.3 Attenuation

The signal attenuation of any pair shall not exceed 3 dB (measured at 10 MHz) for the total length between the transceiver and the station.

7.2.2.4 Velocity of Propagation

The minimum velocity of propagation of the transceiver cable shall be 0.65 c.

7.2.2.5 Pulse Distortion

Pulse distortion shall not exceed ± 1 nS at the end of 50 meters of cable when driven with random 10 Mbit data encoded in accordance with 7.5.1.

7.2.2.6 Resistance

The resistance of the conductors used for the power pair shall not exceed 40 milliohms per meter.

7.2.2.7 Transfer Impedance

The common mode transfer impedance of the transceiver cable shall not exceed the values shown in Figure 7-2 as a function of frequency. The differential mode transfer impedance of the cable with respect to any pair shall be 20 dB lower than the specified common mode transfer impedance.

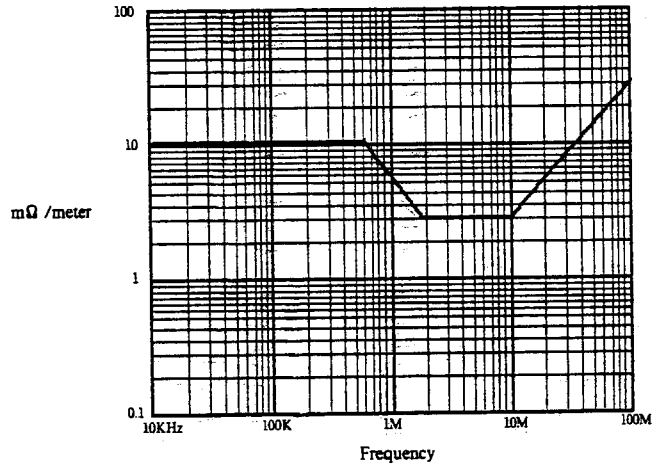


Figure 7-2: Transceiver Cable Transfer Impedance

Amazon
Exhibit 1005
Page 204

7.2.3 Transceiver Cable Connectors

The connectors used at the ends of the transceiver cable shall be 15 conductor 'D' subminiature types (Cinch type DASM-15 or equivalent). The end of the cable that mates with the transceiver must use a female connector with a slide lock assembly (Cinch type DA 51220-1 or equivalent). The transceiver must provide a mating male connector with locking posts. The other end of the transceiver cable (which mates with a female connector at the station) must use a male connector with locking posts (Cinch type D 53018 or equivalent). The station must provide a female connector with the slide lock assembly.

Because of the end-to-end matching of the connectors, transceiver cables may be extended by concatenating transceiver cable sections. (The transceiver cable sections function as 'extension cords'.) A cable with multiple sections must still meet the cable loss characteristics of 7.3.1.1.2.

The pin assignment is given in the following table:

Transceiver Cable Connector Pin Assignment

1.	Shield (See note)		
2.	Collision Presence +	9.	Collision Presence -
3.	Transmit +	10.	Transmit -
4.	Reserved	11.	Reserved
5.	Receive +	12.	Receive -
6.	Power return	13.	Power
7.	Reserved	14.	Reserved
8.	Reserved	15.	Reserved

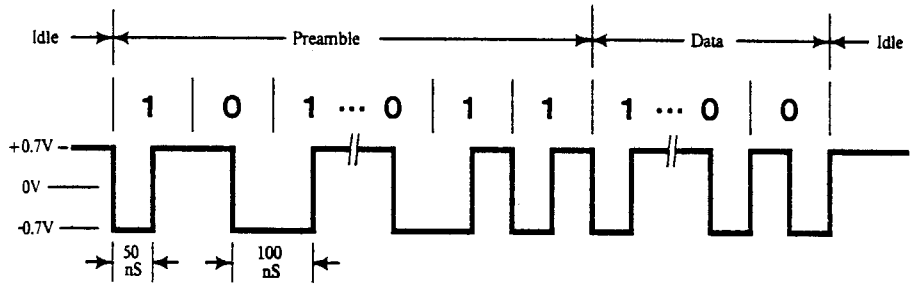
Note: Shield must be terminated to connector shell as well as pin 1.

Metal, metallized plastic, or otherwise shielded connector backshells must be used to ensure shield integrity.

7.2.4 Transceiver Cable Drive

This section describes the requirements for driving any of the signal pairs in the transceiver cable: transmit, receive, and collision presence.

The AC signal levels presented to the transceiver cable shall be ± 700 mV nominal, balanced differential drive into 78 ± 5 Ω. The common mode voltage presented to the transceiver cable shall not exceed that allowed at the receiver, as specified in 7.2.5.2. Signal waveform shall be as shown in Figure 7-3.



1. Voltages are measured differentially at output of transceiver cable driver.
2. Rise and fall times meet 10,000 series ECL requirements.

Figure 7-3: Typical Transceiver Cable Waveform

The transceiver cable driver must be capable of maintaining the specified minimum differential signal into the worst case low cable impedance (73 Ω differential, 18.5 Ω common mode) in the environment specified in section 7-7.

The idle state of the output shall be high (+ 700 mV nominal); the first transition presented is negative-going, the last transition must be positive-going. Note that the presence of AC coupling may cause the voltage as specified at the output of the transceiver cable drive circuit not to appear on the transceiver cable in the idle state.

A typical transceiver cable drive circuit is given in Appendix D.

7.2.5 Transceiver Cable Receive

The following sections specify the requirements for receiving signals from any signal pair in the transceiver cable: transmit, receive, and collision presence. The circuit must be capable of receiving the signals from the transceiver cable driver specified in 7.2.4 through the cable specified in 7.2.2 in the worst case. A typical receive circuit is given in Appendix D.

7.2.5.1 Load Impedance and Termination

The termination impedance shall be 78 Ω ± 1% differential mode, and 18.5 Ω minimum common-mode, over the frequency range of 3-20 MHz.

7.2.5.2 Common Mode and CMRR

The common mode range and the common mode rejection ratio shall be sufficient to maintain a 5:1 signal to noise ratio in the environment specified in 7.7, measured at the input to the transceiver cable receiver. The common mode DC voltage at the input of the receiver shall be in the range of zero to +5 Vdc..

Amazon
Exhibit 1005
Page 205

7.3 Coaxial Cable Compatibility Interface Specifications

The coaxial cable is the common, shared broadcast medium through which stations communicate. It provides one of the compatibility interface points described in 7.1.4.2.

7.3.1 Coaxial Cable Component Specifications

The cable is of constant impedance, coaxial construction. It is terminated at each end by a *terminator* (specified in 7.3.1.3), and connection provided for each transceiver. Coaxial cable connectors are used to make the connection from the cable to the terminators, and between cable sections (if needed). The cable has various electrical and mechanical requirements which must be met to ensure proper operation.

7.3.1.1 Coaxial Cable Parameters

7.3.1.1.1 Characteristic Impedance

The average characteristic impedance of the cable shall be 50 ± 2 Ω, measured according to Mil. Std. C17-E. Periodic variations in impedance along a single piece of cable may be up to ± 3 Ω sinusoidal, centered around the average value, with a period < 2 meters. Note that the proper operation of the network is dependent upon the cable characteristic impedance; its value and tolerance are critical.

7.3.1.1.2 Attenuation

The attenuation of a cable segment shall not exceed 8.5 dB measured at 10 MHz, nor 6.0 dB measured at 5 MHz.

7.3.1.1.3 Velocity of Propagation

The minimum acceptable velocity of propagation is 0.77 c..

7.3.1.1.4 Mechanical Requirements

The cable used should be suitable for routing in various environments, including but not limited to, dropped ceilings, raised floors, and cable troughs. The jacket must provide insulation between the cable sheath and any building structural metal. Also, the cable must be capable of accepting coaxial cable connectors, described in 7.3.1.2. The cable must in addition conform to the following requirements:

1. The center conductor must be 0.0855" ± .0005" diameter solid, tinned copper,
2. The core dielectric material must be foamed,
3. The inside diameter of the innermost shield must be .242" minimum,
4. The outside diameter of the outermost shield must be .326" ± .007",
5. The outermost shield must be greater than 90% coverage tinned copper braid,
6. The jacket O.D. must be 0.405" nominal,
7. The cable concentricity must be 90% minimum.

The cable must also meet applicable flammability criteria and local codes for the installed environment. Different (e.g., polyethylene and Teflon dielectric) types of cable sections may be interconnected, while meeting the sectioning requirements of 7.6.1.

7.3.1.1.5 Pulse Distortion

Pulse distortion shall not exceed ± 7 nS at the end of 500 meters of cable when driven with random 10 Mbit data encoded in accordance with 7.5.1.

7.3.1.1.6 Jacket Marking

The cable jacket must be marked with annular rings in a color contrasting with the background color of the jacket. The rings must be spaced at 2.5 meter ± 5 cm regularly along the entire length of the cable. It is permissible for the 2.5 meter spacing to be interrupted at discontinuities between cable sections joined by connectors. (See 7.6.2 for transceiver placement rules which mandate cable markings.)

7.3.1.1.7 Transfer Impedance

The transfer impedance of the cable shall not exceed the values shown in Figure 7-4 as a function of frequency.

Amazon
Exhibit 1005
Page 206

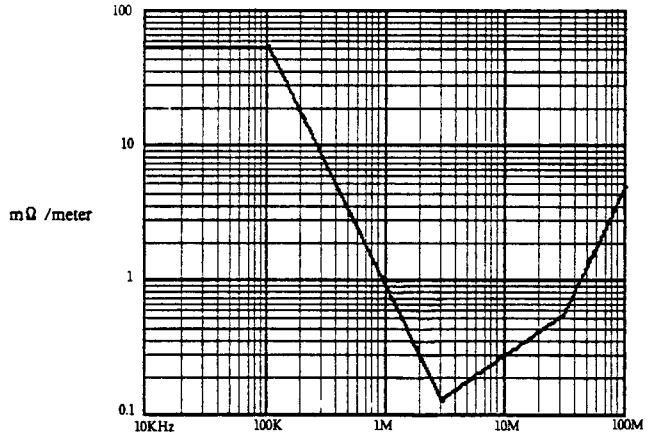


Figure 7-4: Maximum Coaxial Cable Transfer Impedance

7.3.1.2 Coaxial Cable Connectors

Coaxial cable connectors are used to join cable sections and attach terminators. Three types of connectors may be necessary; male plugs, female jacks, and female-to-female barrels. Plugs are used exclusively at the ends of all cable sections. Jacks are used to house cable terminators. Barrels are used to join cable sections.

All connectors are N series, 50 Ω constant impedance types. Since the frequencies present in the transmitted data are well below UHF range (being band-limited to approximately 20 MHz), military versions of the connectors are not required (but are acceptable).

Means must be provided to ensure that the connector shell (which connects to the cable sheath) does not make contact with any building metal, or other unintended conductor. A sleeve or boot to be slid over the connector at installation time is suitable.

7.3.1.3 Coaxial Cable Terminators

Coaxial cable terminators are used to provide a termination impedance for the cable equal in value to its characteristic impedance, thereby eliminating any reflection from the ends of the cables. Terminators shall be packaged within an inline female jack connector. The termination impedance shall be 50 Ω ± 1% measured from 0-50 MHz, with the magnitude of the phase angle of the impedance not to exceed 5 degrees. The terminator power rating shall be 1 watt or greater.

7.3.1.4 Transceiver-to-Coaxial Cable Connections

A means must be provided to allow for attaching a transceiver to the coaxial cable. The connection must disturb the transmission line characteristics of the cable as little as possible; it must present a predictably low shunt capacitance, and therefore a negligibly short stub length. For this reason, the transceiver must be located as close to its cable connection as possible; they are normally considered to be one assembly. Long (greater than 3 cm) connections between the coaxial cable and the input of the transceiver are not acceptable.

The transceiver-to-coaxial cable connection shall present less than 2 picofarads shunt capacitance to the coaxial cable, not including any transceiver electronics. If the design of the connection is such that the coaxial cable must be severed to install the transceiver, the coaxial cable segment must still meet the sectioning requirements of 7.6.1. Any coaxial connectors used on a severed cable must be type N, as specified in 7.3.1.2.

7.3.2 Coaxial Cable Signaling

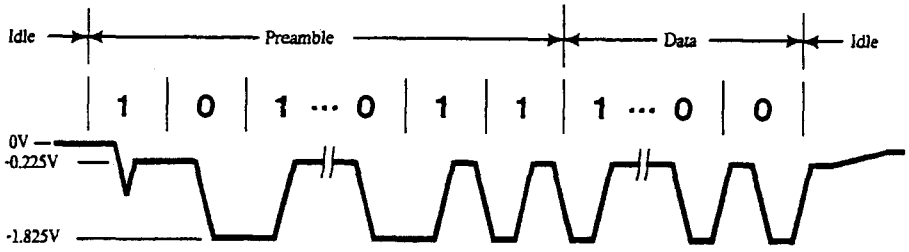
The AC component of the signal on the coaxial cable due to a single transceiver as measured on the coaxial cable immediately adjacent to the transceiver connection shall be ± 16 mA nominal (14 mA min, 19 mA max). The DC component shall be one-half the AC component, plus 4.5 mA (4 mA min, 5 mA max). The actual current measured at a given point on the cable is a function of the transmitted current and the cable loss to the point of measurement. Positive current is defined as current out of the center conductor of the cable (into the transceiver). Cable loss is specified in 7.3.1.1.2.

The 10%-90% rise and fall times shall be 25 ± 5 nsec. Figure 7-5 shows typical waveforms present on the cable. Harmonic content generated from a 10 MHz fundamental periodic input shall meet the following requirements:

Second and Third Harmonics:	-20 dB min
Fourth and Fifth Harmonics:	-30 dB min
Sixth and Seventh Harmonics:	-40 dB min
All Higher Harmonics:	-50 dB min

The signals as generated from the encoder (described in 7.5.1.1) shall appear on the coaxial cable without any inversions.

Amazon
Exhibit 1005
Page 207



1. Voltages given are nominal; worst case is given in text.
2. Rise time is 25 nS nominal.
3. Voltages are measured on coaxial cable adjacent to transceiver.

Figure 7-5: Typical Coaxial Cable Waveform

7.4 Transceiver Specifications

The following sections specify the requirements for a transceiver.

7.4.1 Transceiver-to-Coaxial Cable Interface

The following sections describe the interface between the transceiver and the coaxial cable. Positive current is defined as current into the transceiver (out of the center conductor of the cable).

7.4.1.1 Input Impedance

The shunt capacitance presented to the coaxial cable by the transceiver circuitry (not including the means of attachment to the coaxial cable) shall not exceed 2 picofarads. The shunt resistance presented to the coaxial cable shall be greater than 50 KΩ.

These conditions must be met in both the power off and the power on, not transmitting states.

7.4.1.2 Bias Current

The transceiver must draw between -2 and +50 uA in the power-off and the power-on, not transmitting states.

7.4.1.3 Transmit Output Levels

Signals received from the transceiver cable transmit pair must be transmitted onto the coaxial cable with the characteristics specified in 7.3.2. Note that 7.3.2 specifies the current level on the coaxial cable. Since the coaxial cable proceeds in two directions away from the transceiver, the current into the transceiver is actually twice the current measured on the coaxial cable.

Transmitted output asymmetry shall not exceed 2 ns for a 50/50 duty cycle input on the transceiver cable transmit pair.

7.4.2 Transceiver-to-Transceiver Cable Interface

7.4.2.1 Transmit Pair

The transceiver must present the transceiver cable receive characteristics specified in 7.2.5 to the transmit pair. At the start of a frame transmission, no more than 2 bits (two 100 ns bit cells) of information may be received from the transmit pair and not transmitted onto the coaxial cable. The steady-state propagation delay between the transmit pair input and the coaxial cable output shall not exceed 50 ns. There are no signal inversion between the transceiver cable transmit pair and the coaxial cable.

7.4.2.2 Receive Pair

The transceiver must present the transceiver cable transmit characteristics specified in 7.2.4 to the receive pair. Asymmetry as seen on the receive pair shall not exceed ± 2 nsec for a ± 200 mV peak sinusoidal input from the coaxial cable.

The signal from the coaxial cable shall pass through AC coupling with an appropriate time constant before proceeding to the receive pair. The time constant should compensate for the coaxial cable pulse distortion.

At the start of a frame reception from the coaxial cable, no more than 5 bits (five 100 ns bit cells) of information may be received from the coaxial cable and not transmitted onto the receive pair. In addition, it is permissible for the first bit sent over the receive pair to contain encoding phase violations or invalid data, however all successive bits of the frame shall be valid and meet encoding rules. The steady-state propagation delay between the coaxial cable and the receive pair output shall not exceed 50 ns. There are no signal inversions between the coaxial cable and the transceiver cable receive pair.

Amazon
Exhibit 1005
Page 208

7.4.2.3 Collision Presence Pair

The transceiver must present the transmitter characteristics specified in 7.2.4 to the collision presence pair. The signal presented to the collision presence pair shall be a periodic waveform with a 10 MHz ± 15% frequency. This signal shall be presented to the collision presence pair no more than 5 bit times (500 nS) after the average signal on the coaxial cable at the transceiver exceeds either that which could be produced by two transceiver outputs in the worst case (if the transceiver in question is not transmitting), or that which could be produced by that transceiver alone in the worst case (if that transceiver is transmitting).

7.4.2.4 Power Pair

The transceiver cable provides power which may be used for operation of the transceiver electronics. The power available shall be as described in 7.2.1.4. The distribution impedance of the transceiver cable is 4 Ω maximum, for a 50 meter cable with the resistance specified in 7.2.2.6. In order for the transceiver to derive its operating power from the power pair, circuitry must be employed to provide the required electrical isolation specified in 7.4.3.

7.4.3 Electrical Isolation

The transceiver must provide electrical isolation between the transceiver cable and the coaxial cable. The isolation impedance shall be greater than 250 KΩ, measured between any conductor (including shield) of the transceiver cable and either the center conductor or shield of the coaxial cable, at 60 Hz. The breakdown voltage of the isolation means provided shall be at least 250 VAC, rms.

7.4.4 Reliability

No single nor double component failure within the transceiver electronics shall impede communication among other transceivers on the coaxial cable. Connectors and other passive components comprising the means of connecting the transceiver to the coaxial cable shall be designed to minimize the probability of total network failure.

7.5 Channel Logic

The following sections describe the functions that must be performed to properly interface between the data link and the transceiver cable. They are normally implemented as logic, typically within the same device implementing the data link layer.

7.5.1 Channel Encoding

The channel shall use Manchester phase encoding, with a data rate of 10 Mbps, ± .01%, measured at the encoder clock. Thus, each bit cell is 100 ns long.

The following section describes the requirements for encoding and decoding signals to be transmitted on, or received from the coaxial or transceiver cables.

Amazon
Exhibit 1005
Page 209

7.5.1.1 Encoder

The encoder is used to translate physically separate signals of clock (synchronization) and data into a single, self-synchronizable serial bit stream, suitable for transmission on the coaxial cable by the transceiver.

During the first half of the bit cell time, the serial signal transmitted is the logical complement of the bit value being encoded during that cell. During the second half of the bit cell time, the uncomplemented value of the bit being encoded is transmitted. Therefore, there is always a signal transition (either positive-going or negative-going, depending on the bit being encoded) in the center of each bit cell. A timing diagram for a typical bit stream is given in Figure 7-6.

The encoder output drives the transmit pair of the transceiver cable, and ultimately, the coaxial cable through the transceiver. The encoder output asymmetry must not exceed 0.5 ns. The encoder shall provide the defined output for the first (and all subsequent) bits presented to its input. All information submitted for encoding shall appear at the output of the encoder.

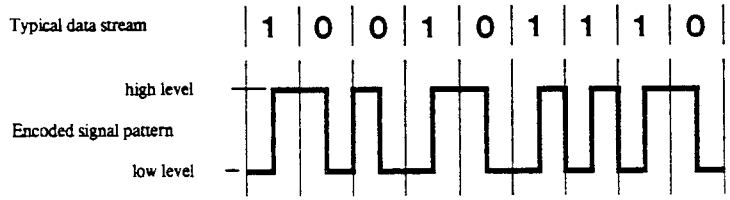


Figure 7-6: Manchester Encoding

7.5.1.2 Decoder

The decoder is used to separate the incoming phase encoded bit stream into a data stream and a clock signal. The decoder must be able to provide data and clock signals usable by the data link under the asymmetry imposed by the worst case system configuration. The decoder must provide usable output (clock and data) after no more than 8 bit cell times after reception of an encoded signal. The first signals received from the transceiver at the beginning of frame reception may not constitute a valid, properly encoded bit; it is possible for the time from the first transition seen to the first true mid-bit cell transition to assume any value from zero to 100 nS.

The decoder input is normally derived from the coaxial cable, through the transceiver cable receive pair. It is not necessary for the decoder to provide usable output when there is a collision on the coaxial cable, regardless of whether the station using that decoder is involved in the collision.

7.5.1.3 Preamble Generation

Because most of the channel circuitry is allowed to provide valid output some number of bit times after being presented valid input, it is necessary for a preamble to be sent before the start of data link information, to allow the channel circuitry to reach its steady-state, with valid outputs throughout the system. Upon request by the data link to transmit the first bit of a new frame, the channel shall first transmit the preamble; a predetermined bit sequence used for channel stabilization and synchronization. If, while transmitting the preamble, the channel logic asserts the collision detect signal as specified in 7.5.2, any remaining preamble bits shall not be sent. The channel should immediately proceed with the transmission of the bit submitted by the data link.

The preamble is a 64 bit pattern to be presented to the channel encoder in the same manner as data link information. The pattern is:

10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101011.

The bits are transmitted in order, from left to right. The nature of the pattern is such that when encoded, it appears as a periodic waveform on the cable, with a 5 MHz frequency. Excepting the final two bits, the only transitions present in the waveform are in the center of the bit cells. This is depicted in Figure 7-7. The last two bits of the preamble contain transitions at both the bit cell centers and the edges, and are used to indicate the end of the preamble, and the beginning of the data link encapsulation portion of the frame. The next bit transmitted is the bit originally submitted by transmission by the data link.

Preamble removal on reception is discussed in 7.5.4.1.

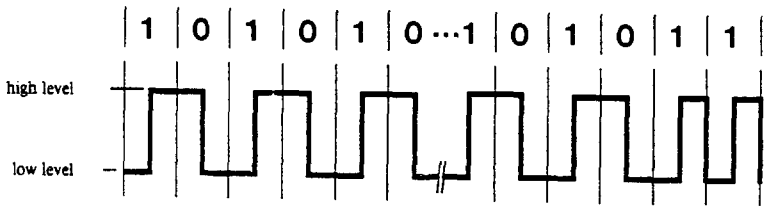


Figure 7-7: Preamble Encoding

7.5.2 Collision Detect Signal

The channel must indicate to the data link when the signals on the coaxial cable imply simultaneous transmission attempts by more than one station. This is normally indicated through the collision presence pair in the transceiver cable, described in 7.2.1.3.

The channel logic must assert the collision detect signal within 2 bit times (200 ns), following the onset of collision presence. This collision detect signal shall be asserted only when the data link is transmitting. A functional logic description of the collision detect signal is shown in Figure 7-8.

Following the loss of collision presence information, the channel must deassert the collision detect signal within 1.6 bit cell times (160 ns).

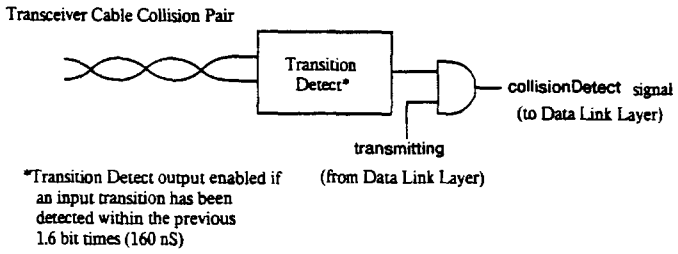


Figure 7-8: Functional Logic of collisionDetect Signal

7.5.3 Carrier Sense Signal

The channel must indicate to the data link the presence of carrier, a signal transmission attempt on the coaxial cable by a station. This is normally indicated through both the receive and collision presence pairs in the transceiver cable, described in 7.2.1.

The carrier sense signal shall be asserted when one or more station is attempting transmission on the cable, regardless of whether the station sensing carrier is transmitting at that time. The channel logic must assert the carrier sense signal within 2 bit times (200 ns) following the onset of carrier presence information. A

Amazon
Exhibit 1005
Page 210

functional logic description of these signals is shown in Figure 7-9.

Following the loss of carrier presence information (receive transitions and collision presence information) the channel must deassert the carrier sense signal within 1.6 bit cell times (160 ns).

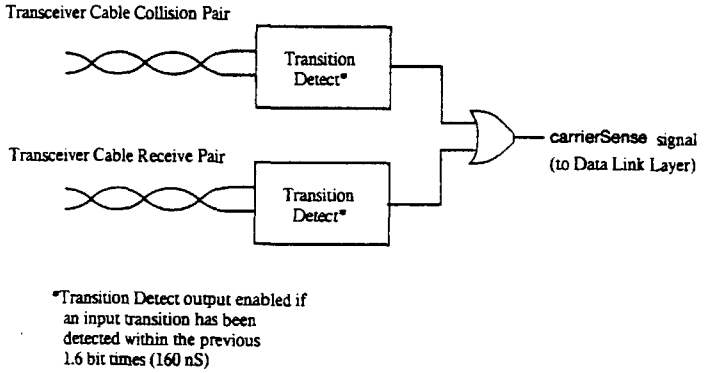


Figure 7-9: Functional Logic of carrierSense Signal

7.5.4 Channel Framing

During reception, the channel must provide the data link with signals to indicate beginning and end of frame.

7.5.4.1 Beginning-of-Frame Sequence

The channel logic recognizes the presence of activity on the medium through the carrier sense signal. This is the first indication that the frame reception process should begin. However, dependent upon the physical configuration of the system, there are some number of preamble bits to be received by the channel before the start of the data link frame as indicated by the double-1 at the end of preamble. In addition, the first signals received from the decoder may be invalid due to the first bit allowance of the transceiver (see 7.4.2.2). The channel must wait no less than 8 bit times (800 nS) before monitoring the output of the decoder for the 'double-1' indicating end of preamble, and beginning of data link frame. Upon reception of the double-1, the channel shall begin passing successive bits to the data link through the defined receive bit stream interface. If, after waiting the required 8 bit times, a 'double-0' is encountered, the physical channel shall not pass any bits of

the current frame to the data link. Normal operation of the data link and channel shall resume on the subsequent frame.

7.5.4.2 End-of-Frame Sequence

As specified in 7.5.3, the carrier sense signal must be deasserted no later than 1.6 bit times (160 ns) after the cessation of activity on the coaxial cable as seen by the channel logic. The channel ensures that no extraneous bits will appear at the end of a frame following the last valid bit.

7.6 Channel Configuration Requirements

7.6.1 Cable Sectioning

The 500 meter maximum length coaxial cable segment need not be made from a single, homogeneous length of cable. The boundary between two cable sections (joined by coaxial connectors; two male plugs and a barrel) represents a signal reflection point due to the impedance discontinuity caused by the batch-to-batch impedance tolerance of the cable. Since the worst-case variation from 50 Ω is 2 Ω (see 7.3.1.1.1), a possible worst-case reflection of 4% may result from the joining of two cable sections. The configuration of long cable segments (up to 500 meters) from smaller sections must be made with care. The following recommendations apply, and are given in order of preference:

1. If possible, the total segment should be made from one homogeneous (no breaks) cable. This is feasible for short segments, and results in minimal reflections from cable impedance discontinuities.
2. If cable segments must be built up from smaller sections, it is highly desirable to ensure that all the sections are from the same manufacturer and lot. This is equivalent to using a single cable, since the cable discontinuities are due to extruder limitations, and not extruder-to-extruder tolerances. There are no restrictions in cable sectioning if this method is used. However, if a cable section in such a system is later replaced, it must be replaced either with another cable from the same manufacturer and lot, or with one of the standard lengths described below.
3. If uncontrolled cable sections must be used in building up a longer segment, the lengths should be chosen such that reflections, when they occur, do not have a high probability of adding in phase. This can be accomplished by using lengths which are odd integral multiples of a half-wavelength in the cable at 5 MHz; this corresponds to using lengths of 23.4, 70.2, and 117 meters (± 0.5 meters) for all sections. These are considered to be the standard lengths for all cable sections. Using these lengths exclusively, any mix or match of cable sections may be used to build up a 500 meter segment without incurring excessive reflections.

59

Amazon
Exhibit 1005
Page 211

4. As a last resort, an arbitrary configuration of cable sections may be employed, if it has been confirmed by analysis or measurement that the worst-case signal reflection due to the impedance discontinuities at any point on the cable does not exceed 7% of the incident wave when driven by a transceiver meeting the specifications of 7.4.

7.6.2 Transceiver Placement

Transceivers and their associated connections to the cable cause signal reflections due to their non-infinite bridging impedance. While this impedance must be implemented as specified in 7.3.1.4 and 7.4.1, the placement of transceivers along the coaxial cable must also be controlled to insure that reflections from transceiver do not add in phase to a significant degree.

Coaxial cables marked as specified in 7.3.1.1.6 have marks at regular 2.5 meters spacing; a transceiver may be placed at any mark on the cable. This guarantees both a minimum spacing between transceivers of 2.5 meters, as well as controlling the relative spacing of transceivers to insure non-alignment on fractional wavelength boundaries.

The total number of transceivers on a cable segment shall not exceed 100.

7.6.3 System Grounding

The sheath conductor of the coaxial cable shall not make electrical contact with any earth reference, building structural metal, ducting, plumbing fixture, or other unintentioned conductor. Insulators may be used to cover any coaxial connectors used to join cable sections and terminators, to insure that this requirement is met. A sleeve or boot attached at installation time is acceptable.

The sheath conductor of the transceiver cable shall be connected to the earth reference or chassis of the device housing the station logic.

60

7.6.4 Repeaters

Repeaters are used to extend the channel length and topology beyond that which could be achieved by a single coaxial cable segment. (See the channel configuration model in 7.1.5.) A repeater requires a transceiver on each of the segments between which it is repeating signals. These transceivers must be as specified in 7.4, and must be counted towards the maximum specified in 7.6.2.

A maximum of two repeaters may be in the signal path between any two transceivers on the channel.

7.6.4.1 Carrier Detect and Transmit Repeat

Repeaters must implement the carrier sense function as specified in 7.5.3 for both segments between which it is connected. Upon detection of carrier from one segment, the repeater must retransmit all received signals from that segment onto the other segment. Signals shall be retimed and amplified as specified in 7.6.4.3. The maximum steady-state propagation delay through the repeater for the repeated signal (not including startup delays, carrier sense delay or retiming delays) shall not exceed 800 nS.

7.6.4.2 Collision Detect and Collision Repeat

Repeaters must implement the collision detect function as specified in 7.5.2 for both segments between which it is connected. If, while repeating signals as specified in 7.6.4.1, collision is detected on either side, the repeater must ensure that all stations involved in the collision recognize the event as a collision, regardless of which side of the repeater the station is on. The maximum time between the recongnition of the collision and the repeating of the collision indication (not including carrier sense of retiming delays) shall not exceed 200 nS.

7.6.4.3 Repeater Signal Regeneration

7.6.4.3.1 Signal Amplification

The repeater (with its associated transceivers) shall ensure that any signals repeated between segments shall have the same amplitude characteristics at the transceiver output of the repeated-to segment as they did at the output of the transmitter on the repeated-from segment, allowing for transceiver output tolerances as specified in 7.4.1.3. Any loss of signal-to-noise ratio due to cable loss and noise pickup is thus regained at the output of the repeater.

7.6.4.3.2 Signal Timing

The repeater must ensure that the symmetry characteristics of the signals at the transceiver output of the repeated-to segment are the same as those at the output of the transmitter on the repeated-from segment, allowing for transceiver and transceiver cable tolerances. Any loss of symmetry due to transceivers and cable distortion is thus regained at the output of the repeater.

Amazon
Exhibit 1005
Page 212

7.7 Environment Specifications

The following sections specify the physical environment in which all channel components must operate to be considered compatible.

7.7.1 Electromagnetic Environment

The physical channel hardware shall meet its specifications when operating in the following ambient plane-wave fields:

2 Volts/Meter from 10 KHz through 30 MHz

5 Volts/Meter from 30 MHz through 1 GHz

7.7.2 Temperature and Humidity

All physical channel hardware, with the possible exception of the channel logic components shall operate over the ambient temperature range of 5 to 50 degrees Celsius, and humidity range of 10% to 95% non-condensing. The channel logic components are normally part of the station hardware, and are thus subject to individual station product requirements. Hardware which does not meet the temperature and humidity requirements specified must state so in its published product specification.

Amazon
Exhibit 1005
Page 213

APPENDIX A: GLOSSARY

This section defines some of the essential terminology associated with the Ethernet.

baseband coaxial system: A system whereby information is directly encoded and impressed on the coaxial transmission medium. One information signal at a time can be present on the medium without disruption (see collision).

binary exponential backoff: The algorithm used to schedule retransmissions after a collision. So called because the interval from which the retransmission time is selected is expanded exponentially with repeated collisions.

broadcast: Describes the class of media for which the Ethernet is designed, in which all stations are capable of receiving a signal transmitted by any other station. Also, describes the mode of usage of such a medium by the Data Link Layer in which all stations are instructed to receive a given frame.

carrier sense: A signal provided by the Physical Layer to the Data Link Layer to indicate that one or more stations are currently transmitting on the channel.

channel logic: The logical functions provided between the transceiver cable and the Data Link, which support the defined interface between the data link and the physical layers.

Client Layer: Collective term used to describe any layer of a network architecture, which use the Ethernet Data Link and Client interface.

coaxial cable: A two-conductor, concentric, constant impedance transmission line.

coaxial cable interface: The electrical, mechanical, and logical interface to the shared coaxial cable medium. This is a mandatory compatibility interface, which must be correctly implemented by every Ethernet implementation.

coaxial cable section: An unbroken piece of coaxial cable, fitted with coaxial connectors at its ends, used to build up coaxial cable segments.

coaxial cable segment: A length of coaxial cable made up from one or more coaxial cable sections and coaxial connectors, terminated at each end in its characteristic impedance. A 500 meter segment is the longest configuration possible without repeaters.

collision: The result of multiple transmissions overlapping in the physical channel, resulting in garbled data and necessitating retransmission.

collision detect: A signal provided by the Physical Layer to the Data Link Layer to indicate that one or more other stations are contending with the local station's

transmission. It can be true only during transmission.

collision enforcement: Transmission of extra, encoded "jam" bits after a collision is detected, to insure that the duration of the collision is sufficient to guarantee its detection by all transmitting stations.

compatibility interfaces: The coaxial cable interface, and the transceiver cable interface, the two points at which hardware compatibility is defined to allow connection of independently designed and manufactured components to the Ethernet.

contention: Interference between colliding transmissions (see collision). Resolution of occasional contention is a normal part of the Ethernet's distributed link management procedure (see CSMA-CD).

controller: The implementation unit which connects a station to the Ethernet, typically comprising part of the Physical Layer, much or all of the Data Link Layer, and appropriate electronics for interfacing to the station.

CSMA-CD: Carrier Sense Multiple Access with Collision Detection, the generic term for the class of link management procedure used by the Ethernet. So called because it a) allows multiple stations to access the broadcast channel at will, b) avoids contention via carrier sense and deference, and c) resolves contention via collision detection and retransmission.

Data Link Layer: The higher of the two layers in the Ethernet design, which implements a medium-independent link level communication facility on top of the physical channel provided by the Physical Layer.

deference: A process by which a data link controller delays its transmission when the channel is busy to avoid contention with ongoing transmissions.

frame check sequence: An encoded value appended to each frame by the Data Link Layer to allow detection of transmission errors in the physical channel.

interframe spacing: An enforced idle time between transmission of successive frames to allow receiving data link controllers and the physical channel to recover.

jam: An encoded bit sequence used for collision enforcement.

Manchester encoding: A means by which separate data and clock signals can be combined into a single, self-synchronizable data stream, suitable for transmission on a serial channel.

multicast: An addressing mode in which a given frame is targeted to a group of logically related stations.

physical address: The unique address value associated with a given station on the network. An Ethernet physical address is defined to be distinct from all other physical addresses on all Ethernets.

Physical Channel: The implementation of the physical layer.

Physical Layer: The lower of the two layers of the Ethernet design, implemented by the physical channel using the specified coaxial cable medium. The Physical Layer insulates the Data Link Layer from medium-dependent physical characteristics.

preamble: A sequence of 64 encoded bits which the Physical Layer transmits before each frame to allow synchronization of clocks and other Physical Layer circuitry at other sites on the channel.

repeater: A device used to extend the length and topology of the physical channel beyond that imposed by a single segment, up to the maximum allowable end-to-end channel length.

round-trip propagation time: In bit times, the time required in the worst-case for a transmitting station's collision detect signal to be asserted due to normal contention for the channel. This delay is the primary component of the slot time.

slot time: A multi-purpose parameter which describes the contention behavior of the Data Link Layer. It serves as a) an upper bound on the collison vulnerability of a given transmission, b) an upper bound on the size of the frame fragment produced by a collision, and c) the scheduling quantum for collision retransmission.

station: A single addressable site on the Ethernet, generally implemented as a computer and appropriate peripherals, and connected to the Ethernet via a controller and a transceiver.

transceiver: The portion of the Physical Layer implementation that connects directly to the coaxial cable and provides both the electronics which send and receive the encoded signals on the cable and the required electrical isolation.

transceiver cable: A four pair, shielded cable used for the transceiver cable interface.

transceiver cable interface: The electrical, mechanical and logical interface which connects the transceiver to the controller. The standard transceiver cable is a recommended compatibility interface.

6.2

Amazon
Exhibit 1005
Page 214

APPENDIX B: NOTES ON ADDRESS AND TYPE ASSIGNMENT, AND LICENSING

Address and Type Assignment

The address and type fields will be administered by Xerox Corporation.

A block of addresses will be assigned to each licensee of Ethernet patents (see below). Others may obtain an address block or type field assignment by request. A nominal fee to cover administrative costs will be charged.

Submit written requests to:

Xerox Corporation
Ethernet Address Administration Office
3333 Coyote Hill Road
Palo Alto, CA 94304

Licensing

Ethernet incorporates features that are protected by one or more patents assigned to Xerox Corporation. Questions on the need for licensing particular uses of this specification should be directed to:

Xerox Corporation
Director of Licensing
Long Ridge Road
Stamford, CT 06904

Amazon
Exhibit 1005
Page 215

APPENDIX C: CRC IMPLEMENTATION

Every frame contains, in its frame check sequence field, a 32-bit cyclic redundancy check (CRC) code. Because the formal mathematical definition of this code (see 6.2.4) is not suggestive of an appropriate implementation, this appendix outlines one possible implementation in terms of a feedback shift register. This type of implementation is likely to be common in practice, but is not a mandatory part of the specification.

The feedback shift register (see Figure C-1) is used to represent division of the pre-scaled message by the generating polynomial. The 32-bit register is accessed via the three signals Input, Output, and Control. When Control = 1, Input bits are shifted into the feedback shift register and also fed directly back to Output. When Control = 0, the feedback paths are disabled and the shift register shifts the complement of its contents to Output.

Before CRC generation at the transmitting end, initialization logic (not shown in Figure C-1) preloads the shift register to all 1's. Control is then held at 1 while the address, type and data fields of the outgoing frame are shifted into Input and the CRC is generated. Meanwhile, the same bits emerging at Output are transmitted over the network. When the last bit of the data field has been processed, Control is set to 0 and the complemented CRC is shifted out for transmission, starting with the x^{31} term (see 6.2.4).

CRC checking at the receiving end also begins with the shift register preloaded to all 1's. Control is then held at 1 while the incoming bits are shifted into Input to regenerate the CRC. When the last bit of the data field has been processed, the shift register should contain the CRC whose binary complement is about to arrive on the network. Since this field boundary cannot be recognized by the receiver, however, Control remains at 1 and the bits of the CRC continue to feed into the the shift register until the end of the entire frame is reached. If the two CRCs match, the final contents of the shift register is the value:

11000111 00000100 11011101 01111011

(where the leftmost bit corresponds to the x^{31} term of the polynomial and the rightmost to the x^0 term). Any other final value indicates a detected error. (The extra logic to test for this value is not shown in Figure C-1).

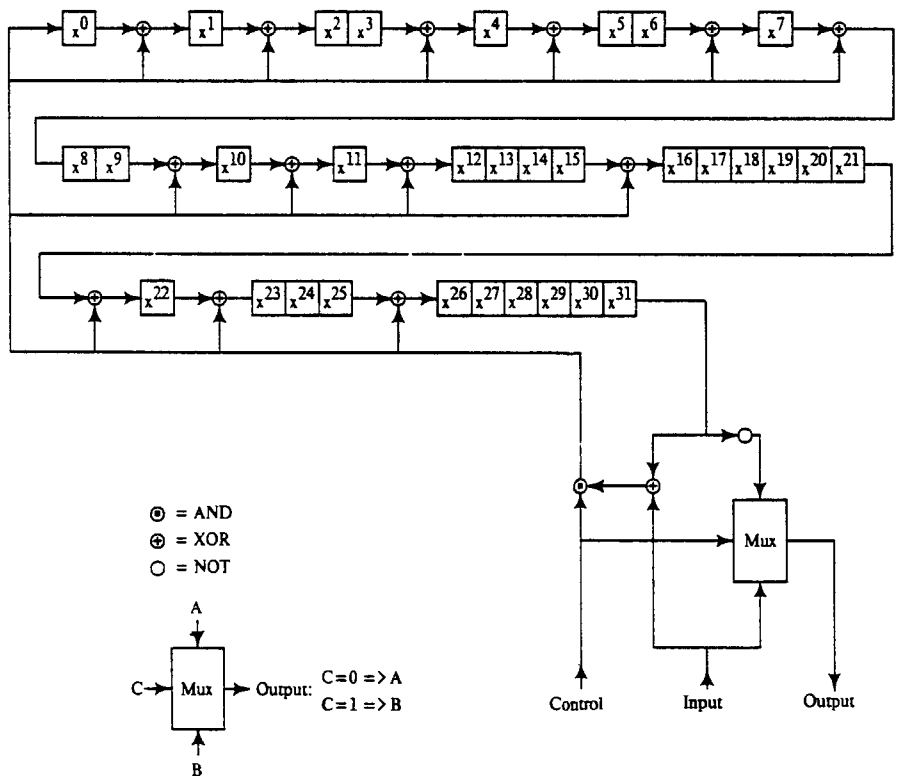


Figure C-1: CRC Implmentation

One potential problem which is avoided in this implementation is insensitivity of the shift register to incoming zero-bits when it is in the all-zero state. Following standard practice, this state is avoided at the beginning and end of the frame by preloading the shift register with all 1-bits, and by inverting each bit of the final CRC. Logically, these correspond, respectively, to the complementing of the first 32 bits of the frame and to the final complementing of the remainder, as specified in the mathematical definition in 6.2.4. See also [9] for further discussion.

Amazon
Exhibit 1005
Page 216

APPENDIX D: IMPLEMENTATION OF TRANSCEIVER CABLE DRIVER AND RECEIVER

This appendix presents circuit digrams for typical implementations of the transceiver cable drivers and receivers. The use of these exact circuits is not necessary for conformance to the specification; equivalent circuits may be used as long as the relevant specifications are met.

Figure D-1 depicts an implementation of the transceiver cable driver specified in 7.2.4. It is suitable for use at either end of the transceiver cable, as necessary; i.e., it would be located at the station end to drive the transmit pair, and at the transceiver end to drive the receive and collision presence pairs. In addition, it is capable of driving suitable isolation circuits required to be located within the transceiver.

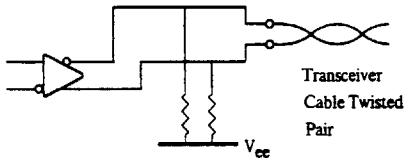


Figure D-1: Typical Transceiver Cable Driver

Figure D-2 depicts an implementation of the transceiver cable receiver specified in 7.2.5. It is suitable for use at either end of the transceiver cable, as necessary; i.e., it would be located at the station end to receive from the receive and collision presence pairs, and at the transceiver to receive from the transmit pair. It is capable of operating through suitable isolation circuits required to be located within the transceiver.

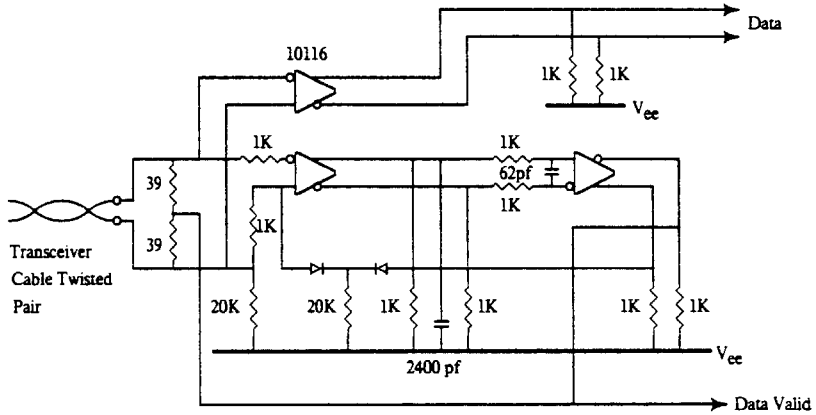


Figure D-2: Typical Transceiver Cable Receiver

Amazon
Exhibit 1005
Page 217

APPENDIX E: INTERFRAME RECOVERY

It is important that data link controller implementations be able to receive a frame that arrives immediately after another frame has been transmitted or received. Here, "immediately" means 9.6 μsec, based on the minimum interframe spacing provided as recovery time for the data link. (See 6.3.2.2) It is important that the data link controller be able to resume reception within that time.

Reception of multiple closely spaced incoming frames is a very desirable capability, and is crucial for stations which tend to communicate with several other stations concurrently. There is one important case in which a data link controller implementation cannot reasonably be expected to receive closely spaced incoming frames: if the station hardware (e.g. I/O bus) is intrinsically unable to accept the bits of a frame at the rate at which they arrive over the network, each incoming frame must be buffered to allow the station to accept it at some lower rate. Assuming limited buffering resources (e.g. a one frame buffer), reception of subsequent frames cannot occur until sufficient buffer space is available. This mode of operation is allowed for low performance stations.

Reception of an incoming frame immediately after transmission of an outgoing frame is a very important capability, even for stations which do not tend to communicate with several other stations concurrently. All stations, low performance to high performance, should allow reception of an incoming frame immediately after transmission of an outgoing frame.

Amazon
Exhibit 1005
Page 218

APPENDIX 1009D

5
A566682
CCR

COMPUTER COMMUNICATION REVIEW

KURT F. WENDT LIBRARY
COLLEGE OF ENGINEERING
OCT 4 1981
UW-MADISON, WI 53706

A Quarterly Publication of the
Special Interest Group on Data Communication

July 1981 Volume 11 Number 3

CONTENTS:

From the Editor 1
From the Chairman 2
ACM National Lecturers 3
Information Institute Proposed by Rep. Brown 4
Protocol Workshop Report; C. Sunshine 7
Clarification of TCP End of Letter; J. Postel 9
Vulnerabilities of Network Control Protocols: An Example; E.C. Rosen 10
An Introduction to the Ethernet Specification; J.F. Shoch .. 17
The Ethernet: Data Link Layer and Physical Layer Specifications 20
Articles in Computer Communications from Other Publications 67
Calendar of Events 70

Amazon
Exhibit 1005
Page 220

COMPUTER COMMUNICATION REVIEW

A QUARTERLY PUBLICATION OF THE SPECIAL INTEREST GROUP ON DATA COMMUNICATION

Executive Committee

David C. Wood (Present Chairman)
Robert E. Kahn (Past Chairman)
Carl A. Sunshine
Helen M. Wood
Peter J. Sevcik

Chairman:

David C. Wood
The MITRE Corporation
1820 Dolley Madison Blvd.
McLean, VA 22102
(703)827-6394
ARPANET: WOOD @ MITRE

Vice-Chairman

Carl A. Sunshine
USC Information Sciences Inst.
4676 Admiralty Way
Marina del Rey, CA 90291
(213)822-1511
ARPANET: SUNSHINE @ USC-ISIF

Secretary-Treasurer:

Helen M. Wood
National Bureau of Standards
A209 Administration Building
Washington, DC 20234
(301)921-2834
ARPANET: WOOD @ NBS-10

Editor:

Peter J. Sevcik
Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, MA 02238
(617)497-3511
ARPANET: SEVCIK @ BBNG

COMPUTER COMMUNICATION REVIEW is an informal quarterly publication of the ACM Special Interest Group on Data Communication. Its scope of interest includes: data communication systems for computers; data communication technology for computers; reliability, security and integrity of data on data communication systems; problems of interfacing communication systems and computer systems; computer communicati system modeling and analysis.

Membership in SIGCOMM (at $9 per annum) is open to ACM members and associate members; student members may join SIGC at $6 per annum; non-members of ACM ma join SIGCOMM at $23 per annum. All memb of SIGCOMM will receive COMPUTER COMMUNICATION REVIEW.

Items attributed to persons will ordinarily be interpreted as personal rather than organizational opinions. Technical papers appearing in the Computer Communication Review are unrefereed.

Note: This issue rec'd at ACM HQ for production on 8/13/81

Amazon
Exhibit 1005
Page 221

The Ethernet

A Local Area Network

Data Link Layer
and
Physical Layer
Specifications







Digital Equipment Corporation
Maynard, MA

Intel Corporation
Santa Clara, CA

Xerox Corporation
Stamford, CT

Version 1.0

September 30, 1980

Amazon
Exhibit 1005
Page 222

This material may be protected by Copyright law (Title 17 U.S. Code)

IMPORTANT INFORMATION AND DISCLAIMERS

1. This specification includes subject matter relating to a patent(s) of Xerox Corporation. No license under such patent(s) is granted by implication, estoppel or otherwise as a result of publication of this specification. Applicable licenses may be obtained from Xerox Corporation.

2. This specification is furnished for informational purposes only. Digital, Intel, and Xerox do not warrant or represent that this specification or any products made in conformance with it will work in the intended manner or be compatible with other products in a network system. Nor do they assume responsibility for any errors that the specification may contain, or have any liabilities or obligations for damages (including but not limited to special, indirect or consequential damages) arising out of or in connection with the use of this specification in any way. Digital, Intel and Xerox products may follow or deviate from the specification without notice at any time.

3. No representations or warranties are made that this specification or anything made from it is or will be free from infringements or patents of third persons.

Preface

This document contains the specification of the Ethernet, a local area network developed jointly by Digital Equipment Corporation, Intel Corporation, and Xerox Corporation. The Ethernet specification arises from an extensive collaborative effort of the three corporations, and several years of work at Xerox on an earlier prototype Ethernet.

This specification is intended as a design reference document, rather than an introduction or tutorial. Readers seeking introductory material are directed to the reference list in Section 2, which cites several papers describing the intent, theory, and history of the Ethernet.

This document contains 7 sections, falling into three main groups:

Sections 1, 2, and 3 provide an overall description of the Ethernet, including its goals, and the scope of the specification.

Sections 4 and 5 describe the architectural structure of the Ethernet in terms of a functional model consisting of two layers, the Data Link Layer and the Physical Layer.

Sections 6 and 7 specify the two layers in detail, providing the primary technical specification of the Ethernet.

Readers wishing to obtain an initial grasp of the organization and content of the specification will be best served by reading Sections 1, 3, and 4. Readers involved in actual implementation of the Ethernet will find Sections 5, 6, and 7 to contain the central material of the specification. Section 2 provides references, and the appendices provide supplementary material.

The approach taken in the specification of the Data Link Layer in Section 6 is a procedural one; in addition to describing the necessary algorithms in English and control flow charts, the specification presents these algorithms in the language Pascal. This approach makes clear the required behavior of Data Link Layer, while leaving individual implementations free to exploit any appropriate technology.

Because the procedural approach is not suitable for specifying the details of the Physical Layer, Section 7 uses carefully worded English prose and numerous figures and tables to specify the necessary parameters of this layer.

Some aspects of the Ethernet are necessarily discussed in more than one place in this specification. Whenever any doubt arises concerning the official definition in such a case, the reader should utilize the Pascal procedural specification of the Data Link Layer in Section 6.5, and the detailed prose specification of the Physical Layer in Sections 7.2 through 7.9.

Amazon
Exhibit 1005
Page 223

One aspect of an overall network architecture which is not addressed by this specification is network management. The network management facility performs operation, maintenence, and planning functions for the network:

- Operation functions include parameter setting, such as address selection.
- Maintenance functions provide for fault detection, isolation, and repair.
- Planning functions include collection of statisical and usage information, necessary for planned network growth.

While network management itself is properly performed outside the Ethernet Data Link and Physical Layers, it requires appropriate additional interfaces to those layers, which will be defined in a subsequent version of this specification.

Amazon
Exhibit 1005
Page 224

Table of Contents

Preface i
1. INTRODUCTION 1
2. REFERENCES 3
3. GOALS AND NON-GOALS 4
3.1 Goals 4
3.2 Non-Goals 5
4. FUNCTIONAL MODEL OF THE ETHERNET ARCHITECTURE 6
4.1 Layering 8
4.2 Data Link Layer 9
4.3 Physical Layer 10
4.4 Ethernet Operation and the Functional Model 12
4.4.1 Transmission Without Contention 12
4.4.2 Reception Without Contention 12
4.4.3 Collisions: Handling of Contention 13
5. INTER-LAYER INTERFACES 15
5.1 Client Layer to Data Link Layer 15
5.2 Data Link Layer to Physical Layer 16
6. ETHERNET DATA LINK LAYER SPECIFICATION 19
6.1 Data link Layer Overview and Model 19
6.2 Frame format 19
6.2.1 Address Fields 21
6.2.1.1 Destination Address Field 21
6.2.1.2 Source Address Field 21
6.2.2 Type Field 21
6.2.3 Data Field 22
6.2.4 Frame Check Sequence Field 22
6.2.5 Frame Size Limitations 22
6.3 Frame Transmission 23
6.3.1 Transmit Data Encapsulation 23
6.3.1.1 Frame Assembly 23
6.3.1.2 Frame Check Sequence Generation 23
6.3.2 Transmit Link Management 23

6.3.2.1 Carrier Deference 23
6.3.2.2 Interframe Spacing 24
6.3.2.3 Collision Handling 24
6.3.2.3.1 Collision Detection and Enforcement 24
6.3.2.3.2 Collision Backoff and Retransmission 25

6.4 Frame Reception 25
6.4.1 Receive Data Decapsulation 25
6.4.1.1 Framing 25
6.4.1.1.1 Maximum Frame Size 26
6.4.1.1.2 Integral Number of Octets in Frame 26
6.4.1.2 Address Recognition 26
6.4.1.2.1 Physical Addresses 26
6.4.1.2.2 Multicast Addresses 26
6.4.1.3 Frame Check Sequence Validation 26
6.4.1.4 Frame Disassembly 27
6.4.2 Receive Link Management 27
6.4.2.1 Collision Filtering 27

6.5 The Data Link Layer Procedural Model 27

6.5.1 Overview of the Procedural Model 27

6.5.1.1 Ground Rules for the Procedural Model 27

6.5.1.2 Use of Pascal in the Procedural Model 29

6.5.2 The Procedural Model 31
6.5.2.1 Global Declarations 34
6.5.2.1.1 Common Constants and Types 34
6.5.2.1.2 Transmit State Variables 35
6.5.2.1.3 Receive State Variables 35
6.5.2.1.4 Summary of Interlayer Interfaces 36
6.5.2.1.5 State Variable Initialization 37
6.5.2.2 Frame Transmission 38
6.5.2.3 Frame Reception 42
6.5.2.4 Common Procedures 44

7. ETHERNET PHYSICAL LAYER SPECIFICATION 45

7.1 Physical Channel Overview and Model 45

7.1.1 Channel Goals and Non-Goals 45
7.1.1.1 Goals 45
7.1.1.2 Non-Goals 45

7.1.2 Characteristics of the Channel 46

7.1.3 Functions Provided by the Channel 46

7.1.4 Implementation of the Channel 46
7.1.4.1 General Overview of Channel Hardware 47
7.1.4.2 Compatibility Interfaces 48

7.1.5 Channel Configuration Model 49

7.1.6 Channel Interfaces 53

7.2 Transceiver Cable Compatibility Interface Specifications 53

7.2.1 Transceiver Cable Signals 53
7.2.1.1 Transmit Signal 53
7.2.1.2 Receive Signal 54
7.2.1.3 Collision Presence Signal 54
7.2.1.4 Power 54

7.2.2 Transceiver Cable Parameters 54
7.2.2.1 Mechanical Configuration 54
7.2.2.2 Characteristic Impedance 55
7.2.2.3 Attenuation 55
7.2.2.4 Velocity of Propagation 55
7.2.2.5 Pulse Distortion 55
7.2.2.6 Resistance 55
7.2.2.7 Transfer Impedance 55

7.2.3 Transceiver Cable Connectors 56

7.2.4 Transceiver Cable Drive 56

7.2.5 Transceiver Cable Receive 57
7.2.5.1 Load Impedance and Termination 57
7.2.5.2 Common Mode and CMRR 57

7.3 Coaxial Cable Compatibility Interface Specifications 58

7.3.1 Coaxial Cable Component Specifications 58
7.3.1.1 Coaxial Cable Parameters 58
7.3.1.1.1 Characteristic Impedance 58
7.3.1.1.2 Attenuation 58
7.3.1.1.3 Velocity of Propagation 58
7.3.1.1.4 Mechanical Requirements 59
7.3.1.1.5 Pulse Distortion 59
7.3.1.1.6 Jacket Marking 59
7.3.1.1.7 Transfer Impedance 59
7.3.1.2 Coaxial Cable Connectors 60
7.3.1.3 Coaxial Cable Terminators 60
7.3.1.4 Transceiver-to-Coaxial Cable Connections 61

7.3.2 Coaxial Cable Signaling 61

7.4 Transceiver Specifications 62

7.4.1 Transceiver-to-Coaxial Cable Interface 62
7.4.1.1 Input Impedance 62
7.4.1.2 Bias Current 62
7.4.1.3 Transmit Output Levels 63

7.4.2 Transceiver-to-Transceiver Cable Interface 63
7.4.2.1 Transmit Pair 63
7.4.2.2 Receive Pair 63
7.4.2.3 Collision Presence Pair 64
7.4.2.4 Power Pair 64

7.4.3 Electrical Isolation 64

Amazon
Exhibit 1005
Page 225

7.4.4 Reliability 64
7.5 Channel Logic 64
7.5.1 Channel Encoding 64
7.5.1.1 Encoder 65
7.5.1.2 Decoder 65
7.5.1.3 Preamble Generation 66
7.5.2 Collision Detect Signal 67
7.5.3 Carrier Sense Signal 67
7.5.4 Channel Framing 68
7.5.4.1 Beginning-of-Frame Sequence 68
7.5.4.2 End-of-Frame Sequence 69
7.6 Channel Configuration Requirements 69
7.6.1 Cable Sectioning 69
7.6.2 Transceiver Placement 70
7.6.3 System Grounding 70
7.6.4 Repeaters 71
7.6.4.1 Carrier Detect and Transmit Repeat 71
7.6.4.2 Collision Detect and Collision Repeat 71
7.6.4.3 Repeater Signal Generation 71
7.6.4.3.1 Signal Amplification 71
7.6.4.3.2 Signal Timing 71
7.7 Environment Specifications 72
7.7.1 Electromagnetic Environment 72
7.7.2 Temperature and Humidity 72

Appendices

APPENDIX A: GLOSSARY 73
APPENDIX B: ASSIGNMENT OF ADDRESS AND TYPE VALUES 76
APPENDIX C: CRC IMPLEMENTATION 77
APPENDIX D: IMPLEMENTATION OF TRANSCEIVER CABLE DRIVER AND RECEIVER 80
APPENDIX E: INTERFRAME RECOVERY 82

Figures and Tables

Figure 4-1: Ethernet Architecture and Typical Implementation 7
Figure 4-2: Architectural Layering 8
Figure 4-3: Data Link Layer Functions 10
Figure 4-4: Physical Layer Functions 11
Figure 6-1: Data Link Layer Frame Format 20
Figure 6-2: Structure of the Data Link Procedural Model 31
Figure 6-3: Control Flow Summary -- Client Layer Processes 32
Figure 6-4: Control Flow Summary -- Data Link Layer Processes 33
Figure 7-1: Physical Channel Configurations 50
Table 7-1: Physical Channel Propagation Delay Budget 52
Figure 7-2: Maximum Transceiver Cable Transfer Impedance 55
Figure 7-3: Typical Transceiver Cable Waveform 57
Figure 7-4: Maximum Coaxial Cable Transfer Impedance 60
Figure 7-5: Typical Coaxial Cable Waveform 62
Figure 7-6: Manchester Encoding 65
Figure 7-7: Preamble Encoding 66
Figure 7-8: Functional Logic of **collisionDetect** Signal 67
Figure 7-9: Functional Logic of **carrierSense** Signal 68
Figure C-1: CRC Implementation 78
Figure D-1: Typical Transceiver Cable Driver 80
Figure D-2: Typical Transceiver Cable Receiver 81

Amazon
Exhibit 1005
Page 226

1. INTRODUCTION

The Ethernet local area network provides a communication facility for high speed data exchange among computers and other digital devices located within a moderate-sized geographic area. Its primary characteristics include:

Physical Layer:

Data rate: 10 Million bits/sec

Maximum station separation: 2.5 Kilometers

Maximum number of stations: 1024

Medium: Shielded coaxial cable, base-band signalling

Topology: Branching non-rooted tree

Data Link Layer:

Link control procedure: Fully distributed peer protocol, with statistical contention resolution (CSMA/CD)

Message protocol: Variable size frames, "best-effort" delivery

The Ethernet, like other local area networks, falls in a middle ground between long distance, low speed networks which carry data for hundreds or thousands of kilometers, and specialized, very high speed interconnections which are generally limited to tens of meters. The Ethernet is intended primarily for use in such areas as office automation, distributed data processing, terminal access, and other situations requiring economical connection to a local communication medium carrying bursty traffic at high peak data rates. Use in situations demanding resistance to hostile environments, real-time response guarantees, and so on, while not specifically excluded, do not constitute the primary environment for which the Ethernet is designed.

The precursor to the Ethernet specified in this document was the "Experimental Ethernet", designed and implemented by Xerox in 1975, and used continually since that time by thousands of stations. The Ethernet defined here builds on that experience, and on the larger base of the combined experience of Digital, Intel, and Xerox in many forms of networking and computer interconnection.

In specifying the Ethernet, this document provides precise detailed definitions of the lowest two layers of an overall network architecture. It thus defines what is generally

Amazon
Exhibit 1005
Page 227

referred to as a *link-level* facility. It does not specify the higher level protocols needed to provide a complete network architecture. Such higher level protocols would generally include such functions as internetwork communication, error recovery, flow control, security measures (e.g. encryption), and other higher level functions that increase the power of the communication facility and/or tailor it to specific applications. In particular, it should be noted that all error recovery functions have been relegated to higher level protocols, in keeping with the low error rates that characterize local networks.

One of the main objectives of this specification is *compatibility*. As stated in Section 3, it is intended that *every implementation of the Ethernet be able to exchange data with every other implementation*. It should be noted that higher level protocols raise their own issues of compatibility over and above those addressed by the Ethernet and other link-level facilities. This does not eliminate the importance of link-level compatibility, however. While the compatibility provided by the Ethernet does not guarantee solutions to higher level compatibility problems, it does provide a context within which such problems can be addressed, by avoiding low level incompatibilities that would make direct communication impossible.

2. REFERENCES

The following three papers describe the Experimental Ethernet, and are reprinted in: "The Ethernet Local Network: Three Reports," Xerox Palo Alto Research Center Technical Report CSL-80-2. (February, 1980.)

[1] Metcalfe, R. M. and Boggs, D. R., "Ethernet: Distributed Packet Switching for Local Computer Networks," *Communications of the ACM 19* 7 (July 1976).

[2] Crane, R. C. and Taft, E. A. "Practical Considerations in Ethernet Local Network Design," Presented at *Hawaii International Conference on System Sciences* (January, 1980).

[3] Shoch, J. F. and Hupp, J. A. "Measured Performance of an Ethernet Local Network," Presented at *Local Area Communications Network Symposium* Boston (May 1979).

The following references describe the ISO Open Systems Model:

[4] Zimmermann, H., "OSI Reference Model -- The ISO Model of Architecture for Open Systems Interconnection," *IEEE Transactions on Communication COM-28* 4 (April 1980).

[5] International Organization for Standardization (ISO), "Reference Model of Open Systems Interconnection," *Document no. ISO/TC97/SC16 N227* (June 1979).

The following references describe the Pascal language (used in the Data Link Layer procedural model) and its derivative Concurrent Pascal:

[6] Jensen, K. and Wirth, N., *Pascal User Manual and Report, 2nd Edition.* Springer-Verlag (1974).

[7] Brinch Hansen, P., *Concurrent Pascal Report.* Technical Report CIT-IS-TR 17, California Institute of Technology (1975).

The following references discuss the CRC code used for the frame check sequence:

[8] Hammond, J. L., Brown, J. E. and Liu, S. S., "Development of a Transmission Error Model and an Error Control Model," Technical Report RADC-TR-75-138, Rome Air Development Center (1975).

[9] Bittel, R., "On Frame Check Sequence (FCS) Generation and Checking.," ANSI working paper X3-S34-77-43, (1977).

Amazon
Exhibit 1005
Page 228

3. GOALS AND NON-GOALS

This section states the assumptions underlying the design of the Ethernet.

3.1 Goals

The goals of the Ethernet design are:

Simplicity: Features which would complicate the design without substantially contributing to the meeting of the other goals have been excluded.

Low cost: Since technological improvements will continue to reduce the overall cost of stations wishing to connect to the Ethernet, the cost of the connection itself should be minimized.

Compatibility: All implementations of the Ethernet should be capable of exchanging data at the data link level. For this reason, the specification avoids optional features, to eliminate the possibility of incompatible variants of the Ethernet.

Addressing flexibility: The addressing mechanisms should provide the capability to target frames to a single node, a group of nodes, or to all nodes on the network.

Fairness: All nodes should have equal access to the network when averaged over time.

Progress: No single node operating in accordance with the protocol should be able to prevent the progress of other nodes.

High speed: The network should operate efficiently at a data rate of 10 Megabits per second.

Low delay: At any given level of offered traffic, the network should introduce as little delay as possible in the transfer of a frame.

Stability: The network should be stable under all load conditions, in the sense that the delivered traffic should be a monotonically non-decreasing function of the total offered traffic.

Maintainability: The Ethernet design should allow for network maintenance, operation, and planning.

Layered Architecture: The Ethernet design should be specified in layered terms to separate the logical aspects of the data link protocol from the physical details of the communication medium.

3.2 Non-Goals

The following are *not* goals of the Ethernet design:

Full duplex: At any given instant, the Ethernet can transfer data from one source station to one or more destination stations. Bi-directional communication is provided by rapid exhange of frames, rather than full duplex operation.

Error control: Error handling at the data link level is limited to detection of bit errors in the physical channel, and the detection and recovery from collisions. Provision of a complete error control facility to handle detected errors is relegated to higher layers of the network architecture.

Security: The data link protocol does not employ encryption or other mechanisms to provide security. Higher layers of the network architecture may provide such facilities as appropriate.

Speed flexibility: This specification defines a physical channel operating at a single fixed data rate of 10 Megabits per second.

Priority: The data link protocol provides no support of priority station operation.

Hostile user: There is no attempt to protect the network from a malicious user at the data link level.

Amazon
Exhibit 1005
Page 229

4. FUNCTIONAL MODEL OF THE ETHERNET ARCHITECTURE

There are two important ways to view the Ethernet design, corresponding to:

Architecture, emphasizing the logical divisions of the system, and how they fit together.

Implementation, emphasizing the actual components, and their packaging and interconnection.

Figure 4-1 illustrates these two views as they apply to a typical implementation, showing how each view groups the various functions.

This document is organized along *architectural* lines, emphasizing the large-scale separation of the Ethernet system into two parts: the *Data Link Layer* and the *Physical Layer*. These layers are intended to correspond closely to the lowest layers of the ISO Model for Open Systems Interconnection [4,5]. Architectural organization of the specification has two main advantages:

Clarity: A clean overall division of the design along architectural lines makes the specification clearer.

Flexibility: Segregation of medium-dependent aspects in the Physical Layer allows the Data Link Layer to apply to transmission media other than the specified coaxial cable.

As is evident in Figure 4-1, the architectural model is based on a set of interfaces different from those emphasized in the implementations. One crucial aspect of the design, however, must be addressed largely in terms of the implementation interfaces: *compatibility*. Two important compatibility interfaces are defined within what is architecturally the Physical Layer:

Coaxial cable interface: To communicate via the Ethernet, all stations must adhere rigidly to the exact specification of coaxial cable signals defined in this document, and to the procedures which define correct behavior of a station. The medium-independent aspects of the Data Link Layer should not be taken as detracting from this point: *communication via the Ethernet requires complete compatibility at the coaxial cable interface.*

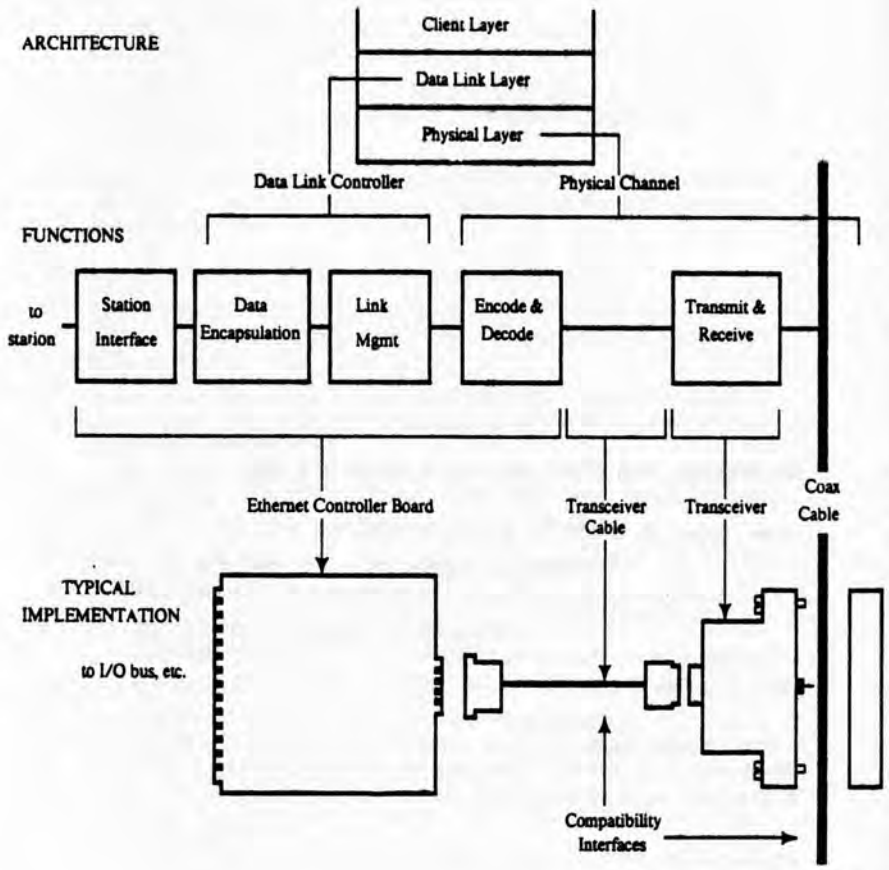


Figure 4-1: Ethernet Architecture and Typical Implementation

Amazon
Exhibit 1005
Page 230

Transceiver cable interface: It is anticipated that most stations will be located some distance away from their connection to the coaxial cable. While it is necessary to place a small amount of circuitry (the *transceiver*) directly adjacent to the coaxial cable, the majority of the electronics (the *controller*) can and should be placed with the station. Since it is desirable for the same transceiver to be usable with a wide variety of stations, a second compatibility interface, the *transceiver cable interface*, is defined. While conformance with this interface is not strictly necessary to insure communication, it is highly recommended, since it allows maximum flexibility in intermixing transceivers and stations.

4.1 Layering

The major division in the Ethernet Architecture is between the Physical Layer and the Data Link Layer, corresponding to the lowest two levels in the ISO model. The higher levels of the overall network architecture, which use the Data Link Layer, will be collectively referred to in this document as the "Client Layer" since, strictly speaking, the identity and function of higher level facilities are outside the scope of this specification. The intent, however, is that the Ethernet Physical and Data Link Layers support the higher layers of the ISO model (Network Layer, Transport Layer, etc.).

The overall structure of the layered architecture is shown in Figure 4-2.

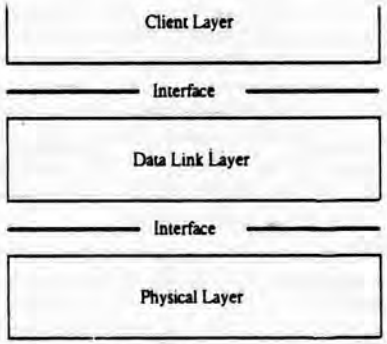


Figure 4-2: Architectural Layering

In the architectural model used here, the layers interact via well defined interfaces.

The interface between the Client Layer and the Data Link Layer includes facilities for transmitting and receiving frames, and provides per-operation status information for use by higher-level error recovery procedures.

The interface between the Data Link Layer and the Physical Layer includes signals for framing (carrier sense, transmit initiation) and contention resolution (collision detect), facilities for passing a pair of serial bit streams (transmit, receive) between the two layers, and a wait function for timing.

These interfaces are described more precisely in Section 5.

As mentioned in the preface, additional interfaces are necessary to allow a higher level network management facility to interact with the Data Link Layer and Physical Layer to perform operation, maintenance and planning functions.

4.2 Data Link Layer

The Data Link Layer defines a medium-independent link level communication facility, built on the medium-dependent physical channel provided by the Physical Layer. It is applicable to a general class of local area broadcast media suitable for use with the channel access discipline known as carrier-sense multiple-access with collision-detection (CSMA-CD). Compatibility with non-contention media (e.g., switched lines, token-passing rings, etc.), while a worthwhile topic for further research, is not addressed in this specification.

The Data Link Layer specified here is intended to be as similar as possible to that described in the ISO model. In a broadcast network like the Ethernet, the notion of a data link between two network entities does not correspond directly to a distinct physical connection. Nevertheless, the two main functions generally associated with a data link control procedure are present:

Data encapsulation

- framing (frame boundary delimitation)
- addressing (handling of source and destination addresses)
- error detection (detection of physical channel transmission errors)

Link management

- channel allocation (collision avoidance)
- contention resolution (collision handling)

This split is reflected in the division of the Data Link Layer into the Data Encapsulation sub-layer and the Link Management sub-layer, as shown in Figure 4-3.

29

Amazon
Exhibit 1005
Page 231

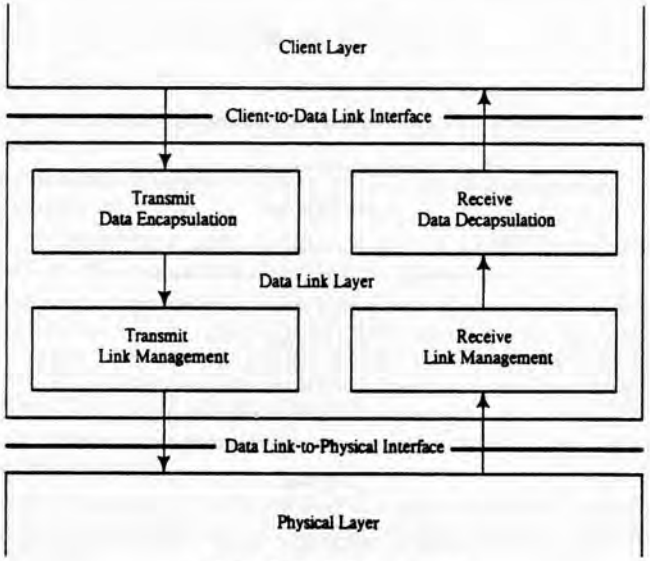


Figure 4-3: Data Link Layer Functions

In terms of the ISO model, the Ethernet Data Link Layer provides a multi-endpoint connection between higher-layer entities wishing to communicate. The connection provided is called a *data link*, and is implemented between two or more Data Link Layer entities called *data link controllers* via a Physical Layer connection called the *physical channel.*

4.3 Physical Layer

The Physical Layer specified in this document provides a 10 MBit/sec physical channel through a coaxial cable medium. Because one purpose of the layered architecture is to insulate the Data Link Layer from the medium-specific aspects of the channel, the Physical Layer completely specifies the essential physical characteristics of the Ethernet, such as data encoding, timing, voltage levels, etc. Implementation details are left unspecified, to retain maximum flexibility for the implementor. In all cases, the criterion applied in distinguishing between essential characteristics and implementation details is *guaranteed compatibility:* any two correct implementations of the Physical Layer specified here will be capable of exchanging data over the coaxial cable, enabling communication between their

respective stations at the Data Link Layer.

The Physical Layer defined in this specification performs two main functions generally associated with physical channel control:

Data encoding

- preamble generation/removal (for synchronization)
- bit encoding/decoding (between binary and phase-encoded form)

Channel access

- bit transmission/reception (of encoded data)
- carrier sense (indicating traffic on the channel)
- collision detection (indicating contention on the channel)

This split is reflected in the division of the Physical Layer into the Data Encoding sub-layer and the Channel Access sub-layer, as shown in Figure 4-4.

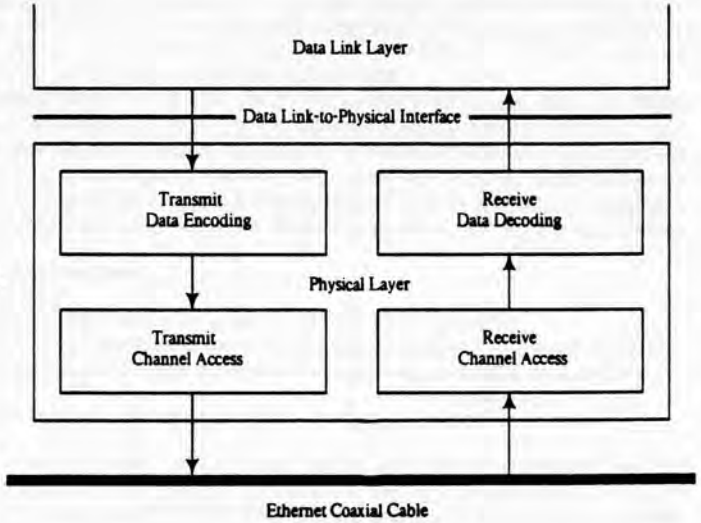


Figure 4-4: Physical Layer Functions

Amazon
Exhibit 1005
Page 232

4.4 Ethernet Operation and the Functional Model

This section provides an overview of frame transmission and reception in terms of the functional model of the architecture. This overview is descriptive, rather than definitional; the formal specifications of the operations described here are given in Sections 6 and 7.

4.4.1 Transmission Without Contention

When the Client Layer requests the transmission of a frame, the Transmit Data Encapsulation component of the Data Link Layer constructs the frame from the client-supplied data and appends a frame check sequence to provide for error detection. The frame is then handed to the Transmit Link Management component for transmission.

Transmit Link Management attempts to avoid contention with other traffic on the channel by monitoring the *carrier sense* signal and *deferring* to passing traffic. When the channel is clear, frame transmission is initiated (after a brief interframe delay to provide recovery time for other data link controllers and for the physical channel). The Data Link Layer then provides a serial stream of bits to the Physical Layer for transmission.

The Data Encoding component of the Physical Layer, before sending the actual bits of the frame, sends an encoded preamble to allow the receivers and repeaters along the channel to synchronize their clocks and other circuitry. It then begins translating the bits of the frame into encoded form and passes them to the Channel Access component for actual transmission over the medium.

The Channel Access component performs the task of actually generating the electrical signals on the medium which represent the bits of the frame. Simultaneously, it monitors the medium and generates the *collison detect* signal, which, in the contention-free case under discussion, remains off for the duration of the frame.

When transmission has completed without contention, the Data Link Layer so informs the Client Layer and awaits the next request for frame transmission.

4.4.2 Reception Without Contention

At the receiving station, the arrival of a frame is first detected by the Receive Channel Access component of the Physical Layer, which responds by synchronizing with the incoming preamble, and by turning on the carrier sense signal. As the encoded bits arrive from the medium, they are passed to the Receive Data Decoding component.

Amazon
Exhibit 1005
Page 233

Receive Data Decoding translates the encoded signal back into binary data and discards the leading bits, up to and including the end of the preamble. It then passes subsequent bits up to the Data Link Layer.

Meanwhile, the Receive Link Management component of the Data Link Layer, having seen carrier sense go on, has been waiting for the incoming bits to be delivered. Receive Link Management collects bits from the Physical Layer as long as the carrier sense signal remains on. When the carrier sense signal goes off, the frame is passed to Receive Data Decapsulation for processing.

Receive Data Decapsulation checks the frame's destination address field to decide whether the frame should be received by this station. If so, it passes the contents of the frame to the Client Layer along with an appropriate status code. The status code is generated by inspecting the frame check sequence to detect any damage to the frame enroute, and by checking for proper octet-boundary alignment of the end of the frame.

4.4.3 Collisions: Handling of Contention

If multiple stations attempt to transmit at the same time, it is possible for their transmitting data link controllers to interfere with each others' transmissions, in spite of their attempts to avoid this by deferring. When two stations' transmissions overlap, the resulting contention is called a *collision*. A given station can experience a collision during the initial part of its transmission (the "collision window"), before its transmitted signal has had time to propagate to all parts of the Ethernet channel. Once the collision window has passed, the station is said to have *acquired* the channel; subsequent collisions are avoided, since all other (properly functioning) stations can be assumed to have noticed the signal (via carrier sense) and to be deferring to it. The time to acquire the channel is thus based on the *round-trip propagation time* of the physical channel.

In the event of a collision, the Transmit Channel Access component of a transmitting station's Physical Layer first notices the interference on the channel and turns on the collision detect signal. This is noticed in turn by the Transmit Link Management component of the Data Link Layer, and collision handling begins. First, Transmit Link Management *enforces* the collision by transmitting a bit sequence called the *jam*. This insures that the duration of the collision is sufficient to be noticed by the other transmitting station(s) involved in the collision. After the jam is sent, Transmit Link Management terminates the transmission and schedules a retransmission attempt for a randomly selected time in the near future. Retransmission is attempted repeatedly in the face of repeated collisions. Since repeated collisions indicate a busy channel, however, Transmit Link Management attempts to adjust to the channel load by *backing off* (voluntarily delaying its own retransmissions to reduce its load on the channel). This is accomplished by

expanding the interval from which the random retransmission time is selected on each retransmission attempt. Eventually, either the transmission succeeds, or the attempt is abandoned on the assumption that the channel has failed or has become overloaded.

At the receiving end, the bits resulting from a collision are received and decoded by the Physical Layer just as are the bits of a valid frame. In particular, collisions do *not* turn on the receiving station's collision detect signal, which is generated only during transmission. Instead, the fragmentary frames received during collisions are distinguished from valid frames by the Data Link's Receive Link Management component, by noting that a collision fragment is always smaller than the shortest valid frame. Such fragments are discarded by Receive Link Management.

Amazon
Exhibit 1005
Page 234

5. INTER-LAYER INTERFACES

The purpose of this section is to provide precise definitions of the interfaces between the architectural layers defined in Section 4. In order to provide such a definition, some precise notation must be adopted. The notation used here is the Pascal language, in keeping with the procedural nature of the formal Data Link Layer specification (see 6.5). Each interface is thus described as a set of procedures and/or shared variables which collectively provide the only valid interactions between layers. The accompanying text describes the meaning of each procedure or variable and points out any implicit interactions among them.

Note that the description of the interfaces in Pascal is a notational technique, and in no way implies that they can or should be implemented in software. This point is discussed more fully in 6.5, which provides complete Pascal declarations for the data types used in the remainder of this section. Note also that the "synchronous" (one frame at a time) nature of the frame transmission and reception operations is a property of the architectural interface between the Client Layer and the Data Link Layer, and need not be reflected in the implementation interface between a station and its controller.

5.1 Client Layer to Data Link Layer

The two primary services provided to the Client Layer by the Data Link Layer are transmission and reception of frames. The interface through which the Client Layer uses the facilities of the Data Link Layer therefore consists of a pair of functions.

Functions:
TransmitFrame
ReceiveFrame

Each of these functions has the components of a frame as its parameters (input or output), and returns a status code as its result.

The Client Layer transmits a frame by invoking *TransmitFrame*:

```
function TransmitFrame (
  destinationParam: AddressValue;
  sourceParam: AddressValue;
  typeParam: TypeValue;
  dataParam: DataValue): TransmitStatus;
```

The *TransmitFrame* operation is synchronous, in the sense that its duration is the entire attempt to transmit the frame, so that when the operation completes, transmission has either succeeded or failed, as indicated by the resulting status code:

```
type TransmitStatus = (transmitOK, excessiveCollisionError);
```

Successful transmission is indicated by the status code *transmitOK*; the code *excessiveCollisionError* indicates that the transmission attempt was aborted due to excessive collisions, because of heavy traffic or a network failure. Implementations may define additional implementation-dependent status codes if necessary.

The Client Layer accepts incoming frames by invoking *ReceiveFrame*:

```
function ReceiveFrame (
  var destinationParam: AddressValue;
  var sourceParam: AddressValue;
  var typeParam: TypeValue;
  var dataParam: DataValue): ReceiveStatus;
```

The *ReceiveFrame* operation is synchronous, in the sense that the operation does not complete until a frame has been received. The fields of the frame are delivered via the output parameters, along with a status code:

```
type ReceiveStatus = (receiveOK, frameCheckError, alignmentError);
```

Successful reception is indicated by the status code *receiveOK*. The code *frameCheckError* indicates that the frame received was damaged by a transmission error in the physical channel. The code *alignmentError* indicates that the frame received was damaged, and that in addition, its length was not an integral number of octets. Implementations may define additional implementation-dependent status codes if necessary.

33

5.2 Data Link Layer to Physical Layer

The interface through which the Data Link Layer uses the facilities of the Physical Layer consists of a function, a pair of procedures and three Boolean variables.

Function:	Variables:
ReceiveBit	*collisionDetect*
Procedures:	*carrierSense*
TransmitBit	*transmitting*
Wait	

During transmission, the contents of an outgoing frame are passed from the Data Link Layer to the Physical Layer via repeated use of the *TransmitBit* operation:

```
procedure TransmitBit (bitParam: Bit);
```

Each invocation of *TransmitBit* passes one new bit of the outgoing frame to the Physical Layer. The *TransmitBit* operation is synchronous, in the sense that the duration of the operation is the entire transmission of the bit, so that when the operation completes, the Physical Layer is ready to accept the next bit immediately. (Note: this does not imply that all invocations of *TransmitBit* are of exactly equal duration; for example, if the Physical Layer must perform some initial processing -- e.g., preamble generation -- before transmitting the first bit of a frame, the first

Amazon
Exhibit 1005
Page 235

invocation of *TransmitBit* may take significantly longer.)

The overall event of data being transmitted is signaled to the Physical Layer via the variable *transmitting*:

```
var transmitting: Boolean;
```

Before sending the first bit of a frame, the Data Link Layer sets *transmitting* to *true*, to inform the Physical Link that a stream of bits will be presented via the *TransmitBit* operation. After the last bit of the frame has been presented, the Data Link Layer sets *transmitting* to *false* to indicate the end of the frame.

The presence of a collision in the physical channel is signaled to the Data Link Layer via the variable *collisionDetect*:

```
var collisionDetect: Boolean;
```

The *collisionDetect* signal remains *true* during the duration of the collision. (Note: Since an entire collision may occur during the first invocation of *TransmitBit* -- e.g., during preamble removal -- the Data Link Layer must handle this possibility by monitoring *collisionDetect* concurrently with its transmission of outgoing bits. See 6.5 for details.)

The *collisionDetect* signal is generated only during transmission and is never *true* at any other time; in particular, it cannot be used during frame reception to detect collisions between overlapping transmissions from two or more other stations.

During reception, the contents of an incoming frame are retrieved from the Physical Layer by the Data Link Layer via repeated use of the *ReceiveBit* operation:

```
function ReceiveBit: Bit;
```

Each invocation of *ReceiveBit* retrieves one new bit of the incoming frame (i.e., not including any preamble bits) from the Physical Layer. The *ReceiveBit* operation is synchronous, in the sense that its duration is the entire reception of a single bit. (As with *TransmitBit*, the first invocation of *ReceiveBit* make take significantly longer -- e.g., due to preamble removal). Upon receiving a bit, the Data Link Layer must immediately request the next bit until all bits of the frame have have been received. (See 6.5 for details.)

The overall event of data being received is signaled to the Data Link Layer via the variable *carrierSense*:

```
var carrierSense: Boolean;
```

When the Physical Layer sets *carrierSense* to *true*, the Data Link Layer must immediately begin retrieving the incoming bits via the *ReceiveBit* operation. When *carrierSense* subsequently becomes *false*, the Data Link Layer can begin processing the received bits as a completed frame. Note that the *true/false*

transitions of *carrierSense* are not defined to be precisely synchronized with the beginning and end of the frame, but may precede the beginning and lag the end, respectively. If an invocation of *ReceiveBit* is pending when *carrierSense* becomes *false*, *ReceiveBit* returns an undefined value, which should be discarded by the Data Link Layer. (See 6.5 for details.)

The Data Link Layer must also monitor the value of *carrierSense* to defer its own transmissions when the channel is busy.

The Physical Layer also provides the procedure *Wait*:

```
procedure Wait (bitTimes: integer);
```

This procedure waits for the specified number of bit times. This allows the Data Link Layer to measure time intervals in units of the (physical-channel-dependent) bit time.

Another important property of the Physical Layer which is an implicit part of the interface presented to the Data Link Layer is the *round-trip propagation time* of the physical channel. This figure represents the maximum time required for a signal to propagate from one end of the network to the other, and for a collision to propagate back. The round-trip propagation time is primarily (but not entirely) a function of the physical size of the network. The round-trip propagation time of the Physical Layer is defined to be at most 450 bit times (see 7.1.2).

34

6. ETHERNET DATA LINK LAYER SPECIFICATION

6.1 Data Link Layer Overview and Model

As defined in Section 4, the Ethernet Architecture consists of the Data Link Layer, and below it, the Physical Layer. Furthermore, the Data Link Layer is divided into two sub-layers (see Figure 4-3).

Data encapsulation
- framing
- addressing
- error detection

Link management
- channel allocation
- contention resolution

This model is used throughout this section to structure the detailed specification of the Data Link Layer. An English description of the Data Link Layer is given in 6.2, 6.3, and 6.4. A more precise algorithmic definition is given in 6.5, which provides a procedural model for the Data Link Layer in the form of a program in the language Pascal. Note that whenever there is any apparent ambiguity concerning the definition of some aspect of the Data Link Layer, it is the Pascal procedural specification in 6.5 which should be consulted for the definitive statement.

6.2 Frame Format

The data encapsulation function of the Data Link Layer comprises the construction and processing of frames. The subfunctions of framing, addressing, and error detection are reflected in the frame format as follows:

Framing: No explicit framing information is needed, since the necessary framing cues (*carrierSense* and *transmitting*) are present in the interface to the Physical Layer.

Addressing: Two address fields are provided to identify the source and destination stations for the frame.

Error detection: A Frame Check Sequence field is provided for detection of transmission errors.

Figure 6-1 shows the five fields of a frame: the addresses of the frame's source and destination, a type field for use by higher layers (see 6.2.2), a data field containing the transmitted data, and the frame check sequence field containing a cyclic redundancy check value to detect transmission errors. Of these five fields, all are of fixed size except the data field, which may contain any integral number of octets between the minimum and maximum values specified below (see 6.2.5).

Amazon
Exhibit 1005
Page 236

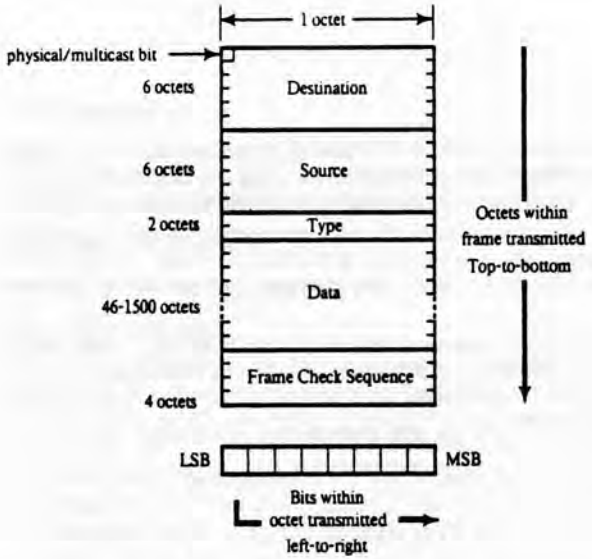


Figure 6-1: Data Link Layer Frame Format

Relative to Figure 6-1, the octets of a frame are transmitted from top to bottom, and the bits of each octet are transmitted from left to right.

NOTE

This document does *not* define an order of transmission for the octets of standard multi-octet data types (strings, integers, etc), since no values of such data types appear in the data link frame format. The order in which implementations of the Ethernet store the octets of a frame in computer memory, and the manner in which higher level protocols interpret the contents of the data field as values of various multi-octet data types, are beyond the scope of this specification.

The Ethernet itself is also totally insensitive to the interpretation of bits within a octet as constituting the digits of an 8-digit binary numeric value. Since some uniform convention is helpful, however, in avoiding needless incompatibility among different station types, the interpretation is arbitrarily defined to be that the left-most bit (first transmitted) is the low-order (2^0) digit and the right-most bit (last transmitted) is the high-order (2^7) digit.

6.2.1 Address Fields

Data link addresses are 6 octets (48 bits) in length. A data link address is of one of two types:

Physical address: The unique address associated with a particular station on the Ethernet. A station's physical address should be distinct from the physical address of any other station on *any* Ethernet.

Multicast address: A multi-destination address, associated with one or more stations on a given Ethernet. There are two kinds of multicast address:

- *Multicast-group address:* An address associated by higher-level convention with a group of logically related stations.
- *Broadcast address:* A distinguished, predefined multicast address which always denotes the set of *all* stations on a given Ethernet.

The first bit of a data link address distinguishes physical from multicast addresses:

0 ➡ physical address

1 ➡ multicast address

In either case, the remainder of the first octet and all of the subsequent octets form a 47-bit pattern. In the case of the broadcast address, this pattern consists of 47 one-bits. There is no standard "null" address value.

The procedures for assigning suitably unique values for physical and multicast addresses are discussed in Appendix B.

6.2.1.1 Destination Address Field

The destination address field specifies the station(s) for which the frame is intended. It may be a physical or multicast (including broadcast) address. For details of address recognition by the receiving station(s), see 6.4.1.2.

6.2.1.2 Source Address Field

The source address field specifies the station sending the frame. The source address field is not interpreted at the Data Link Layer. It is specified at the data link level because a uniform convention for the placement of this field is crucial for most higher level protocols.

6.2.2 Type Field

The type field consists of a two-octet value reserved for use by higher levels (in particular, to identify the Client Layer protocol associated with the frame). The type field is uninterpreted at the Data Link Layer. It is specified at this level because a uniform convention for the placement and value assignment of this field is crucial if multiple higher level protocols are to be able to share the same Ethernet network without conflict. Appendix B discusses the assignment of type field values.

35

Amazon
Exhibit 1005
Page 237

6.2.3 Data Field

The data field contains a sequence of n octets, where $46 \leq n \leq 1500$ Within this range, full data transparency is provided, in the sense that any arbitrary sequence of octet values may appear in the data field.

6.2.4 Frame Check Sequence Field

The frame check sequence (FCS) field contains a 4-octet (32-bit) cyclic redundancy check (CRC) value. This value is computed as a function of the contents of the source, destination, type and data fields (i.e., all fields except the frame check sequence field itself). The encoding is defined by the generating polynomial:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

(This polynomial is also used in the Autodin-II network; its properties are investigated in [8].)

Mathematically, the CRC value corresponding to a given frame is defined by the following procedure:

1. The first 32 bits of the frame are complemented.
2. The n bits of the frame are then considered to be the coefficients of a polynomial M(x) of degree n-1. (The first bit of the destination address field corresponds to the x^{n-1} term and the last bit of the data field corresponds to the x^0 term.)
3. M(x) is multiplied by x^{32} and divided by G(x), producing a remainder R(x) of degree ≤ 31.
4. The coefficients of R(x) are considered to be a 32-bit sequence.
5. The bit sequence is complemented and the result is the CRC.

The 32 bits of the CRC value are placed in the frame check sequence field so that the x^{31} term is the leftmost bit of the first octet, and the x^0 term is the rightmost bit of the last octet. (The bits of the CRC are thus transmitted in the order x^{31}, x^{30},...,x^1, x^0)

Appendix C discusses CRC implementation issues.

6.2.5 Frame Size Limitations

Given the limitations on the size of the data field specified in 6.2.3 and the 18 octet total size for the other four fields, the smallest valid frame contains 64 octets and the largest valid frame contains 1518 octets.

Amazon
Exhibit 1005
Page 238

6.3 Frame Transmission

The Data Link frame transmission and reception are as follows:

Frame transmission includes data encapsulation and link management aspects:

Transmit Data Encapsulation includes the assembly of the outgoing frame (from the values provided by the Client Layer) and frame check sequence generation.

Transmit Link Management includes carrier deference, interframe spacing, collision detection and enforcement, and collision backoff and retransmission.

The performance of these functions by a transmitting data link controller interacts with corresponding actions by other data link controllers to jointly implement the Ethernet data link protocol.

6.3.1 Transmit Data Encapsulation

6.3.1.1 Frame Assembly

The fields of the data link frame are set to the values provided by the Client Layer as arguments to the *TransmitFrame* operation (see 5.1), with the exception of the frame check sequence, which is set to the CRC value generated by the data link controller.

6.3.1.2 Frame Check Sequence Generation

The CRC value defined in 6.2.4 is generated and inserted in the frame check sequence field, following the fields supplied by the Client Layer. Appendix C discusses CRC implementation.

6.3.2 Transmit Link Management

6.3.2.1 Carrier Deference

Even when it has nothing to transmit, the data link controller monitors the physical channel for traffic by watching the *carrierSense* signal provided by the Physical Layer. Whenever the channel is busy, the data link controller *defers* to the passing frame by delaying any pending transmission of its own. After the last bit of the passing frame (i.e., when *carrierSense* changes from true to false), the data link controller continues to defer for 9.6 μsec to provide proper interframe spacing (see 6.3.2.2). At the end of that time, if it has a frame waiting to be transmitted, transmission is initiated independent of the value of *carrierSense*. When transmission has completed (or immediately, if there was nothing to transmit) the data link controller resumes its original monitoring of *carrierSense*.

When a frame is submitted by the Client Layer for transmission, the transmission is initiated as soon as possible, but in conformance with the rules of deference stated above.

6.3.2.2 Interframe Spacing

As defined in 6.3.2.1, the rules for deferring to passing frames insure a minimum interframe spacing of 9.6 μsec. This is intended to provide interframe recovery time for other data link controllers and for the physical channel.

Note that 9.6 μsec is the minimum value of the interframe spacing. If necessary for implementation reasons, a transmitting controller may use a larger value with a resulting decrease in its throughput. The value should not exceed 10.6 μsec.

6.3.2.3 Collision Handling

Once a data link controller has finished deferring and has started transmission, it is still possible for it to experience contention for the channel. As discussed in 4.4.3, collisions can occur until acquisition of the network has been accomplished through the deference of all other stations' data link controllers.

The dynamics of collision handling are largely determined by a single parameter called the *slot time*. This single parameter describes three important aspects of collision handling:

- It is an upper bound on the acquisition time of the network.
- It is an upper bound on the length of a frame fragment generated by a collision. (See 6.4.2.1)
- It is the scheduling quantum for retransmission. (See 6.3.2.3.2)

In order to fulfill all three functions, the slot time must be larger than the sum of the Physical Layer round-trip propagation time (450 bit times; see 7.1.2) and the Data Link Layer maximum jam time (48 bit times, see 6.3.2.3.1). The slot time is defined to be 512 bit times.

6.3.2.3.1 Collision Detection and Enforcement

Collisions are detected by monitoring the ***collisionDetect*** signal provided by the Physical Layer. When a collision is detected during a frame transmission, the transmission is not terminated immediately. Instead, the transmission continues until at least 32 (but not more than 48) additional bits have been transmitted (counting from the time ***collisionDetect*** went on). This collision enforcement or "jam" guarantees that the duration of the collision is sufficient to insure its detection by all transmitting stations on the network. The content of the jam is unspecified; it may be any fixed or variable pattern convenient to the data link controller implementation, but should not be the 32-bit CRC value corresponding to the (partial) frame transmitted prior to the jam.

Amazon
Exhibit 1005
Page 239

6.3.2.3.2 Collision Backoff and Retransmission

When a transmission attempt has terminated due to a collision, it is retried by the transmitting data link controller until either it is successful, or 16 attempts (the original attempt plus 15 retries) have been made and all have terminated due to collisions. Note that all attempts to transmit a given frame are completed before any subsequent outgoing frames are transmitted. The scheduling of the retransmissions is determined by a controlled randomization process called "truncated binary exponential backoff". At the end of enforcing a collision (jamming), the data link controller delays before attempting to retransmit the frame. The delay is an integral multiple of the slot time. (See 6.3.2.3). The number of slot times to delay before the n^{th} retransmission attempt is chosen as a uniformly distributed random integer r in the range $0 \leq r < 2^k$ where $k = \min(n, 10)$. If all 16 attempts fail, this event is reported as an error.

Note that the values given above define the most aggressive behavior that a station may exhibit in attempting to retransmit after a collision. In the course of implementing the retransmission scheduling procedure, a station may introduce extra delays which will degrade its own throughput, but in no case may a station's retransmission scheduling result in a lower average delay between retransmission attempts than the procedure defined above.

6.4 Frame Reception

Frame reception includes both data decapsulation and link management aspects:

Receive Data Decapsulation comprises framing, address recognition, frame check sequence validation, and frame disassembly to pass the fields of the received frame to the Client Layer.

Receive Link Management's main function is the filtering of collision fragments from complete incoming frames.

The performance of these functions by a receiving data link controller interacts with corresponding actions by other data link controllers to jointly implement the Ethernet data link protocol.

6.4.1 Receive Data Decapsulation

6.4.1.1 Framing

The data link controller recognizes the boundaries of an incoming frame by monitoring the ***carrierSense*** signal provided by the Physical Layer. There are two possible length errors that can occur, which indicate ill-framed data: the frame may be too long, or its length may not be an integral number of octets.

6.4.1.1.1 Maximum Frame Size

The receiving data link controller is not required to enforce the frame size limit specified in 6.2.5, but it is allowed to truncate frames longer than 1518 octets and report this event as an (implementation-dependent) error.

6.4.1.1.2 Integral Number of Octets in Frame

Since the format of a valid frame specifies an integral number of octets, only a collision or an error can produce a frame with a length that is not an integral multiple of 8. Complete frames (i.e., not rejected as collision fragments; see 6.4.2.1) that do not contain an integral number of octets are truncated to the nearest octet boundary. If frame check sequence validation (see 6.4.1.3) detects an error in such a frame, the status code *alignmentError* is reported.

6.4.1.2 Address Recognition

The Ethernet data link controller is capable of recognizing physical and multicast addresses, as defined in 6.2.1.

6.4.1.2.1 Physical Addresses

The data link controller recognizes and accepts any frame whose destination field contains the physical address of the station.

The physical address of each station is set by network management to a unique value associated with the station, and distinct from the address of any other station on any Ethernet. The setting of the station's physical address by network management allows multiple data link controllers connected to single station all to respond to the same physical address. The procedures for allocating unique addresses are discussed in Appendix B.

6.4.1.2.2 Multicast Addresses

The data link controller recognizes and accepts any frame whose destination field contains the broadcast address.

The data link controller is capable of activating some number of multicast-group addresses as specified by higher layers. The data link controller recognizes and accepts any frame whose destination field contains an active multicast-group address. An active multicast-group address may be deactivated.

6.4.1.3 Frame Check Sequence Validation

FCS validation is essentially identical to FCS generation. If the bits of the incoming frame (exclusive of the FCS field itself) do not generate a CRC value identical to the one received, an error has occurred and is reported as such. Implementation issues are discussed in Appendix C.

6.4.1.4 Frame Disassembly

The frame is disassembled and the fields are passed to the Client Layer via the output parameters of the *ReceiveFrame* operation (see 5.1).

6.4.2 Receive Link Management

6.4.2.1 Collision Filtering

As specified in 6.2.5, the smallest valid frame must contain at least 64 octets. Any frame containing less than 64 octets is presumed to be a fragment resulting from a collision and is discarded by the receiving data link controller. Since occasional collisions are a normal part of the link management procedure, the discarding of such a fragment is not reported as an error to the Client Layer.

6.5 The Data Link Layer Procedural Model

6.5.1 Overview of the Procedural Model

The functions of the Ethernet Data Link Layer are presented below, modeled as a program written in the language Pascal [6]. This procedural model is intended as the primary specification of the functions to be provided in any Ethernet Data Link Layer implementation. It is important to distinguish, however, between the model and a real implementation. The model is optimized for simplicity and clarity of presentation, while any realistic implementation must place heavier emphasis on such constraints as efficiency and suitability to a particular implementation technology or computer architecture. In this context, several important properties of the procedural model must be considered.

6.5.1.1 Ground Rules for the Procedural Model

a) First, it must be emphasized that the description of the Data Link Layer in a programming language is in no way intended to imply that a data link controller must be implemented as a program executed by a computer. The implementation may consist of any appropriate technology including hardware, firmware, software, or any combination.

b) Similarly, it must be emphasized that it is the *behavior* of Data Link Layer implementations that must match the specification, *not* their internal structure. The internal details of the procedural model are useful only to the extent that they help specify that behavior clearly and precisely.

38

Amazon
Exhibit 1005
Page 240

c) The handling of incoming and outgoing frames is rather stylized in the procedural model, in the sense that frames are handled as single entities by most of the Data Link Layer and are only serialized for presentation to the Physical Layer. In reality, many data link controller implementations will instead handle frames serially on a bit, octet or word basis. A serial implementation would typically perform the required functions (address recognition, frame check sequence generation/validation, etc.) in an overlapped, pipelined fashion. This approach has not been reflected in the procedural model, since this would only complicate the description of the functions without changing them in any way.

d) The model consists of algorithms designed to be executed by a number of concurrent processes; these algorithms collectively implement the Ethernet data link control procedure. The timing dependencies introduced by the need for concurrent activity are resolved in two ways:

- *Processes vs. External events:* It is assumed that the algorithms are executed "very fast" relative to external events, in the sense that a process never falls behind in its work and fails to respond to an external event in a timely manner. For example, when a frame is to be received, it is assumed that the data link procedure *ReceiveFrame* is always called well before the frame in question has started to arrive.

- *Processes vs. Processes:* Among processes, no assumptions are made about relative speeds of execution. This means that each interaction between two processes must be structured to work correctly independent of their respective speeds. Note, however, that the timing of interactions among processes is often, in part, an indirect reflection of the timing of external events, in which case appropriate timing assumptions may still be made.

It is intended that the concurrency in the model reflect the parallelism intrinsic to the task of implementing the Ethernet data link, although the actual parallel structure of the implementations is likely to vary.

6.5.1.2 Use of Pascal in the Procedural Model

Pascal was chosen for the procedural model because of its relative simplicity and clarity, and its general acceptance.

Several observations need to be made about the way in which Pascal is used for the model, including:

a) Some limitations of the language have been circumvented in order to simplify the specification:

1) The elements of the program (variables, procedures, etc) are presented in logical groupings, in top-down order. Certain Pascal ordering restrictions have thus been circumvented to improve readability.

2) The **process** and **cycle** constructs of the Pascal derivative Concurrent Pascal [7] have been introduced to indicate the sites of autonomous concurrent activity. As used here, a process is simply a parameterless procedure that begins execution at "the beginning of time" rather than being invoked by a procedure call. A cycle statement represents the main body of a process and is executed repeatedly forever.

3) The lack of variable array bounds in the language has been circumvented by treating frames as if they are always of a single fixed size (which is never actually specified). In fact, of course, the size of a frame depends on the size of its data field, hence the value of the "pseudo-constant" *frameSize* should be thought of as varying in the long-term, even though it is fixed for any given frame.

4) The use of a variant record to represent a frame (both as fields and as bits) follows the letter but not the spirit of the Pascal Report, since it allows the underlying representation to be viewed as two different data types. (It also assumes that this representation is as shown in Figure 6-1.)

b) The model makes no use of any explicit interprocess synchronization primitives. Instead, all interprocess interaction is done via carefully stylized manipulation of shared variables. For example, some variables are set by only one process and inspected by another process in such a manner that the net result is independent of their execution speeds. While such techniques are not generally suitable for the construction of large concurrent programs, they simplify the model and more nearly resemble the methods appropriate to the most likely implementation technologies (e.g. microcode, hardware state-machines, etc.)

Amazon
Exhibit 1005
Page 241

6.5.2 Procedural Model

The procedural model used here is based on five cooperating concurrent processes. Of these, three are actually defined in the Data Link Layer. The remaining two processes are provided by the Client Layer and utilize the interface operations provided by the Data Link Layer. The five processes are thus:

Client Layer:

Frame Transmitter Process Frame Receiver Process

Data Link Layer:

Bit Transmitter Process Bit Receiver Process

Deference Process

This organization of the model is illustrated in Figure 6-2, and reflects the fact that the communication of entire frames is initiated by the Client Layer, while the timing of collision backoff and of individual bit transfers is based on interactions between the Data Link Layer and the Physical-Layer-dependent bit-time.

Figure 6-2 depicts the static structure of the procedural model, showing how the various processes and procedures interact by invoking each other. Figures 6-3 and 6-4 summarize the dynamic behavior of the model during transmission and reception, focusing on the steps that must be performed, rather than the procedural structure which performs them. The usage of the shared state variables is not depicted in the figures, but is described in the comments in 6.5.2.1.

Amazon
Exhibit 1005
Page 242

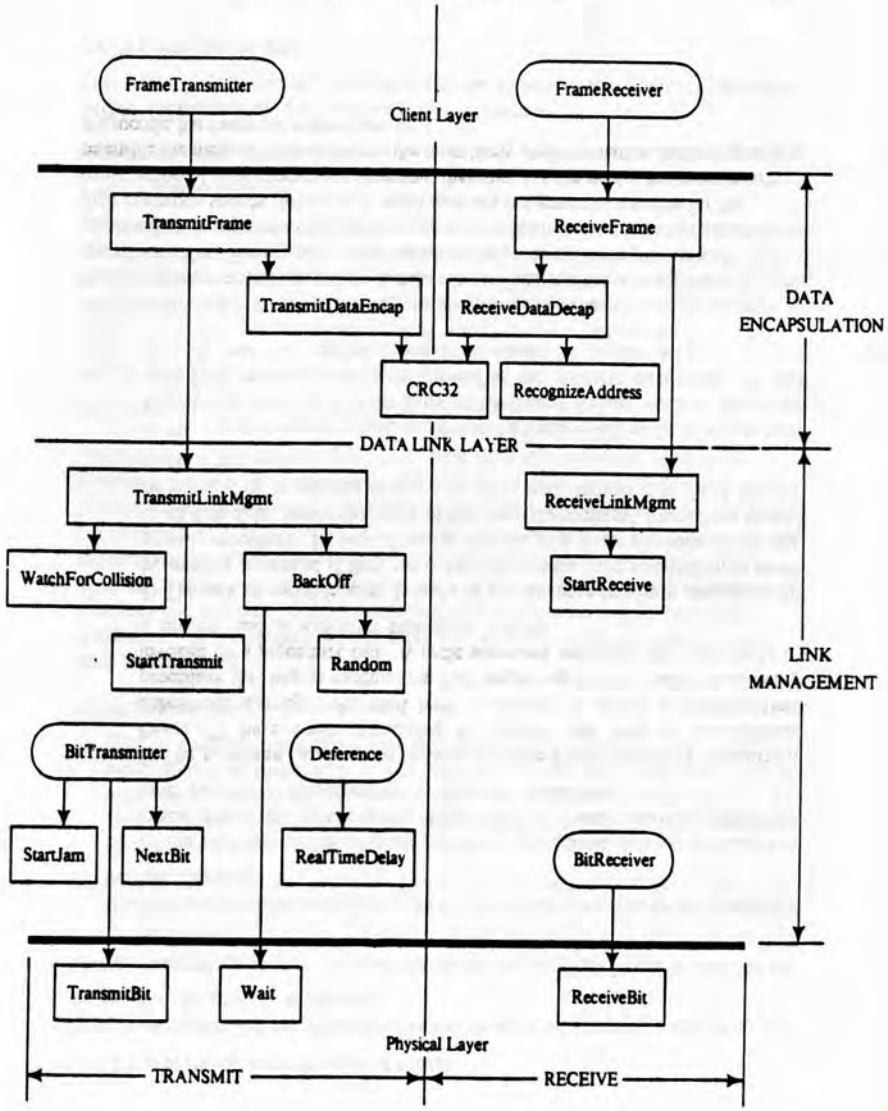


Figure 6-2: Structure of the Data Link Procedural Model

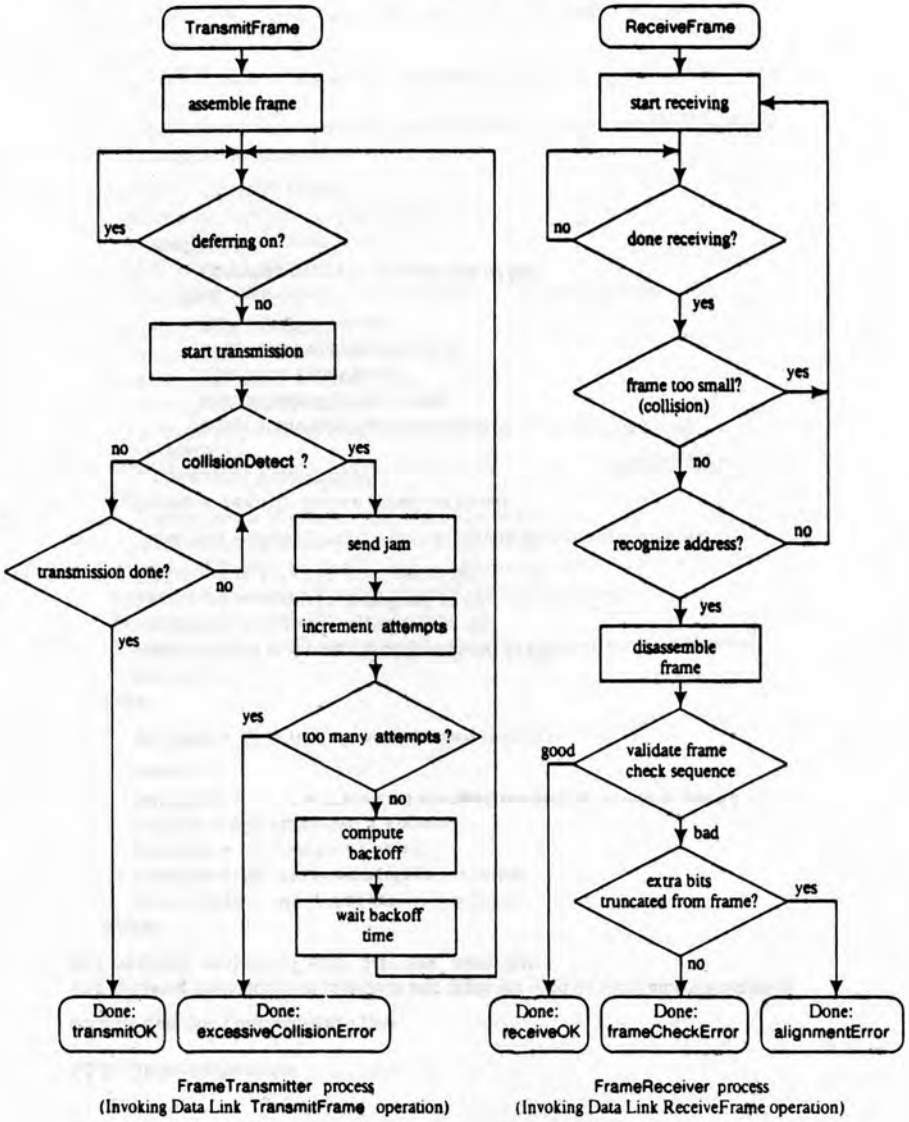


Figure 6-3: Control Flow Summary -- Client Layer Processes

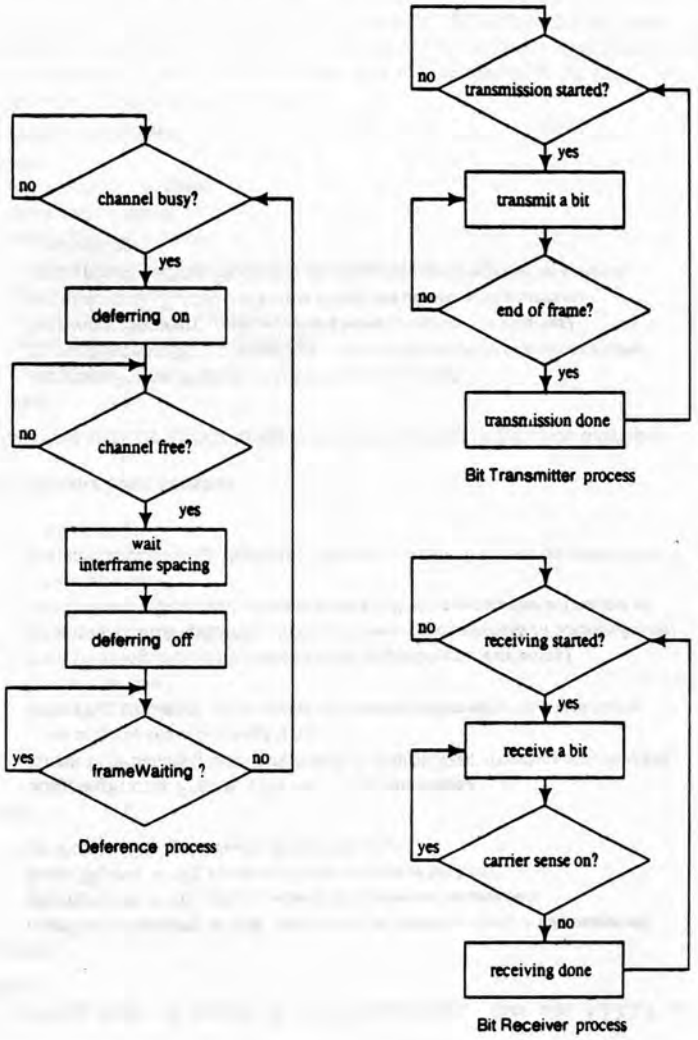


Figure 6-4: Control Flow Summary -- Data Link Layer Processes

Amazon
Exhibit 1005
Page 243

6.5.2.1 Global Declarations

6.5.2.1.1 Common Constants and Types

The following declarations of constants and types are used by the frame transmission and reception sections of each data link controller:

```
const
    addressSize = 48;  {48 bit address = 6 octets}
    typeSize = 16;  {16 bit protocol type = 2 octets}
    dataSize = ...;  {see 6.5.1.2, note 3}
    crcSize = 32;  {32 bit CRC = 4 octets}
    frameSize = ...;  { = 2*addressSize + typeSize + dataSize + crcSize...see 6.5.1.2,
    note 3}
    slotTime = 512;  {unit of time for collision handling}
type
    Bit = 0..1;
    AddressValue = array [1..addressSize] of Bit;
    TypeValue = array [1..typeSize] of Bit;
    DataValue = array [1..dataSize] of Bit;
    CRCValue = array [1..crcSize] of Bit;
    ViewPoint = (fields, bits);  {Two ways to view the contents of a frame}
    Frame = record  {Format of data link frame}
      case view: ViewPoint of
        fields: (
            destinationField: AddressValue;
            sourceField: AddressValue;
            typeField: TypeValue;
            dataField: DataValue;
            fcsField: CRCValue);
        bits: (
            contents: array [1..frameSize] of Bit)
      end; {Frame}
```


6.5.2.1.2 Transmit State Variables

The following items are specific to frame transmission. (See also 6.5.2.1.4 on interfaces)

```
const
    interFrameSpacing = 9.6;  {minimum time between frames, in microseconds}
    attemptLimit = 16;  {Max number of times attempt transmission}
    backOffLimit = 10;  {Limit on number of times to back off}
    jamSize = 32;  {jam may be 32 to 48 bits long}
var
    outgoingFrame: Frame;  {The frame to be transmitted}
    currentTransmitBit, lastTransmitBit: 1..frameSize;  {Positions of current and
      last outgoing bits in outgoingFrame}
    deferring: Boolean;  {True implies any pending transmission must wait for the
      channel to clear}
    frameWaiting: Boolean;  {Indicates that outgoingFrame is deferring}
    attempts: 0..attemptLimit;  {Number of transmission attempts on outgoingFrame}
    newCollision: Boolean;  {Indicates that a collision has occurred but has not yet
      been jammed}
    transmitSucceeding: Boolean;  {Running indicator of whether transmission is
      succeeding}
```

6.5.2.1.3 Receive State Variables

The following items are specific to frame reception. (See also 6.5.2.1.4 on interfaces)

```
var
    incomingFrame: Frame;  {The frame being received}
    currentReceiveBit: 1..frameSize;  {Position of current bit in incomingFrame}
    receiving: Boolean;  {Indicates that a frame reception is in progress}
    excessBits: 0..7;  {Count of excess trailing bits beyond octet boundary}
    receiveSucceeding: Boolean;  {Running indicator of whether reception is
      succeeding}
```

Amazon
Exhibit 1005
Page 244

6.5.2.1.4 Summary of Interlayer Interfaces

The interface to the Client Layer, defined in 5.1, is summarized below:

```
type
  TransmitStatus = (transmitOK, excessiveCollisionError); {Result of
    TransmitFrame operation}
  ReceiveStatus = (receiveOK, frameCheckError, alignmentError); {Result
    of ReceiveFrame operation}

function TransmitFrame (
  destinationParam: AddressValue;
  sourceParam: AddressValue;
  typeParam: typeValue;
  dataParam: DataValue): TransmitStatus; {Transmits one frame}

function ReceiveFrame (
  var destinationParam: AddressValue;
  var sourceParam: AddressValue;
  var typeParam: TypeValue;
  var dataParam: DataValue): ReceiveStatus; {Receives one frame}
```

The interface to the Physical Layer, defined in 5.2, is summarized below:

```
var
    carrierSense: Boolean; {Indicates incoming bits}
    transmitting: Boolean; {Indicates outgoing bits}
    collisionDetect: Boolean; {Indicates channel contention}
procedure TransmitBit (bitParam: Bit); {Transmits one bit}
function ReceiveBit: Bit; {Receives one bit}
procedure Wait (bitTimes: integer); {Waits for indicated number of bit-times}
```


Amazon
Exhibit 1005
Page 245

6.5.2.1.5 State Variable Initialization

The procedure *Initialize* must be run when the Data Link Layer begins operation, before any of the processes begin execution. *Initialize* sets certain crucial shared state variables to their initial values. (All other global variables are appropriately reinitialized before each use.) *Initialize* then waits for the channel to be idle, and starts operation of the various processes.

```
procedure Initialize;
begin
  frameWaiting := false;
  deferring := false;
  newCollision := false;
  transmitting := false; {In interface to Physical Layer; see below}
  receiving := false;
  while carrierSense do nothing;
  {Start execution of all processes}
end; {Initialize}
```

6.5.2.2 Frame Transmission

The algorithms in this section define data link frame transmission.

The function *TransmitFrame* implements the frame transmission operation provided to the Client Layer:

```
function TransmitFrame (
    destinationParam: AddressValue;
    sourceParam: AddressValue;
    typeParam: typeValue;
    dataParam: DataValue): TransmitStatus;
  procedure TransmitDataEncap; ... (nested procedure; see body below)
begin
  TransmitDataEncap;
  TransmitFrame := TransmitLinkMgmt
end; (TransmitFrame)
```

First, *TransmitFrame* calls the internal procedure *TransmitDataEncap* to construct the frame. It then calls *TransmitLinkMgmt* to perform the actual transmission. The *TransmitStatus* returned indicates the success or failure of the transmission attempt.

TransmitDataEncap builds the frame and places the 32-bit CRC in the frame check sequence field:

```
procedure TransmitDataEncap;
begin
  with outgoingFrame do
  begin (assemble frame)
    view := fields;
    destinationField := destinationParam;
    sourceField := sourceParam;
    typeField := typeParam;
    dataField := dataParam;
    fcsField := CRC32(outgoingFrame);
    view := bits
  end (assemble frame)
end; (TransmitDataEncap)
```


Amazon
Exhibit 1005
Page 246

TransmitLinkMgmt attempts to transmit the frame, deferring first to any passing traffic. If a collision occurs, transmission is terminated properly and retransmission is scheduled following a suitable backoff interval:

```
function TransmitLinkMgmt: TransmitStatus;
begin
  attempts := 0; transmitSucceeding := false;
  while attempts < attemptLimit and not transmitSucceeding do
  begin (loop)
    if attempts > 0 then BackOff;
    frameWaiting := true;
    while deferring do nothing;  (defer to passing frame, if any)
    frameWaiting := false;
    StartTransmit;
    while transmitting do WatchForCollision;
    attempts := attempts + 1
  end; (loop)
  if transmitSucceeding then TransmitLinkMgmt := transmitOK
  else TransmitLinkMgmt := excessiveCollisionError
end; (TransmitLinkMgmt)
```

Each time a frame transmission attempt is initiated, *StartTransmit* is called to alert the *BitTransmitter* process that bit transmission should begin:

```
procedure StartTransmit;
begin
  currentTransmitBit := 1;
  lastTransmitBit := frameSize;
  transmitSucceeding := true;
  transmitting := true
end; (StartTransmit)
```

Once frame transmission has been initiated, *TransmitLinkMgmt* monitors the channel for contention by repeatedly calling *WatchForCollision:*

```
procedure WatchForCollision;
begin
  if transmitSucceeding and collisionDetect then
  begin
    newCollision := true;
    transmitSucceeding := false
  end
end; (WatchForCollision)
```

WatchForCollision, upon detecting a collision, updates *newCollision* to insure proper jamming by the *BitTransmitter* process.

After transmission of the jam has completed, if *TransmitLinkMgmt* determines that another attempt should be made, *BackOff* is called to schedule the next attempt to retransmit the frame.

```
var maxBackOff: 2..1024; (Working variable of BackOff)

procedure BackOff;
begin
  if attempts = 1 then maxBackOff := 2 else if attempts ≤ backOffLimit
    then maxBackOff := maxBackOff*2;
  Wait(slotTime*Random(0, maxBackOff))
end; (BackOff)

function Random (low, high: integer): integer;
begin
  Random := ...{uniformly distributed random integer r such that low ≤ r <
  high}
end; (Random)
```

BackOff performs the truncated binary exponential backoff computation and then waits for the selected multiple of the slot time.

The *Deference* process runs asynchronously to continuously compute the proper value for the variable *deferring*.

```
process Deference;
begin
  cycle (main loop)
    while not carrierSense do nothing; (watch for carrier to appear)
    deferring := true; (delay start of new transmissions)
    while carrierSense do nothing; (wait for carrier to disappear)
    RealTimeDelay(interFrameSpacing);
    deferring := false; (allow new transmissions to proceed)
    while frameWaiting do nothing (allow waiting transmission (if any))
  end (main loop)
end; (Deference)

procedure RealTimeDelay (usec: real);
begin
  {Wait for the specified number of microseconds}
end; (RealTimeDelay)
```


The *BitTransmitter* process runs asynchronously, transmitting bits at a rate determined by the Physical Layer's *TransmitBit* operation:

```
process BitTransmitter;
begin
  cycle (outer loop)
    while transmitting do
    begin (inner loop)
      TransmitBit(outgoingFrame[currentTransmitBit]); (send next bit to
      Physical Layer)
      if newCollision then StartJam else NextBit
    end (inner loop)
  end (outer loop)
end; (BitTransmitter)

procedure NextBit;
begin
  currentTransmitBit := currentTransmitBit + 1;
  transmitting := (currentTransmitBit ≤ lastTransmitBit)
end; (NextBit)

procedure StartJam;
begin
  currentTransmitBit := 1;
  lastTransmitBit := jamSize;
  newCollision := false
end; (StartJam)
```

BitTransmitter, upon detecting a new collision, immediately enforces it by calling *StartJam* to initiate the transmission of the jam. The jam may contain 32 to 48 bits of arbitrary data. (*StartJam* uses the first 32 bits of the frame, merely to simplify this program).

Amazon
Exhibit 1005
Page 247

6.5.2.3 Frame Reception

The algorithms in this section define data link frame reception:

The procedure *ReceiveFrame* implements the frame reception operation provided to the Client Layer:

```
function ReceiveFrame (
    var destinationParam: AddressValue;
    var sourceParam: AddressValue;
    var typeParam: TypeValue;
    var dataParam: DataValue): ReceiveStatus;
  function ReceiveDataDecap: ReceiveStatus; ... (nested function; see body
  below)
begin
  repeat
    ReceiveLinkMgmt;
    ReceiveFrame := ReceiveDataDecap;
  until receiveSucceeding
end; (ReceiveFrame)
```

ReceiveFrame calls *ReceiveLinkMgmt* to receive the next valid frame, and then calls the internal procedure *ReceiveDataDecap* to return the frame's fields to the Client Layer if the frame's address indicates that it should do so. The returned *ReceiveStatus* indicates the presence or absence of detected transmission errors in the frame.

```
function ReceiveDataDecap: ReceiveStatus;
begin
  receiveSucceeding := RecognizeAddress
    (incomingFrame.destinationField);
  if receiveSucceeding then with incomingFrame do
  begin (disassemble frame)
    view := fields;
    destinationParam := destinationField;
    sourceParam := sourceField;
    typeParam := typeField;
    dataParam := dataField;
    if fcsField = CRC32(incomingFrame) then ReceiveDataDecap := receiveOK
    else if excessBits = 0 then ReceiveDataDecap := frameCheckError
    else ReceiveDataDecap := alignmentError;
    view := bits
  end (disassemble frame)
end; (ReceiveDataDecap)
```


Amazon
Exhibit 1005
Page 248


```
function RecognizeAddress (address: AddressValue): Boolean;
begin
  RecognizeAddress := ... (Returns true for the set of physical, broadcast, and
    multicast-group addresses corresponding to this station)
end; (RecognizeAddress)
```

ReceiveLinkMgmt attempts repeatedly to receive the bits of a frame, discarding any fragments from collisions by comparing them to the minimum valid frame size:

```
procedure ReceiveLinkMgmt;
begin
  repeat
    StartReceive;
    while receiving do nothing; (wait for frame to finish arriving)
    excessBits := frameSize mod 8;
    frameSize := frameSize - excessBits; (truncate to octet boundary)
    receiveSucceeding := (frameSize ≥ slotTime); (reject collision fragments)
  until receiveSucceeding
end; (ReceiveLinkMgmt)

procedure StartReceive;
begin
  currentReceiveBit := 1;
  receiving := true
end; (StartReceive)
```

The *BitReceiver* process run asynchronously, receiving bits from the channel at the rate determined by the Physical Layer's *ReceiveBit* operation:

```
process BitReceiver;
  var b: Bit;
begin
  cycle (outer loop)
    while receiving do
    begin (inner loop)
      b := ReceiveBit; (Get next bit from physical link)
      if carrierSense then
      begin(append bit to packet)
        incomingFrame[currentReceiveBit] := b;
        currentReceiveBit := currentReceiveBit + 1
      end; (append bit to packet)
      receiving := carrierSense
    end (inner loop)
  end (outer loop)
end; (BitReceiver)
```

6.5.2.4 Common procedures

The function *CRC32* is used by both the transmit and receive algorithms to generate a 32 bit CRC value:

```
function CRC32 (f: Frame): CRCValue;
begin
  CRC32 := {The 32-bit CRC as defined in 6.2.4}
end; {CRC32}
```

Purely to enhance readability, the following procedure is also defined:

```
procedure nothing; begin end;
```

The idle state of a process (i.e., while waiting for some event) is cast as repeated calls on this procedure.

7. ETHERNET PHYSICAL LAYER SPECIFICATION: Baseband Coaxial System

7.1 Physical Channel Overview and Model

The Ethernet physical channel (henceforth referred to as the channel) provides the lowest layer in the Ethernet architecture. It performs all the functions needed to transmit and receive data at the physical level, while supporting the Data Link to Physical Layer Interface described in 5.2.

This section describes the requirements for interface and compatibility with a baseband coaxial implementation of the channel.

7.1.1 Channel Goals and Non-goals

This section states the objectives underlying the design of the channel.

7.1.1.1 Goals

The following are the goals of the channel:

1. Provide a means for communication between Ethernet Data Link Entities.
2. Define physical interfaces which can be implemented compatibly among different manufacturers of hardware.
3. Provide all clocks, synchronization, and timing required for both itself and the Ethernet Data Link.
4. Provide high bandwidth and low bit error rates.
5. Provide for ease of installability and serviceability.
6. Provide for high network availability.
7. Support the Ethernet Data Link to Physical Link interface.
8. Low cost.

7.1.1.2 Non-Goals

The following are not goals of the baseband coaxial channel design:

1. Operation at data rates other than 10 megabits per second.
2. Operation with media other than the specified coaxial cable.
3. Simultaneous use of the channel by transmitters using signals not specified in this document.
4. Protection against a malicious user or a malfunctioning Data Link Entity is not provided by the channel as specified. However, higher layers (above the Data Link) and/or physical security means may be employed to acheive this.

Amazon
Exhibit 1005
Page 249

7.1.2 Characteristics of the Channel

The channel provides (and the data link assumes) the following characteristics:

1. The ability to send and receive information (non-simultaneously) between any two or more data link entities on the same network.
2. The ability to detect the presence of another station's transmission while not transmitting (carrier sense).
3. The ability to detect the presence of another station's transmission while transmitting (collision detect).
4. A total worst-case round trip signal propagation delay (including actual propagation time, synchronization time for all intervening electronics, and signal rise time degradation) of 450 bit times (equal to 45 μs for this 10 Mbit channel).

7.1.3 Functions Provided by the Channel

The channel hardware provides the following functions in the performance of its role:

1. Means for transmitting and receiving serial bit streams between the data link layer and the media.
2. Generation of clock for synchronization and timing.
3. Means for detecting carrier (non-idle channel).
4. Means for detecting collisions (simultaneous transmission attempts by multiple stations).
5. Coding and decoding of the data link bit stream into a self-synchronizable sequence of electrical signals suitable for transmission on the media provided by the channel.
6. Generation and removal of coding-specific preamble information (a synchronizing header sequence inserted before the first bit of the frame) to ensure that all channel electronics are brought to a known steady-state before the data link frame is transmitted.

7.1.4 Implementation of the Channel

The physical channel specification is implementation dependent; most of the channel hardware is fully specified, and little leeway is given to the individual designer. This is done in the interest of compatibility; any system which allows different implementors to use different channel cables, connectors, clock speeds and the like will not be compatible across manufacturer boundaries. Only the design of channel components which are not critical to system compatibility is left to the implementor.

7.1.4.1 General Overview of Channel Hardware

The channel minimally consists of the following functional blocks:

1. The passive broadcast medium (coaxial cable),
2. The transceiver (transmitter-receiver for the coaxial cable),
3. The means for connecting transceivers to a coaxial cable segment and for connecting coaxial cable segments together,
4. The channel clock,
5. The channel data encoder and decoder,
6. The preamble generator and remover,
7. The carrier and collision detect circuits.

The coaxial medium is the only element common to the entire network. A transceiver is required for each station connected to the medium. The transceiver must be located adjacent to the coaxial cable. The latter four components are generally located within, and tightly coupled to, the station hardware implementing the data link function.

It may be useful to be able to physically separate the transceiver from the rest of the channel hardware. This allows topological flexibility, packaging advantages, and improved system availability, as well as allowing for independent manufacture of station hardware and transceivers. To ensure that compatibility is maintained, a physical interface (known as the *transceiver cable*) is identified and specified to connect the transceiver to the station.

Finally, it may be necessary to add *repeaters* to the system, to reach the maximum allowable distance between stations, and to provide additional topological flexibility. Repeaters are implemented using standard transceivers, plus a simple, non-buffered finite state machine.

48

Amazon
Exhibit 1005
Page 250

7.1.4.2 Compatibility Interfaces

There are a number of possibilities for implementing systems or subsystems compatible in whole or in part with this specification. It is important that all implementations be compatible at some point, so that heterogenous systems from different manufacturers' implementations can be interconnected on the same medium. It is not necessary in every case to implement all of the components described herein; e.g., it is possible to design an integrated station/transceiver (without requiring the transceiver cable). The implementor must make the required trade-offs between topological flexibility, system availability, configurability, user needs, and cost when designing the system.

For a device to be considered compatible, it must meet the applicable requirements at either the transceiver cable or the coaxial cable interface, as appropriate, in addition to the Data Link compatibility required for all stations connected to the network.

All Ethernets must be compatible at the coaxial cable.

If a transceiver cable is used, it should be the one specified in this document. This allows device manufacturers to build hardware compatible with the Ethernet at the transceiver cable level, without concerning themselves with the details of transceiver implementation. Devices implementing transceiver cable compatibility should be capable of using transceivers designed and built by another manufacturer, on the specified coaxial cable.

Equipment designed for connection to the specified coaxial cable either without a physically separate transceiver or with a non-standard transceiver cable interface will be capable of communication. However, a sacrifice may have been made with respect to interchangeability with other stations.

This scheme of multiple compatibility interfaces allows individual designers some flexibility in making system tradeoffs, yet allows cable manufacturers, transceiver manufacturers and systems manufacturers to use standard commodity parts to produce a compatible communications system.

7.1.5 Channel Configuration Model

Certain physical limits have been placed on the physical channel. These revolve mostly around maximum cable lengths (or maximum propagation times), as these affect the slot time as defined in the data link. While the precise specification (in later sections) specify these maxima in terms of propagation times, they were derived from the physical configuration model described here.

The maximum configuration is as follows:

1. A coaxial cable, terminated in its characteristic impedance at each end, constitutes a cable *segment*. A segment may contain a maximum of 500 meters of coaxial cable.
2. A maximum of 2 repeaters in the path between any two stations. Repeaters do not have to be located at the ends of segments, nor is the user limited to one repeater per segment. In fact, repeaters can be used not only to extend the length of the channel, but to extend the topology from one to three-dimensional. Repeaters occupy transceiver positions on each cable segment and count towards the maximum number of transceivers on a segment just as do the logically distinguishable stations.
3. A maximum total coaxial cable length along the longest path between any two transceivers of 1500 meters. The propagation velocity of the coaxial cable is assumed to be 0.77 c worst-case. (c is the velocity of light *in vacuo*; 300,000 kilometers per second.) The total round-trip delay for all the coaxial cable in the system is therefore 13 μs worst-case.
4. A maximum of 50 meters of transceiver cable between any station and its associated transceiver. Note that in the worst case the signal must pass through six 50 meter transceiver cables, one at the transmitting station, one at the receiving station, and 2 at each repeater (two repeaters possible). The propagation velocity of the transceiver cable is assumed to be .65 c worst-case. The total round-trip delay for all the transceiver cables is therefore 3.08 μs worst-case.
5. A maximum of 1000 meters of point-to-point link anywhere in the system. This will typically be used as a way of linking cable segments in different buildings. Note that a repeater with this internal point-to-point link can be used to repeat signals between segments many hundreds of meters apart. The worst-case propagation velocity of the link cable is assumed to be .65 c; the round-trip propagation delay for 1000 meters is 10.26 μs.

Table 7-1 summarizes the allocation of the round-trip propagation delay to the individual components in the channel. Figure 7-1 shows a minimum, typical, and large-scale channel configuration.

Amazon
Exhibit 1005
Page 251

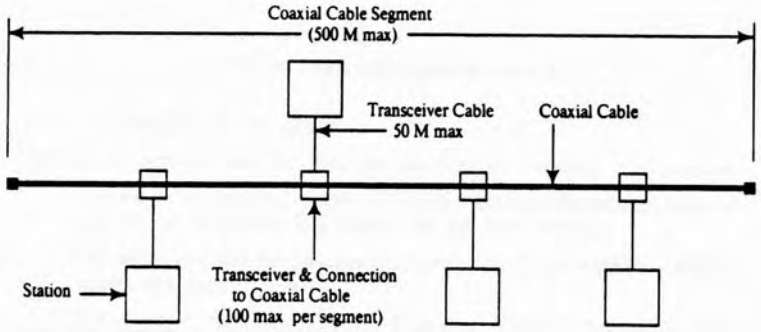


Figure 7-1a: Minimal Configuration

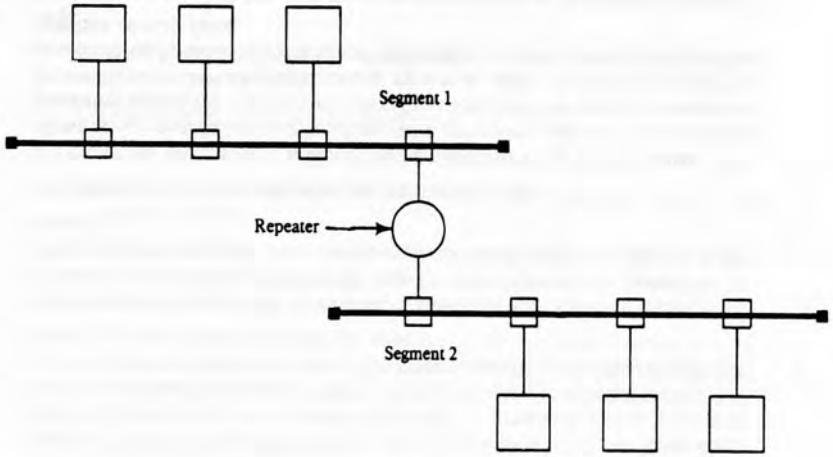


Figure 7-1b: A Typical Medium-scale Configuration

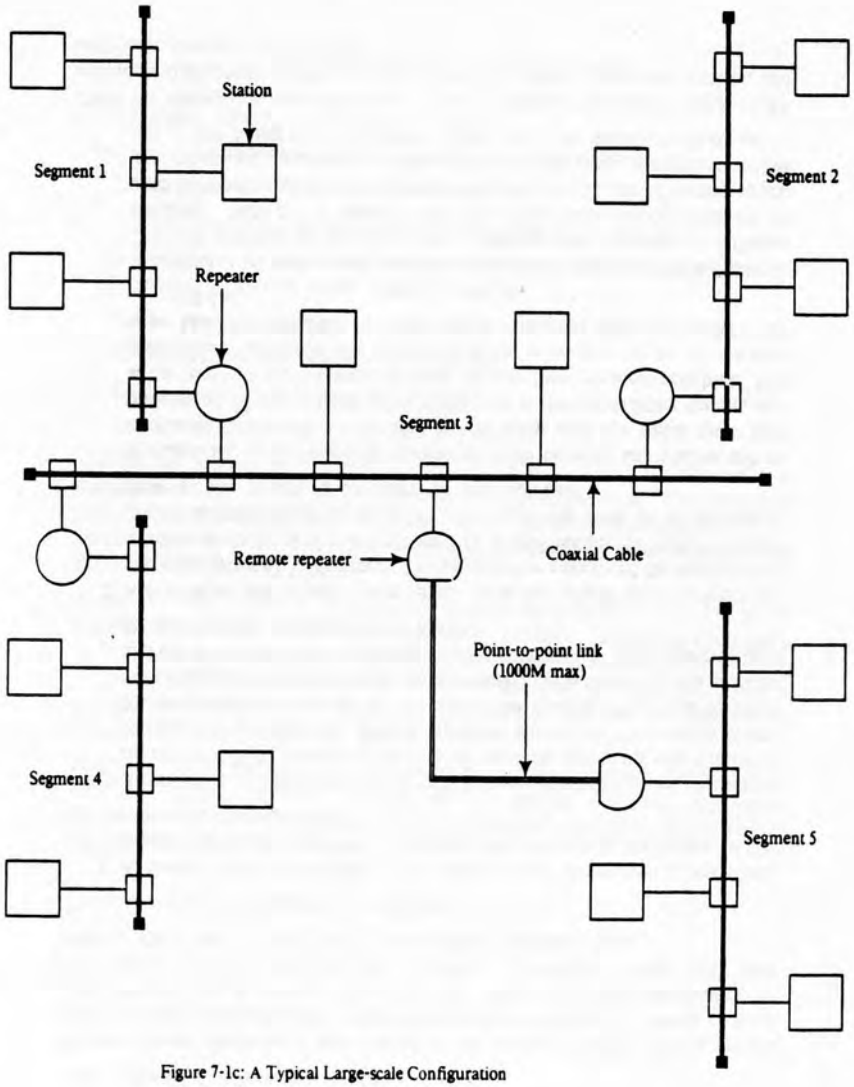


Figure 7-1c: A Typical Large-scale Configuration

Amazon
Exhibit 1005
Page 252

Table 7-1: Physical Channel Propagation Delay Budget (Note 1)

Element	Unit Steady-State Delay	Unit Startup Delay	# Units Forward Path (Note 2)	# Units Return Path	Total Delay
Encoder	0.1 μS	0	3	3	0.60 μS
Transceiver Cable	5.13 nS/M	0	300 M	300 M	3.08 μS
Transceiver (transmit path)	0.50 μS	0.2 μS	3	3	1.50 μS
Transceiver (receive path)	0.50 μS	0.5 μS	3	0	1.65 μS
Transceiver (collision path)	0	0.5 μS	0	3	1.50 μS
Coaxial Cable	4.33 nS/M	0	1500 M	1500 M	13.00 μS
Point-to-Point Link Cable	5.13 nS/M	0	1000 M	1000 M	10.26 μS
Repeater (repeat path)	0.8 μS	0	2	0	1.60 μS
Repeater (collision path)	0.2 μS	0	0	2	0.40 μS
Decoder	0.1 μS	0.8 μS	2	0	1.80 μS
Carrier Sense	0	0.2 μS	3	0	0.60 μS
Collision Detect	0	0.2 μS	0	3	0.60 μS
Signal Rise Time (to 70% in 500 M) (Note 3)	0	0.1 μS	3	0	0.30 μS
Signal Rise Time (50% to 94% in 500 M) (Note 4)	0	2.7 μS	0	3	8.10 μS
			Total Worst-Case Round-Trip Delay		44.99 μS

Note 1: All quantities given are worst-case (both number of units and unit delays per unit).

Note 2: The propagation delay has been separated into "forward-path" and "return path" delay. This is because in one direction it is carrier sense which is being propagated through the channel, and in the return direction it is collision detect which is being propagated. The two signals have different propagation delays.

Note 3: In the worst-case, the propagated signal must reach 70% of its final value to be detected as valid carrier at the end of 500 meters of coaxial cable. This rise time must be included in the propagation delay budget.

Note 4: In the worst-case the propagated collision on the return path must reach 94% of its final value to be detected as a collision at the end of 500 meters of coaxial cable.

7.1.6 Channel Interfaces

The channel specification hinges around three well-defined entities; the transceiver and coaxial cables (shown as compatibility interfaces in Figure 4-1), and the logical interface between the physical channel and the data link controller (shown in Figure 4-4). Note that the former two are physical interfaces specific to the channel, and are specified in the interest of compatibility. The latter is provided as a means by which the data link controller can interact with the physical channel.

The channel access component of the logical interface (discussed in 4.4.1) comprises the collision and carrier detect functions described in 7.5.2 and 7.5.3, as well as the actual transmission of signals on the media. The data encoding and decoding functions described in 4.4.1 comprise the generation and decomposition of encoded signals suitable for transmission (described in 7.5.1), the generation and removal of code-specific preamble (described in 7.5.1.3 and 7.5.4.1), and the serial bit stream interface between the layers.

Section 5 describes the interface between the data link and physical layers as a series of Pascal procedures, functions, and shared variables. The data link specification in section 6 shows how the data link uses this interface to communicate between client layers. However, this specification will not attempt to model the operation of the physical channel in Pascal. The interface between layers is supported by the physical hardware which provides the ability to send and receive bit streams, provide timing, and signal carrier sense and collision detect to the data link.

The remainder of this section specifies the requirements for compatibility at both the transceiver cable and the coaxial cable. In addition, the specifications for the transceiver, which interfaces the transceiver cable to the coaxial cable is given, as well as the specification for the logic required between the transceiver cable and the interface to the data link.

7.2 Transceiver Cable Compatibility Interface Specifications

The transceiver cable is the means by which a physically separate transceiver is connected to a station. It provides one of the compatibility interfaces described in 7.1.4.2.

7.2.1 Transceiver Cable Signals

The transceiver cable carries four signals: Transmit, Receive, Collision Presence, and Power. Each signal is carried on a twisted pair of conductors in the cable.

7.2.1.1 Transmit Signal

The transmit pair carries encoded data for which the data link is requesting transmission on the channel. This signal is generated by the data encoder, with the transceiver cable drive characteristics specified in 7.2.4.

Amazon
Exhibit 1005
Page 253

7.2.1.2 Receive Signal

The receive pair carries encoded data from the transceiver to the station. It typically goes to the data decoder and the carrier sense circuitry. In the steady-state, all transitions and lack of transitions on the coaxial cable become transitions and lack of transitions on the receive pair, with the transceiver cable drive characteristics specified in 7.2.4. (During start-up, the first few bits may be absorbed by the transceiver to attain steady-state.)

In the case of a station transmitting without collision interference, the station's own transmit transitions on the coaxial cable will also appear on the receive pair, after a delay due to propagation through the transceiver. During collisions (whether or not that transceiver is involved in the collision) transitions on the receive lead are undefined; they may or may not meet decoder phase requirements, or they may not be present at all for extended periods. Thus the receive signal on the transceiver cable cannot be used alone to deterministically generate the carrier sense signal. This is described in more detail in 7.5.3.

7.2.1.3 Collision Presence Signal

The collision presence pair is used by the transceiver to indicate the presence of multiple transmission attempts on the coaxial cable. This is done by transmitting a square wave with a 10MHz fundamental frequency through the standard transceiver cable driver (described in 7.2.4). An oscillator is used instead of a simple level shift to allow AC coupling at the transceiver. Transceivers use the collision presence signal to indicate one of two conditions: the transceiver is transmitting and there is an attempt by another station to transmit at the same time, or there is a simultaneous transmission attempt by three or more stations regardless of whether the transceiver in question is transmitting.

7.2.1.4 Power

A pair of wires is designated for providing power to the transceiver. When the transceiver cable is implemented, the station end of the cable must supply a voltage between +12 and +15 Vdc ± 5% with at least 0.5 Amperes available to the cable for remotely powering the transceiver. The power source must meet applicable requirements for UL Class 2 wiring devices.

7.2.2 Transceiver Cable Parameters

7.2.2.1 Mechanical Configuration

The transceiver cable consists of four stranded, twisted pair conductors, plus an overall shield and insulating jacket. The conductor and jacket insulating material may be polyethylene or other suitable material. The flammability characteristics of the insulating material must be suitable for the installed environment.

Amazon
Exhibit 1005
Page 254

7.2.2.2 Characteristic Impedance

The differential mode characteristic impedance of all pairs shall be 78 Ω, ± 5 Ω, in the configuration.

7.2.2.3 Attenuation

The signal attenuation of any pair shall not exceed 3 dB (measured at 10 MHz) for the total length between the transceiver and the station.

7.2.2.4 Velocity of Propagation

The minimum velocity of propagation of the transceiver cable shall be 0.65 c.

7.2.2.5 Pulse Distortion

Pulse distortion shall not exceed ± 1 nS at the end of 50 meters of cable when driven with random 10 Mbit data encoded in accordance with 7.5.1.

7.2.2.6 Resistance

The resistance of the conductors used for the power pair shall not exceed 40 milliohms per meter.

7.2.2.7 Transfer Impedance

The common mode transfer impedance of the transceiver cable shall not exceed the values shown in Figure 7-2 as a function of frequency. The differential mode transfer impedance of the cable with respect to any pair shall be 20 dB lower than the specified common mode transfer impedance.

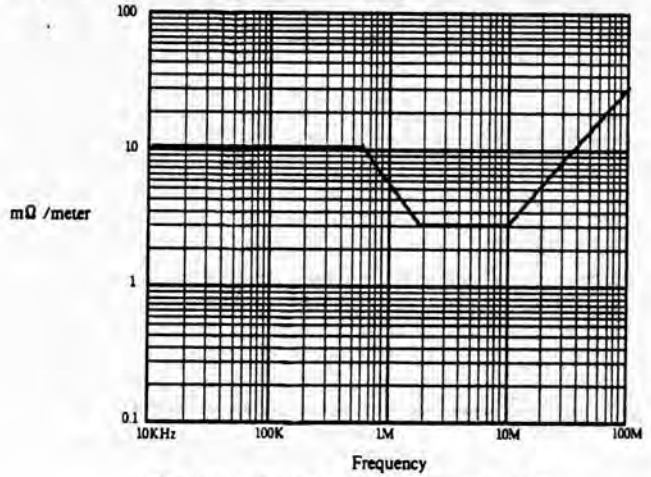


Figure 7-2: Transceiver Cable Transfer Impedance

7.2.3 Transceiver Cable Connectors

The connectors used at the ends of the transceiver cable shall be 15 conductor 'D' subminiature types (Cinch type DASM-15 or equivalent). The end of the cable that mates with the transceiver must use a female connector with a slide lock assembly (Cinch type DA 51220-1 or equivalent). The transceiver must provide a mating male connector with locking posts. The other end of the transceiver cable (which mates with a female connector at the station) must use a male connector with locking posts (Cinch type D 53018 or equivalent). The station must provide a female connector with the slide lock assembly.

Because of the end-to-end matching of the connectors, transceiver cables may be extended by concatenating transceiver cable sections. (The transceiver cable sections function as 'extension cords'.) A cable with multiple sections must still meet the cable loss characteristics of 7.3.1.1.2.

The pin assignment is given in the following table:

Transceiver Cable Connector Pin Assignment

1.	Shield (See note)		
2.	Collision Presence +	9.	Collision Presence -
3.	Transmit +	10.	Transmit -
4.	Reserved	11.	Reserved
5.	Receive +	12.	Receive -
6.	Power return	13.	Power
7.	Reserved	14.	Reserved
8.	Reserved	15.	Reserved

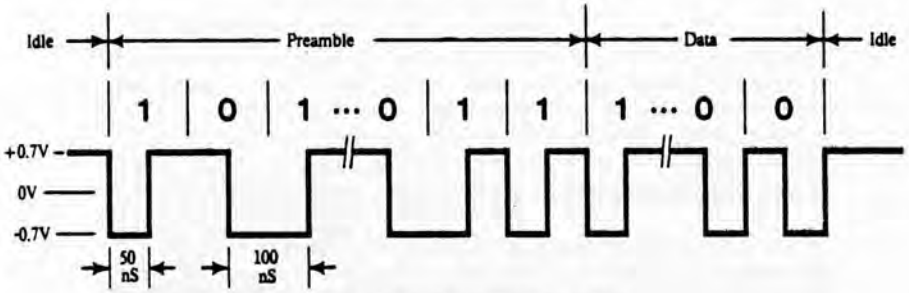
Note: Shield must be terminated to connector shell as well as pin 1.

Metal, metallized plastic, or otherwise shielded connector backshells must be used to ensure shield integrity.

7.2.4 Transceiver Cable Drive

This section describes the requirements for driving any of the signal pairs in the transceiver cable: transmit, receive, and collision presence.

The AC signal levels presented to the transceiver cable shall be ± 700 mV nominal, balanced differential drive into 78 ± 5 Ω. The common mode voltage presented to the transceiver cable shall not exceed that allowed at the receiver, as specified in 7.2.5.2. Signal waveform shall be as shown in Figure 7-3.



1. Voltages are measured differentially at output of transceiver cable driver.
2. Rise and fall times meet 10,000 series ECL requirements.

Figure 7-3: Typical Transceiver Cable Waveform

The transceiver cable driver must be capable of maintaining the specified minimum differential signal into the worst case low cable impedance (73 Ω differential, 18.5 Ω common mode) in the environment specified in section 7-7.

The idle state of the output shall be high (+ 700 mV nominal); the first transition presented is negative-going, the last transition must be positive-going. Note that the presence of AC coupling may cause the voltage as specified at the output of the transceiver cable drive circuit not to appear on the transceiver cable in the idle state.

A typical transceiver cable drive circuit is given in Appendix D.

7.2.5 Transceiver Cable Receive

The following sections specify the requirements for receiving signals from any signal pair in the transceiver cable: transmit, receive, and collision presence. The circuit must be capable of receiving the signals from the transceiver cable driver specified in 7.2.4 through the cable specified in 7.2.2 in the worst case. A typical receive circuit is given in Appendix D.

7.2.5.1 Load Impedance and Termination

The termination impedance shall be 78 Ω ± 1% differential mode, and 18.5 Ω minimum common-mode, over the frequency range of 3-20 MHz.

7.2.5.2 Common Mode and CMRR

The common mode range and the common mode rejection ratio shall be sufficient to maintain a 5:1 signal to noise ratio in the environment specified in 7.7, measured at the input to the transceiver cable receiver. The common mode DC voltage at the input of the receiver shall be in the range of zero to +5 Vdc..

Amazon
Exhibit 1005
Page 255

7.3 Coaxial Cable Compatibility Interface Specifications

The coaxial cable is the common, shared broadcast medium through which stations communicate. It provides one of the compatibility interface points described in 7.1.4.2.

7.3.1 Coaxial Cable Component Specifications

The cable is of constant impedance, coaxial construction. It is terminated at each end by a *terminator* (specified in 7.3.1.3), and connection provided for each transceiver. Coaxial cable connectors are used to make the connection from the cable to the terminators, and between cable sections (if needed). The cable has various electrical and mechanical requirements which must be met to ensure proper operation.

7.3.1.1 Coaxial Cable Parameters

7.3.1.1.1 Characteristic Impedance

The average characteristic impedance of the cable shall be 50 ± 2 Ω, measured according to Mil. Std. C17-E. Periodic variations in impedance along a single piece of cable may be up to ± 3 Ω sinusoidal, centered around the average value, with a period < 2 meters. Note that the proper operation of the network is dependent upon the cable characteristic impedance; its value and tolerance are critical.

7.3.1.1.2 Attenuation

The attenuation of a cable segment shall not exceed 8.5 dB measured at 10 MHz, nor 6.0 dB measured at 5 MHz.

7.3.1.1.3 Velocity of Propagation

The minimum acceptable velocity of propagation is 0.77 c..

7.3.1.1.4 Mechanical Requirements

The cable used should be suitable for routing in various environments, including but not limited to, dropped ceilings, raised floors, and cable troughs. The jacket must provide insulation between the cable sheath and any building structural metal. Also, the cable must be capable of accepting coaxial cable connectors, described in 7.3.1.2. The cable must in addition conform to the following requirements:

1. The center conductor must be 0.0855" ± .0005" diameter solid, tinned copper,
2. The core dielectric material must be foamed,
3. The inside diameter of the innermost shield must be .242" minimum,
4. The outside diameter of the outermost shield must be .326" ± .007",
5. The outermost shield must be greater than 90% coverage tinned copper braid,
6. The jacket O.D. must be 0.405" nominal,
7. The cable concentricity must be 90% minimum.

The cable must also meet applicable flammability criteria and local codes for the installed environment. Different (e.g., polyethylene and Teflon dielectric) types of cable sections may be interconnected, while meeting the sectioning requirements of 7.6.1.

7.3.1.1.5 Pulse Distortion

Pulse distortion shall not exceed ± 7 nS at the end of 500 meters of cable when driven with random 10 Mbit data encoded in accordance with 7.5.1.

7.3.1.1.6 Jacket Marking

The cable jacket must be marked with annular rings in a color contrasting with the background color of the jacket. The rings must be spaced at 2.5 meter ± 5 cm regularly along the entire length of the cable. It is permissible for the 2.5 meter spacing to be interrupted at discontinuities between cable sections joined by connectors. (See 7.6.2 for transceiver placement rules which mandate cable markings.)

7.3.1.1.7 Transfer Impedance

The transfer impedance of the cable shall not exceed the values shown in Figure 7-4 as a function of frequency.

Amazon
Exhibit 1005
Page 256

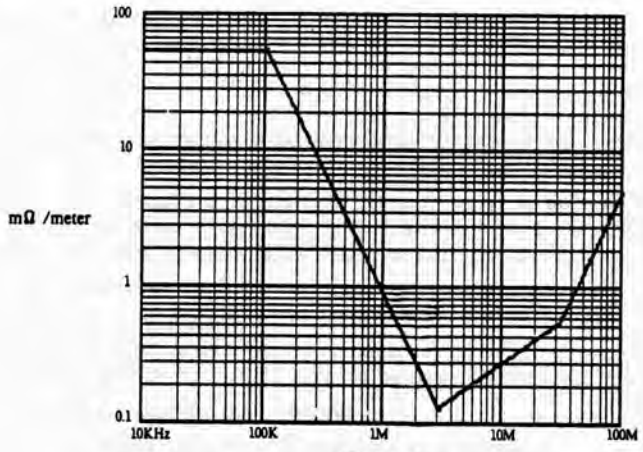


Figure 7-4: Maximum Coaxial Cable Transfer Impedance

7.3.1.2 Coaxial Cable Connectors

Coaxial cable connectors are used to join cable sections and attach terminators. Three types of connectors may be necessary; male plugs, female jacks, and female-to-female barrels. Plugs are used exclusively at the ends of all cable sections. Jacks are used to house cable terminators. Barrels are used to join cable sections.

All connectors are N series, 50 Ω constant impedance types. Since the frequencies present in the transmitted data are well below UHF range (being band-limited to approximately 20 MHz), military versions of the connectors are not required (but are acceptable).

Means must be provided to ensure that the connector shell (which connects to the cable sheath) does not make contact with any building metal, or other unintended conductor. A sleeve or boot to be slid over the connector at installation time is suitable.

7.3.1.3 Coaxial Cable Terminators

Coaxial cable terminators are used to provide a termination impedance for the cable equal in value to its characteristic impedance, thereby eliminating any reflection from the ends of the cables. Terminators shall be packaged within an inline female jack connector. The termination impedance shall be 50 Ω ± 1% measured from 0-50 MHz, with the magnitude of the phase angle of the impedance not to exceed 5 degrees. The terminator power rating shall be 1 watt or greater.

7.3.1.4 Transceiver-to-Coaxial Cable Connections

A means must be provided to allow for attaching a transceiver to the coaxial cable. The connection must disturb the transmission line characteristics of the cable as little as possible; it must present a predictably low shunt capacitance, and therefore a negligibly short stub length. For this reason, the transceiver must be located as close to its cable connection as possible; they are normally considered to be one assembly. Long (greater than 3 cm) connections between the coaxial cable and the input of the transceiver are not acceptable.

The transceiver-to-coaxial cable connection shall present less than 2 picofarads shunt capacitance to the coaxial cable, not including any transceiver electronics. If the design of the connection is such that the coaxial cable must be severed to install the transceiver, the coaxial cable segment must still meet the sectioning requirements of 7.6.1. Any coaxial connectors used on a severed cable must be type N, as specified in 7.3.1.2.

7.3.2 Coaxial Cable Signaling

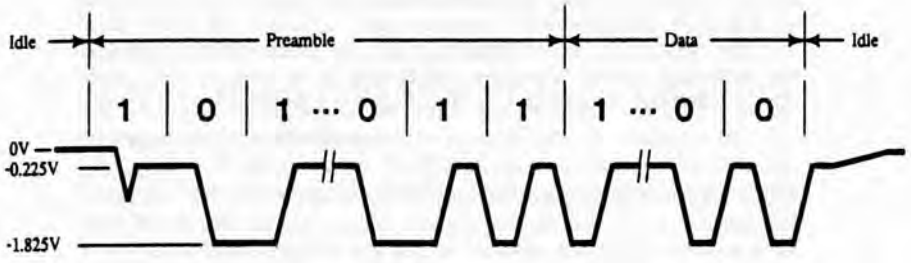
The AC component of the signal on the coaxial cable due to a single transceiver as measured on the coaxial cable immediately adjacent to the transceiver connection shall be ± 16 mA nominal (14 mA min, 19 mA max). The DC component shall be one-half the AC component, plus 4.5 mA (4 mA min, 5 mA max). The actual current measured at a given point on the cable is a function of the transmitted current and the cable loss to the point of measurement. Positive current is defined as current out of the center conductor of the cable (into the transceiver). Cable loss is specified in 7.3.1.1.2.

The 10%-90% rise and fall times shall be 25 ± 5 nsec. Figure 7-5 shows typical waveforms present on the cable. Harmonic content generated from a 10 MHz fundamental periodic input shall meet the following requirements:

Second and Third Harmonics: -20 dB min
Fourth and Fifth Harmonics: -30 dB min
Sixth and Seventh Harmonics: -40 dB min
All Higher Harmonics: -50 dB min

The signals as generated from the encoder (described in 7.5.1.1) shall appear on the coaxial cable without any inversions.

Amazon
Exhibit 1005
Page 257



1. Voltages given are nominal; worst case is given in text.
2. Rise time is 25 nS nominal.
3. Voltages are measured on coaxial cable adjacent to transceiver.

Figure 7-5: Typical Coaxial Cable Waveform

56

7.4 Transceiver Specifications

The following sections specify the requirements for a transceiver.

7.4.1 Transceiver-to-Coaxial Cable Interface

The following sections describe the interface between the transceiver and the coaxial cable. Positive current is defined as current into the transceiver (out of the center conductor of the cable).

7.4.1.1 Input Impedance

The shunt capacitance presented to the coaxial cable by the transceiver circuitry (not including the means of attachment to the coaxial cable) shall not exceed 2 picofarads. The shunt resistance presented to the coaxial cable shall be greater than 50 KΩ.

These conditions must be met in both the power off and the power on, not transmitting states.

7.4.1.2 Bias Current

The transceiver must draw between -2 and +50 uA in the power-off and the power-on, not transmitting states.

Amazon
Exhibit 1005
Page 258

7.4.1.3 Transmit Output Levels

Signals received from the transceiver cable transmit pair must be transmitted onto the coaxial cable with the characteristics specified in 7.3.2. Note that 7.3.2 specifies the current level on the coaxial cable. Since the coaxial cable proceeds in two directions away from the transceiver, the current into the transceiver is actually twice the current measured on the coaxial cable.

Transmitted output asymmetry shall not exceed 2 ns for a 50/50 duty cycle input on the transceiver cable transmit pair.

7.4.2 Transceiver-to-Transceiver Cable Interface

7.4.2.1 Transmit Pair

The transceiver must present the transceiver cable receive characteristics specified in 7.2.5 to the transmit pair. At the start of a frame transmission, no more than 2 bits (two 100 ns bit cells) of information may be received from the transmit pair and not transmitted onto the coaxial cable. The steady-state propagation delay between the transmit pair input and the coaxial cable output shall not exceed 50 ns. There are no signal inversion between the transceiver cable transmit pair and the coaxial cable.

7.4.2.2 Receive Pair

The transceiver must present the transceiver cable transmit characteristics specified in 7.2.4 to the receive pair. Asymmetry as seen on the receive pair shall not exceed ± 2 nsec for a ± 200 mV peak sinusoidal input from the coaxial cable.

The signal from the coaxial cable shall pass through AC coupling with an appropriate time constant before proceeding to the receive pair. The time constant should compensate for the coaxial cable pulse distortion.

At the start of a frame reception from the coaxial cable, no more than 5 bits (five 100 ns bit cells) of information may be received from the coaxial cable and not transmitted onto the receive pair. In addition, it is permissible for the first bit sent over the receive pair to contain encoding phase violations or invalid data, however all successive bits of the frame shall be valid and meet encoding rules. The steady-state propagation delay between the coaxial cable and the receive pair output shall not exceed 50 ns. There are no signal inversions between the coaxial cable and the transceiver cable receive pair.

7.4.2.3 Collision Presence Pair

The transceiver must present the transmitter characteristics specified in 7.2.4 to the collision presence pair. The signal presented to the collision presence pair shall be a periodic waveform with a 10 MHz ± 15% frequency. This signal shall be presented to the collision presence pair no more than 5 bit times (500 nS) after the average signal on the coaxial cable at the transceiver exceeds either that which could be produced by two transceiver outputs in the worst case (if the transceiver in question is not transmitting), or that which could be produced by that transceiver alone in the worst case (if that transceiver is transmitting).

7.4.2.4 Power Pair

The transceiver cable provides power which may be used for operation of the transceiver electronics. The power available shall be as described in 7.2.1.4. The distribution impedance of the transceiver cable is 4 Ω maximum, for a 50 meter cable with the resistance specified in 7.2.2.6. In order for the transceiver to derive its operating power from the power pair, circuitry must be employed to provide the required electrical isolation specified in 7.4.3.

7.4.3 Electrical Isolation

The transceiver must provide electrical isolation between the transceiver cable and the coaxial cable. The isolation impedance shall be greater than 250 KΩ, measured between any conductor (including shield) of the transceiver cable and either the center conductor or shield of the coaxial cable, at 60 Hz. The breakdown voltage of the isolation means provided shall be at least 250 VAC, rms.

7.4.4 Reliability

No single nor double component failure within the transceiver electronics shall impede communication among other transceivers on the coaxial cable. Connectors and other passive components comprising the means of connecting the transceiver to the coaxial cable shall be designed to minimize the probability of total network failure.

7.5 Channel Logic

The following sections describe the functions that must be performed to properly interface between the data link and the transceiver cable. They are normally implemented as logic, typically within the same device implementing the data link layer.

7.5.1 Channel Encoding

The channel shall use Manchester phase encoding, with a data rate of 10 Mbps, ± .01%, measured at the encoder clock. Thus, each bit cell is 100 ns long.

The following section describes the requirements for encoding and decoding signals to be transmitted on, or received from the coaxial or transceiver cables.

7.5.1.1 Encoder

The encoder is used to translate physically separate signals of clock (synchronization) and data into a single, self-synchronizable serial bit stream, suitable for transmission on the coaxial cable by the transceiver.

During the first half of the bit cell time, the serial signal transmitted is the logical complement of the bit value being encoded during that cell. During the second half of the bit cell time, the uncomplemented value of the bit being encoded is transmitted. Therefore, there is always a signal transition (either positive-going or negative-going, depending on the bit being encoded) in the center of each bit cell. A timing diagram for a typical bit stream is given in Figure 7-6.

The encoder output drives the transmit pair of the transceiver cable, and ultimately, the coaxial cable through the transceiver. The encoder output asymmetry must not exceed 0.5 ns. The encoder shall provide the defined output for the first (and all subsequent) bits presented to its input. All information submitted for encoding shall appear at the output of the encoder.

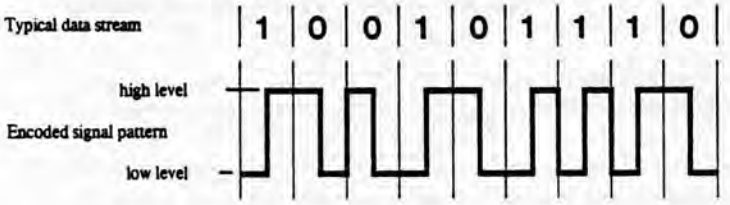


Figure 7-6: Manchester Encoding

7.5.1.2 Decoder

The decoder is used to separate the incoming phase encoded bit stream into a data stream and a clock signal. The decoder must be able to provide data and clock signals usable by the data link under the asymmetry imposed by the worst case system configuration. The decoder must provide usable output (clock and data) after no more than 8 bit cell times after reception of an encoded signal. The first signals received from the transceiver at the beginning of frame reception may not constitute a valid, properly encoded bit; it is possible for the time from the first transition seen to the first true mid-bit cell transition to assume any value from zero to 100 nS.

The decoder input is normally derived from the coaxial cable, through the transceiver cable receive pair. It is not necessary for the decoder to provide usable output when there is a collision on the coaxial cable, regardless of whether the station using that decoder is involved in the collision.

57

Amazon
Exhibit 1005
Page 259

7.5.1.3 Preamble Generation

Because most of the channel circuitry is allowed to provide valid output some number of bit times after being presented valid input, it is necessary for a preamble to be sent before the start of data link information, to allow the channel circuitry to reach its steady-state, with valid outputs throughout the system. Upon request by the data link to transmit the first bit of a new frame, the channel shall first transmit the preamble; a predetermined bit sequence used for channel stabilization and synchronization. If, while transmitting the preamble, the channel logic asserts the collision detect signal as specified in 7.5.2, any remaining preamble bits shall not be sent. The channel should immediately proceed with the transmission of the bit submitted by the data link.

The preamble is a 64 bit pattern to be presented to the channel encoder in the same manner as data link information. The pattern is:

10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101011.

The bits are transmitted in order, from left to right. The nature of the pattern is such that when encoded, it appears as a periodic waveform on the cable, with a 5 MHz frequency. Excepting the final two bits, the only transitions present in the waveform are in the center of the bit cells. This is depicted in Figure 7-7. The last two bits of the preamble contain transitions at both the bit cell centers and the edges, and are used to indicate the end of the preamble, and the beginning of the data link encapsulation portion of the frame. The next bit transmitted is the bit originally submitted by transmission by the data link.

Preamble removal on reception is discussed in 7.5.4.1.

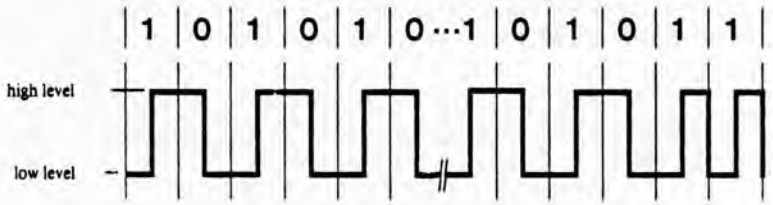


Figure 7-7: Preamble Encoding

7.5.2 Collision Detect Signal

The channel must indicate to the data link when the signals on the coaxial cable imply simultaneous transmission attempts by more than one station. This is normally indicated through the collision presence pair in the transceiver cable, described in 7.2.1.3.

The channel logic must assert the collision detect signal within 2 bit times (200 ns), following the onset of collision presence. This collision detect signal shall be asserted only when the data link is transmitting. A functional logic description of the collision detect signal is shown in Figure 7-8.

Following the loss of collision presence information, the channel must deassert the collision detect signal within 1.6 bit cell times (160 ns).

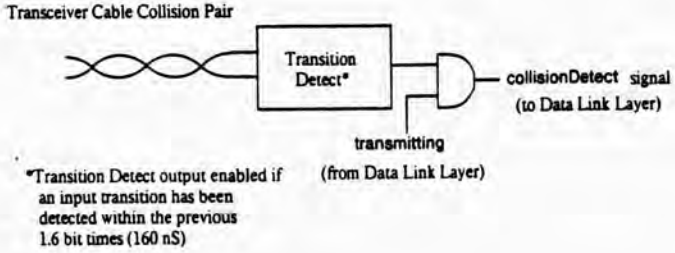


Figure 7-8: Functional Logic of collisionDetect Signal

7.5.3 Carrier Sense Signal

The channel must indicate to the data link the presence of carrier, a signal transmission attempt on the coaxial cable by a station. This is normally indicated through both the receive and collision presence pairs in the transceiver cable, described in 7.2.1.

The carrier sense signal shall be asserted when one or more station is attempting transmission on the cable, regardless of whether the station sensing carrier is transmitting at that time. The channel logic must assert the carrier sense signal within 2 bit times (200 ns) following the onset of carrier presence information. A

Amazon
Exhibit 1005
Page 260

functional logic description of these signals is shown in Figure 7-9.

Following the loss of carrier presence information (receive transitions and collision presence information) the channel must deassert the carrier sense signal within 1.6 bit cell times (160 ns).

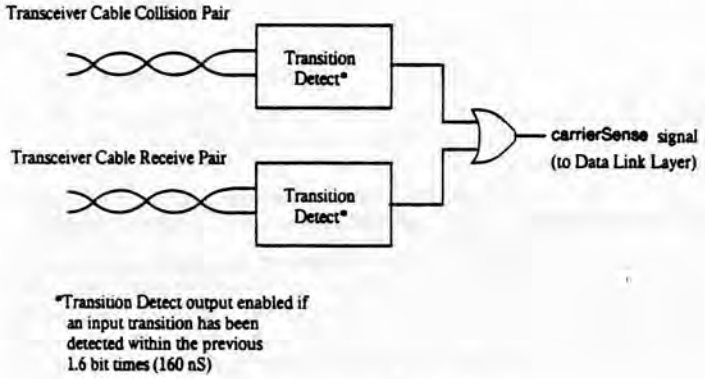


Figure 7-9: Functional Logic of carrierSense Signal

7.5.4 Channel Framing

During reception, the channel must provide the data link with signals to indicate beginning and end of frame.

7.5.4.1 Beginning-of-Frame Sequence

The channel logic recognizes the presence of activity on the medium through the carrier sense signal. This is the first indication that the frame reception process should begin. However, dependent upon the physical configuration of the system, there are some number of preamble bits to be received by the channel before the start of the data link frame as indicated by the double-1 at the end of preamble. In addition, the first signals received from the decoder may be invalid due to the first bit allowance of the transceiver (see 7.4.2.2). The channel must wait no less than 8 bit times (800 nS) before monitoring the output of the decoder for the 'double-1' indicating end of preamble, and beginning of data link frame. Upon reception of the double-1, the channel shall begin passing successive bits to the data link through the defined receive bit stream interface. If, after waiting the required 8 bit times, a 'double-0' is encountered, the physical channel shall not pass any bits of

the current frame to the data link. Normal operation of the data link and channel shall resume on the subsequent frame.

7.5.4.2 End-of-Frame Sequence

As specified in 7.5.3, the carrier sense signal must be deasserted no later than 1.6 bit times (160 ns) after the cessation of activity on the coaxial cable as seen by the channel logic. The channel ensures that no extraneous bits will appear at the end of a frame following the last valid bit.

7.6 Channel Configuration Requirements

7.6.1 Cable Sectioning

The 500 meter maximum length coaxial cable segment need not be made from a single, homogeneous length of cable. The boundary between two cable sections (joined by coaxial connectors; two male plugs and a barrel) represents a signal reflection point due to the impedance discontinuity caused by the batch-to-batch impedance tolerance of the cable. Since the worst-case variation from 50 Ω is 2 Ω (see 7.3.1.1.1), a possible worst-case reflection of 4% may result from the joining of two cable sections. The configuration of long cable segments (up to 500 meters) from smaller sections must be made with care. The following recommendations apply, and are given in order of preference:

1. If possible, the total segment should be made from one homogeneous (no breaks) cable. This is feasible for short segments, and results in minimal reflections from cable impedance discontinuities.
2. If cable segments must be built up from smaller sections, it is highly desirable to ensure that all the sections are from the same manufacturer and lot. This is equivalent to using a single cable, since the cable discontinuities are due to extruder limitations, and not extruder-to-extruder tolerances. There are no restrictions in cable sectioning if this method is used. However, if a cable section in such a system is later replaced, it must be replaced either with another cable from the same manufacturer and lot, or with one of the standard lengths described below.
3. If uncontrolled cable sections must be used in building up a longer segment, the lengths should be chosen such that reflections, when they occur, do not have a high probability of adding in phase. This can be accomplished by using lengths which are odd integral multiples of a half-wavelength in the cable at 5 MHz; this corresponds to using lengths of 23.4, 70.2, and 117 meters (± 0.5 meters) for all sections. These are considered to be the standard lengths for all cable sections. Using these lengths exclusively, any mix or match of cable sections may be used to build up a 500 meter segment without incurring excessive reflections.

Amazon
Exhibit 1005
Page 261

4. As a last resort, an arbitrary configuration of cable sections may be employed, if it has been confirmed by analysis or measurement that the worst-case signal reflection due to the impedance discontinuities at any point on the cable does not exceed 7% of the incident wave when driven by a transceiver meeting the specifications of 7.4.

7.6.2 Transceiver Placement

Transceivers and their associated connections to the cable cause signal reflections due to their non-infinite bridging impedance. While this impedance must be implemented as specified in 7.3.1.4 and 7.4.1, the placement of transceivers along the coaxial cable must also be controlled to insure that reflections from transceiver do not add in phase to a significant degree.

Coaxial cables marked as specified in 7.3.1.1.6 have marks at regular 2.5 meters spacing; a transceiver may be placed at any mark on the cable. This guarantees both a minimum spacing between transceivers of 2.5 meters, as well as controlling the relative spacing of transceivers to insure non-alignment on fractional wavelength boundaries.

The total number of transceivers on a cable segment shall not exceed 100.

7.6.3 System Grounding

The sheath conductor of the coaxial cable shall not make electrical contact with any earth reference, building structural metal, ducting, plumbing fixture, or other unintentioned conductor. Insulators may be used to cover any coaxial connectors used to join cable sections and terminators, to insure that this requirement is met. A sleeve or boot attached at installation time is acceptable.

The sheath conductor of the transceiver cable shall be connected to the earth reference or chassis of the device housing the station logic.

60

7.6.4 Repeaters

Repeaters are used to extend the channel length and topology beyond that which could be achieved by a single coaxial cable segment. (See the channel configuration model in 7.1.5.) A repeater requires a transceiver on each of the segments between which it is repeating signals. These transceivers must be as specified in 7.4, and must be counted towards the maximum specified in 7.6.2.

A maximum of two repeaters may be in the signal path between any two transceivers on the channel.

7.6.4.1 Carrier Detect and Transmit Repeat

Repeaters must implement the carrier sense function as specified in 7.5.3 for both segments between which it is connected. Upon detection of carrier from one segment, the repeater must retransmit all received signals from that segment onto the other segment. Signals shall be retimed and amplified as specified in 7.6.4.3. The maximum steady-state propagation delay through the repeater for the repeated signal (not including startup delays, carrier sense delay or retiming delays) shall not exceed 800 nS.

7.6.4.2 Collision Detect and Collision Repeat

Repeaters must implement the collision detect function as specified in 7.5.2 for both segments between which it is connected. If, while repeating signals as specified in 7.6.4.1, collision is detected on either side, the repeater must ensure that all stations involved in the collision recognize the event as a collision, regardless of which side of the repeater the station is on. The maximum time between the recongnition of the collision and the repeating of the collision indication (not including carrier sense of retiming delays) shall not exceed 200 nS.

7.6.4.3 Repeater Signal Regeneration

7.6.4.3.1 Signal Amplification

The repeater (with its associated transceivers) shall ensure that any signals repeated between segments shall have the same amplitude characteristics at the transceiver output of the repeated-to segment as they did at the output of the transmitter on the repeated-from segment, allowing for transceiver output tolerances as specified in 7.4.1.3. Any loss of signal-to-noise ratio due to cable loss and noise pickup is thus regained at the output of the repeater.

7.6.4.3.2 Signal Timing

The repeater must ensure that the symmetry characteristics of the signals at the transceiver output of the repeated-to segment are the same as those at the output of the transmitter on the repeated-from segment, allowing for transceiver and transceiver cable tolerances. Any loss of symmetry due to transceivers and cable distortion is thus regained at the output of the repeater.

Amazon
Exhibit 1005
Page 262

7.7 Environment Specifications

The following sections specify the physical environment in which all channel components must operate to be considered compatible.

7.7.1 Electromagnetic Environment

The physical channel hardware shall meet its specifications when operating in the following ambient plane-wave fields:

2 Volts/Meter from 10 KHz through 30 MHz

5 Volts/Meter from 30 MHz through 1 GHz

7.7.2 Temperature and Humidity

All physical channel hardware, with the possible exception of the channel logic components shall operate over the ambient temperature range of 5 to 50 degrees Celsius, and humidity range of 10% to 95% non-condensing. The channel logic components are normally part of the station hardware, and are thus subject to individual station product requirements. Hardware which does not meet the temperature and humidity requirements specified must state so in its published product specification.

Amazon
Exhibit 1005
Page 263

APPENDIX A: GLOSSARY

This section defines some of the essential terminology associated with the Ethernet.

baseband coaxial system: A system whereby information is directly encoded and impressed on the coaxial transmission medium. One information signal at a time can be present on the medium without disruption (see collision).

binary exponential backoff: The algorithm used to schedule retransmissions after a collision. So called because the interval from which the retransmission time is selected is expanded exponentially with repeated collisions.

broadcast: Describes the class of media for which the Ethernet is designed, in which all stations are capable of receiving a signal transmitted by any other station. Also, describes the mode of usage of such a medium by the Data Link Layer in which all stations are instructed to receive a given frame.

carrier sense: A signal provided by the Physical Layer to the Data Link Layer to indicate that one or more stations are currently transmitting on the channel.

channel logic: The logical functions provided between the transceiver cable and the Data Link, which support the defined interface between the data link and the physical layers.

Client Layer: Collective term used to describe any layer of a network architecture, which use the Ethernet Data Link and Client interface.

coaxial cable: A two-conductor, concentric, constant impedance transmission line.

coaxial cable interface: The electrical, mechanical, and logical interface to the shared coaxial cable medium. This is a mandatory compatibility interface, which must be correctly implemented by every Ethernet implementation.

coaxial cable section: An unbroken piece of coaxial cable, fitted with coaxial connectors at its ends, used to build up coaxial cable segments.

coaxial cable segment: A length of coaxial cable made up from one or more coaxial cable sections and coaxial connectors, terminated at each end in its characteristic impedance. A 500 meter segment is the longest configuration possible without repeaters.

collision: The result of multiple transmissions overlapping in the physical channel, resulting in garbled data and necessitating retransmission.

collision detect: A signal provided by the Physical Layer to the Data Link Layer to indicate that one or more other stations are contending with the local station's

transmission. It can be true only during transmission.

collision enforcement: Transmission of extra, encoded "jam" bits after a collision is detected, to insure that the duration of the collision is sufficient to guarantee its detection by all transmitting stations.

compatibility interfaces: The coaxial cable interface, and the transceiver cable interface, the two points at which hardware compatibility is defined to allow connection of independently designed and manufactured components to the Ethernet.

contention: Interference between colliding transmissions (see collision). Resolution of occasional contention is a normal part of the Ethernet's distributed link management procedure (see CSMA-CD).

controller: The implementation unit which connects a station to the Ethernet, typically comprising part of the Physical Layer, much or all of the Data Link Layer, and appropriate electronics for interfacing to the station.

CSMA-CD: Carrier Sense Multiple Access with Collision Detection, the generic term for the class of link management procedure used by the Ethernet. So called because it a) allows multiple stations to access the broadcast channel at will, b) avoids contention via carrier sense and deference, and c) resolves contention via collision detection and retransmission.

Data Link Layer: The higher of the two layers in the Ethernet design, which implements a medium-independent link level communication facility on top of the physical channel provided by the Physical Layer.

deference: A process by which a data link controller delays its transmission when the channel is busy to avoid contention with ongoing transmissions.

frame check sequence: An encoded value appended to each frame by the Data Link Layer to allow detection of transmission errors in the physical channel.

interframe spacing: An enforced idle time between transmission of successive frames to allow receiving data link controllers and the physical channel to recover.

jam: An encoded bit sequence used for collision enforcement.

Manchester encoding: A means by which separate data and clock signals can be combined into a single, self-synchronizable data stream, suitable for transmission on a serial channel.

multicast: An addressing mode in which a given frame is targeted to a group of logically related stations.

physical address: The unique address value associated with a given station on the network. An Ethernet physical address is defined to be distinct from all other physical addresses on all Ethernets.

Amazon
Exhibit 1005
Page 264

Physical Channel: The implementation of the physical layer.

Physical Layer: The lower of the two layers of the Ethernet design, implemented by the physical channel using the specified coaxial cable medium. The Physical Layer insulates the Data Link Layer from medium-dependent physical characteristics.

preamble: A sequence of 64 encoded bits which the Physical Layer transmits before each frame to allow synchronization of clocks and other Physical Layer circuitry at other sites on the channel.

repeater: A device used to extend the length and topology of the physical channel beyond that imposed by a single segment, up to the maximum allowable end-to-end channel length.

round-trip propagation time: In bit times, the time required in the worst-case for a transmitting station's collision detect signal to be asserted due to normal contention for the channel. This delay is the primary component of the slot time.

slot time: A multi-purpose parameter which describes the contention behavior of the Data Link Layer. It serves as a) an upper bound on the collison vulnerability of a given transmission, b) an upper bound on the size of the frame fragment produced by a collision, and c) the scheduling quantum for collision retransmission.

station: A single addressable site on the Ethernet, generally implemented as a computer and appropriate peripherals, and connected to the Ethernet via a controller and a transceiver.

transceiver: The portion of the Physical Layer implementation that connects directly to the coaxial cable and provides both the electronics which send and receive the encoded signals on the cable and the required electrical isolation.

transceiver cable: A four pair, shielded cable used for the transceiver cable interface.

transceiver cable interface: The electrical, mechanical and logical interface which connects the transceiver to the controller. The standard transceiver cable is a recommended compatibility interface.

APPENDIX B: NOTES ON ADDRESS AND TYPE ASSIGNMENT, AND LICENSING

Address and Type Assignment

The address and type fields will be administered by Xerox Corporation.

A block of addresses will be assigned to each licensee of Ethernet patents (see below). Others may obtain an address block or type field assignment by request. A nominal fee to cover administrative costs will be charged.

Submit written requests to:

Xerox Corporation
Ethernet Address Administration Office
3333 Coyote Hill Road
Palo Alto, CA 94304

Licensing

Ethernet incorporates features that are protected by one or more patents assigned to Xerox Corporation. Questions on the need for licensing particular uses of this specification should be directed to:

Xerox Corporation
Director of Licensing
Long Ridge Road
Stamford, CT 06904

APPENDIX C: CRC IMPLEMENTATION

Every frame contains, in its frame check sequence field, a 32-bit cyclic redundancy check (CRC) code. Because the formal mathematical definition of this code (see 6.2.4) is not suggestive of an appropriate implementation, this appendix outlines one possible implementation in terms of a feedback shift register. This type of implementation is likely to be common in practice, but is not a mandatory part of the specification.

The feedback shift register (see Figure C-1) is used to represent division of the pre-scaled message by the generating polynomial. The 32-bit register is accessed via the three signals Input, Output, and Control. When Control = 1, Input bits are shifted into the feedback shift register and also fed directly back to Output. When Control = 0, the feedback paths are disabled and the shift register shifts the complement of its contents to Output.

Before CRC generation at the transmitting end, initialization logic (not shown in Figure C-1) preloads the shift register to all 1's. Control is then held at 1 while the address, type and data fields of the outgoing frame are shifted into Input and the CRC is generated. Meanwhile, the same bits emerging at Output are transmitted over the network. When the last bit of the data field has been processed, Control is set to 0 and the complemented CRC is shifted out for transmission, starting with the x^{31} term (see 6.2.4).

CRC checking at the receiving end also begins with the shift register preloaded to all 1's. Control is then held at 1 while the incoming bits are shifted into Input to regenerate the CRC. When the last bit of the data field has been processed, the shift register should contain the CRC whose binary complement is about to arrive on the network. Since this field boundary cannot be recognized by the receiver, however, Control remains at 1 and the bits of the CRC continue to feed into the the shift register until the end of the entire frame is reached. If the two CRCs match, the final contents of the shift register is the value:

11000111 00000100 11011101 01111011

(where the leftmost bit corresponds to the x^{31} term of the polynomial and the rightmost to the x^0 term). Any other final value indicates a detected error. (The extra logic to test for this value is not shown in Figure C-1).

Amazon
Exhibit 1005
Page 265

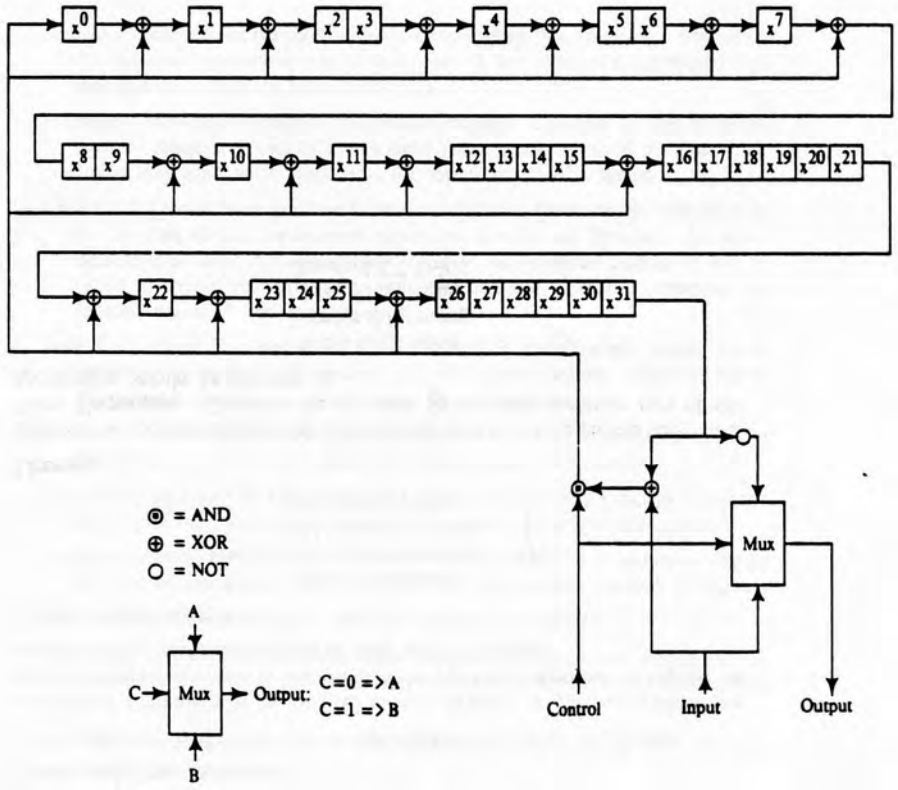


Figure C-1: CRC Implmentation

One potential problem which is avoided in this implementation is insensitivity of the shift register to incoming zero-bits when it is in the all-zero state. Following standard practice, this state is avoided at the beginning and end of the frame by preloading the shift register with all 1-bits, and by inverting each bit of the final CRC. Logically, these correspond, respectively, to the complementing of the first 32 bits of the frame and to the final complementing of the remainder, as specified in the mathematical definition in 6.2.4. See also [9] for further discussion.

64

Amazon
Exhibit 1005
Page 266

APPENDIX D: IMPLEMENTATION OF TRANSCEIVER CABLE DRIVER AND RECEIVER

This appendix presents circuit digrams for typical implementations of the transceiver cable drivers and receivers. The use of these exact circuits is not necessary for conformance to the specification; equivalent circuits may be used as long as the relevant specifications are met.

Figure D-1 depicts an implementation of the transceiver cable driver specified in 7.2.4. It is suitable for use at either end of the transceiver cable, as necessary; i.e., it would be located at the station end to drive the transmit pair, and at the transceiver end to drive the receive and collision presence pairs. In addition, it is capable of driving suitable isolation circuits required to be located within the transceiver.

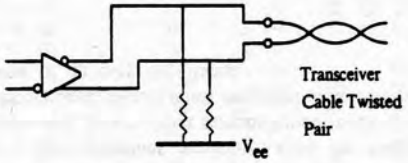


Figure D-1: Typical Transceiver Cable Driver

Figure D-2 depicts an implementation of the transceiver cable receiver specified in 7.2.5. It is suitable for use at either end of the transceiver cable, as necessary; i.e., it would be located at the station end to receive from the receive and collision presence pairs, and at the transceiver to receive from the transmit pair. It is capable of operating through suitable isolation circuits required to be located within the transceiver.

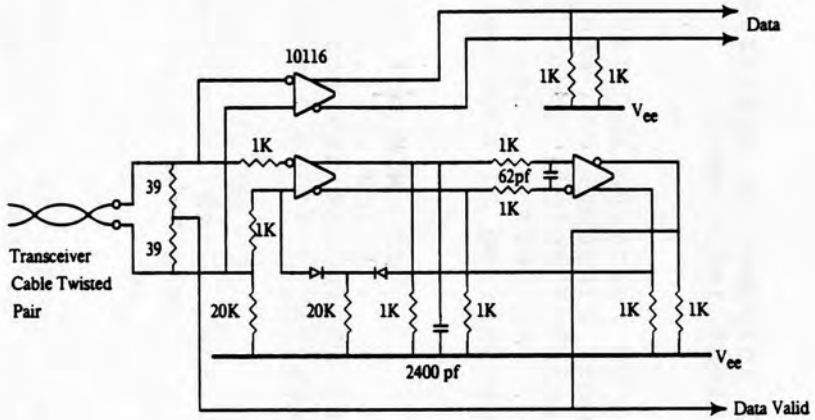


Figure D-2: Typical Transceiver Cable Receiver

Amazon
Exhibit 1005
Page 267

APPENDIX E: INTERFRAME RECOVERY

It is important that data link controller implementations be able to receive a frame that arrives immediately after another frame has been transmitted or received. Here, "immediately" means 9.6 μsec, based on the minimum interframe spacing provided as recovery time for the data link. (See 6.3.2.2) It is important that the data link controller be able to resume reception within that time.

Reception of multiple closely spaced incoming frames is a very desirable capability, and is crucial for stations which tend to communicate with several other stations concurrently. There is one important case in which a data link controller implementation cannot reasonably be expected to receive closely spaced incoming frames: if the station hardware (e.g. I/O bus) is intrinsically unable to accept the bits of a frame at the rate at which they arrive over the network, each incoming frame must be buffered to allow the station to accept it at some lower rate. Assuming limited buffering resources (e.g. a one frame buffer), reception of subsequent frames cannot occur until sufficient buffer space is available. This mode of operation is allowed for low performance stations.

Reception of an incoming frame immediately after transmission of an outgoing frame is a very important capability, even for stations which do not tend to communicate with several other stations concurrently. All stations, low performance to high performance, should allow reception of an incoming frame immediately after transmission of an outgoing frame.

Amazon
Exhibit 1005
Page 268

APPENDIX 1009E

APPENDIX 1009E
Early Citations to Ethernet

Source Documents

Digital Equipment Corporation, Intel Corportation, Xerox Corporation, (1980, September), The Ethernet : A Local Area Network : Data Link Layer and Physical Layer Specifications, Version 1.0, September 30, 1980. Also in *Computer Communication Review*, vol. 11, no. 3 (July 1981), 20–66.

Early Citations (in chronological order)

Medvitz, C. M. (1982). A concurrent simulation of ethernet.

Shepherd, D., & Corcoran, P.M. (1982). A gateway development system. Microprocess. Microsystems, 6, 21-24.

Oppen, D.C., & Dalal, Y.K. (1983). The clearinghouse: a decentralized agent for locating named objects in a distributed environment. ACM Trans. Inf. Syst., 1, 230-253.

Arato, A., Sarkadi-Nagy, I., Sulyán, J., & Telbisz, F. (1983). A local area network architecture tailored to laboratory environments. Interfaces in Computing, 1(3), 255-262.

Arthurs, E., & Stuck, B. (1984). A Modified Access Policy for ETHERNET TM Version 1.0 Data Link Layer. IEEE transactions on communications, 32(8), 977-979.

Cheriton, D. R., & Mann, T. P. (1989). Decentralizing a global naming service for improved performance and fault tolerance. ACM Transactions on Computer Systems (TOCS), 7(2), 147-183.

Deering, S. E., & Cheriton, D. R. (1990). Multicast routing in datagram internetworks and extended LANs. ACM Transactions on Computer Systems (TOCS), 8(2), 85-110.

APPENDIX 1011A

LIBRARY OF CONGRESS | CATALOG

Refine Your Search 1 of 1

BOOK

Foundations of multithreaded, parallel, and distributed programming

Full Record MARC Tags

Personal name
Andrews, Gregory R.

Main title
Foundations of multithreaded, parallel, and distributed programming / Gregory R. Andrews.

Published/Created
Reading, Mass. : Addison-Wesley, c2000.

Request this Item LC Find It Item Availability

LCCN Permalink	https://lccn.loc.gov/99045285
Description	xx, 664 p. : ill. ; 24 cm.
ISBN	0201357526
LC classification	QA76.58 .A57 2000
Portion of title	Multithreaded, parallel, and distributed programming
LC Subjects	Parallel programming (Computer science) Electronic data processing--Distributed processing. Threads (Computer programs)
Browse by shelf order	QA76.58
Notes	Includes bibliographical references and index.
LCCN	99045285
Dewey class no.	004/.3
Other system no.	(DLC) 99045285
Type of material	Book

Amazon
Exhibit 1005
Page 272

Item Availability

CALL NUMBER	QA76.58 .A57 2000 LANDOVR Copy 1
Request in	Jefferson or Adams Building Reading Rooms - STORED OFFSITE
Status	Not Charged

Amazon
Exhibit 1005
Page 273

LIBRARY OF CONGRESS | CATALOG

Refine Your Search 1 of 1

BOOK

Foundations of multithreaded, parallel, and distributed programming

Full Record MARC Tags

000		01230cam a22003014a 4500
001		11770186
005		20000620075910.0
008		990810s2000 maua b 001 0 eng
035	__	|**a** (DLC) 99045285
906	__	|**a** 7 |**b** cbc |**c** orignew |**d** 1 |**e** ocip |**f** 19 |**g** y-gencatlg
925	0_	|**a** acquire |**b** 1 shelf copy |**x** policy default
955	__	|**a** pc06 to ja00 08-10-99; jf09 to subj 8-10-99; jf11 to sl 08-11-99 ;jf12; ; to Children's Lit. 08-23-99; aa03 08-30-99; bk rec'd, to CIP ver. ps15 04-06-00; CIP ver. jg15 05-03-00; ja15 to BCCD 06-20-00
010	__	|**a** 99045285
020	__	|**a** 0201357526
040	__	|**a** DLC |**c** DLC |**d** DLC
042	__	|**a** pcc |**a** lcac
050	00	|**a** QA76.58 |**b** .A57 2000
082	00	|**a** 004/.3 |**2** 21
100	1_	|**a** Andrews, Gregory R.
245	10	|**a** Foundations of multithreaded, parallel, and distributed programming / |**c** Gregory R. Andrews.
246	30	|**a** Multithreaded, parallel, and distributed programming
260	__	|**a** Reading, Mass. : |**b** Addison-Wesley, |**c** c2000.
300	__	|**a** xx, 664 p. : |**b** ill. ; |**c** 24 cm.
504	__	|**a** Includes bibliographical references and index.
650	_0	|**a** Parallel programming (Computer science)
650	_0	|**a** Electronic data processing |**x** Distributed processing.
650	_0	|**a** Threads (Computer programs)
985	__	|**e** GAP

Request this Item LC Find It Item Availability

Item Availability

Amazon
Exhibit 1005
Page 274

CALL NUMBER	QA76.58 .A57 2000 LANDOVR Copy 1
Request in	Jefferson or Adams Building Reading Rooms - STORED OFFSITE
Status	Not Charged

Amazon
Exhibit 1005
Page 275

APPENDIX 1011B

Copyright United States Copyright Office

The Library has opened access to the reading rooms by appointment only. More. The Jefferson Building has reopened to visitors via timed, ticketed entry. More.

Help | Search | History | Titles | Start Over

Public Catalog

Copyright Catalog (1978 to present)
Search Request: Left Anchored Title = Foundations of multithreaded, parallel, and distributed programming
Search Results: Displaying 1 of 1 entries

previous next

Labeled View

Foundations of multithreaded, parallel, and distributed programming /...

Type of Work: Text
Registration Number / Date: TX0005144213 / 2000-02-22
Title: Foundations of multithreaded, parallel, and distributed programming / Gregory R. Andrews.
Imprint: Reading, MA : Addison-Wesley, c2000.
Description: 664 p.
Copyright Claimant: Addison Wesley Longman, Inc.
Date of Creation: 1998
Date of Publication: 2000-02-09
Names: Andrews, Gregory R.
Addison Wesley Longman, Inc.

previous next

Save, Print and Email (Help Page)

Select Download Format Full Record Format for Print/Save

Enter your email address: Email

Amazon
Exhibit 1005
Page 277

Amazon
Exhibit 1005
Page 278

APPENDIX 1011C

http://www.cs.arizona.edu/people/greg/mpdbook/ Go MAY SEP APR
223 captures
26 May 2000 - 24 Apr 2021
1999 2000 2001
About this capture



Foundations of Multithreaded, Parallel, and Distributed Programming

Gregory R. Andrews

"Concurrent hardware is more prevalent than ever, and concurrent programming is more relevant than ever." (from the Preface)

This Web site contains materials to support the use of *Foundations of Multithreaded, Parallel, and Distributed Programming*. I expect the contents of these pages to evolve. Please let me know if there is additional information that you would find useful. I would welcome contributions from readers and adopters of the book. The bottom of this page contains information on how to contact me.

The picture on the cover was taken near Flora, Norway by Philippe Colombi. Click on the image for a larger version.

Books

The Wayback Machine - https://web.archive.org/web/20000919040130/http://www.cs.arizona.edu:80/peop…



Foundations of Multithreaded, Parallel, and Distributed Programming

Gregory R. Andrews

"Concurrent hardware is more prevalent than ever, and concurrent programming is more relevant than ever." (from the Preface)

This Web site contains materials to support the use of *Foundations of Multithreaded, Parallel, and Distributed Programming*. I expect the contents of these pages to evolve. Please let me know if there is additional information that you would find useful. I would welcome contributions from readers and adopters of the book. The bottom of this page contains information on how to contact me.

The picture on the cover was taken near Flora, Norway by Philippe Colombi. Click on the image for a larger version.

Instructor's Manual

The following items are available to assist course instructors:

Postscript and PDF versions of the figures and some of the larger displays in the text.

The source code for the programs in case study sections.

A discussion of ways to use the book and corresponding course outlines.

The syllabus and versions of my lecture notes for Spring 2000, the first term I used the book in my class.

The homework assignments, projects, and midterms from my Spring 2000 and Spring 1999 courses. Answers are not available on the Web, but I would be happy to send them to instructors.

Amazon
Exhibit 1005
Page 281

Languages, Libraries, and Tools

Links to information on the languages, libraries, and tools described in the book, as well as a few general links.

Ordering Information

The book's ISBN number is 0-201-35752-6.

Addison Wesley has two Web pages for the book. Visit this one to request an examination copy. Visit this one for additional information.

Supplements

HTML versions of the book's preface, table of contents, glossary, and index. These can be searched using your Web browser.

The errata file contains corrections to the known errors. Most are simple formatting mistakes, but there are a couple major errors. If you find a mistake in the text, please send me email.

Greg Andrews
Phone: 520-621-4239
Fax: 520-621-4246
E-Mail: greg@cs.arizona.edu
Snail Mail: Dept. of Computer Science, The University of Arizona, Tucson, AZ 85721-0077

Last updated August 1, 2000

Amazon
Exhibit 1005
Page 282

APPENDIX 1011D

http://www.awlonline.com/aw/productList/0,1991,CS1105~CS,00.html Go
JUN SEP JAN
16
1999 2000 2001
6 captures
7 Apr 2000 - 17 Jun 2001
About this capture

Addison Wesley - Distributed Systems

For a complete list of all higher education and professional titles, press here.

- Distributed Systems: Concepts and Design, 3/e
  0-201-61918-0
  George Coulouris , Jean Dollimore , Tim Kindberg (2001)

- Foundations of Multithreaded, Parallel, and Distributed Programming
  0-201-35752-6
  Gregory Andrews (2000)

- Essential COM
  0-201-63446-5
  Don Box (1998)

http://www.awlonline.com/product/0,2627,0201357526,00.html Go
DEC JAN FEB
24 captures
19 Jan 2001 - 14 Apr 2021
2000 19 2001 2005
About this capture

Foundations of Multithreaded, Parallel, and Distributed Programming

Gregory R. Andrews, University of Arizona

Copyright 2000, 664 pp.
Paper format
ISBN 0-201-35752-6

book feedback | exam copy | mailing list

Course List:

Concurrent Programming/Parallel Programming
Operating Systems--Advanced
Introduction to Operating Systems (Grad Level)
Parallel Computing
Distributed Systems

Author's Homepage http://www.cs.arizona.edu/people/greg/mpdbook/

The Wayback Machine - https://web.archive.org/web/20010119113900/http://www.awlonline.com:80/product...

Foundations of Multithreaded, Parallel, and Distributed Programming

Gregory R. Andrews, University of Arizona

Copyright 2000, 664 pp.
Paper format
ISBN 0-201-35752-6

book feedback | exam copy | mailing list

Course List:

Concurrent Programming/Parallel Programming
Operating Systems--Advanced
Introduction to Operating Systems (Grad Level)
Parallel Computing
Distributed Systems

Author's Homepage

http://www.cs.arizona.edu/people/greg/mpdbook/

Author Bio

Gregory Andrews received a B.S. degree in Mathematics from Stanford University in 1969 and a Ph.D. degree in Computer Science from the University of Washington in 1974. From 1974-79 he was an Assistant Professor at Cornell University. Since 1979 he has been at The University of Arizona, where he is currently Professor of Computer Science. From 1986-93 he chaired the department; in 1986 he received a distinguished teaching award.

Greg has been on the editorial board of *Information Processing Letters* since 1979. He was the general chair of the Twelfth ACM Symposium on Operating Systems Principles in 1989 and has been on the program committees of numerous conferences. From 1988-92 he was on advisory committees for the computing directorate of the National Science Foundation. Since 1991 he has been on the Board of Directors of the Computing Research Association (CRA).

Greg's research interests include all aspects of concurrent programming. A long-term project has been the design and implementation of the SR programming language. Current work focuses on the development of Filaments, a software package that provides efficient fine-grain parallelism on a variety of parallel machines.

Amazon
Exhibit 1005
Page 286

Description

Foundations of Multithreaded, Parallel, and Distributed Programming covers, and then applies, the core concepts and techniques needed for an introductory course in this subject. Its emphasis is on the practice and application of parallel systems, using real-world examples throughout.

Greg Andrews teaches the fundamental concepts of multithreaded, parallel and distributed computing and relates them to the implementation and performance processes. He presents the appropriate breadth of topics and supports these discussions with an emphasis on performance.

Back Cover Copy

Greg Andrews teaches the fundamental concepts of multithreaded, parallel and distributed computing and relates them to the implementation and performance processes. He presents the appropriate breadth of topics and supports these discussions with an emphasis on performance.

Features

- Emphasizes how to solve problems, with correctness the primary concern and performance an important, but secondary, concern
- Includes a number of case studies which cover such topics as pthreads, MPI, and OpenMP libraries, as well as programming languages like Java, Ada, high performance Fortran, Linda, Occam, and SR
- Provides examples using Java syntax and discusses how Java deals with monitors, sockets, and remote method invocation
- Covers current programming techniques such as semaphores, locks, barriers, monitors, message passing, and remote invocation
- Concrete examples are executed with complete programs, both shared and distributed
- Sample applications include scientific computing and distributed systems

Features

- Emphasizes how to solve problems, with a primary concern for correctness and then performance.
- Includes a number of case studies which cover such topics as pthreads, MPI, and OpenMP libraries, as well as programming languages like Java, Ada, high performance Fortran, Linda, Occam, and SR.
- Provides examples using Java syntax and discusses how Java deals with monitors, sockets, and remote method invocation.
- Covers current programming techniques such as semaphores, locks, barriers, monitors, message passing, and remote invocation.
- Concrete examples are executed with complete programs, both shared and distributed.
- Sample applications include scientific computing and distributed systems.

Table of Contents

1. The Concurrent Computing Landscape.
The Essence of Concurrent Programming.
Hardware Architectures.
Single Processor Machines.
Shared-Memory Multiprocessors.
Multicomputers Networks.
Applications and Programming Styles.

Amazon
Exhibit 1005
Page 287

Iterative Parallelism: Matrix Multiplication.
Recursive Parallelism: Adaptive Quadrature.
Producers and Consumers: Unix Pipes.
Clients and Servers: Remote Files.
Peers: Distributed Matrix Multiplication.
Summary of Programming Notation.
Declarations Sequential Statements.
Concurrent Statements and Process Declarations.
Comments.

I. SHARED VARIABLE PROGRAMMING.

2. Processes and Synchronization.
States, Actions, Histories, and Properties.
Parallelization: Finding Patterns in Files.
Synchronization: The Maximum of an Array.
Atomic Actions and Await Statements.
Fine-Grained Atomicity.
Specifying Synchronization: The Await Statement.
Finding Patterns in a File Revisited.
A Synopsis of Axiomatic Semantics.
Formal Logical Systems.
A Programming Logic.
Semantics of Concurrent Execution.
Techniques for Avoiding Interference.
Disjoint Variables Weakened Assertions.
Global Invariants.
Synchronization.
An Example: The Array Copy Problem Revisited.
Safety and Liveness Properties.
Proving Safety Properties.
Scheduling Policies and Fairness.

3. Locks and Barriers.
The Critical Section Problem.
Critical Sections: Spin Locks.
Test and Set.
Test and Test and Set.
Implementing Await Statements.
Critical Sections: Fair Solutions.
The Tie-Breaker Algorithm.
The Ticket Algorithm.
The Bakery Algorithm.
Barrier Synchronization.
Shared Counter.
Flags and Coordinators.
Symmetric Barriers.
Data Parallel Algorithms.
Parallel Prefix Computations.
Operations on Linked Lists.
Grid Computations: Laplace's Equation.
Synchronous Multiprocessors.

Amazon
Exhibit 1005
Page 288

Parallel Computing with a Bag of Tasks.
Matrix Multiplication.
Adaptive Quadrature.

4. Semaphores.
Syntax and Semantics.
Basic Problems and Techniques.
Critical Sections: Mutual Exclusion.
Barriers: Signaling Events.
Producers and Consumers: Split Binary Semaphores.
Bounded Buffers: Resource Counting.
The Dining Philosophers.
Readers and Writers.
Readers/Writers as an Exclusion Problem.
Readers/Writers Using Condition Synchronization.
The Technique of Passing the Baton.
Alternative Scheduling Policies.
Resource Allocation and Scheduling.
Problem Definition and General Solution Pattern.
Shortest-Job-Next Allocation.
Case Study: Pthreads.
Thread Creation.
Semaphores.
Example: A Simple Producer and Consumer.

5. Monitors.
Syntax and Semantics.
Mutual Exclusion.
Condition Variables.
Signaling Disciplines.
Additional Operations on Condition Variables.
Synchronization Techniques.
Bounded Buffers: Basic Condition Synchronization.
Readers and Writers: Broadcast Signal.
Shortest-Job-Next Allocation: Priority Wait.
Interval Timer: Covering Conditions.
The Sleeping Barber: Rendezvous.
Disk Scheduling: Program Structures.
Scheduler as a Separate Monitor.
Scheduler as an Intermediary.
Scheduler as a Nested Monitor.
Case Study: Java.
The Threads Class.
Synchronized Methods.
Parallel Readers/Writers.
Exclusive Readers/Writers.
True Readers/Writers.
Case Study: Pthreads.
Locks and Condition Variables.
Example: Summing the Elements of a Matrix.

6. Implementations.
A Single-Processor Kernel.

Amazon
Exhibit 1005
Page 289

A Multiprocessor Kernel.
Implementing Semaphores in a Kernel.
Implementing Monitors in a Kernel.
Implementing Monitors Using Semaphores.

II. DISTRIBUTED PROGRAMMING.

7. Message Passing.
Asynchronous Message Passing.
Filters: A Sorting Network.
Clients and Servers.
Active Monitors.
A Self-Scheduling Disk Driver.
File Servers: Conversational Continuity.
Interacting Peers: Exchanging Values.
Synchronous Message Passing.
Case Study: CSP.
Communication Statements.
Guarded Communication.
Example: The Sieve of Eratosthenes.
Case Study: Linda.
Tuple Space and Process Interaction.
Example: Prime Numbers with a Bag of Tasks.
Case Study: MPI.
Basic Functions.
Global Communication and Synchronization.
Case Study: Java.
Networks and Sockets.
Example: A Remote File Reader.

8. RPC and Rendezvous.
Remote Procedure Call.
Synchronization in Modules.
A Time Server Caches in a Distributed File System.
A Sorting Network of Merge Filters.
Interacting Peers: Exchanging Values.
Rendezvous.
Input Statements.
Client/Server Examples.
A Sorting Network of Merge Filters.
Interacting Peers: Exchanging Values.
A Multiple Primitives Notation.
Invoking and Servicing Operations.
Examples.
Readers/Writers Revisited.
Encapsulated Access.
Replicated Files.
Case Study: Java.
Remote Method Invocation.
Example: A Remote Database.
Case Study: Ada.
Tasks.

Amazon
Exhibit 1005
Page 290

Rendezvous.
Protected Types.
Example: The Dining Philosophers.
Case Study: SR.
Resources and Globals.
Communication and Synchronization.
Example: Critical Section Simulation.

9. Paradigms for Process Interaction.
Managers/Workers (Distributed Bag of Tasks).
Sparse Matrix Multiplication.
Adaptive Quadrature Revisited.
Heartbeat Algorithms.
Image Processing: Region Labeling.
Cellular Automata: The Game of Life.
Pipeline Algorithms.
A Distributed Matrix Multiplication Pipeline.
Matrix Multiplication by Blocks.
Probe/Echo Algorithms.
Broadcast in a Network.
Computing the Topology of a Network.
Broadcast Algorithms.
Logical Clocks and Event Ordering.
Distributed Semaphores.
Token-Passing Algorithms.
Distributed Mutual Exclusion.
Termination Detection in a Ring.
Termination Detection in a Graph.
Replicated Servers.
Distributed Dining Philosophers.
Decentralized Dining Philosophers.

10. Implementations.
Asynchronous Message Passing.
Shared-Memory Kernel.
Distributed Kernel.
Synchronous Message Passing.
Direct Communication Using Asynchronous Messages.
Guarded Communication Using a Clearing House.
RPC and Rendezvous.
RPC in a Kernel.
Rendezvous Using Asynchronous Message Passing.
Multiple Primitives in a Kernel.
Distributed Shared Memory.
Implementation Overview.
Page Consistency Protocols.

III. PARALLEL PROGRAMMING:

Speedup and Efficiency.
Overheads and Challenges.

11. Scientific Computing.

Amazon
Exhibit 1005
Page 291

Grid Computations.
Laplace's Equation.
Sequential Jacobi Iteration.
Shared Variable Program.
Message Passing Program.
Red/Black Successive Over Relaxation.
Multigrid Methods.
Particle Computations.
The Gravitational #N-Body Problem.
Shared Variable Program.
Message Passing Programs.
Approximate Methods.
Matrix Computations.
Gaussian Elimination.
LU Decomposition.
Shared Variable Program.
Message Passing Program.

12. Languages, Compilers, Libraries, and Tools.
Parallel Programming Libraries.
Case Study: Pthreads.
Case Study: MPI.
Case Study: OpenMP.
Parallelizing Compilers.
Dependence Analysis.
Program Transformations.
Other Programming Models.
Imperative Languages.
Coordination Languages Data Parallel Languages.
Functional Languages.
Abstract Models.
Case Study: High Performance Fortran (HPF).
Parallel Programming Tools.
Performance Measurement and Visualization.
Metacomputers and Metacomputing.
Case Study: The Globus Toolkit.

top

Amazon
Exhibit 1005
Page 292

APPENDIX 1011E

Foundations of multithreaded, parallel, and distributed programming



Responsibility	Gregory R. Andrews.
Imprint	Reading, Mass. : Addison-Wesley, c2000.
Physical description	xx, 664 p. : ill. ; 24 cm.

At the library

SAL3 (off-campus storage)
No public access

Stacks	Request
QA76.58 .A57 2000	Available

More options

Find it at other libraries via WorldCat

Description

Creators/Contributors

Author/Creator
Andrews, Gregory R.

Contents/Summary

Bibliography
Includes bibliographical references and index.

Contents
- 1. The Concurrent Computing Landscape. The Essence of Concurrent Programming. Hardware Architectures. Single Processor Machines. Shared-Memory Multiprocessors. Multicomputers Networks. Applications and Programming Styles. Iterative Parallelism: Matrix Multiplication. Recursive Parallelism:

Find it at other libraries via WorldCat

(Limited preview)

Applications and Programming Styles. Iterative Parallelism: Matrix Multiplication. Recursive Parallelism: Adaptive Quadrature. Producers and Consumers: Unix Pipes. Clients and Servers: Remote Files. Peers: Distributed Matrix Multiplication. Summary of Programming Notation. Declarations Sequential Statements. Concurrent Statements and Process Declarations. Comments. I. SHARED VARIABLE PROGRAMMING.

- 2. Processes and Synchronization. States, Actions, Histories, and Properties. Parallelization: Finding Patterns in Files. Synchronization: The Maximum of an Array. Atomic Actions and Await Statements. Fine-Grained Atomicity. Specifying Synchronization: The Await Statement. Finding Patterns in a File Revisited. A Synopsis of Axiomatic Semantics. Formal Logical Systems. A Programming Logic. Semantics of Concurrent Execution. Techniques for Avoiding Interference. Disjoint Variables Weakened Assertions. Global Invariants. Synchronization. An Example: The Array Copy Problem Revisited. Safety and Liveness Properties. Proving Safety Properties. Scheduling Policies and Fairness.
- 3. Locks and Barriers. The Critical Section Problem. Critical Sections: Spin Locks. Test and Set. Test and Test and Set. Implementing Await Statements. Critical Sections: Fair Solutions. The Tie-Breaker Algorithm. The Ticket Algorithm. The Bakery Algorithm. Barrier Synchronization. Shared Counter. Flags and Coordinators. Symmetric Barriers. Data Parallel Algorithms. Parallel Prefix Computations. Operations on Linked Lists. Grid Computations: Laplace's Equation. Synchronous Multiprocessors. Parallel Computing with a Bag of Tasks. Matrix Multiplication. Adaptive Quadrature.
- 4. Semaphores. Syntax and Semantics. Basic Problems and Techniques. Critical Sections: Mutual Exclusion. Barriers: Signaling Events. Producers and Consumers: Split Binary Semaphores. Bounded Buffers: Resource Counting. The Dining Philosophers. Readers and Writers. Readers/Writers as an Exclusion Problem. Readers/Writers Using Condition Synchronization. The Technique of Passing the Baton. Alternative Scheduling Policies. Resource Allocation and Scheduling. Problem Definition and General Solution Pattern. Shortest-Job-Next Allocation. Case Study: Pthreads. Thread Creation. Semaphores. Example: A Simple Producer and Consumer.
- 5. Monitors. Syntax and Semantics. Mutual Exclusion. Condition Variables. Signaling Disciplines. Additional Operations on Condition Variables. Synchronization Techniques. Bounded Buffers: Basic Condition Synchronization. Readers and Writers: Broadcast Signal. Shortest-Job-Next Allocation: Priority Wait. Interval Timer: Covering Conditions. The Sleeping Barber: Rendezvous. Disk Scheduling: Program Structures. Scheduler as a Separate Monitor. Scheduler as an Intermediary. Scheduler as a Nested Monitor. Case Study: Java. The Threads Class. Synchronized Methods. Parallel Readers/Writers. Exclusive Readers/Writers. True Readers/Writers. Case Study: Pthreads. Locks and Condition Variables. Example: Summing the Elements of a Matrix.
- 6. Implementations. A Single-Processor Kernel. A Multiprocessor Kernel. Implementing Semaphores in a Kernel. Implementing Monitors in a Kernel. Implementing Monitors Using Semaphores. II. DISTRIBUTED PROGRAMMING.
- 7. Message Passing. Asynchronous Message Passing. Filters: A Sorting Network. Clients and Servers. Active Monitors. A Self-Scheduling Disk Driver. File Servers: Conversational Continuity. Interacting Peers: Exchanging Values. Synchronous Message Passing. Case Study: CSP. Communication Statements. Guarded Communication. Example: The Sieve of Eratosthenes. Case Study: Linda. Tuple Space and

Process Interaction. Example: Prime Numbers with a Bag of Tasks. Case Study: MPI. Basic Functions. Global Communication and Synchronization. Case Study: Java. Networks and Sockets. Example: A Remote File Reader.

- 8. RPC and Rendezvous. Remote Procedure Call. Synchronization in Modules. A Time Server Caches in a Distributed File System. A Sorting Network of Merge Filters. Interacting Peers: Exchanging Values. Rendezvous. Input Statements. Client/Server Examples. A Sorting Network of Merge Filters. Interacting Peers: Exchanging Values. A Multiple Primitives Notation. Invoking and Servicing Operations. Examples. Readers/Writers Revisited. Encapsulated Access. Replicated Files. Case Study: Java. Remote Method Invocation. Example: A Remote Database. Case Study: Ada. Tasks. Rendezvous. Protected Types. Example: The Dining Philosophers. Case Study: SR. Resources and Globals. Communication and Synchronization. Example: Critical Section Simulation.
- 9. Paradigms for Process Interaction. Managers/Workers (Distributed Bag of Tasks). Sparse Matrix Multiplication. Adaptive Quadrature Revisited. Heartbeat Algorithms. Image Processing: Region Labeling. Cellular Automata: The Game of Life. Pipeline Algorithms. A Distributed Matrix Multiplication Pipeline. Matrix Multiplication by Blocks. Probe/Echo Algorithms. Broadcast in a Network. Computing the Topology of a Network. Broadcast Algorithms. Logical Clocks and Event Ordering. Distributed Semaphores. Token-Passing Algorithms. Distributed Mutual Exclusion. Termination Detection in a Ring. Termination Detection in a Graph. Replicated Servers. Distributed Dining Philosophers. Decentralized Dining Philosophers.
- 10. Implementations. Asynchronous Message Passing. Shared-Memory Kernel. Distributed Kernel. Synchronous Message Passing. Direct Communication Using Asynchronous Messages. Guarded Communication Using a Clearing House. RPC and Rendezvous. RPC in a Kernel. Rendezvous Using Asynchronous Message Passing. Multiple Primitives in a Kernel. Distributed Shared Memory. Implementation Overview. Page Consistency Protocols. III. PARALLEL PROGRAMMING: Speedup and Efficiency. Overheads and Challenges.
- 11. Scientific Computing. Grid Computations. Laplace's Equation. Sequential Jacobi Iteration. Shared Variable Program. Message Passing Program. Red/Black Successive Over Relaxation. Multigrid Methods. Particle Computations. The Gravitational #N-Body Problem. Shared Variable Program. Message Passing Programs. Approximate Methods. Matrix Computations. Gaussian Elimination. LU Decomposition. Shared Variable Program. Message Passing Program.
- 12. Languages, Compilers, Libraries, and Tools. Parallel Programming Libraries. Case Study: Pthreads. Case Study: MPI. Case Study: OpenMP. Parallelizing Compilers. Dependence Analysis. Program Transformations. Other Programming Models. Imperative Languages. Coordination Languages Data Parallel Languages. Functional Languages. Abstract Models. Case Study: High Performance Fortran (HPF). Parallel Programming Tools. Performance Measurement and Visualization. Metacomputers and Metacomputing. Case Study: The Globus Toolkit. 0201357526T04062001.
- (source: Nielsen Book Data)

Summary

Foundations of Multithreaded, Parallel, and Distributed Programming covers, and then applies, the core concepts and techniques needed for an introductory course in this subject. Its emphasis is on the practice and application of parallel systems, using real-world examples throughout. Greg Andrews teaches the

Amazon
Exhibit 1005
Page 296

- (source: Nielsen Book Data)

Summary

Foundations of Multithreaded, Parallel, and Distributed Programming covers, and then applies, the core concepts and techniques needed for an introductory course in this subject. Its emphasis is on the practice and application of parallel systems, using real-world examples throughout. Greg Andrews teaches the fundamental concepts of multithreaded, parallel and distributed computing and relates them to the implementation and performance processes. He presents the appropriate breadth of topics and supports these discussions with an emphasis on performance.
(source: Nielsen Book Data)

Subjects

Subjects

Parallel programming (Computer science)

Electronic data processing > Distributed processing.

Threads (Computer programs)

Bibliographic information

Publication date

2000

Title variation

Multithreaded, parallel, and distributed programming

ISBN

0201357526

9780201357523

Librarian View

LEADER 09577pam a22003614a 4500			
001 a4287207			
003 SIRSI			
005 20211009050004.0			
008 990810s2000 maua b 001 0 eng			
010			a| 99045285
020			a| 0201357526
020			a| 9780201357523
035			a| (OCoLC-M)42296220
035			a| (OCoLC-I)275690480
040			a| DLC c| DLC d| NhCcYBP
042			a| pcc a| lcac
050	0	0	a| QA76.58 b| .A57 2000
082	0	0	a| 004/.3 2| 21
100	1		a| Andrews, Gregory R. =| ^A620964
245	1	0	a| Foundations of multithreaded, parallel, and distributed programming / c| Gregory R . Andrews.
246	3	0	a| Multithreaded, parallel, and distributed programming
260			a| Reading, Mass. : b| Addison-Wesley, c| c2000.
300			a| xx, 664 p. : b| ill. ; c| 24 cm.
504			a| Includes bibliographical references and index.
596			a| 31
650		0	a| Parallel programming (Computer science) =| ^A1046897
650		0	a| Electronic data processing x| Distributed processing. =| ^A1013917
650		0	a| Threads (Computer programs) =| ^A1255270

905 0 a| 1. The Concurrent Computing Landscape. The Essence of Concurrent Programming . Hardware Architectures. Single Processor Machines. Shared-Memory Multiprocessor s. Multicomputers Networks. Applications and Programming Styles. Iterative Parallelism: Matrix Multiplication. Recursive Parallelism: Adaptive Quadrature. Producers and Consumers: Unix Pipes. Clients and Servers: Remote Files. Peers: Distributed Matrix Multiplication. Summary of Programming Notation. Declarations Sequential Statements. Concurrent Statements and Process Declarations. Comments. I. SHARED VARIABLE PROGRAMMING. 2. Processes and Synchronization. States, Actions, Histories, and Properties. Parallelization: Finding Patterns in Files. Synchronization: The Maximum of an Array. Atomic Actions and Await Statements. Fine-Grained Atomicity. Specifying Synchronization: The Await Statement. Finding Patterns in a File Revisited. A Synopsis of Axiomatic Semantics. Formal Logical Systems. A Programming Logic. Semantics of Concurrent Execution. Techniques for Avoiding Interference. Disjoint Variables Weakened Assertions. Global Invariants. Synchronization. An Example: The Array Copy Problem Revisited. Safety and Liveness Properties. Proving Safety Properties. Scheduling Policies and Fairness. 3. Locks and Barriers. The Critical Section Problem. Critical Sections: Spin Locks. Test and Set. Test and Test and Set. Implementing Await Statements. Critical Sections: Fair Solutions. The Tie-Breaker Algorithm. The Ticket Algorithm. The Bakery Algorithm. Barrier Synchronization. Shared Counter. Flags and Coordinators. Symmetric Barriers. Data Parallel Algorithms. Parallel Prefix Computations. Operations on Linked Lists. Grid Computations: Laplace's Equation. Synchronous Multiprocessors. Parallel Computing with a Bag of Tasks. Matrix Multiplication. Adaptive Quadrature. 4. Semaphores. Syntax and Semantics. Basic Problems and Techniques. Critical Sections: Mutual Exclusion. Barriers: Signaling Events. Producers and Consumers: Split Binary Semaphores. Bounded Buffers: Resource Counting. The Dining Philosophers. Readers and Writers. Readers/Writers as an Exclusion Problem. Readers/Writers Using Condition Synchronization. The Technique of Passing the Baton. Alternative Scheduling Policies. Resource Allocation and Scheduling. Problem Definition and General Solution Pattern. Shortest-Job-Next Allocation. Case Study: Pthreads. Thread Creation. Semaphores. Example: A Simple Producer and Consumer. 5. Monitors. Syntax and Semantics. Mutual Exclusion. Condition Variables. Signaling Disciplines. Additional Operations on Condition Variables. Synchronization Techniques. Bounded Buffers: Basic Condition Synchronization. Readers and Writers: Broadcast Signal. Shortest-Job-Next Allocation: Priority Wait. Interval Timer: Covering Conditions. The Sleeping Barber: Rendezvous. Disk Scheduling: Program Structures. Scheduler as a Separate Monitor. Scheduler as an Intermediary. Scheduler as a Nested Monitor. Case Study: Java. The Threads Class. Synchronized Methods. Parallel Readers/Writers. Exclusive Readers/Writers. True Readers/Writers. Case Study: Pthreads. Locks and Condition Variables. Example: Summing the Elements of a Matrix. 6. Implementations. A Single-Processor Kernel. A Multiprocessor Kernel. Implementing Semaphores in a Kernel. Implementing Monitors in a Ker

Amazon
Exhibit 1005
Page 299

cessor Kernel. Implementing Semaphores in a Kernel. Implementing Monitors in a Kernel. Implementing Monitors Using Semaphores. II. DISTRIBUTED PROGRAMMING. 7. Message Passing. Asynchronous Message Passing. Filters: A Sorting Network. Clients and Servers. Active Monitors. A Self-Scheduling Disk Driver. File Servers: Conversational Continuity. Interacting Peers: Exchanging Values. Synchronous Message Passing. Case Study: CSP. Communication Statements. Guarded Communication. Example: The Sieve of Eratosthenes. Case Study: Linda. Tuple Space and Process Interaction. Example: Prime Numbers with a Bag of Tasks. Case Study: MPI. Basic Functions. Global Communication and Synchronization. Case Study: Java. Networks and Sockets. Example: A Remote File Reader. 8. RPC and Rendezvous. Remote Procedure Call. Synchronization in Modules. A Time Server Caches in a Distributed File System. A Sorting Network of Merge Filters. Interacting Peers: Exchanging Values. Rendezvous. Input Statements. Client/Server Examples. A Sorting Network of Merge Filters. Interacting Peers: Exchanging Values. A Multiple Primitives Notation. Invoking and Servicing Operations. Examples. Readers/Writers Revisited. Encapsulated Access. Replicated Files. Case Study: Java. Remote Method Invocation. Example: A Remote Database. Case Study: Ada. Tasks. Rendezvous. Protected Types. Example: The Dining Philosophers. Case Study: SR. Resources and Globals. Communication and Synchronization. Example: Critical Section Simulation. 9. Paradigms for Process Interaction. Managers/Workers (Distributed Bag of Tasks). Sparse Matrix Multiplication. Adaptive Quadrature Revisited. Heartbeat Algorithms. Image Processing: Region Labeling. Cellular Automata: The Game of Life. Pipeline Algorithms. A Distributed Matrix Multiplication Pipeline. Matrix Multiplication by Blocks. Probe/Echo Algorithms. Broadcast in a Network. Computing the Topology of a Network. Broadcast Algorithms. Logical Clocks and Event Ordering. Distributed Semaphores. Token-Passing Algorithms. Distributed Mutual Exclusion. Termination Detection in a Ring. Termination Detection in a Graph. Replicated Servers. Distributed Dining Philosophers. Decentralized Dining Philosophers. 10. Implementations. Asynchronous Message Passing. Shared-Memory Kernel. Distributed Kernel. Synchronous Message Passing. Direct Communication Using Asynchronous Messages. Guarded Communication Using a Clearing House. RPC and Rendezvous. RPC in a Kernel. Rendezvous Using Asynchronous Message Passing. Multiple Primitives in a Kernel. Distributed Shared Memory. Implementation Overview. Page Consistency Protocols. III. PARALLEL PROGRAMMING: Speedup and Efficiency. Overheads and Challenges. 11. Scientific Computing. Grid Computations. Laplace's Equation. Sequential Jacobi Iteration. Shared Variable Program. Message Passing Program. Red/Black Successive Over Relaxation. Multigrid Methods. Particle Computations. The Gravitational #N-Body Problem. Shared Variable Program. Message Passing Programs. Approximate Methods. Matrix Computations. Gaussian Elimination. LU Decomposition. Shared Variable Program. Message Passing Program. 12. Languages, Compilers, Libraries, and Tools. Parallel Programming Libraries. Case Study: Pthreads. Case Study: MPI. Case Study: OpenMP. Parallelizing Compilers. D

	ase Study: Pthreads. Case Study: MPI. Case Study: OpenMP. Parallelizing Compilers. D ependence Analysis. Program Transformations. Other Programming Models. Imperati ve Languages. Coordination Languages. Data Parallel Languages. Functional Languag es. Abstract Models. Case Study: High Performance Fortran (HPF). Parallel Programmi ng Tools. Performance Measurement and Visualization. Metacomputers and Metacom puting. Case Study: The Globus Toolkit. 0201357526T04062001. l| Nielsen x| 9780201 57523 x| 20160528
916	a| DATECATALOGED b| 20000210
920	b| Foundations of Multithreaded, Parallel, and Distributed Programming covers, and t hen applies, the core concepts and techniques needed for an introductory course in t his subject. Its emphasis is on the practice and application of parallel systems, using r eal-world examples throughout. Greg Andrews teaches the fundamental concepts o f multithreaded, parallel and distributed computing and relates them to the implemen tation and performance processes. He presents the appropriate breadth of topics and supports these discussions with an emphasis on performance. I| Nielsen x| 97802013 57523 x| 20160528
919	a| exclude from BorrowDirect b| HathiTrust ETAS
999	a| QA76.58 .A57 2000 w| LC d| 1 i| 36105028524887 d| 9/7/2012 e| 8/24/2012 l| STACKS m| SAL3 n| 31 q| S r| Y s| Y t| STKS-MONO u| 2/3/2000 z| TAX=8.25

APPENDIX 1011F

APPENDIX 1011F
Early Citations to Andrews

Source Document

Andrews, G. R., (2000), Foundations of Multithreaded, Parallel, and Distributed Programming, Addison-Wesley, **Ex. 1011**

Early Citations from 2000 (in chronological order)

Zhou, J., & Tai, K. C. (2000, April). Deadlock analysis of client/server programs. In *Proceedings 20th IEEE International Conference on Distributed Computing Systems* (pp. 484-491). IEEE.

Alba, E., Cotta, C., Dıaz, M., Soler, E., & Troya, J. M. (2000, May). MALLBA: Middleware for Communications in a Geographically Distributed Optimization System.

Lowenthal, D. K. (2000, June). Accurately selecting block size at runtime in pipelined parallel programs. *International Journal of Parallel Programming*, *28*(3), 245-274.

Karacali, B., Tai, K. C., & Vouk, M. A. (2000, June). Deadlock detection of EFSMs using simultaneous reachability analysis. In *Proceeding International Conference on Dependable Systems and Networks. DSN 2000* (pp. 315-324). IEEE.

Karaçali, B., & Tai, K. C. (2000, August). Model checking based on simultaneous reachability analysis. In *International SPIN Workshop on Model Checking of Software* (pp. 34-53). Springer, Berlin, Heidelberg.

Smith, M. L. (Fall 2000). *View-centric reasoning about parallel and distributed computation*. University of Central Florida.

APPENDIX 1011G

Foundations of Multithreaded, Parallel, and Distributed Programming



Gregory R. Andrews

Amazon
Exhibit 1005
Page 305

Foundations of Multithreaded, Parallel, and Distributed Programming

Amazon
Exhibit 1005
Page 306

Foundations of Multithreaded, Parallel, and Distributed Programming

Gregory R. Andrews
University of Arizona

Addison-Wesley

An imprint of Addison Wesley Longman, Inc.

Reading, Massachusetts • Menlo Park, California •
New York • Harlow, England • Don Mills, Ontario •
Sydney • Mexico City • Madrid • Amsterdam

Amazon
Exhibit 1005
Page 307

Acquisions Editor: *Maite Suarez-Rivas*
Associate Editor: *Jason Miranda*
Project Manager: *Trillium Project Management*
Production Assistant: *Tracy Treeful*
Copyeditor: *Stephanie Magean*
Proofreader: *Trillium Project Management*
Cover Designer: *Night & Day Design*
Composition: *Chiron, Inc.*

Library of Congress Cataloging-in-Publication Data
Andrews, Gregory R.
Foundations of multithreaded, parallel, and distributed programming / Gregory R. Andrews.
p. cm.
Includes bibliographical references.
ISBN 0-201-35752-6
1. Parallel programming (Computer science) 2. Electronic data processing—Distributed processing. 3. Threads (Computer programs) I. Title: Multithreaded, parallel, and distributed programming. II. Title.

QA76.58.A57 2000
004′.3--dc21

99-045285

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Addison-Wesley was aware of a trademark claim, the designations have been printed in initial caps or all caps.

The programs and applications presented in this book have been included for their instructional value. They have been tested with care but are not guaranteed for any purpose. The publisher and author do not offer any warranties or representations, nor do they accept any liabilities with respect to the programs or applications.

Access the latest information about Addison-Wesley titles from our World Wide Web site: www.awlonline.com

Copyright © 2000 by Addison Wesley Longman, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America.

1 2 3 4 5 6 7 8 9 10 CRS 0302010099

Amazon
Exhibit 1005
Page 308

Preface

Concurrent programming originated in 1962 with the invention of channels, which are independent device controllers that make it possible to have a CPU execute a new application program at the same time that I/O operations are being executed on behalf of other, suspended application programs. Hence, concurrent programming—the word *concurrent* means happening at the same time—was initially of concern to operating systems designers. In the late 1960s, hardware designers developed multiple processor machines. This presented not only a challenge for operating systems designers but also an opportunity that application programmers could exploit.

The first major concurrent programming challenge was to solve what is now called the critical section problem. It and related problems (dining philosophers, readers/writers, etc.) led to a spate of papers in the 1960s. To harness the challenge, people developed synchronization primitives such as semaphores and monitors to simplify the programmer's task. By the mid-1970s, people came to appreciate the necessity of using formal methods to help control the inherent complexity of concurrent programs.

Computer networks were introduced in the late 1970s and early 1980s. The Arpanet supported wide-area computing, and the Ethernet established local-area networks. Networks gave rise to distributed programming, which was a major topic of the 1980s and became even more important in the 1990s. The essence of distributed programming is that processes interact by means of message passing rather than by reading and writing shared variables.

Now, at the dawn of a new century, we have seen the emergence of massively parallel processing—in which tens, hundreds, or even thousands of processors are used to solve a single problem. We have also seen the emergence of

Amazon
Exhibit 1005
Page 309

client/server computing, the Internet, and the World Wide Web. Finally, we are beginning to see multiprocessor workstations and PCs. Concurrent hardware is more prevalent than ever, and concurrent programming is more relevant than ever.

This is my third book, another attempt to capture a part of the history of concurrent programming. My first book—*Concurrent Programming: Principles and Practice*, published in 1991—gives a broad, reference-level coverage of the period between 1960 and 1990. Because new problems, programming mechanisms, and formal methods were significant topics in those decades, the book focuses on them.

My second book—*The SR Programming Language: Concurrency in Practice*, published in 1993—summarizes my work with Ron Olsson in the late 1980s and early 1990s on a specific language that can be used to write concurrent programs for both shared- and distributed-memory machines. The SR book is pragmatic rather than formal, showing how to solve numerous problems in a single programming language.

This book is an outgrowth of my past and a reflection of what I think is important now and will be in the future. I have drawn heavily from material in the *Concurrent Programming* book, but I have completely rewritten every section that I have retained and have rewritten examples to use pseudo-C rather than SR. I have added new material throughout, especially on parallel scientific programming. I have also included case studies of the most important languages and software libraries, with complete sample programs. Finally, I have a new vision for the role of this book—in classrooms and in personal libraries.

A New Vision and Role

Parallel and distributed computing are today pervasive concepts. As usual in computer science, advances have been led by hardware designers, who keep building bigger, faster, more powerful computers and communication networks. For the most part, they are succeeding—witness the stock market!

New computers and networks create new challenges and opportunities, and for once software designers are no longer all that far behind. Consider Web browsers, Internet commerce, Pthreads, Java, MPI, and so on—and again, witness the stock market! These software products are specifically designed to take advantage of concurrency in hardware and applications. In short, much of the computing world is now concurrent!

Aspects of concurrent computing—whether multithreaded, parallel, or distributed—are now covered in almost every advanced course in computer science.

Amazon
Exhibit 1005
Page 310

Reflecting the history of the topic, operating systems courses lead the way—covering topics like multithreading, communication protocols, and distributed file systems. Architecture courses cover multiprocessors and networks. Compilers courses cover compilation issues for parallel machines. Theory courses cover models for parallel computing. Algorithms courses cover parallel algorithms. Database courses cover locking and distributed databases. Graphics courses make use of parallelism for rendering and ray tracing. The list goes on. In addition, concurrent computing has become a fundamental tool in a wide range of science and engineering disciplines.

The main purpose of this book—as reflected in the title—is to lay the foundations for programming multithreaded, parallel, and distributed computations. The specific goal is to provide a text that can be used for, or to create, an advanced undergraduate/beginning graduate course. Whenever a topic in computing has become pervasive, as concurrency surely has, we have added foundation courses to provide students with basic knowledge of a topic. Similarly, whenever a topic has become well understood, as concurrency now is, we have migrated the topic to the core curriculum.

I have tried to cover those aspects of parallel and distributed computing that I think every computer science student should know. This includes basic principles, programming techniques, major applications, implementations, and performance issues. In addition, I have included case studies of many of the most important languages and software libraries: Pthreads (three chapters), Java (three chapters), CSP, Linda, MPI (two chapters), Ada, SR, and OpenMP. Each case study describes relevant parts of the language or library and then presents a complete sample program. (Sources for the programs are available on this book's Web site. In addition, I summarize several additional languages, models, and tools for parallel scientific computing in Chapter 12.

On the other hand, no single book can cover everything—and still be affordable—so students and instructors may wish to augment this text with others. The Historical Notes and the References at the end of each chapter describe additional material and provide pointers for further study. This book's Web pages provide further information and links to relevant material.

Content Overview

This book contains 12 chapters. Chapter 1 summarizes basic concepts of concurrency, hardware, and applications. Then it examines five representative applications: matrix multiplication, adaptive quadrature, Unix pipes, file systems, and distributed matrix multiplication. Each application is simple yet useful; together

Amazon
Exhibit 1005
Page 311

they illustrate the range of issues and five recurring programming styles. The last section of Chapter 1 summarizes the programming notation that is used in the text.

The remaining chapters are organized into three parts. Part 1 describes concurrent programming mechanisms that use shared variables, and hence that are directly suitable for shared-memory machines. Chapter 2 introduces fundamental concepts of processes and synchronization; the chapter uses a series of examples to illustrate the key points and ends with a discussion of the formal semantics of concurrency. Understanding the semantic concepts will help you understand parts of later chapters, but it should also be sufficient to refer back to them when necessary. Chapter 3 shows how to implement and use locks and barriers; it also describes data parallel algorithms and a parallel programming technique called a bag of tasks. Chapter 4 describes semaphores and gives numerous examples of how to use them. Semaphores were the first high-level concurrent programming mechanism and remain one of the most important. Chapter 5 covers monitors in detail. Monitors were introduced in a seminal 1974 paper, somewhat lost favor in the 1980s and early 1990s, but have regained importance with the Java language. Finally, Chapter 6 describes how to implement processes, semaphores, and monitors on both uniprocessors and multiprocessors.

Part 2 covers distributed programming, in which processes communicate and synchronize by means of messages. Chapter 7 describes message passing using `send` and `receive` primitives. It shows how to use these primitives to program filters (which have one-way communication), clients and servers (which have two-way communication), and interacting peers (which have back and forth communication). Chapter 8 examines two additional communication primitives: remote procedure call (RPC) and rendezvous. With these, a client process initiates a communication by issuing a `call`—which is implicitly a `send` followed by a `receive`; the communication is serviced either by a new process (RPC) or by a rendezvous with an existing process. Chapter 9 describes several paradigms for process interaction in distributed programs. These include three that are commonly used in parallel computations—manager/workers, heartbeat, and pipeline—as well as four that arise in distributed systems—probe/echo, broadcast, token passing, and replicated servers. Finally, Chapter 10 describes how to implement message passing, RPC, and rendezvous. That chapter also shows how to implement what is called a distributed shared memory, which supports a shared-memory programming model in a distributed environment.

Part 3 covers parallel programming, especially for high-performance scientific computations. (Many other kinds of parallel computations are described in earlier chapters and in the exercises of several chapters.) The goal of a parallel program is speedup—namely, to solve a problem faster by using multiple

Amazon
Exhibit 1005
Page 312

processors. Parallel programs are written using shared variables or message passing; hence they employ the techniques described in Parts 1 and 2. Chapter 11 examines the three major classes of scientific computing applications: grid, particle, and matrix computations. These arise in simulating (modeling) physical and biological systems; matrix computations are also used for such things as economic forecasting. Chapter 12 surveys the most important tools that are used to write parallel scientific computations: libraries (Pthreads, MPI, and OpenMP), parallelizing compilers, languages and models, and higher-level tools such as metacomputations.

The end of each chapter provides historical notes, references, and an extensive set of exercises. The historical notes summarize the origin and evolution of each topic and how the topics relate to each other. The notes also describe the papers and books listed in the reference section. The exercises explore the topics covered in each chapter and also introduce additional applications.

Classroom Use

I use this book each spring at the University of Arizona for a class of about 60 students. About half the students are undergraduates and half are graduates. Most are computer science majors; the class is an elective for them, but most majors end up taking it. The class is also taken each year by a few Ph.D. students from other departments who have an interest in computational science and parallel computing. Most of the undergraduates are simultaneously taking our undergraduate operating systems course; the CS graduate students have all had such a course and many have also taken our advanced operating systems course.

Our operating systems classes, like those elsewhere, examine the synchronization *needs* of operating systems, and explore how processes, synchronization, and other OS topics are implemented, mostly on single processor machines. My class shows how to *use* concurrency as a general programming tool in a broad range of applications. I look at programming techniques, higher-level concepts, parallel computing, and distributed computing. In a sense, my class is to an OS class somewhat like what a comparative programming languages class is to a compilers class.

In the first half of my class, I cover Chapter 2 in some detail but then go fairly rapidly through the other chapters in Part I, emphasizing topics not covered in the OS classes: the Test-and-Test-and-Set protocol, the Ticket Algorithm, barriers, passing the baton for semaphores, some of the monitor programming techniques, and the multiprocessor kernel. In addition, students are introduced to the Pthreads library and Java's threads and synchronized methods in homework

Amazon
Exhibit 1005
Page 313

assignments. I also include a parallel computing project after my lectures on barriers (drawing material from Chapter 11).

In the second half of my class I cover much of Part 2 of the book on distributed programming. We look at message passing, and how it is used to program both distributed systems and distributed parallel computations. Then we look at RPC and rendezvous, their realization in Java and Ada, and distributed systems applications. Finally, we look at each of the paradigms in Chapter 9, again using applications from parallel and distributed systems to motivate and illustrate them. Students are introduced to the MPI library in a homework assignment and again use Java.

During the term I give four homework assignments, two in-class exams, a parallel computing project, and a final project. (See the book's Web site for samples.) Each homework assignment consists of written problems from the exercises, as well as one or two programming problems. Graduate students are required to do all problems; undergraduates do two-thirds of the problems (of their choice). Exams are administered similarly: graduates do all problems, undergraduates choose a subset of the problems to answer. At Arizona we use SR in the earliest programming assignments, and students have the option of using SR for many of the later assignments, but they are encouraged to use Pthreads, Java, and MPI.

The parallel programming project addresses one of the problems in Chapter 11 (often a grid computation). Students write programs and do timing experiments on a small shared-memory multiprocessor. Graduate students implement harder algorithms and are also required to write a fairly detailed report on their experiments. The final project is a paper or programming project on some aspect of distributed computing. Students get to choose what they want to do, subject to my approval. Most students do implementation projects; many work in pairs. The students produce an amazing variety of interesting systems, and most include a graphical user interface.

Even though the students in my class have a wide range of backgrounds, almost all give the class an excellent evaluation. Undergraduates frequently remark on how well it meshes with their OS class; they like seeing concurrency from another perspective, learning about multiprocessors, and seeing a broad range of applications and tools. Graduate students remark on how the class ties together a variety of things they had heard something about, and also introduces them to the breadth of concurrent programming; many of them would, however, like to see more coverage of parallel computing. Eventually we hope to be able to offer separate classes for undergraduates and graduates. I would not change much for the undergraduate-only class, but I would cover slightly fewer topics in somewhat more depth. For the graduate version I would spend less time on

Amazon
Exhibit 1005
Page 314

material they were familiar with (Part 1) and more time on parallel applications and parallel computing tools (Part 3).

This book is ideally suited for classes that cover the range of concurrent programming mechanisms, tools, and applications. Almost all students will make use of *some* of what is covered here, but not many will do just parallel computing, just distributed systems, or just programming in Java. However, the book can also be used as a supplement to classes with a more specialized focus. For example, a parallel computing class might use this book together with one that covers parallel algorithms and applications.

Online Resources

The "home page" for the book is located at

```
http://www.cs.arizona.edu/people/greg/mpdbook
```

The Web site contains the source for the larger programs in the text, links to software and information on the languages and libraries described in the case studies, and a variety of other useful links. It also contains extensive information from my Arizona class, including the syllabus, lecture notes, and copies of homework assignments, projects, and exams. Information on this book is also available at

```
http://www.awlonline.com/cs
```

Despite my best efforts, this book no doubt contains errors, so there will be (too soon I'm sure) an errata page. Contact your Addison Wesley representative for information on obtaining a copy of the Instructor's Manual for this book. Other material will no doubt be added in the future.

Acknowledgments

I received many useful suggestions from the reviewers of earlier drafts of this book. Martti Tienari and his students at the University of Helsinki found several subtle errors. My most recent Ph.D. students, Vince Freeh and Dave Lowenthal, provided comments on new chapters and helped debug my thinking, if not my programs, over the past several years. The students in my spring 1999 class served as guinea pigs and found several embarrassing errors. Andy Bernat also provided feedback from his spring 1999 class at the University of Texas, El Paso. Thank you to the following reviewers for their invaluable feedback: Jaspal Subhlok (Carnegie Mellon University), Boleslaw Szymanski (Rensselaer

Amazon
Exhibit 1005
Page 315

Polytechnic Institute), G.S. Stiles (Utah State University), Narsingh Deo (University of Central Florida), Janet Hartman (Illinois State University), Nan C. Schaller (Rochester Institute of Technology), and Mark Fienup (University of Northern Iowa).

The National Science Foundation has supported my research for many years, most recently through grants CCR-9415303 and ACR-9720738. A research infrastructure grant from the NSF (CDA-9500991), helped support the equipment used to prepare the book and test the programs.

Most importantly, I want to thank my wife, Mary, who once again put up with the thousands of hours I spent working on this book—despite having vowed "never again" after finishing the SR book.

Greg Andrews
Tucson, Arizona

Amazon
Exhibit 1005
Page 316

Table of Contents

Preface v

Chapter 1: The Concurrent Computing Landscape 1

1.1 The Essence of Concurrent Programming 2
1.2 Hardware Architectures 4
1.2.1 Processors and Caches 4
1.2.2 Shared-Memory Multiprocessors 6
1.2.3 Distributed-Memory Multicomputers and Networks 8
1.3 Applications and Programming Styles 10
1.4 Iterative Parallelism: Matrix Multiplication 13
1.5 Recursive Parallelism: Adaptive Quadrature 17
1.6 Producers and Consumers: Unix Pipes 19
1.7 Clients and Servers: File Systems 21
1.8 Peers: Distributed Matrix Multiplication 23
1.9 Summary of Programming Notation 26
1.9.1 Declarations 26
1.9.2 Sequential Statements 27
1.9.3 Concurrent Statements, Processes, and Procedures 29
1.9.4 Comments 31
Historical Notes 31
References 33
Exercises 34

Amazon
Exhibit 1005
Page 317

Part 1: Shared-Variable Programming 39

Chapter 2: Processes and Synchronization 41

2.1 States, Actions, Histories, and Properties 42
2.2 Parallelization: Finding Patterns in a File 44
2.3 Synchronization: The Maximum of an Array 48
2.4 Atomic Actions and Await Statements 51
2.4.1 Fine-Grained Atomicity 51
2.4.2 Specifying Synchronization: The Await Statement 54
2.5 Producer/Consumer Synchronization 56
2.6 A Synopsis of Axiomatic Semantics 57
2.6.1 Formal Logical Systems 58
2.6.2 A Programming Logic 59
2.6.3 Semantics of Concurrent Execution 62
2.7 Techniques for Avoiding Interference 65
2.7.1 Disjoint Variables 65
2.7.2 Weakened Assertions 66
2.7.3 Global Invariants 68
2.7.4 Synchronization 69
2.7.5 An Example: The Array Copy Problem Revisited 70
2.8 Safety and Liveness Properties 72
2.8.1 Proving Safety Properties 73
2.8.2 Scheduling Policies and Fairness 74
Historical Notes 77
References 80
Exercises 81

Chapter 3: Locks and Barriers 93

3.1 The Critical Section Problem 94
3.2 Critical Sections: Spin Locks 97
3.2.1 Test and Set 98
3.2.2 Test and Test and Set 100
3.2.3 Implementing Await Statements 101
3.3 Critical Sections: Fair Solutions 104
3.3.1 The Tie-Breaker Algorithm 104
3.3.2 The Ticket Algorithm 108
3.3.3 The Bakery Algorithm 111
3.4 Barrier Synchronization 115
3.4.1 Shared Counter 116
3.4.2 Flags and Coordinators 117

Amazon
Exhibit 1005
Page 318

3.4.3 Symmetric Barriers 120
3.5 Data Parallel Algorithms 124
3.5.1 Parallel Prefix Computations 124
3.5.2 Operations on Linked Lists 127
3.5.3 Grid Computations: Jacobi Iteration 129
3.5.4 Synchronous Multiprocessors 131
3.6 Parallel Computing with a Bag of Tasks 132
3.6.1 Matrix Multiplication 133
3.6.2 Adaptive Quadrature 134
Historical Notes 135
References 139
Exercises 141

Chapter 4: Semaphores 153
4.1 Syntax and Semantics 154
4.2 Basic Problems and Techniques 156
4.2.1 Critical Sections: Mutual Exclusion 156
4.2.2 Barriers: Signaling Events 156
4.2.3 Producers and Consumers: Split Binary Semaphores 158
4.2.4 Bounded Buffers: Resource Counting 160
4.3 The Dining Philosophers 164
4.4 Readers and Writers 166
4.4.1 Readers/Writers as an Exclusion Problem 167
4.4.2 Readers/Writers Using Condition Synchronization 169
4.4.3 The Technique of Passing the Baton 171
4.4.4 Alternative Scheduling Policies 175
4.5 Resource Allocation and Scheduling 178
4.5.1 Problem Definition and General Solution Pattern 178
4.5.2 Shortest-Job-Next Allocation 180
4.6 Case Study: Pthreads 184
4.6.1 Thread Creation 185
4.6.2 Semaphores 186
4.6.3 Example: A Simple Producer and Consumer 186
Historical Notes 188
References 190
Exercises 191

Chapter 5: Monitors 203
5.1 Syntax and Semantics 204
5.1.1 Mutual Exclusion 206

Amazon
Exhibit 1005
Page 319

5.1.2 Condition Variables 207
5.1.3 Signaling Disciplines 208
5.1.4 Additional Operations on Condition Variables 212
5.2 Synchronization Techniques 213
5.2.1 Bounded Buffers: Basic Condition Synchronization 213
5.2.2 Readers and Writers: Broadcast Signal 215
5.2.3 Shortest-Job-Next Allocation: Priority Wait 217
5.2.4 Interval Timer: Covering Conditions 218
5.2.5 The Sleeping Barber: Rendezvous 221
5.3 Disk Scheduling: Program Structures 224
5.3.1 Using a Separate Monitor 228
5.3.2 Using an Intermediary 230
5.3.3 Using a Nested Monitor 235
5.4 Case Study: Java 237
5.4.1 The Threads Class 238
5.4.2 Synchronized Methods 239
5.4.3 Parallel Readers/Writers 241
5.4.4 Exclusive Readers/Writers 243
5.4.5 True Readers/Writers 245
5.5 Case Study: Pthreads 246
5.5.1 Locks and Condition Variables 246
5.5.2 Example: Summing the Elements of a Matrix 248
Historical Notes 250
References 253
Exercises 255

Chapter 6: Implementations 265
6.1 A Single-Processor Kernel 266
6.2 A Multiprocessor Kernel 270
6.3 Implementing Semaphores in a Kernel 276
6.4 Implementing Monitors in a Kernel 279
6.5 Implementing Monitors Using Semaphores 283
Historical Notes 284
References 286
Exercises 287

Part 2: Distributed Programming 291

Chapter 7: Message Passing 295
7.1 Asynchronous Message Passing 296

Amazon
Exhibit 1005
Page 320

7.2 Filters: A Sorting Network 298
7.3 Clients and Servers 302
7.3.1 Active Monitors 302
7.3.2 A Self-Scheduling Disk Server 308
7.3.3 File Servers: Conversational Continuity 311
7.4 Interacting Peers: Exchanging Values 314
7.5 Synchronous Message Passing 318
7.6 Case Study: CSP 320
7.6.1 Communication Statements 321
7.6.2 Guarded Communication 323
7.6.3 Example: The Sieve of Eratosthenes 326
7.6.4 Occam and Modern CSP 328
7.7 Case Study: Linda 334
7.7.1 Tuple Space and Process Interaction 334
7.7.2 Example: Prime Numbers with a Bag of Tasks 337
7.8 Case Study: MPI 340
7.8.1 Basic Functions 341
7.8.2 Global Communication and Synchronization 343
7.9 Case Study: Java 344
7.9.1 Networks and Sockets 344
7.9.2 Example: A Remote File Reader 345
Historical Notes 348
References 351
Exercises 353

Chapter 8: RPC and Rendezvous 361
8.1 Remote Procedure Call 362
8.1.1 Synchronization in Modules 364
8.1.2 A Time Server 365
8.1.3 Caches in a Distributed File System 367
8.1.4 A Sorting Network of Merge Filters 370
8.1.5 Interacting Peers: Exchanging Values 371
8.2 Rendezvous 373
8.2.1 Input Statements 374
8.2.2 Client/Server Examples 376
8.2.3 A Sorting Network of Merge Filters 379
8.2.4 Interacting Peers: Exchanging Values 381
8.3 A Multiple Primitives Notation 382
8.3.1 Invoking and Servicing Operations 382
8.3.2 Examples 384

Amazon
Exhibit 1005
Page 321

8.4 Readers/Writers Revisited 386
8.4.1 Encapsulated Access 387
8.4.2 Replicated Files 389
8.5 Case Study: Java 393
8.5.1 Remote Method Invocation 393
8.5.2 Example: A Remote Database 395
8.6 Case Study: Ada 397
8.6.1 Tasks 398
8.6.2 Rendezvous 399
8.6.3 Protected Types 401
8.6.4 Example: The Dining Philosophers 403
8.7 Case Study: SR 406
8.7.1 Resources and Globals 406
8.7.2 Communication and Synchronization 408
8.7.3 Example: Critical Section Simulation 409
Historical Notes 411
References 415
Exercises 416

Chapter 9: Paradigms for Process Interaction 423
9.1 Manager/Workers (Distributed Bag of Tasks) 424
9.1.1 Sparse Matrix Multiplication 424
9.1.2 Adaptive Quadrature Revisited 428
9.2 Heartbeat Algorithms 430
9.2.1 Image Processing: Region Labeling 432
9.2.2 Cellular Automata: The Game of Life 435
9.3 Pipeline Algorithms 437
9.3.1 A Distributed Matrix Multiplication Pipeline 438
9.3.2 Matrix Multiplication by Blocks 441
9.4 Probe/Echo Algorithms 444
9.4.1 Broadcast in a Network 444
9.4.2 Computing the Topology of a Network 448
9.5 Broadcast Algorithms 451
9.5.1 Logical Clocks and Event Ordering 452
9.5.2 Distributed Semaphores 454
9.6 Token-Passing Algorithms 457
9.6.1 Distributed Mutual Exclusion 457
9.6.2 Termination Detection in a Ring 460
9.6.3 Termination Detection in a Graph 462
9.7 Replicated Servers 465

Amazon
Exhibit 1005
Page 322

9.7.1 Distributed Dining Philosophers 466
9.7.2 Decentralized Dining Philosophers 467
Historical Notes 471
References 474
Exercises 477

Chapter 10: Implementations 487
10.1 Asynchronous Message Passing 488
10.1.1 Shared-Memory Kernel 488
10.1.2 Distributed Kernel 491
10.2 Synchronous Message Passing 496
10.2.1 Direct Communication Using Asynchronous Messages 497
10.2.2 Guarded Communication Using a Clearinghouse 498
10.3 RPC and Rendezvous 504
10.3.1 RPC in a Kernel 504
10.3.2 Rendezvous Using Asynchronous Message Passing 507
10.3.3 Multiple Primitives in a Kernel 509
10.4 Distributed Shared Memory 515
10.4.1 Implementation Overview 516
10.4.2 Page Consistency Protocols 518
Historical Notes 520
References 521
Exercises 522

Part 3: Parallel Programming 527

Chapter 11: Scientific Computing 533
11.1 Grid Computations 534
11.1.1 Laplace's Equation 534
11.1.2 Sequential Jacobi Iteration 535
11.1.3 Jacobi Iteration Using Shared Variables 540
11.1.4 Jacobi Iteration Using Message Passing 541
11.1.5 Red/Black Successive Over-Relaxation (SOR) 546
11.1.6 Multigrid Methods 549
11.2 Particle Computations 553
11.2.1 The Gravitational N-Body Problem 554
11.2.2 Shared-Variable Program 555
11.2.3 Message-Passing Programs 559

Amazon
Exhibit 1005
Page 323

11.2.4 Approximate Methods 569
11.3 Matrix Computations 573
11.3.1 Gaussian Elimination 573
11.3.2 LU Decomposition 575
11.3.3 Shared-Variable Program 576
11.3.4 Message-Passing Program 581
Historical Notes 583
References 584
Exercises 585

Chapter 12: Languages, Compilers, Libraries, and Tools 591
12.1 Parallel Programming Libraries 592
12.1.1 Case Study: Pthreads 593
12.1.2 Case Study: MPI 593
12.1.3 Case Study: OpenMP 595
12.2 Parallelizing Compilers 603
12.2.1 Dependence Analysis 604
12.2.2 Program Transformations 607
12.3 Languages and Models 614
12.3.1 Imperative Languages 616
12.3.2 Coordination Languages 619
12.3.3 Data Parallel Languages 620
12.3.4 Functional Languages 623
12.3.5 Abstract Models 626
12.3.6 Case Study: High-Performance Fortran (HPF) 629
12.4 Parallel Programming Tools 633
12.4.1 Performance Measurement and Visualization 633
12.4.2 Metacomputers and Metacomputing 634
12.4.3 Case Study: The Globus Toolkit 636
Historical Notes 638
References 642
Exercises 644

Glossary 647

Index 657

Amazon
Exhibit 1005
Page 324

569
573
573
575
576
581
583
584
585

591
592
593
593
595
603
604
607
614
616
619
620
623
626
629
633
633
634
636
638
642
644

647

657

1

The Concurrent Computing Landscape

Imagine the following scenario:

> Several cars want to drive from point A to point B. They can compete for space on the same road and end up either following each other or competing for positions (and having accidents!). Or they could drive in parallel lanes, thus arriving at about the same time without getting in each other's way. Or they could travel different routes, using separate roads.

This scenario captures the essence of concurrent computing: There are multiple tasks that need to be done (cars moving). Each can execute one at a time on a single processor (road), execute in parallel on multiple processors (lanes in a road), or execute on distributed processors (separate roads). However, tasks often need to synchronize to avoid collisions or to stop at traffic lights or stop signs.

This book is an "atlas" of concurrent computing. It examines the kinds of cars (processes), the routes they might wish to travel (applications), the patterns of roads (hardware), and the rules of the road (communication and synchronization). So load up the tank and get started.

In this chapter we describe the legend of the concurrent programming map. Section 1.1 introduces fundamental concepts. Sections 1.2 and 1.3 describe the kinds of hardware and applications that make concurrent programming both interesting and challenging. Sections 1.4 through 1.8 describe and illustrate five recurring programming styles: iterative parallelism, recursive parallelism, producers and consumers, clients and servers, and interacting peers. The last section defines the programming notation that is used in the examples and the remainder of the book.

Later chapters cover applications and programming techniques in detail. The chapters are organized into three parts—shared-variable programming,

Amazon
Exhibit 1005
Page 325

distributed (message-based) programming, and parallel programming. The introduction to each part and each chapter gives a road map, summarizing where we have been and where we are going.

1.1 The Essence of Concurrent Programming

A concurrent program contains two or more processes that work together to perform a task. Each process is a sequential program—namely, a sequence of statements that are executed one after another. Whereas a sequential program has a *single thread of control*, a concurrent program has *multiple threads of control*.

The processes in a concurrent program work together by *communicating* with each other. Communication is programmed using shared variables or message passing. When shared variables are used, one process writes into a variable that is read by another. When message passing is used, one process sends a message that is received by another.

Whatever the form of communication, processes also need to *synchronize* with each other. There are two basic kinds of synchronization: mutual exclusion and condition synchronization. *Mutual exclusion* is the problem of ensuring that *critical sections* of statements do not execute at the same time. *Condition synchronization* is the problem of delaying a process until a given condition is true. As an example, communication between a producer process and a consumer process is often implemented using a shared memory buffer. The producer writes into the buffer; the consumer reads from the buffer. Mutual exclusion is required to ensure that the producer and consumer do not access the buffer at the same time, and hence that a partially written message is not read prematurely. Condition synchronization is used to ensure that a message is not read by the consumer until after it has been written by the producer.

The history of concurrent programming has followed the same stages as other experimental areas of computer science. The topic arose due to opportunities presented by hardware developments and has evolved in response to technological changes. Over time, the initial, *ad hoc* approaches have coalesced into a collection of core principles and general programming techniques.

Concurrent programming originated in the 1960s within the context of operating systems. The motivation was the invention of hardware units called *channels* or *device controllers*. These operate independently of a controlling processor and allow an I/O operation to be carried out concurrently with continued execution of program instructions by the central processor. A channel communicates with the central processor by means of an *interrupt*—a hardware signal that says "stop what you are doing and start executing a different sequence of instructions."

Amazon
Exhibit 1005
Page 326

The programming challenge—indeed the intellectual challenge—that resulted from the introduction of channels was that now parts of a program could execute in an unpredictable order. Hence, if one part of a program is updating the value of a variable, an interrupt might occur and lead to another part of the program trying to change the value of the variable. This specific problem came to be known as the *critical section problem*, which we cover in detail in Chapter 3.

Shortly after the introduction of channels, hardware designers conceived of multiprocessor machines. For a couple decades these were too expensive to be widely used, but now all large machines have multiple processors. Indeed, the largest machines have hundreds of processors and are often called *massively parallel processors*, or MPPs. Even most personal computers will soon have a few processors.

Multiprocessor machines allow different application programs to execute at the same time on different processors. They also allow a *single* application program to execute faster if it can be rewritten to use multiple processors. But how does one synchronize the activity of concurrent processes? How can one use multiple processors to make an application run faster?

To summarize, channels and multiprocessors provide both opportunities and challenges. When writing a concurrent program one has to make decisions about what kinds of processes to employ, how many to use, and how they should interact. These decisions are affected by the application and by the underlying hardware on which the program will run. Whatever choices are made, the key to developing a *correct* program is to ensure that process interaction is properly synchronized.

This book examines all aspects of concurrent programming. However, we focus on *imperative programs* with explicit concurrency, communication, and synchronization. In particular, the programmer has to specify the actions of each process and how they communicate and synchronize. This contrasts with *declarative programs*—e.g., functional or logic programs—in which concurrency is implicit and there is no reading and writing of a program state. In declarative programs, independent parts of the program may execute in parallel; they communicate and synchronize implicitly when one part depends on the results produced by another. Although the declarative approach is interesting and important—see Chapter 12 for more information—the imperative approach is much more widely used. In addition, to implement a declarative program on a traditional machine, one has to write an imperative program.

We also focus on concurrent programs in which process execution is *asynchronous*—namely, each process executes at its own rate. Such programs can be executed by interleaving the processes on a single processor or by executing the processes in parallel on a multiple instruction stream, multiple data stream (MIMD) multiprocessor. This class of machines includes shared-memory

Amazon
Exhibit 1005
Page 327

order. In fact, there are `n!` different orders in which the array of processes could write the values (`n` factorial is the number of permutations of `n` values).

We will declare and call procedures and functions in essentially the same way as in C. Two simple examples are

```
int addOne(int v) {   # an integer function
  return (v + 1);
}
main() {   # a "void" procedure
  int n, sum;
  read(n);   # read an integer from stdin
  for [i = 1 to n]
    sum = sum + addOne(i);
  write("the final value is", sum);
}
```

If the input value `n` is `5`, this program writes the line:

```
the final value is 20
```

1.9.4 Comments

We will write comments in one of two ways. One-line comments begin with the sharp character, `#`, and terminate with the end of the line. Multiple-line comments begin with `/*` and end with `*/`. We use `#` to introduce one-line comments because the C++/Java one-line comment symbol, `//`, has long been used in concurrent programming as the separator between arms of concurrent statements.

An *assertion* is a predicate that specifies a condition that is assumed to be true at a certain point in a program. (Assertions are described in detail in Chapter 2.) Assertions can be thought of as very precise comments, so we will specify them by lines that begin with two sharp characters, as in

```
## x > 0
```

This comment asserts that `x` is positive.

Historical Notes

As noted in the text, concurrent programming originated in the 1960s after the introduction of independent device controllers (channels). Operating systems were the first software systems to be organized as multithreaded concurrent programs. The research and initial prototypes leading to modern operating systems

Amazon
Exhibit 1005
Page 328

occurred in the late 1960s and early 1970s. The first textbooks on operating systems appeared in the early 1970s.

The introduction of computer networks in the 1970s led to the development of distributed systems. The invention of the Ethernet in the late 1970s greatly accelerated the pace of activity. Almost immediately there was a plethora of new languages, algorithms, and applications. Subsequent hardware developments spurred further activity. For example, once workstations and local area networks became relatively cheap, people developed client/server computing; more recently the Internet has spawned Java, Web browsers, and a plethora of new applications.

Multiprocessors were invented in the 1970s, the most notable machines being the Illiac SIMD multiprocessors developed at the University of Illinois. Early machines were costly and specialized, however, so for many years high-performance scientific computing was done on vector processors. This started to change in the mid-1980s with the invention of hypercube machines at the California Institute of Technology and their commercial introduction by Intel. Then Thinking Machines introduced the massively parallel Connection Machine. Also, Cray Research and other makers of vector processors started to produce multiprocessor versions of their machines. For several years, numerous companies (and machines) came, and almost as quickly went. However, the set of players and machines is now fairly stable, and high-performance computing is pretty much synonymous with massively parallel processing.

The Historical Notes in subsequent chapters describe developments that are covered in the chapters themselves. Described below are several general references on hardware, operating systems, distributed systems, and parallel computing. These are the books the author has consulted most often while writing this text. (If there is a topic you wish to learn more about, you can also try a good Internet search engine.)

The now classic textbook on computer architecture is Hennessy and Patterson [1996]. If you want to learn how caches, interconnection networks, and multiprocessors work, start there. Hwang [1993] describes high-performance computer architectures. Almasi and Gottlieb [1994] give an overview of parallel computing applications, software, and architectures.

The most widely used textbooks on operating systems are Tanenbaum [1992] and Silberschatz, Peterson, and Galvin [1998]. Tanenbaum [1995] also gives an excellent overview of distributed operating systems. Mullender [1993] contains a superb collection of chapters on all aspects of distributed systems, including reliability and fault tolerance, transaction processing, file systems, real-time systems, and security. Two additional textbooks—Bacon [1998] and Bernstein and Lewis [1993]—cover multithreaded concurrent systems, including distributed database systems.

Amazon
Exhibit 1005
Page 329

machine. This makes SIMD machines attractive for large problems that can be solved using data parallel algorithms. On the other hand, SIMD machines are special purpose in the sense that the entire machine executes one program at a time. (This is the main reason they have fallen out of favor.) Moreover, it is a challenge for the programmer to keep the processors busy doing useful work. In the above algorithm, for example, fewer and fewer processors update `sum[i]` on each iteration, yet each has to evaluate the condition in the `if` statement and then, if the condition is false, has to delay until all other processes have updated `sum`. In general, the execution time of an `if` statement is the *total* of the execution time for *every* branch, even if the branch is not taken. For example, the time to execute an `if/then/else` statement on each processor is the sum of the times to evaluate the condition, execute the `then` part, and execute the `else` part.

3.6 Parallel Computing with a Bag of Tasks

Many iterative problems can be solved using a data parallel programming style, as shown in the previous section. Recursive problems that arise from the divide and conquer paradigm can be parallelized by executing recursive calls in parallel rather than sequentially, as was shown in Section 1.5.

This section presents a different way to implement parallel computations using what is called a *bag of tasks*. A task is an independent unit of work. Tasks are placed in a bag that is shared by two or more worker processes. Each worker executes the following basic code:

```
while (true) {
  get a task from the bag;
  if (no more tasks)
    break;      # exit the while loop
  execute the task, possibly generating new ones;
}
```

This approach can be used to implement recursive parallelism, in which case the tasks are the recursive calls. It can also be used to solve iterative problems having a fixed number of independent tasks.

The bag-of-tasks paradigm has several useful attributes. First, it is quite easy to use. One has only to define the representation for a task, implement the bag, program the code to execute a task, and figure out how to detect termination. Second, programs that use a bag of tasks are *scalable*, meaning that one can use any number of processors merely by varying the number of workers. (However, the performance of the program might not scale.) Finally, the paradigm makes it

Amazon
Exhibit 1005
Page 330

easy to implement *load balancing*. If tasks take different amounts of time to execute, some workers will probably execute more than others. However, as long as there are enough more tasks than workers (e.g., two or three times as many), the total amount of computation executed by each worker should be approximately the same.

Below we show how to use a bag of tasks to implement matrix multiplication and adaptive quadrature. In matrix multiplication there is a fixed number of tasks. With adaptive quadrature, tasks are created dynamically. Both examples use critical sections to protect access to the bag and use barrier-like synchronization to detect termination.

3.6.1 Matrix Multiplication

Consider again the problem of multiplying two $\mathtt{n} \times \mathtt{n}$ matrices `a` and `b`. This requires computing $\mathtt{n}^2$ inner products, one for each combination of a row of `a` and a column of `b`. Each inner product is an independent computation, so each could be executed in parallel. However, suppose we are going to execute the program on a machine with `PR` processors. Then we would like to employ `PR` worker processes, one per processor. To balance the computational load, each should compute about the same number of inner products. In Section 1.4 we statically assigned parts of the computation to each worker. Here, we will use a bag of tasks, and have each worker grab a task whenever it is needed.

If `PR` is much smaller than `n`, a good size for a task would be one or a few rows of the result matrix `c`. (This leads to reasonable data locality for matrices `a` and `c`, assuming matrices are stored in row-major order.) For simplicity, we will use single rows. Initially, the bag contains `n` tasks, one per row. Since these can be ordered in any way we wish, we can represent the bag by simply counting rows:

```
int nextRow = 0;
```

A worker gets a task out of the bag by executing the atomic action:

```
⟨ row = nextRow; nextRow++; ⟩
```

where `row` is a local variable. The bag is empty if `row` is at least `n`. The atomic action above is another instance of drawing a ticket. It can be implemented using a Fetch-and-Add instruction, if one is available, or by using locks to protect the critical section.

Figure 3.20 contains an outline of the full program. We assume that the matrices have been initialized. Workers compute inner products in the usual way. The program terminates when each worker has broken out of its `while` loop. If

Amazon
Exhibit 1005
Page 331

```
int nextRow = 0; # the bag of tasks
double a[n,n], b[n,n], c[n,n];

process Worker[w = 1 to P] {
  int row;
  double sum;  # for inner products
  while (true) {
    # get a task
    ⟨ row = nextRow; nextRow++; ⟩
    if (row >= n)
      break;
    compute inner products for c[row,*];
  }
}
```

Figure 3.20 Matrix multiplication using a bag of tasks.

we want to detect this, we could employ a shared counter, `done`, that is initially zero and that is incremented atomically before a worker executes `break`. Then if we want to have the last worker print the results, we could add the following code to the end of each worker:

```
if (done == n)
  print matrix c;
```

This uses `done` like a counter barrier.

3.6.2 Adaptive Quadrature

Recall that the quadrature problem is to approximate the integral of a function `f(x)` from `a` to `b`. With adaptive quadrature, one first computes the midpoint `m` between `a` and `b`. Then one approximates three areas: from `a` to `m`, from `m` to `b`, and from `a` to `b`. If the sum of the two smaller areas is within some acceptable tolerance of the larger area, then the approximation is considered good enough. If not, the larger problem is divided into two subproblems and the process is repeated.

We can use the bag-of-tasks paradigm as follows to implement adaptive quadrature. Here, a task is an interval to examine; it is defined by the end points of the interval, the value of the function at those endpoints, and the approximation of the area for that interval. Initially there is just one task, for the entire interval `a` to `b`.

Amazon
Exhibit 1005
Page 332

A worker repeatedly takes a task from the bag and executes it. In contrast to the matrix multiplication program, however, the bag may (temporarily) be empty and a worker might thus have to delay before getting a task. Moreover, executing one task will usually lead to producing two smaller tasks, which a worker needs to put back into the bag. Finally, it is more difficult to determine when all the work has been done because it is not sufficient simply to wait until the bag is empty. (Indeed, the bag will *be* empty the minute the first task is taken.) Rather, all the work is done only when both (1) the bag is empty and (2) each worker is waiting to get another task.

Figure 3.21 contains a program for adaptive quadrature using a bag of tasks. The bag is represented by a queue and a counter. Another counter keeps track of the number of idle workers. All the work has been finished when `size` is `0` and `idle` is `n`. Note that the program contains several atomic actions. These are needed to protect the critical sections that access the shared variables. All but one of the atomic actions are unconditional, so they can be protected using locks. However, the one `await` statement has to be implemented using the more elaborate protocol in Section 3.2 or using a more powerful synchronization mechanism such as semaphores or monitors.

The program in Figure 3.21 makes excessive use of the bag of tasks. In particular, when a worker decides to generate two tasks, it puts both of them in the bag, then loops around and takes a new task out of the bag (perhaps even one that it just put in). Instead, we could have a worker put one task in the bag and keep the other. Once the bag contains enough work to lead to a balanced computational load, a further optimization would be to have a worker execute a task fully, using sequential recursion, rather than putting more tasks into the bag.

Historical Notes

The critical section problem was first described by Edsger Dijkstra [1965]. Because the problem is fundamental, it has been studied by scores of people who have published literally hundreds of papers on the topic. This chapter has presented four of the most important solutions. (Raynal [1986] has written an entire book on mutual exclusion algorithms.) Although devising busy-waiting solutions was early on mostly an academic exercise—since busy waiting is inefficient on a single processor—the advent of multiprocessors has spurred renewed interest in such solutions. Indeed, all multiprocessors now provide instructions that support at least one busy-waiting solution. The exercises at the end of this chapter describe most of these instructions.

Dijkstra's paper [1965] presented the first `n`-process software solution. It is an extension of the first two-process solution designed by Dutch mathematician T. Dekker (see the exercises). However, Dijkstra's original formulation of the

Amazon
Exhibit 1005
Page 333

```
type task = (double left, right, fleft, fright, lrarea);
queue bag(task);          # the bag of tasks
int size;                 # number of tasks in bag
int idle = 0;             # number of idle workers
double total = 0.0;       # the total area

compute approximate area from a to b;
insert task (a, b, f(a), f(b), area) in the bag;
count = 1;

process Worker[w = 1 to PR] {
  double left, right, fleft, fright, lrarea;
  double mid, fmid, larea, rarea;
  while (true) {
    # check for termination
    ⟨ idle++;
      if (idle == n && size == 0) break; ⟩
    # get a task from the bag
    ⟨ await (size > 0)
      remove a task from the bag;
      size--; idle--; ⟩
    mid = (left+right) / 2;
    fmid = f(mid);
    larea = (fleft+fmid) * (mid-left) / 2;
    rarea = (fmid+fright) * (right-mid) / 2;
    if (abs((larea+rarea) - lrarea) > EPSILON) {
      ⟨ put (left, mid, fleft, fmid, larea) in the bag;
        put (mid, right, fmid, fright, rarea) in the bag;
        size = size + 2; ⟩
    } else
      ⟨ total = total + lrarea; ⟩
  }
  if (w == 1)   # worker 1 prints the result
    printf("the total is %f\n", total);
}
```

Figure 3.21 Adaptive quadrature using a bag of tasks.

problem did not require the eventual entry property (3.5). Donald Knuth [1966] was the first to publish a solution that also ensures eventual entry.

The tie-breaker algorithm was discovered by Gary Peterson [1981]; it is often called Peterson's algorithm as a result. This algorithm is particularly simple for two processes, unlike the earlier solutions of Dekker, Dijkstra, and others.

Amazon
Exhibit 1005
Page 334

Peterson's algorithm also generalizes readily to an `n`-process solution, as shown in Figure 3.7. That solution requires that a process go through all `n-1` stages, even if no other process is trying to enter its critical section. Block and Woo [1990] present a variation that requires only `m` stages if just `m` processes are contending for entry (see the exercises).

The bakery algorithm was devised by Leslie Lamport [1974]. (Figure 3.11 contains an improved version that appeared in Lamport [1979].) In addition to being more intuitive than earlier critical section solutions, the bakery algorithm allows processes to enter in essentially FIFO order. It also has the interesting property that it is tolerant of some hardware failures. First, if one process reads `turn[i]` while another process is setting it in the entry protocol, the read can return any value between `1` and the value being written. Second, a process `CS[i]` may fail at any time, assuming it immediately sets `turn[i]` to `0`. However, the value of `turn[i]` can become unbounded in Lamport's solution if there is always at least one process in its critical section.

The ticket algorithm is the simplest of the `n`-process solutions to the critical section problem; it also allows processes to enter in the order in which they draw numbers. The ticket algorithm can in fact be viewed as an optimization of the bakery algorithm. However, it requires a Fetch-and-Add instruction, something that few machines provide (see Almasi and Gottlieb [1994] for machines that do).

Harry Jordan [1978] was one of the first to recognize the importance of barrier synchronization in parallel iterative algorithms; he is credited with coining the term barrier. Although barriers are not nearly as commonplace as critical sections, they too have been studied by dozens of people who have developed several different implementations. The combining tree barrier in Figures 3.13 and 3.14 is similar to one devised by Yew, Tzeng, and Lawrie [1987]. The butterfly barrier was devised by Brooks [1986]. Hensgen, Finkel, and Manber [1988] developed the dissemination barrier; their paper also describes a tournament barrier, which is similar in structure to a combining tree. Gupta [1989] describes a "fuzzy barrier," which includes a region of statements a process can execute while it waits for a barrier; his paper also describes a hardware implementation of fuzzy barriers.

As described in several places in the chapter, busy-waiting implementations of locks and barriers can lead to memory and interconnection-network contention. Thus it is important for delayed processes to spin on the contents of local memory, e.g., cached copies of variables. Chapter 8 of Hennessy and Patterson's architecture book [1996] discusses synchronization issues in multiprocessors, includes an interesting historical perspective, and provides an extensive set of references.

An excellent paper by Mellor-Crummy and Scott [1991] analyzes the performance of several lock and barrier protocols, including most of those described

Amazon
Exhibit 1005
Page 335

in the text. That paper also presents a new, list-based lock protocol and a scalable, tree-based barrier with only local spinning. A recent paper [McKenney 1996] describes different kinds of locking patterns in parallel programs and gives guidelines for selecting which locking primitives to use. Another recent paper [Savage et al. 1997] describes a tool called Eraser that can detect data races due to synchronization mistakes in lock-based multithreaded programs.

On multiprogrammed (time-sliced) systems, performance can suffer greatly if a process is preempted while holding a lock. One way to overcome this problem is to use *nonblocking*—or *lock free*—algorithms instead of using mutual exclusion locks. An implementation of a data structure is nonblocking if some process will complete an operation in a finite number of steps, regardless of the execution speed of other processes. Herlihy and Wing [1990] introduces this concept and presents a nonblocking implementation of concurrent queues. In Herlihy and Wing's implementation, the insert operation is also *wait free*—i.e., every insert operation will complete in a finite number of steps, again regardless of execution speeds. Herlihy [1990] presents a general methodology for implementing highly concurrent data structures, and Herlihy [1991] contains a comprehensive discussion of wait-free synchronization. A recent paper by Michael and Scott [1998] evaluates the performance of nonblocking implementations of several data structures; they also consider the alternative of lock-based implementations that are modified to be preemption safe. They conclude that nonblocking implementations are superior for those data structures for which they exist (e.g., queues), and that preemption-safe locks outperform normal locks on multiprogrammed systems.

Data parallel algorithms are most closely associated with massively parallel machines since those machines permit thousands of data elements to be operated on in parallel. Many of the earliest examples of data parallel algorithms were designed for the NYU Ultracomputer [Schwartz 1980]; all have the characteristic attribute that execution time is logarithmic in the size of the data. The Ultracomputer is a MIMD machine, and its designers realized the importance of efficient critical section and barrier synchronization protocols. Consequently, they implemented a Replace-and-Add operation in an early hardware prototype [Gottlieb et al. 1983]; this instruction adds a value to a memory word and then returns the result. However, more recent versions of the Ultracomputer provide a Fetch-and-Add operation instead [Almasi & Gottlieb 1994], which returns the value of a memory word before adding to it. This change was made because Fetch-and-Add, unlike Replace-and-Add, generalizes to any binary combining operator—for example, Fetch-and-Max or Fetch-and-And. This general kind of instruction is called a Fetch-and-Φ operation.

The introduction of the Connection Machine in the mid-1980s spurred renewed interest in data parallel algorithms. That machine was designed by

Amazon
Exhibit 1005
Page 336

Daniel Hillis [1985] as part of his doctoral dissertation at MIT. (Precious few dissertations have been as influential!) The Connection Machine was a SIMD machine, so barrier synchronization was automatically provided after every machine instruction. The Connection Machine also had thousands of processing elements; this enables a data parallel algorithm to have a very fine granularity—for example, a processor can be assigned to every element of an array. Hillis and Steele [1986] give an overview of the original Connection Machine and describe several interesting data parallel algorithms.

The bag-of-tasks paradigm for parallel computing was introduced by Carriero, Gelernter, and Leichter [1986]. That paper shows how to implement the task bag in the Linda programming notation, which we describe in Chapter 7. They called the associated programming model *replicated workers*, because any number of workers can share the bag. Some now call this the *work farm model*, because the tasks in the bag are farmed out to the workers. We prefer the phrase "bag of tasks" because it characterizes the essence of the approach: a shared bag of tasks.

References

Almasi, G. S., and A. Gottlieb. 1994. *Highly Parallel Computing*, 2nd ed. Menlo Park, CA: Benjamin/Cummings.

Block, K., and T.-K. Woo. 1990. A more efficient generalization of Peterson's mutual exclusion algorithm. *Information Processing Letters* 35 (August): 219–22.

Brooks, E. D., III. 1986. The butterfly barrier. *Int. Journal of Parallel Prog.* 15, 4 (August): 295–307.

Carriero, N., D. Gelernter, and J. Leichter. 1986. Distributed data structures in Linda. *Thirteenth ACM Symp. on Principles of Prog. Langs.*, January, pp. 236–42.

Dijkstra, E. W. 1965. Solution of a problem in concurrent programming control. *Comm. ACM* 8, 9 (September): 569.

Gottlieb, A., B. D. Lubachevsky, and L. Rudolph. 1983. Basic techniques for the efficient coordination of very large numbers of cooperating sequential processors. *ACM Trans. on Prog. Languages and Systems* 5, 2 (April): 164–89.

Gupta, R. 1989. The fuzzy barrier: a mechanism for high speed synchronization of processors. *Third Int. Conf. on Architectural Support for Prog. Languages and Operating Systems*, April, pp. 54–63.

Amazon
Exhibit 1005
Page 337

This statement evaluates `n` calls of `f` in parallel and assigns the results to shared array `a`. The corresponding C-Linda code is

```
for (i = 1; i <= n; i++)
  EVAL("a", i, f(i));
```

This forks `n` process tuples; each evaluates one call of `f(i)`. When a process tuple terminates, it turns into a data tuple containing the array name, an index, and the value of that element of `a`.

TS can also be used to implement conventional message passing. For example, a process can send a message to a channel `ch` by executing

```
OUT("ch", expressions);
```

Again, the name of a shared object is turned into a tag on a tuple. Another process can receive the message by executing

```
IN("ch", variables);
```

However, this statement extracts any matching tuple. If it is important that messages be received in the order they are sent, the programmer has to impose an ordering. This can be done by using counters to put sequence numbers on messages, and then reading them in sequence-number order.

7.7.2 Example: Prime Numbers with a Bag of Tasks

In Section 7.6, we showed how to generate prime numbers using the Sieve of Eratosthenes. In that approach, candidate numbers pass through a pipeline of filter processes. Each filter holds one prime factor and checks whether that prime divides each candidate; if not, the candidate might be prime so it is passed to the next filter process in the pipeline.

We can also generate primes using the bag-of-tasks paradigm (which was introduced in Section 3.6). As usual, the workers share a bag of tasks. In this case, the bag contains the candidate numbers; it is stored in tuple space. A manager process deposits candidates into the bag and gathers results.

Each worker repeatedly removes a candidate from the bag and determines whether it is prime by dividing it by smaller primes. But how can a worker acquire all the primes it needs? The answer is to check candidates in strictly increasing order. In this way, the prime factors needed to check a new candidate will already have been generated before they are needed. A worker might have to delay—waiting for another worker to generate a smaller prime—but workers will not deadlock since all smaller primes will eventually be generated.

Amazon
Exhibit 1005
Page 338

Figure 7.16 contains a C-Linda program that generates the first `limit` primes using a bag of tasks stored in tuple space. The structure of a C-Linda program is essentially that of a C program. However, the main routine is named `real_main` rather than `main`.

```
# include "linda.h"
# define LIMIT 1000      /* upper bound for limit */

void worker() {
  int primes[LIMIT] = {2,3};  /* table of primes */
  int numPrimes = 1, i, candidate, isprime;

  /* repeatedly get candidates and check them */
  while(true) {
    if (RDP("stop"))  /* check for termination */
      return;
    IN("candidate", ?candidate);     /* get candidate */
    OUT("candidate", candidate+2);  /* output next one */
    i = 0; isprime = 1;
    while (primes[i]*primes[i] <= candidate) {
      if (candidate%primes[i] == 0) { /* not prime */
        isprime = 0; break;
      }
      i++;
      if (i > numPrimes) {  /* need another prime */
        numPrimes++;
        RD("prime", numPrimes, ?primes[numPrimes]);
      }
    }
    /* tell manager the result */
    OUT("result", candidate, isprime);
  }
}

real_main(int argc, char *argv[]) {
  int primes[LIMIT] = {2,3};  /* my table of primes */
  int limit, numWorkers, i, isprime;
  int numPrimes = 2, value = 5;
  limit = atoi(argv[1]);  /* read command line */
  numWorkers = atoi(argv[2]);
```

Amazon
Exhibit 1005
Page 339

```
    /* create workers and put first candidate in bag */
    for (i = 1; i <= numWorkers; i++)
      EVAL("worker", worker());
    OUT("candidate", value);

    /* get results from workers in increasing order */
    while (numPrimes < limit) {
      IN("result", value, ?isprime);
      if (isprime) {    /* put value in table and TS */
        primes[numPrimes] = value;
        OUT("prime", numPrimes, value);
        numPrimes++;
      }
      value = value + 2;
    }
    /* tell workers to quit, then print the primes */
    OUT("stop");
    for (i = 0; i < limit; i++)
      printf("%d\n", primes[i]);
}
```

Figure 7.16 Prime number generation in C-Linda.

The main routine executes first as a single process. It begins by reading command-line arguments to determine how many primes to generate and how many workers to use. It then uses `EVAL` to fork the worker processes and uses `OUT` to deposit the first candidate, `5`, in TS.

The main routine then assumes the role of a manager process. In particular, it uses `IN` to receive the results from workers; these are received in order so that primes can be counted. When `real_main` receives a prime, it puts the prime and its count into TS and updates its local table. For example, `5` is the third prime, `7` is the fourth, and so on.

The program terminates when `real_main` has received `limit` primes. The manager tells the workers to stop by depositing a `stop` token in TS, then it prints the primes stored in its local table.

Each worker process repeatedly gets a candidate number from TS, checks its primality, and sends the result to the manager by depositing the result in TS. After a worker extracts a candidate number using `IN`, it deposits the next odd candidate into TS. Thus, there is at most one candidate number in TS at a time, and the candidates are increasing odd integers.

Amazon
Exhibit 1005
Page 340

Each worker keeps a local table of prime factors. It extends this table when it needs another prime factor to check a candidate number. It gets the prime by reading it from TS. To avoid deadlock, workers read primes in strictly increasing order.

On each iteration, a worker uses the conditional read primitive, `RDP`, to see if it should stop. This primitive returns true if there is a `stop` tuple in TS; otherwise it returns false. Since the workers will not see the `stop` tuple until after the manager has acquired `limit` primes, more candidates than necessary will be checked. This is hard to avoid without having the processes exchange many more messages.

7.8.1

The program in Figure 7.16 is not the most efficient one we could devise. This is because each candidate is deposited and extracted twice—once in a `candidate` tuple and then in a `result` tuple. Accessing TS is much less efficient than accessing local variables. To increase the use of local variables, we could have the manager deposit ranges of candidates—say 10 or 20 numbers at a time. We could then have the workers compute all primes in a given range and send the collective result to the manager. We could also seed the tables of primes in the manager and workers with more of the small primes.

7.8 Case Study: MPI

The Message Passing Interface (MPI) is a library of message-passing routines. When MPI is used, the processes in a distributed program are written in a sequential language such as C or Fortran; they communicate and synchronize by calling functions in the MPI library.

The MPI application programmer's interface (API) was defined in the mid-1990s by a large group of people from academia, government, and industry. The interface reflects people's experiences with earlier message-passing libraries, such as PVM. The goal of the group was to develop a single library that could be implemented efficiently on the variety of multiple processor machines. MPI has now become the de facto standard, and several implementations exist.

MPI programs have what is called an SPMD style—single program, multiple data. In particular, every processor executes a copy of the same program. (In an application written in `C`, the `main` routine in the program is started on every processor.) Each instance of the program can determine its own identity and hence take different actions. The instances interact by calling MPI library functions. The MPI functions support process-to-process communication, group communication, setting up and managing communication groups, and interacting with the environment.

Amazon
Exhibit 1005
Page 341

The first three paradigms are commonly used in parallel computations; the other four arise in distributed systems. We illustrate how the paradigms can be used to solve a variety of problems, including sparse matrix multiplication, image processing, distributed matrix multiplication, computing the topology of a network, distributed mutual exclusion, distributed termination detection, and decentralized dining philosophers. We later use the three parallel computing paradigms to solve the scientific computing problems in Chapter 11. Additional applications are described in the exercises, including sorting and the traveling salesman.

9.1 Manager/Workers (Distributed Bag of Tasks)

We introduced the bag-of-tasks paradigm in Section 3.6 and showed how to implement it using shared variables for communication and synchronization. Recall the basic idea: Several worker processes share a bag that contains independent tasks. Each worker repeatedly removes a task from the bag, executes it, and possibly generates one or more new tasks that it puts into the bag. The benefits of this approach to implementing a parallel computation are that it is easy to vary the number of workers and it is relatively easy to ensure that each does about the same amount of work.

Here we show how to implement the bag-of-tasks paradigm using message passing rather than shared variables. This is done by employing a manager process to implement the bag, hand out tasks, collect results, and detect termination. The workers get tasks and deposit results by communicating with the manager. Thus the manager is essentially a server process and the workers are clients.

The first example below shows how to multiply sparse matrices (matrices for which most entries are zero). The second example revisits the quadrature problem and uses a combination of static and adaptive integration intervals. In both examples, the total number of tasks is fixed, but the amount of work per task is variable.

9.1.1 Sparse Matrix Multiplication

Assume that `A` and `B` are `n` × `n` matrices. The goal, as before, is to compute the matrix product `A` × `B` = `C`. This requires computing n^2 inner products. Each inner product is the sum (plus reduction) of the pairwise products of two vectors of length `n`.

A matrix is said to be *dense* if most entries are nonzero and it is *sparse* if most entries are zero. If `A` and `B` are dense matrices, then `C` will also be dense

Amazon
Exhibit 1005
Page 342

(unless there is considerable cancellation in the inner products). On the other hand, if either `A` or `B` is sparse, then `C` will also be sparse. This is because each zero in `A` or `B` will lead to a zero in `n` vector products and hence will not contribute to the result of `n` of the inner products. For example, if some row of `A` contains all zeros, then the corresponding row of `C` will also contain all zeros.

Sparse matrices arise in many contexts, such as numerical approximations to partial differential equations and large systems of linear equations. A tridiagonal matrix is an example of a sparse matrix; it has zeros everywhere except on the main diagonal and the two diagonals immediately above and below the main diagonal. When we know that matrices are sparse, we can save space by storing only the nonzero entries, and we can save time when multiplying matrices by ignoring entries that are zero.

We will represent sparse matrix `A` as follows by storing information about the rows of `A`:

```
int lengthA[n];
pair *elementsA[n];
```

The value of `lengthA[i]` is the number of nonzero entries in row `i` of `A`. Variable `elementsA[i]` points to a list of the nonzero entries in row `i`. Each entry is represented by a pair (record): an integer column index and a double precision data value. Thus if `lengthA[i]` is `3`, then there are three pairs in the list `elementsA[i]`. These will be ordered by increasing values of the column indexes. As a concrete example, consider the following:

```
lengthA  elementsA

  1      (3, 2.5)
  0
  0
  2      (1, -1.5)   (4, 0.6)
  0
  1      (0, 3.4)
```

This represents a six by six matrix containing four nonzero elements: one in row `0` column `3`, one in row `3` column `1`, one in row `3` column `4`, and one in row `5`, column `0`.

We will represent matrix `C` in the same way as `A`. However, to facilitate matrix multiplication, we will represent matrix `B` by columns rather than rows. In particular, `lengthB` indicates the number of nonzero elements in each column of `B`, and `elementsB` contains pairs of row index and data value.

Computing the matrix product of `A` and `B` requires as usual examining the n^2 pairs of rows and columns. With sparse matrices, the most natural size for a

Amazon
Exhibit 1005
Page 343

```
module Manager
  type pair = (int index, double value);
  op getTask(result int row, len; result pair [*]elems);
  op putResult(int row, len; pair [*]elems);
body Manager
  int lengthA[n], lengthC[n];
  pair *elementsA[n], *elementsC[n];
  # matrix A is assumed to be initialized
  int nextRow = 0, tasksDone = 0;
  process manager {
    while (nextRow < n or tasksDone < n) {
      # more tasks to do or more results needed
      in getTask(row, len, elems) ->
          row = nextRow;
          len = lengthA[i];
          copy pairs in *elementsA[i] to elems;
          nextRow++;
      [] putResult(row, len, elems) ->
          lengthC[row] = len;
          copy pairs in elems to *elementsC[row];
          tasksDone++;
      ni
    }
  }
end Manager
```

Figure 9.1 (a) Sparse matrix multiplication: Manager process.

task is one row of the result matrix `C`, because an entire row is represented by a single vector of (column, value) pairs. Thus there are as many tasks as there are rows of `A`. (An obvious optimization would be to skip over rows of `A` that are entirely zeros—namely, those for which `lengthA[i]` is `0`, because the corresponding rows of `C` would also be all zeros. This is not likely in realistic applications, however.)

Figure 9.1 contains code to implement sparse matrix multiplication using a manager and several worker processes. We assume that matrix `A` is already initialized in the manager and that each worker contains an initialized copy of matrix `B`. The processes interact using the rendezvous primitives of Chapter 8, because that leads to the simplest program. To use asynchronous message passing, the manager would be programmed in the style of an active monitor (Section 7.3) and the `call` in the workers would be replaced by `send` then `receive`.

Amazon
Exhibit 1005
Page 344

```
process worker[w = 1 to numWorkers] {
  int lengthB[n];
  pair *elementsB[n];  # assumed to be initialized
  int row, lengthA, lengthC;
  pair *elementsA, *elementsC;
  int r, c, na, nb;  # used in computing
  double sum;        #   inner products
  while (true) {
    # get a row of A, then compute a row of C
    call getTask(row, lengthA, elementsA);
    lengthC = 0;
    for [i = 0 to n-1]
      INNER_PRODUCT(i);  # see body of text
    send putResult(row, lengthC, elementsC);
  }
}
```

Figure 9.1 (b) Sparse matrix multiplication: Worker processes.

The manager process in Figure 9.1 (a) implements the bag and accumulates results. It services two operations: `getTask` and `putResult`. Integer `nextRow` identifies the next task—namely, the next row. Integer `tasksDone` counts the number of results that have been returned. When a task is handed out, `nextRow` is incremented; when a result is returned, `tasksDone` is incremented. When both values are equal to `n`, all tasks have been completed.

The code for the worker processes is shown in Figure 9.1 (b). Each worker repeatedly gets a new task, performs it, then sends the result to the manager. A task consists of computing a row of the result matrix `C`, so a worker executes a `for` loop to compute `n` inner products, one for every column of `B`. However, the code for computing an inner product of two sparse vectors differs significantly from the loop for computing an inner product of two dense vectors. In particular, the meaning of `INNER_PRODUCT(i)` in the worker code is

```
sum = 0.0; na = 1; nb = 1;
c = elementsA[na]->index;      # column in row of A
r = elementsB[i][nb]->index;   # row in column of B
while (na <= lengthA and nb <= lengthB) {
  if (r == c) {
    sum += elementsA[na]->value *
           elementsB[i][nb]->value;
    na++; nb++;
```

Amazon
Exhibit 1005
Page 345

```
      c = elementsA[na]->index;
      r = elementsB[i][nb]->index;
    } else if (r < c) {
      nb++; r = elementsB[i][nb]->index;
    } else { # r > c
      na++; c = elementsA[na]->index;
    }
  }
  if (sum != 0.0) {  # extend row of C
    elementsC[lengthC] = pair(i, sum);
    lengthC++;
  }
```

The basic idea is to scan through the sparse representations for the row of `A` and column `i` of `B`. (The above code assumes that the row and column each contain at least one nonzero element.) The inner product will be nonzero only if there are pairs of values that have the *same* column index `c` in `A` and row index `r` in `B`. The `while` loop above finds all these pairs and adds up their products. If the `sum` is nonzero when the loop terminates, then a new pair is added to the vector representing the row of `C`. (We assume that space for the elements has already been allocated.) After all `n` inner products have been computed, a worker sends the row of `C` to the manager, then calls the manager to get another task.

9.1.2 Adaptive Quadrature Revisited

The quadrature problem was introduced in Section 1.5, where we presented static (iterative) and dynamic (recursive) algorithms for approximating the integral of function `f(x)` from `a` to `b`. Section 3.6 showed how to implement adaptive quadrature using a shared bag of tasks.

Here we use a manager to implement a distributed bag of tasks. However, we use a combination of the static and dynamic algorithms rather than the "pure" adaptive quadrature algorithm used in Figure 3.21. In particular, we divide the interval from `a` to `b` into a fixed number of subintervals and then use the adaptive quadrature algorithm within each subinterval. This combines the simplicity of an iterative algorithm with the greater accuracy of an adaptive algorithm. In particular, by using a static number of tasks, the manager and workers are simpler and there are far fewer interactions between them.

Figure 9.2 contains the code for the manager and workers. Because the manager is essentially a server process, we again use rendezvous for interaction

Amazon
Exhibit 1005
Page 346

```
module Manager
  op getTask(result double left, right);
  op putResult(double area);
body Manager
  process manager {
    double a, b;          # interval to integrate
    int numIntervals;     # number of intervals to use
    double width = (b-a)/numIntervals;
    double x = a, totalArea = 0.0;
    int tasksDone = 0;
    while (tasksDone < numIntervals) {
      in getTask(left, right) st x < b ->
          left = x; x += width; right = x;
      [] putResult(area) ->
          totalArea += area;
          tasksDone++;
      ni
    }
  print the result totalArea;
  }
end Manager

double f() { ... }          # function to integrate
double quad(...) { ... }    # adaptive quad function

process worker[w = 1 to numWorkers] {
  double left, right, area = 0.0;
  double fleft, fright, lrarea;
  while (true) {
    call getTask(left, right);
    fleft = f(left); fright = f(right);
    lrarea = (fleft + fright) * (right - left) / 2;
    # calculate area recursively as shown in Section 1.5
    area = quad(left, right, fleft, fright, lrarea);
    send putResult(area);
  }
}
```

Figure 9.2 Adaptive quadrature using manager/workers paradigm.

Amazon
Exhibit 1005
Page 347

between the manager and workers. Thus the manager has the same structure as that shown in Figure 9.1 (a) and again exports two operations, `getTask` and `putResult`. However, the operations have different parameters, because now a task is defined by the endpoints of an interval, `left` and `right`, and a result consists of the area under `f(x)` over that interval. We assume the values of `a`, `b`, and `numIntervals` are given, perhaps as command-line arguments. From these values, the manager computes the width of each interval. The manager then loops, accepting calls of `getTask` and `putResult`, until it has received one result for each interval (and hence task). Notice the use of a synchronization expression "`st x< b`" in the arm for `getTask` in the input statement; this prevents `getTask` from handing out another task when the bag is empty.

The worker processes in Figure 9.2 share or have their own copies of the code for functions `f` and `quad`, where `quad` is the recursive function given in Section 1.5. Each worker repeatedly gets a task from the manager, calculates the arguments needed by `quad`, calls `quad` to approximate the area under `f(left)` to `f(right)`, then sends the result to the manager.

When the program in Figure 9.2 terminates, every worker will be blocked at its call of `getTask`. This is often harmless, as here, but at times it can indicate deadlock. We leave to the reader modifying the program so that the workers terminate normally. (*Hint:* Make `getTask` a function that returns `true` or `false`.)

In this program the amount of work per task varies, depending on how rapidly the function `f` varies. Thus, if there are only about as many tasks as workers, the computational load will almost certainly be unbalanced. On the other hand, if there are too many tasks, then there will be unnecessary interactions between the manager and workers, and hence unnecessary overhead. The ideal is to have just enough tasks so that, on average, each worker will be responsible for about the same total amount of work. A reasonable heuristic is to have from two to three times as many tasks as workers, which here means that `numIntervals` should be two to three times larger than `numWorkers`.

9.2 Heartbeat Algorithms

The bag-of-tasks paradigm is useful for solving problems that have a fixed number of independent tasks or that result from use of the divide-and-conquer paradigm. The *heartbeat paradigm* is useful for many data parallel iterative applications. In particular, it can be used when the data is divided among workers, each is responsible for updating a particular part, and new data values depend on values held by workers or their immediate neighbors. Applications include

Amazon
Exhibit 1005
Page 348

```
process Worker[w = 1 to PR] {
  # run the simulation with time steps of DT
  for [time = start to finish by DT] {
    calculateForces(w);
    barrier(w);
    moveBodies(w);
    barrier(w);
  }
}
```

Figure 11.11 Shared-variable program for the n-body problem.

so that the workload is balanced. We also need to minimize communication overhead or at least to make it small relative to the amount of useful work. These are challenging issues because the n-body problem involves calculating forces between every pair of bodies, and hence every worker process needs to interact with all the others.

A useful starting point is to consider whether we can use any of the three process interaction paradigms at the start of Chapter 9: manager/workers (distributed bag of tasks), heartbeat, and pipeline. There are well-defined tasks and load balancing is an issue, so the manager/workers paradigm is appropriate. We can also use a heartbeat algorithm by dividing up the bodies and having workers exchange information; however, it is not sufficient to have each worker just exchange information with one or two neighbors. Finally, we can also use a pipeline algorithm and have bodies flow from worker to worker. Below we describe how to solve the n-body problem using each of these paradigms, then we discuss the tradeoffs between them.

Manager/Workers Program

First consider using the manager/workers paradigm (Section 9.1). With this approach the manager maintains a bag of tasks; workers repeatedly get a task, do it, then return the result to the manager. For the n-body problem, there are two phases. In the first, the tasks are to calculate the forces between every pair of bodies. In the second, the tasks are to move the bodies. The majority of the computation—and the part that can be imbalanced—is calculating forces. Hence it makes sense to use dynamic tasks for the force calculations but to use static tasks (one per worker) for moving bodies.

Amazon
Exhibit 1005
Page 349

Suppose there are `PR` processors and that we want to use `PR` worker processes. (Hence, the manager will execute on the same processor as one of the workers.) Assume for simplicity that `n` is a multiple of `PR`. When using the manager/workers paradigm, we want to have at least twice as many tasks as workers to balance the load, but we do not want too many tasks or the workers will spend too much time communicating with the manager. One way to divide up the force calculations for the `n` bodies is as follows:

> Divide the bodies into `PR` blocks of size `n/PR`. The first block contains the first `n/PR` bodies, the second the next `n/PR` bodies, and so on.
>
> Form pairs (i, j) for each different combination of block numbers. There will be `PR*(PR+1)/2` such pairs—the sum of the integers from `1` to `PR`.
>
> Let each pair represent a task—for example, the task for pair (i, j) is to calculate the forces between the bodies in block i and those in block j.

As a concrete example, suppose `PR` is four. Then the ten tasks are the pairs

(1, 1) (1, 2) (1, 3) (1, 4) (2, 2) (2, 3) (2, 4) (3, 3) (3, 4) (4, 4)

The manager process initializes the bag with the pairs defining the tasks. Workers repeatedly get a task from the manager and calculate the forces specified by the task. To do so, a worker needs to know the positions, velocities, and masses of the bodies in one or two blocks. The manager could send these along with the task, but we can greatly reduce the length of messages by having each worker keep its own copy of the data for each body. Similarly, we can get rid of the need to send results to the manager if each worker keeps track of the forces it has calculated for each body.

After calculating all forces—namely when the bag of tasks is empty—the workers need to exchange forces and then move the bodies. The simplest static assignment of work for the move-bodies phase is to have each worker move the bodies in one block: worker 1 moves block 1, worker 2 moves block 2, and so on. Hence, each worker `w` needs the force vectors for the bodies in block `w`. We could use the manager to collect and disseminate the forces calculated by each worker, but it will be more efficient if each worker sends the forces for a block of bodies directly to the worker that will move those bodies. Alternatively, if we have global communication primitives—such as those in the MPI library—the workers can use them to broadcast and reduce (add) the values in the force vectors. After each worker moves its block of bodies, the workers need to exchange the new positions and velocities of the bodies. Again we could use point-to-point messages or global communication to effect this exchange.

Amazon
Exhibit 1005
Page 350

Figure 11.12 contains an outline of the code for a manager/workers program. The outer loop in each process executes once for each time step in the simulation. The inner loop in the manager process iterates `numTasks+PR` times, where `numTasks` is the number of tasks in the bag. On the last `PR` iterations, the bag will be empty, so the manager sends the pair (0, 0) as a "bag is empty" sentinel. When each of the `PR` workers gets this sentinel, it breaks out of the loop that calculates forces.

Heartbeat Program

A heartbeat algorithm (Section 9.2) is the message-passing analog of a shared-variable algorithm that has computational phases followed by barriers. The shared-variable program in Figure 11.11 has this structure, so we can modify it as follows to use message passing: (1) each worker is assigned a subset of the bodies; (2) every worker first calculates the forces for its set of bodies then exchanges forces with the other workers; (3) every worker moves its bodies; (4) the workers exchange the new positions and velocities of the bodies. This pattern is repeated for each time step in the simulation.

We can assign bodies to worker processes using any of the allocation patterns described in Figure 11.10: blocks, stripes, or reverse stripes. If we use a stripes or reverse stripes allocation, then the computational load will be much more balanced than if we use fixed-size blocks. However, each worker would need to have its own copy of the position, velocity, force, and mass vectors. Moreover, each worker would need to exchange entire vectors with every other worker after each phase. This leads to a large number of long messages.

If we assign blocks of bodies to workers, we will use about half as many messages and they will be shorter. (We will also need less local storage in the workers.) To see why, consider the example of four workers and ten tasks described in the previous section. With a block assignment, worker 1 handles the first four tasks: (1, 1), (1, 2), (1, 3), and (1, 4). Namely, it handles all force calculations involving the bodies in block 1. To do so, it needs positions and velocities for the bodies in blocks 2, 3, and 4, and it needs to return forces for those blocks. However, worker 1 never needs to tell another worker the position or velocity of its bodies and it never needs to get force information from any other worker. At the other extreme, worker 4 is responsible only for the task (4, 4). It does not need to know the position or velocity of any other body and it does not need to send forces to any other worker.

If the blocks are all the same size, then the workload will be unbalanced, which would more than offset any gain from sending fewer messages. However,

Amazon
Exhibit 1005
Page 351

Sorting algorithms (cont.)
merge network, 300–1
quicksort, 17
Soundness, of a logic, 58
Spanning tree, 445
Sparse matrix multiplication, 424–28
Spatial locality, 5
Speedup, of a parallel computation, 528
Spin lock, 99
Spin loop, 55
Spinning process, 57
Split binary semaphore, 159
SPMD (single program multiple data), 316, 340, 541
SR language, 406–11, 615–16
history and references, 414
SST disk scheduling, 227, 310
Stable marriage problem, 90, 356, 420
Stable prefix, of a message queue, 455
StarMod language, 412
Starvation, absence of. *See* Fairness
State, of a
process, 267
program, 42
Strength reduction transformation, 536
Strip mining, of loops, 611
Stripes, allocation by, 557
Strips, allocation by, 557
Strongly fair scheduling policy, 76
Structure of distributed programs. *See* Distributed program structures
Successive over relaxation, 546–49
Symmetric barrier, 121
Symmetric multiprocessor (SMP), 6
Synchronization, 2. *See also* Condition synchronization; Mutual exclusion
specified using `await`, 54–55
used for avoiding interference, 69–70
Synchronizing Resources. *See* SR language
Synchronous execution, 131
Synchronous message passing, 318–20. *See also* CSP language
implementation of, 496–503
Synchronous multiprocessor, 131
Synchronous `send` statement, 318

Task parallel program, 11
Temporal locality, 5
Temporal logic, 80
Termination detection problem, 460–65
Test-and-test-and-set protocol, 101
Test-and-Set instruction, 99
Textual substitution, 60
Thread of control, 2, 184
Ticket algorithm, 108–11, 137
Tie-breaker algorithm, 104–8, 136
Tightly coupled machine, 9
Tiling, of loops, 612

Time server. *See also* Interval timer
using remote procedure call, 365–67
using rendezvous, 378
Timestamp, 453
Token-passing algorithm, 423, 457–65
decentralized dining philosophers, 467–71
distributed mutual exclusion, 457–60
termination detection, 460–65
Topology of a network problem, 448
Total correctness, 43, 59
Tournament barrier, 146
Transformations, in parallelizing compilers, 608
Transputer machine, 329
Traveling salesman problem, 479–80
Tree code, 570
Tree-structured barrier, 118–20
Triangular matrix, 574
Triple, in a programming logic, 59
Tuple space, 334. *See also* Linda language
Turing Plus language, 251

UMA multiprocessor, 6
Unconditional atomic action, 55
Unconditionally fair scheduling policy, 75
Uniform memory access time, 6
Unisex bathroom problem, 200
Unix pipes, 20
Unroll and jam transformation, 610

V cycle, 552
`V` operation, 154–55. *See also* Semaphore
meaning of name, 188
Visualization tools, 634

W cycle, 552
Wait-free synchronization, 138
`wait` statement, 207–9
implementation of, 281, 285
Water molecule problem, 198, 257
Wave front synchronization, 580, 612
Weakened assertions, 66–68
Weakly fair scheduling policy, 75
Weighted voting, 392–93, 413
Work farm model, 139. *See also* Bag-of-tasks paradigm; Manager/workers paradigm
Writers' preference, 177
Write set, 45, 65

ZPL language, 615, 621–23, 640

Amazon
Exhibit 1005
Page 352

Parallel Programming/Concurrent Programming/Operating Systems

Foundations of Multithreaded, Parallel, and Distributed Programming

Gregory R. Andrews, The University of Arizona

Greg Andrews has produced a first-rate textbook for students that also serves as an authoritative and highly readable reference for professional software developers. I recommend it to anyone interested in stepping past the particular constructs found, for example, in the Java or Ada programming languages or the POSIX threads library, to gain a deeper understanding of underlying programming principles and practices. Andrews succeeds in capturing the conceptual unity of concurrent, distributed, and parallel programming, while still conveying the diversity of expression, specialized algorithms, and design and analysis techniques associated with each.

—**Professor Doug Lea**, State University of New York at Oswego, and author of *Concurrent Programming in Java: Design Principles and Patterns, Second Edition*

Andrews does a very good job of presenting the different methods of implementing concurrent systems at both low- and high-levels.

—**Professor Dyke Stiles**
Utah State University

The clarity of explanations and the use of the same problem in different paradigms to compare and contrast implementations, [as well as] the coded examples and accompanying explanations, are excellent.

—**Professor Janet Hartman**
Illinois State University

ADDISON-WESLEY

Addison-Wesley is an imprint of Addison Wesley Longman, Inc.

X0039ETGJV
FOUNDATIONS OF MULTITHREADED, PA
Used, Good

ramming
USED - ACCEPTABLE

Fulfilled by Amazon

Amazon
Exhibit 1005
Page 353

APPENDIX 1012A

Adaptive cluster computing using JavaSpaces

Publisher: IEEE

Cite This PDF

J. Batheja ; M. Parashar All Authors

9 Paper Citations
3 Patent Citations
20 Full Text Views

Abstract

Document Sections

1. Introduction
2. A Framework for Adaptive Parallel Computing on Clusters
3. Experimental Evaluation of the Framework
4. Conclusions

Authors

Figures

Abstract:
We present the design, implementation and evaluation of a framework that uses JavaSpaces [1] to support this type of opportunistic adaptive parallel/distributed computing over networked clusters in a non-intrusive manner. The framework targets applications exhibiting coarse-grained parallelism and has three key features: (1) portability across heterogeneous platforms, (2) minimal configuration overheads for participating nodes, and (3) automated system state monitoring (using SNMP) to ensure nonintrusive behavior. Experimental results presented in this paper demonstrate that for applications that can be broken into coarse-grained, relatively independent tasks, the opportunistic adaptive parallel computing framework can provide performance gains. Furthermore, the results indicate that monitoring and reacting to the current system state minimizes the intrusiveness of the framework.

Published in: Proceedings 2001 IEEE International Conference on Cluster Computing

Date of Conference: 14-17 Oct. 2001

DOI: 10.1109/CLUSTR.2001.959993

Date Added to IEEE *Xplore*: 07 August 2002

Publisher: IEEE

Print ISBN:0-7695-1116-3

Conference Location: Las Vegas, NV, USA

More Like This

Matlab and Java based virtual and remote laboratories for control engineering
2009 17th Mediterranean Conference on Control and Automation
Published: 2009

A Case Study on the Relationship between Code Ownership and Refactoring Activities in a Java Software System
2016 IEEE/ACM 7th International Workshop on Emerging Trends in Software Metrics (WETSoM)
Published: 2016

Show More

IEEE Authors:
Increase Your Research Impact

Feedback

Authors
Figures
References
Citations
Keywords
Metrics
Footnotes

Date of Conference: 14-17 Oct. 2001
DOI: 10.1109/CLUSTR.2001.959993
Date Added to IEEE *Xplore*: 07 August 2002
Publisher: IEEE
Print ISBN:0-7695-1116-3
Conference Location: Las Vegas, NV, USA

First Page of the Article

Adaptive Cluster Computing using JavaSpaces[1]

Jyoti Batheja and Manish Parashar
The Applied Software Systems Laboratory
Department of Electrical and Computer Engineering, Rutgers University
94 Brett Road, Piscataway, NJ 08854
{jbatheja,parashar}@caip.rutgers.edu

Abstract

In this paper we present the design, implementation and evaluation of a framework that uses JavaSpaces [1] to support this type of opportunistic adaptive parallel/distributed computing over networked clusters in a non-intrusive manner. The framework targets applications exhibiting coarse-grained parallelism and has three key features: (1) portability across heterogeneous platforms, (2) minimal configuration overheads for participating nodes, and (3) automated system state monitoring (using SNMP) to ensure non-intrusive behavior. Experimental results presented in this paper demonstrate that for applications that can be broken into coarse-grained, relatively independent tasks, the opportunistic adaptive parallel computing framework can provide performance gains. Furthermore, the results indicate that monitoring and reacting to the current system state minimizes the intrusiveness of the framework.

Keywords: *Adaptive cluster computing, Parallel/Distributed computing, JavaSpaces, Jini, SNMP.*

1. Introduction

resources, the configurations and capabilities of these resources, and the available software, services and tools on the systems. This heterogeneity must be hidden from the application and addressed in a seamless manner, so that the application can uniformly exploit available parallelism. 2) **Intrusiveness:** A local user should not be able to perceive that local resources are being stolen for foreign computations. Furthermore, inclusion into the framework should require minimal modifications to existing (legacy) code or standard practices. 3) **System configuration and management overhead:** Incorporating a new resource into the cycle stealing resource cluster may require system configuration and software installation. These modifications and overheads must be minimized so that clusters can be expanded on the fly to utilize all available resources. 4) **Adaptability to system and network dynamics:** The availability and state of system and network resources in a cluster can be unpredictable and highly dynamic. These dynamics must be handled to ensure reliable application execution. 5) **Security and privacy:** Secure and safe access to resources in the cluster must be guaranteed so as to provide assurance to users making their systems available for external computations. Policies must be defined and

Increase Your Research Impact
Add executable code to your research articles
UPLOAD YOUR CODE
CODE OCEAN
IEEE
Feedback

Proceedings of the 2001 IEEE International Conference on Cluster Computing (CLUSTER'01)
0-7695-1116-3/02 $17.00 © 2002 IEEE

IEEE COMPUTER SOCIETY

Hide First Page Preview

1. Introduction

Traditional High Performance Computing (HPC) is based on massively parallel processors, supercomputers or high-end workstation clusters connected by high-speed networks. These resources are relatively expensive, and are dedicated to specialized parallel and distributed applications. Exploiting available idle resources in a networked system can provide a more cost effective alternative for certain applications. However, there are a number of challenges that must be addressed before such opportunistic adaptive cluster computing can be a truly viable option. These include: 1) Heterogeneity: Cluster environments are typically heterogeneous in the type of resources, the configurations and capabilities of these resources, and the available software, services and tools on the systems. This heterogeneity must be hidden from the application and addressed in a seamless manner, so that the application can uniformly exploit available parallelism. 2) Intrusiveness: A local user should not be able to perceive that local resources are being stolen for foreign computations. Furthermo[illegible]hould require minimal modifications to existing (legacy) code or standard practices. 3) System configuration and management overhead: Incorporating a new resource into the cycle stealing resource cluster may require system configuration and software installation. These modifications and overheads must be minimized so that clusters can be expanded on the fly to utilize all available resources. 4) Adaptability to system and network dynamics: The availability and state of system and network

Sign in to Continue Reading

Authors
Figures
References
Citations

Feedback

Amazon
Exhibit 1005
Page 357

References

Citations

Keywords

Metrics

Footnotes

IEEE Personal Account

CHANGE USERNAME/PASSWORD

Purchase Details

PAYMENT OPTIONS

VIEW PURCHASED DOCUMENTS

Profile Information

COMMUNICATIONS PREFERENCES

PROFESSION AND EDUCATION

TECHNICAL INTERESTS

Need Help?

US & CANADA: +1 800 678 4333

WORLDWIDE: +1 732 981 0060

CONTACT & SUPPORT

Follow

About IEEE *Xplore* | Contact Us | Help | Accessibility | Terms of Use | Nondiscrimination Policy | IEEE Ethics Reporting | Sitemap | Privacy & Opting Out of Cookies

A not-for-profit organization, IEEE is the world's largest technical professional organization dedicated to advancing technology for the benefit of humanity.

© Copyright 2021 IEEE - All rights reserved. Use of this web site signifies your agreement to the terms and conditions.

Feedback

Proceedings

2001 IEEE International Conference on Cluster Computing

Amazon
Exhibit 1005
Page 359

Proceedings

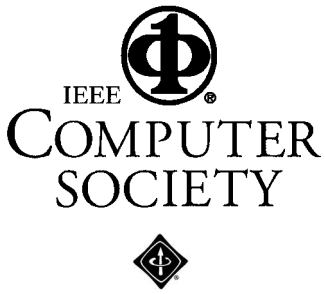
2001 IEEE International Conference on Cluster Computing

8–11 October 2001

Newport Beach, California, USA

Edited by
Daniel S. Katz, Thomas Sterling, Mark Baker,
Larry Bergman, Marcin Paprzycki, and Rajkumar Buyya

Sponsored by the
IEEE Computer Society Task Force on Cluster Computing



Los Alamitos, California
Washington · Brussels · Tokyo

Amazon
Exhibit 1005
Page 360

Copyright © 2001 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved

Copyright and Reprint Permissions: Abstracting is permitted with credit to the source. Libraries may photocopy beyond the limits of US copyright law, for private use of patrons, those articles in this volume that carry a code at the bottom of the first page, provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923.

Other copying, reprint, or republication requests should be addressed to: IEEE Copyrights Manager, IEEE Service Center, 445 Hoes Lane, P.O. Box 133, Piscataway, NJ 08855-1331.

The papers in this book comprise the proceedings of the meeting mentioned on the cover and title page. They reflect the authors' opinions and, in the interests of timely dissemination, are published as presented and without change. Their inclusion in this publication does not necessarily constitute endorsement by the editors, the IEEE Computer Society, or the Institute of Electrical and Electronics Engineers, Inc.

IEEE Computer Society Order Number PR01116
ISBN 0-7695-1116-3
ISBN 0-7695-1118-X (microfiche)
Library of Congress Number 2001093692

Additional copies may be ordered from:

IEEE Computer Society
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1314
Tel: + 1 714 821 8380
Fax: + 1 714 821 4641
http://computer.org/
csbooks@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: + 1 732 981 0060
Fax: + 1 732 981 9667
http://shop.ieee.org/store/
customer-service@ieee.org

IEEE Computer Society
Asia/Pacific Office
Watanabe Bldg., 1-4-2
Minami-Aoyama
Minato-ku, Tokyo 107-0062
JAPAN
Tel: + 81 3 3408 3118
Fax: + 81 3 3408 3553
tokyo.ofc@computer.org

Editorial production by Anne Jacobs

Cover art production by Joe Daigle/Studio Productions

Printed in the United States of America by The Printing House



CLUSTER2001

Table of Contents

Message from the Task Force Co-Chairs xi

Message from the Conference General Co-Chairs xiii

Message from the Program Chairs xv

IEEE Cluster 2001 Conference Organization Committee xvii

Cluster 2001 Technical Program Committee xix

Keynote

"Architecture Recapitulates Phylogeny": How Scalability Requires Specialization 3
S. Oberlin

Networking I

Early Experiences with the Myricom 2000 Switch on an SMP Beowulf-Class Cluster for Unstructured Adaptive Meshing 7
C. D. Norton and T. A. Cwik

Using Multirail Networks in High-Performance Clusters 15
S. Coll, E. Frachtenberg, F. Petrini, A. Hoisie, and L. Gurvits

I/O

Next Generation Parallel Virtual File System (invited) 27
W. Ligon

Experiences with Oasis+: A Fault Tolerant Storage System 29
D. Watson, Y. Luo, and B. D. Fleisch

Clusterfile: A Flexible Physical Layout Parallel File System 37
F. Isailă and W. F. Tichy

Cluster Management I

High Performance Computing with Microsoft Windows 2000 (invited) 47
D. A. Lifka

GulfStream—A System for Dynamic Topology Management in Multi-Domain Server Farms 55
S. A. Fakhouri, G. Goldszmidt, M. Kalantar, J. A. Pershing, and I. Gupta

Cluster Rolling Upgrade Using Multiple Version Support 63
E. T. Roush

Amazon
Exhibit 1005
Page 362

Alternative Hardware

Dense Computing with Transmeta's Crusoe (invited) 73
D. Gaudet

PVL: An Object Oriented Software Library for Parallel Signal Processing (invited) 74
E. Rutledge and J. Kepner

Numerically-Intensive "Plug-and-Play" Parallel Computing 75
D. E. Dauger and V. K. Decyk

Applications I

Parallel Standard Cell Placement on a Cluster of Workstations 85
F. H. Khundakjie, P. H. Madden, N. B. Abu-Ghazaleh, and M. C. Yildiz

Stingray: Cone Tracing Using a Software DSM for SCI Clusters 95
A. Meyer and E. Cecchet

Parallel and Adaptive Reduction of Hyperspectral Data to Intrinsic Dimensionality 102
T. El-Ghazawi, S. Kaewpijit, and J. Le Moigne

Scheduling

Workload Management: More Than Just Job Scheduling (invited) 113
J. P. Jones, B. Nitzberg, and B. Henderson

Scheduling Parallel Adaptive Applications in Networks of Workstations and Clusters of Processors 116
D. Kebbal, E.-G. Talbi, and J.-M. Geib

Sharing Partitionable Workloads in Heterogeneous NOWs: Greedier Is Not Better 124
A. L. Rosenberg

Plenary Talks

High Performance Computing and Trends: Connected Computational Requirements with Computing Resources 135
J. Dongarra

Future Directions for the Scyld Advanced Beowulf Management System 136
D. Becker

OSCAR and the Beowulf Arms Race for the "Cluster Standard" 137
S. L. Scott

Posters

CPPvm—Parallel Programming in C++ 141
S. Görzig

Amazon
Exhibit 1005
Page 363

Dynamic Cluster Configuration and Management Using JavaSpaces 145
K. A. Hawick and H. A. James

Communication with Threads in Software DSMs 149
W. Hu, G. Shi, and F. Zhang

IEEE 1394 and RFC 2734: A Viable HSI for Hypercubes 155
K. L. Keville

Hierarchical Cluster for Scalable Web Servers 156
J. Hong and D. Kim

A Low-Cost, High-Density Mounting System for Computer Clusters 157
J. F. Kolen and T. Hutcheson

Table-Comparison Prefetching in VIA-Based Parallel File System 163
Y.-Y. Lee, C.-Y. Kim, and D.-W. Seo

Cluster Trend: Data Speaking 168
P. Liu and Z. Du

Performance Evaluation of Parallel File Systems for PC Clusters and ASCI Red 172
S. Garg and J. Mache

A High-Availability Clustering Architecture with Data Integrity Guarantees 178
R. Rabbat, T. McNeal, and T. Burke

PVM Support for Clusters 183
P. L. Springer

A Distributed Object Model for Solving Irregularly Structured Problems on Cluster 187
Y. Sun and C.-L. Wang

Keynote

Towards High-Level Programming Support for Scientific Computing on Clusters 193
H. P. Zima

Networking II

Performance and Analysis of the PCI-DDC Remote-Write Implementation 197
É. Renault and P. David

GMSOCKS—A Direct Sockets Implementation on Myrinet 204
M. Fischer

A Reconfigurable Extension to the Network Interface of Beowulf Clusters 212
K. D. Underwood, R. R. Sass, and W. B. Ligon, III

Middleware I

Intelligence in High Performance Network Interface Card:
Hardware vs. Software (invited) 225
B. Tourancheau and R. Westrelin

Amazon
Exhibit 1005
Page 364

An SCI-Based Software VIA System for PC Clustering 227
J. Shin, S.-W. Chung, and W. Hahn

A Parallel Programming Environment for a V-Bus Based PC-Cluster 235
S. S. Lim, Y. Paek, K. H. Park, and J. Hoeflinger

Cluster Management II

Cluster Software Tools: Beauty is in Simplicity (invited)..... 247
P. Uthayopas

Dynamic Allocation of Nodes on a Large Space-Shared Cluster 250
L. Kennicott and L. A. Fisk

NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters 258
P. M. Papadopoulos, M. J. Katz, and G. Bruno

Algorithms I

Graph Partitioning for Dynamic, Adaptive and Multi-Phase Scientific Simulations (invited) 271
K. Schloegel, G. Karypis, and V. Kumar

Approximation Algorithms for Data Distribution with Load Balancing of Web Servers 274
L.-C. Chen and H.-A. Choi

A Class of Loop Self-Scheduling for Heterogeneous Clusters 282
A. T. Chronopoulos, R. Andonie, M. Benche, and D. Grosu

Applications II

A Visualization Tool for Analyzing Cluster Performance Data..... 295
R. Haynes, P. Crossno, and E. Russell

Semi-Continuous Transmission in Cluster-Based Video Servers..... 303
S. Irani and N. Venkatasubramanian

Building a Scalable Web Server with Global Object Space Support on Heterogeneous Clusters..... 313
G. Chen, C.-L. Wang, and F. C. M. Lau

Programming Models

Adaptive Cluster Computing Using JavaSpaces..... 323
J. Batheja and M. Parashar

Programmability and Performance of M++ Self-Migrating Threads..... 331
M. Fukuda, N. Suzuki, L. M. Campos, and S. Kobayashi

Cluster-Based Computing with Active, Persistent Objects on the Web 341
F. Sommers, S. Ghandeharizadeh, and S. Gao

Amazon
Exhibit 1005
Page 365

Plenary Talks

Cluster and Grid Superservers: The Dawning Experiences in China 351
Z. Xu, N. Sun, D. Meng, and W. Li

Teragrids: The Future of Terascale Cluster Computing 359
D. A. Reed

Programming with MPI on Clusters 360
E. Lusk

Keynote

Recent Advances in Cluster Networks..... 365
C. L. Seitz

Networking III

Aspects of the InfiniBand™ Architecture (invited)..... 369
G. F. Pfister

DECK-SCI: High-Performance Communication and Multithreading for SCI Clusters..... 372
F. A. D. de Oliveira, R. B. Ávila, M. E. Barreto, P. O. A. Navaux, and C. A. F. De Rose

Dynamic Reconfiguration in High-Speed Computer Clusters..... 380
D. Avresky, N. Natchev, and V. Shurbanov

Middleware II

The Lightweight Protocol CLIC: Performance of an MPI Implementation on CLIC..... 391
A. F. Díaz, J. Ortega, A. Cañas, F. J. Fernández, and A. Prieto

SOVIA: A User-Level Sockets Layer over Virtual Interface Architecture..... 399
J.-S. Kim, K. Kim, and S.-I. Jung

Reliably Locking System V Shared Memory for User Level Communication in Linux..... 409
F. Seifert and W. Rehm

Algorithms II

The Master-Slave Paradigm with Heterogeneous Processors..... 419
O. Beaumont, A. Legrand, and Y. Robert

An Efficiency and Scalability Model for Heterogeneous Clusters..... 427
L. Pastor and J. L. Bosque Orero

Adaptive Runtime Partitioning of AMR Applications on Heterogeneous Clusters..... 435
S. Sinha and M. Parashar

Amazon
Exhibit 1005
Page 366

Plenary Talks

The Grid, Virtual Organizations, and Problem-Solving Environments 445
D. W. Walker

Scalable Cluster Computing—Opportunities and Challenges 448
W. J. Camp

SCore: An Integrated Cluster System Software Package for High Performance Cluster Computing..... 449
A. Hori

Tutorials

Advanced Cluster Programming with MPI 453
B. Gropp

Design and Implementation of Highly Available Linux Clusters..... 454
I. Haddad and F. Rossi

The Scyld Beowulf System..... 455
D. Becker

The Parallel Virtual File System for Commodity Clusters..... 456
W. Ligon and R. Ross

Emerging Technologies for MultiCluster/Grid Computing 457
R. Buyya and M. Baker

Author Index..... 459

Amazon
Exhibit 1005
Page 367

APPENDIX 1012B

Adaptive Cluster Computing using JavaSpaces[1]

Jyoti Batheja and Manish Parashar
The Applied Software Systems Laboratory
Department of Electrical and Computer Engineering, Rutgers University
94 Brett Road, Piscataway, NJ 08854
{jbatheja,parashar}@caip.rutgers.edu

Abstract

In this paper we present the design, implementation and evaluation of a framework that uses JavaSpaces [1] to support this type of opportunistic adaptive parallel/distributed computing over networked clusters in a non-intrusive manner. The framework targets applications exhibiting coarse-grained parallelism and has three key features: (1) portability across heterogeneous platforms, (2) minimal configuration overheads for participating nodes, and (3) automated system state monitoring (using SNMP) to ensure non-intrusive behavior. Experimental results presented in this paper demonstrate that for applications that can be broken into coarse-grained, relatively independent tasks, the opportunistic adaptive parallel computing framework can provide performance gains. Furthermore, the results indicate that monitoring and reacting to the current system state minimizes the intrusiveness of the framework.

Keywords: *Adaptive cluster computing, Parallel/Distributed computing, JavaSpaces, Jini, SNMP.*

1. Introduction

Traditional High Performance Computing (HPC) is based on massively parallel processors, supercomputers or high-end workstation clusters connected by high-speed networks. These resources are relatively expensive, and are dedicated to specialized parallel and distributed applications. Exploiting available idle resources in a networked system can provide a more cost effective alternative for certain applications. However, there are a number of challenges that must be addressed before such opportunistic adaptive cluster computing can be a truly viable option. These include: 1) **Heterogeneity:** Cluster environments are typically heterogeneous in the type of resources, the configurations and capabilities of these resources, and the available software, services and tools on the systems. This heterogeneity must be hidden from the application and addressed in a seamless manner, so that the application can uniformly exploit available parallelism. 2) **Intrusiveness:** A local user should not be able to perceive that local resources are being stolen for foreign computations. Furthermore, inclusion into the framework should require minimal modifications to existing (legacy) code or standard practices. 3) **System configuration and management overhead:** Incorporating a new resource into the cycle stealing resource cluster may require system configuration and software installation. These modifications and overheads must be minimized so that clusters can be expanded on the fly to utilize all available resources. 4) **Adaptability to system and network dynamics:** The availability and state of system and network resources in a cluster can be unpredictable and highly dynamic. These dynamics must be handled to ensure reliable application execution. 5) **Security and privacy:** Secure and safe access to resources in the cluster must be guaranteed so as to provide assurance to users making their systems available for external computations. Policies must be defined and enforced to ensure that external application tasks adhere to the limits and restrictions set on resource/data access and utilization.

Recent advances in opportunistic computing have followed two approaches, *Job level parallelism* and *Adaptive parallelism*. In the job level parallelism approach, entire application jobs are allocated to available idle resources for computation, and are migrated across resources as resources become unavailable. The Condor system [2] supports cluster based job level parallelism. In the adaptive approach, the available processors are treated as part of a dynamic resource pool. Each processor in the pool aggressively competes for application tasks. This

[1] The research presented in this paper is based upon work supported by the National Science Foundation under Grant Number ACI 9984357 (CAREERS) awarded to Manish Parashar.

0-7695-1116-3/01 $10.00 © 2001 IEEE

Amazon
Exhibit 1005
Page 369

approach targets applications that can be decomposed into independent tasks. Adaptive computing techniques can be cluster based or web based. Cluster based systems exploit available resources within a local networked cluster. Web based approach extends this model to exploit resources connected over the Internet. Systems supporting adaptive parallelism include cluster-based systems such as Piranha [3][4], Atlas [5] and ObjectSpace/Anaconda [6], and web based systems such as Charlotte [7][8], Javelin [9], and ParaWeb [10].

This paper presents the design, implementation and evaluation of a framework for adaptive and opportunistic cluster computing based on JavaSpaces that address the issues outlined above. The framework has three key features: (1) portability across heterogeneous platforms, (2) minimal configuration overheads and runtime class loading at the participating nodes, and (3) automated system sate monitoring (using SNMP – Simple Network Management Protocol [11][12]) to ensure non-intrusive behavior.

The rest of this paper is organized as follows. Section 2 describes the architecture and operation of the proposed framework. Section 3 presents an experimental evaluation of the framework. Section 4 presents our conclusions and outlines current and future work.

2. A Framework for Adaptive Parallel Computing on Clusters

The framework presented in this paper employs JavaSpaces to facilitate adaptive (opportunistic) master-worker parallel computing on networked clusters. It targets applications that are sufficiently complex and require parallel computing, that are divisible into relatively coarse-grained subtasks that can be solved independently, and where the subtasks have small input/output sizes.

2.1. Framework Architecture

A schematic overview of the framework architecture is shown in Figure 2. The framework consists of 3 key components: the Master Module, the Worker Module, and the Network Management Module.

JavaSpaces and Parallel Computation: JavaSpaces is a Java implementation of a tuple-space [3][4] system, and is provided as a Jini service. A JavaSpaces is a shared, network accessible repository for Java objects [13], and provides a programming model that views applications as a collection of processes cooperating via the flow of objects into and out of one or more spaces. The JavaSpaces API leverages the Java type semantics and provides operations to store, retrieve and lookup Java objects in the space.

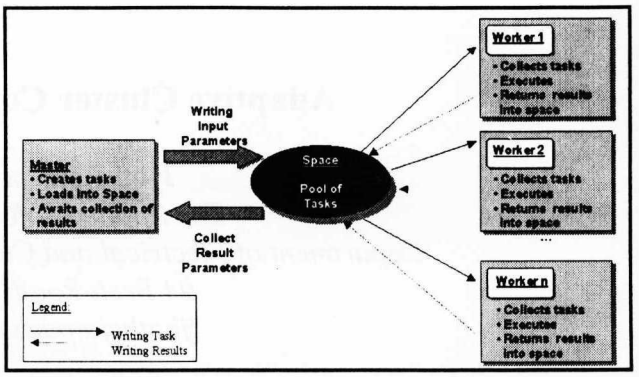


Figure 1. Master-worker parallel computing using JavaSpaces

JavaSpaces naturally supports the master-worker parallel/distributed-computing paradigm. In this paradigm (see Figure 1), the application is distributed across available worker nodes using the *bag-of-tasks* model. The master produces independent application tasks and puts them into the space. The worker consumes the tasks from the space, computes on these tasks, and places results back into the space. This approach supports coarse-grained applications that can be partitioned into relatively independent tasks.

Master Module: The master module runs as an application level process on the master processor. It hosts the JavaSpaces service and registers it with the Jini substrate. The master module defines the problem domain for a given application. It decomposes the application into independent tasks that are JavaSpaces enabled[2], and places these tasks into the space.

Worker Module: The worker module runs as an application level process on the workers nodes. It is a thin module and is configured at runtime using the *Remote Node Configuration Engine*, which enables worker classes, providing the solution content for the application, to be loaded at runtime. This minimizes the overheads of deploying application code. The worker module operation is controlled by the network management module using the *Rule-Base Protocol* as described later. Note that the workers need not be Jini aware in order to interact with the master module. Interaction between the master and the worker processes is via virtual, shared JavaSpaces

Network Management Module: The network management module serves two functions, viz. monitoring the state of worker machines, and providing a decision-making mechanism to facilitate the utilization of idle resources and ensure non-intrusive code execution. In order to exploit idle resources while maintaining non-

[2] JavaSpace required the Objects being passed across the Space to be in a Serializable format. In order to transfer an entry to or from a remote space, the proxy to the remote space implementation first serializes the fields and then transmits it into the space.

Amazon
Exhibit 1005
Page 370

intrusiveness at the remote nodes, it is critical that the framework monitors the state of the worker nodes, and uses this state information to drive the scheduling of tasks on workers. The *Monitoring Agent* component of the network management module performs this task. It monitors the state of registered workers and uses defined policies to decide on the worker's availability. The policies are maintained by the *Inference Engine* component and enforced using the *Rule-Base Protocol.*

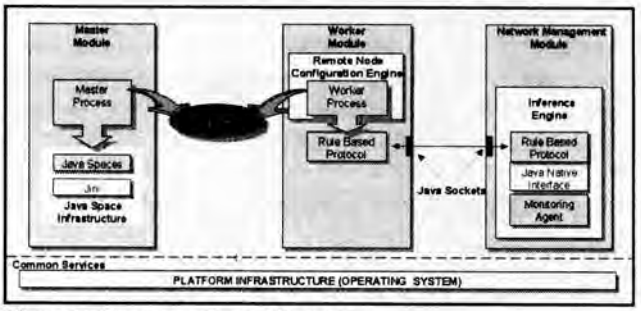


Figure 2. Framework architecture

The monitoring agent uses SNMP [11][12] to monitor remote worker nodes. The SNMP layer consists of two components: the manager component that runs on the SNMP server and the worker-agent component that runs on the worker nodes to be monitored. The monitoring agent uses JNI (Java Native Interface) to access the SNMP layer and query the worker-agents to get relevant system state parameters such as CPU load and available memory.

2.2. Framework Operation

The framework implements the master-worker pattern with JavaSpaces as the backbone. The overall operation of the framework consists of three potentially overlapping phases, viz. task-planning, compute, and result-aggregation. During the *task-planning phase*, the master process first decomposes the application problem into sub tasks. It then iterates through the application tasks, creates a task entry for each task, and writes the tasks entry into the JavaSpaces. During the *compute phase,* the worker process retrieves these tasks from the space using a simple value-based look up. Each task object is identified by a unique ID and the space in which it resides. If a matching task object is not available immediately, the worker process waits until one arrives. The worker classes are downloaded at runtime using the Remote Node Configuration Engine. Results obtained from executing the computations are returned to the space. Worker operation during the compute phase is monitored and managed by the network management module using the rule-base protocol as described in section 2.2.1. During the *result aggregation* phase, the master module removes results written into the space by the workers, and aggregates them into the final solution.

If the resource utilization on a worker node becomes intolerable, the network management module sends a stop/pause signal to the worker process. On receiving the signal, the worker completes the execution of the current task and returns its results into the space. It then enters the stop/pause state and does not accept tasks until it receives a start/resume signal.

Remote Node Configuration: Remote node configuration uses the dynamic class loading mechanism provided by the Java Virtual Machine, which supports locating and fetching the required class files, consulting the security policy, and defining the class object with the appropriate permissions. Required worker classes are downloaded from a web server residing at the master in the form of executable jar files.

Our modification of the network launcher [14] provides mechanisms to intercept calls from the inference engine component of the network management module and use them to signal the executing worker code. This enables the network management module to manage workers using the rule-base protocol. As preempting worker execution to process the signal may result in the current task being lost, the node configuration engine waits for the worker to complete its current task, and forwards the signal before the worker fetches the next task.

2.2.1. Dynamic Worker Management for Adaptive Cluster Computing. The SNMP client, which is part of the worker module, initiates the workers participation in the parallel computation by registering with the SNMP server at the network management module (see Figure 3).

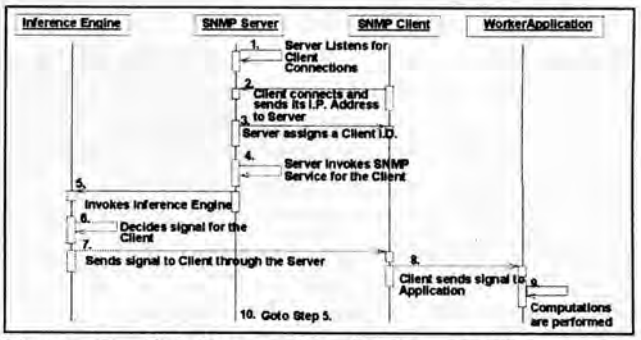


Figure 3. Rule-base protocol for adaptive worker management

The inference engine, also at the network management module, maintains a list of registered workers. It assigns a unique ID to the new worker and adds its IP address to the list. The SNMP server then continues to monitor the state of workers in its list.

The primary SNMP parameter monitored is the average worker CPU utilization. As these values are

Amazon
Exhibit 1005
Page 371

returned they are added to the respective entry in the worker list. Based on this return value and programmed threshold ranges, the inference engine makes a decision on the worker's current availability status and passes an appropriate signal back to the worker. Threshold values are based on heuristics. The rule-base currently defines 4 types of signals in response to the varying load conditions at a worker, viz. *Start, Stop, Pause* and *Resume*. Based on the signal it receives, the worker can be in 3 possible states: *Running, Paused,* or *Stopped*. The worker state transitions are shown in Figure 4 and are described below.

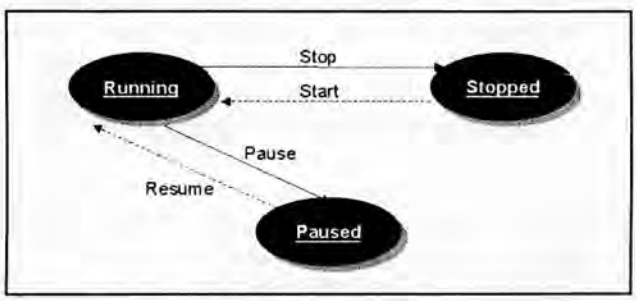


Figure 4. Worker state transition diagram

Running: The worker enters the running state in response to a *Start* or *Resume* signal. *Start* or *Resume* signals are sent when the CPU load at the worker is in the range of 0% - 25%. While in this state, worker is considered idle and can start participating in the parallel application. On receiving a *Start* signal, the worker initiates a new runtime process. The new thread first goes through the remote class loading phase and then starts off a worker thread for task execution. If the worker receives a *Resume* signal, however, it does not require loading of the worker classes since they are already loaded into the worker's memory. It simply removes the lock on the interrupted execution thread and resumes computations.

Stopped: The worker enters this state in response to a *Stop* signal. This may be due to a sustained increase in CPU load caused by a higher priority (possibly interactive) job being executed. The cutoff threshold value for the *Stopped* state is in the range of 50% to 100%. While in this state the node can no longer be used for computations. On receiving the *Stop* signal, the executing worker thread is interrupted and shutdown/cleanup mechanisms are initiated. The shutdown mechanism ensures that the currently executing task completes and its results are written into the space. After cleanup, the worker thread is killed and control returns to the parent process. The next time this worker becomes available, a transition to the *Running* state will require the worker classes to be reloaded.

Paused: The worker enters this state in response to the *Pause* signal. This state indicates that the worker node is experiencing increased CPU loads and is not currently idle, and hence it should temporarily not be used for computation. However, the load increase might be transient and the node could be reused for computation in the near future. Threshold values for the *Paused* state are in the range of 25% - 50%. Upon receiving this signal, the worker backs off, but unlike the stop state the back off is temporary, i.e. until it gets the resume signal. This minimizes worker initialization and class loading overheads for transient load fluctuations. As in the stop state, the pause goes into effect only after the worker writes the results of the currently executing task into the space. However the worker process is not destroyed in this state but only interrupts the execution until the resume signal is received hence bypassing the overhead associated with remote node configuration.

3. Experimental Evaluation of the Framework

The JavaSpaces-based opportunistic cluster-computing framework is experimentally evaluated using two "real-world" applications: (1) a financial application that uses Monte Carlo (MC) simulation for option pricing, and (2) a scientific ray tracing application. The evaluation consists of three experiments. The objective of the first experiment is to study the scalability of the application and our framework, and to demonstrate the potential advantage of using clusters for parallel computing. The second experiment measures the costs of adapting to system state. It measures the overheads of monitoring the workers, signaling, and state-transitions at the workers. We used a set of synthetic load generators to simulate dynamic load conditions at different worker nodes. Finally, the third experiment demonstrates the ability of our framework to adapt to the cluster dynamics.

The experiments are conducted on PC clusters running Windows NT (version 4.0). The parallel ray tracing applications are evaluated on a five PC cluster, with an 800MHz. Intel Pentium III processors and 256 MB RAM. The option-pricing scheme is evaluated on a larger cluster with thirteen PCs. The PCs in this cluster had 300 MHz. processors and 64MB RAM. Due to the high memory requirements of the Jini infrastructure, the master module in both cases runs on an 800 MHz. Intel Pentium III processor PC with 256 MB RAM.

3.1. Parallel Monte Carlo Simulation for Stock Option Pricing

A stock option is a derivative, that is, its pricing value is derived from something else such as the underlying security, the option type (call or put), the strike price, and an expiration date. These financial terms are explained in greater depth in [15]. Parameters such as varying interest

Amazon
Exhibit 1005
Page 372

rates and complex contingencies can prohibit analytical computation of options and other derivative prices. Monte Carlo (MC) simulation, using statistical properties of assumed random sequences is an established tool for pricing of derivative securities. In our implementation we use MC simulations, based on the Broadie and Glasserman MC algorithm [16], to model the behavior of options and account for the various factors affecting its price.

Implementation Overview: The simulation domain is divided into independent tasks and the MC simulations are performed in parallel on these tasks. The total number of simulations is defined by an external input. Each MC task consists of two iterations, the first one obtains a high estimate and the second one obtains a low estimate. For the experimental evaluation presented below, the number of simulations was set to 5000. The problem domain is divided into 50 tasks, each comprising of 100 simulations. As each MC simulation consists of two independent iterations, a total of 100 sub-tasks were created and put into the JavaSpaces. The workers take the task from the space and perform the MC simulations.

3.2. Parallel Ray Tracing

Ray tracing [17] is an image generation technique that simulates light behavior in a scene by following light rays from an observer as they interact with the scene and the light sources. Ray tracing algorithms estimate the intensity and wavelengths of light entering the lens of a virtual camera in a simulated environment. The quantities are estimated at discreet points in the image plane that correspond to pixels. Such applications are ideal candidates for the replicated-worker pattern as they are made up of a number of independent and computationally identical tasks.

Implementation Overview: Rendering an image involves iterating through all the pixels in the plane and computing a color value for each pixel. The computation is identical for all pixels - only the parameters describing the pixel's position differ. In our experiments the 600X600 image plane was divided into rectangular slices of 25X600 thus creating 24 independent tasks. The input for each task consisted of the four coordinates describing the region of computation. The output produced by each task was relatively large, consisting of an array of pixel values.

3.3. Experimental Results

3.3.1. Scalability Analysis. This experiment measures the overall scalability of the application and the framework. In this experiment, we measure the Max Worker Time, Task Planning Time, Task Aggregation Time, and Parallel Time for the different applications as the number of worker nodes is increased. The computation time at a worker is measured from the time it first accesses a task from the space to the time it puts its final result back into the space. Max Worker Time is the maximum of the worker computation times among all workers participating in the application. Task Planning Time is measured at the master process and is the time required for the task-planning phase. This involves dividing the application into tasks and placing these tasks into the space. Task Aggregation Time is also measured at the master process and is the time required for collecting the results returned by the workers from the space, and aggregating them into a meaningful solution. The task aggregation time is expected to follow the maximum worker time, since the master needs to wait for the last task to complete before it finishes aggregating the results. Finally, Parallel Time is measured at the master process and is the time required for the entire application computation from start to finish. It includes the times listed above.

Parallel Monte Carlo Simulation for Stock Option Pricing: The results of the scalability experiment for the option pricing application are plotted in Figure 5.

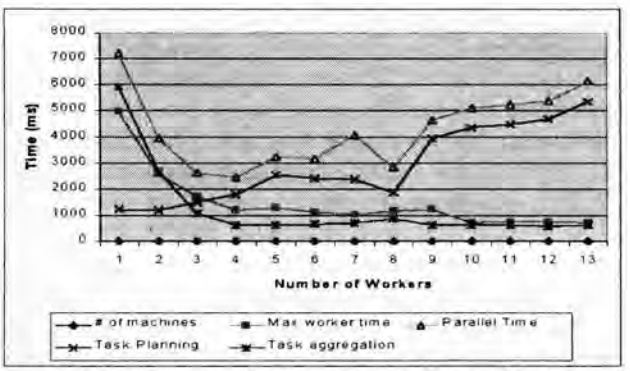


Figure 5. Scalability analysis - option pricing application

As seen in the figure, there is an initial speedup as the number of workers is increased to 4. The speedup deteriorates after that. Initially, as the number of workers increase, the total tasks are more evenly distributed across the available workers causing the maximum worker time to decrease. As expected, the initial part of the Parallel Time curve (up to 4 processors) closely follows the Maximum Worker Time curve. As the number of workers increase beyond 4, the amount of work is no longer sufficient to keep the workers busy, and the Task Planning Time now dominates Parallel Time. Here, the workers are able to complete their tasks and return the results to the space much faster than the master is able to create new tasks and put them into the space. As a result, the workers remain idle waiting for a task to become available, causing the scalability to deteriorate. This indicates that the framework favors computationally intensive coarse-grained tasks.

Amazon
Exhibit 1005
Page 373

Parallel Ray Tracing: The results of the scalability experiment for the parallel ray tracing application are plotted in Figure 6.

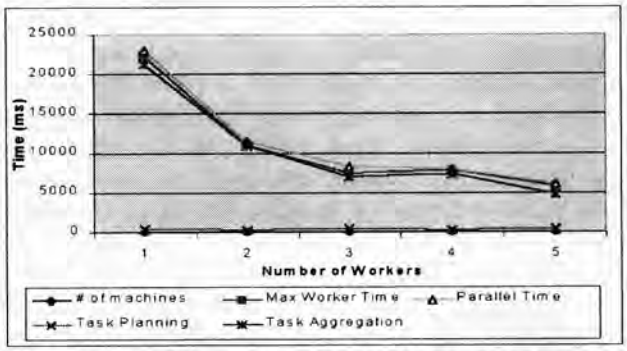


Figure 6. Scalability analysis – ray tracing application

As seen in the figure, Max Worker Time scales reasonably well for this application. This is because worker computations in this application are computationally intensive. The Parallel Time is dominated by the maximum worker time and results in good overall application scalability. Note that the Task Planning Time curve is constant at 500 ms. in this case. The Task Aggregation Time curve follows the Max Worker Time curve as expected. Embarrassingly parallel applications with coarse grained computationally intensive task, such as the parallel ray tracing application, scale well and are suited to the JavaSpaces-based cluster computing framework presented.

3.3.2. Adaptation Protocol Analysis. In this experiment, we analyze the overheads involved in signaling worker nodes and adapting to their current CPU load. In order to enable repeatable loading sequences for the experiments, we implemented two load simulators as part of the experimental setup. Load simulator 1 simulates different types of data transfers, such as RTP packets for voice traffic, HTTP traffic, and multimedia traffic over HTTP via Java sockets, originating at the workers. This load simulator was designed to raise the CPU usage level on the worker from 30% to 50%. The second load simulator (load simulator 2) raised the CPU utilization of the worker machines to 100%. The results of this experiment for the three applications are presented below. Each result consists of two parts: Part (a) plots the CPU usage history on the worker machine throughout the experiment. Part (b) provides an analysis of the signaling times, and lists the Client Signal and Worker Signal times. Client Signal time is the time at which the SNMP client on the worker machine receives the signal. Worker Signal time is the time taken for the signal to be interpreted by the worker and the required action completed. The key observation in this experiment is that the adaptation overhead is minimal in all cases. Furthermore, the large remote class loading overhead at the workers is avoided in the case of transient load increases using the pause/resume states.

Parallel Monte Carlo Simulation for Stock Option Pricing:

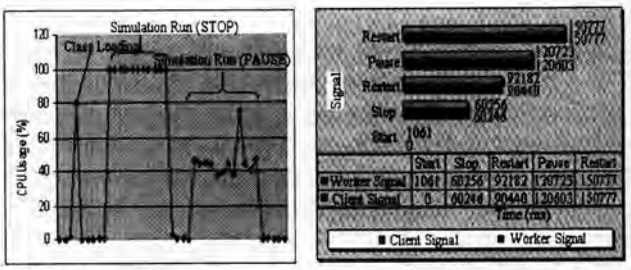


Figure 7. Adaptation protocol analysis - option pricing application (7a. worker CPU usage; 7b. worker reaction times)

The results of this experiment for the stock option pricing application are plotted in Figures 7(a) and 7(b). These plots show the worker behavior under simulated load conditions. In Figure 7(a), the peaks represent the times when the worker receives and reacts to the signals. The first peak is at 80% CPU usage and occurs when the worker is started (i.e. a *Start* signal). This sudden load increase is due to the remote loading of the worker implementation. The next peak at 100% CPU usage occurs when load simulator 2 is started on the worker. This causes a *Stop* signal to be sent to the worker and directs the worker to back off. Load simulator 2 is then stopped allowing the worker to once again become available and to do work. Load simulator 1 is now started causing the next peak at 46% CPU usage. A *Pause* signal is now sent to the worker temporarily suspending work execution. Finally, the simulator 1 is stopped causing a *Resume* signal to be sent to the worker. As seen in Figure 7(b) the worker reaction times to the signal received is minimal in each case.

Parallel Ray Tracing: The results of this experiment for the ray tracing application are plotted in Figures 8(a) and 8(b). As seen in Figure 8(a), the first peak is at 42% CPU usage and occurs when the worker is started (i.e. a *Start* signal). This sudden load increase is once again due to the remote loading of the worker implementation. The next peak at 100% CPU usage occurs when load simulator 2 is started on the worker. This causes a *Stop* signal to be sent to the worker and directs the worker to back off. Load simulator 2 is then stopped allowing the worker to once again become available and to do work. Load simulator 1 is now started raising the CPU load to 50% to 55%. A *Pause* signal is now sent to the worker temporarily suspending work execution. Finally, the simulator 1 is

Amazon
Exhibit 1005
Page 374

stopped causing a *Resume* signal to be sent to the worker. As seen in Figure 8(b) the worker reaction times to the signal received is once again minimal in each case. The Ray Tracing application is resource intensive as illustrated by the various intermittent peaks at 78 to 100% CPU usage. These spikes occur when the task is being computed at the worker node.

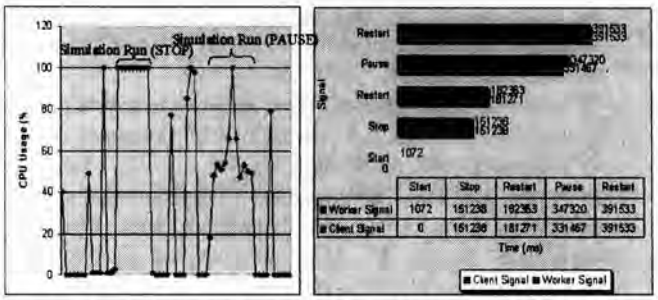


Figure 8. Adaptation protocol analysis - ray tracing application (8a. worker CPU usage; 8b. worker reaction times)

3.3.3. Analysis of Dynamic Worker Behavior Patterns under Varying Load Conditions: This experiment studies the dynamic behavior patterns at the workers under varying load conditions. It consists of three runs: In the first run none of the workers were loaded. In the second and third runs, the load simulator used to simulate high CPU loads are run on 25% and 50% of available workers respectively. Two plots are presented for each application run in this experiment. The first plot presents an analysis of the application behavior under the different load conditions. The four parameters measured are Maximum Worker Time, Maximum Master Overhead, Task Planning and Aggregation Time, and Total Parallel Time. Maximum Worker Time is the maximum value for worker computation time across all workers participating in the application. Maximum Master Overhead is the maximum instantaneous time taken by the master for task planning and aggregation for a particular task. Both the maximum worker time and the maximum master overhead are expected to remain constant for all three runs of the experiment. Task Planning and Aggregation Time is total time taken by the master during the task planning and aggregation phases. Finally Total Parallel Time is the time taken for the execution of the entire application and is measured at the master processor. Both the Task Planning and Aggregation Time and the Total Parallel Time are expected to increase with increased load on the worker machines. The second plot shows the work distribution among all the workers for the three cases.

Parallel Monte Carlo Simulation for Stock Option Pricing:

The results of this experiment for the option pricing application are plotted in Figures 9(a) and 9(b).

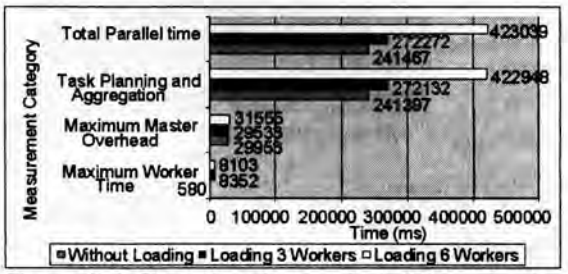


Figure 9(a). Execution time analysis (12 workers) - option pricing application

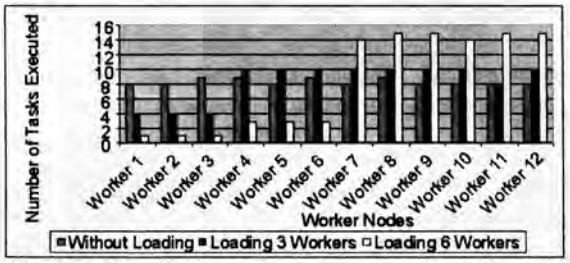


Figure 9 (b). Tasks executed per worker (12 workers) - option pricing application

As seen in Figure 9(a), as the number of workers being loaded increases, the total parallel computation time increases. This is because the computational tasks that would have been normally executed by the loaded workers are now offloaded and picked up by the available workers. The task planning and aggregation times also increase, as the master now has to wait for the worker with the maximum number of tasks to return all its results into the space. The maximum master overhead and the maximum worker time remains the same across all three runs as expected. Figure 9(b) illustrates how task scheduling adapts to the varying load conditions. It shows that the number of tasks executed by a worker depends on its current load. Loaded workers execute fewer tasks causing the available workers to execute larger number of tasks.

Parallel Ray Tracing: The results of this experiment for the ray tracing application are plotted in Figures 10(a) and 10(b). As expected (see Figure 10(a)), the total parallel computation time increases as the number of workers loaded increases. The task planning and aggregation times also increase as before. In this application, the Max Worker Time and the Maximum Master Overhead also increase. This increase was due to an increased latency experienced by the worker while returning tasks to the space. A possible cause for this latency is the system or network conditions at that instant. Note that Jini being a network-based protocol does not offer any real-time

Amazon
Exhibit 1005
Page 375

guarantees. Figure 10(b) illustrates the task distribution across the workers for the three cases.

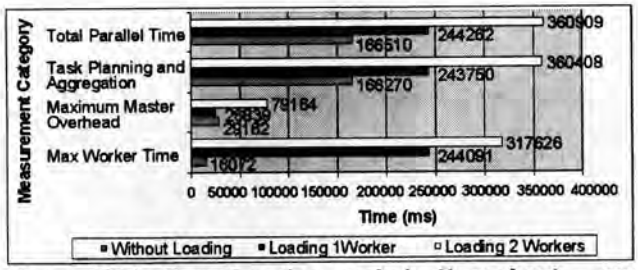


Figure 10(a). Execution time analysis (4 workers) – ray tracing application

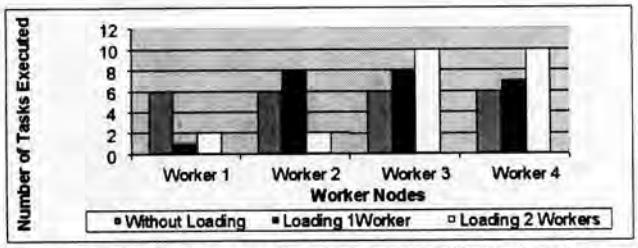


Figure 10(b). Tasks executed per worker (4 workers) – ray tracing application

4. Conclusions

This paper presented the design, implementation and evaluation of a framework for opportunistic parallel computing on networked clusters using JavaSpaces. The framework builds on Jini and JavaSpaces technologies. It provides support for global deployment of application code and remote configuration management of worker nodes, and uses an SNMP system state monitor to ensure non-intrusiveness. The experimental evaluation shows that the framework provides good scalability for coarse-grained tasks. Furthermore, using the system state monitor and triggering heuristics the framework can support adaptive parallelism and minimizes intrusiveness. The results also show that the signaling times between the worker and network management modules and the overheads for adaptation to cluster dynamics is insignificant.

We are currently investigating ways to reduce the overheads during task planning and allocation phases. Furthermore, several application-specific optimizations can be introduced to improve performance. As future work, we envision incorporating a distributed JavaSpaces model to avoid a single point of resource contention or failure. The Jini community is also investigating this area. Finally, the current implementation of the framework does not provide fault tolerance. We are also looking into comparing our approach with approaches that exhibit Job level parallelism through check pointing and process migration as well as comparing different paradigms for parallel processing. We are investigating the transaction management service provided by Jini to address this issue.

5. References

[1] Sun Microsystems. JavaSpaces, www.javasoft.com/products/javaspaces/specs/ (1998)

[2] M. Lizkow, M. Livney, and M. Mukta, "Condor: A hunter of idle workstations", In Proceedings of the 8th International conference on Distributed Computing Systems, San Jose, June1998, pp. 104-111.

[3] M. Carriero, E. Freeman, D. Gelernter, and D. Kaminsky, "Adaptive Parallelism and Piranha", IEEE Computer, Vol. 28, No. 1, January 1995, pp. 40-49.

[4] D. Gelernter and D. Kaminsky, "Supercomputing out of recycled garbage: Prelimnary Experience with Piranha" In Proceedings of the 6th ACM International Conference on Supercomputing, July 1992, pp. 417-427.

[5] J. Baldeschwieler, R. Blumofe, and E. Brewer, "ATLAS: An Infrastructure for Global Computing", In Proceedings of the 7th ACM SIGOPS European Workshop: Systems support for Worldwide Applications, September 1996, pp. 165-172.

[6] P. Ledru, "Adaptive Parallelism: An Early Experiment with Java™ Remote Method Invocation", Technical Report, CS Department, University of Alabama, 1997.

[7] A. Baratloo, M. Karaul, Z. Kedem, and P. Wyckoff, "Charlotte: Metacomputing on the Web", In Proceedings of the 9th ISCA International Conference on Parallel and Distributed Computing Systems (PDCS), September 1996, pp. 181-188.

[8] A. Baratloo, M. Karaul, H. Karl, and Z. Kedem, "An Infrastructure for Network Computing with Java Applets", Concurrency: Practice and Experience, Vol. 10, September 1998, pp. 1029-1041.

[9] B. Christiansen, P. Cappello, M. Ionescu, M. Neary, K. Schauser, and D. Wu, "Javelin: Internet-based parallel computing using Java", Concurrency: Practice and Experience, Vol.9, November 1997, pp. 1139-1160.

[10] T. Brecht, H. Sandhu, J. Talbott, and M. Shan, "ParaWeb: Towards worldwide supercomputing", In Proceedings of the 7th ACM SIGOPS European Workshop, September 1996, pp. 181-188.

[11] SNMP Documentation, http://www.snmpinfo.com

[12] J. Murray, "Windows NT SNMP", O'Reilly Publications, January 1998.

[13] E. Freeman, S. Hupfer, K. Arnold, "JavaSpaces Principles, Patterns, and Practice", Addison Wesley, June 1999.

[14] M. Noble, "Tonic: A Java TupleSpaces Benchmark Project", http://hea-www.harvard.edu/~mnoble/tonic/doc

[15] Glossary of Financial Terms http://www.centrex.xom/terms.html

[16] Broadie and Glasserman MC Algorithm for Option Pricing http://www.puc-rio.br/marco.ind/monte-carlo.html

[17] A. Heirich and J. Arvo, "A Competitive analysis of Load Balancing Strategies for Parallel Ray Tracing", In Journal of Supercomputing, 1998.

Amazon
Exhibit 1005
Page 376

Proceedings

2001 IEEE International Conference on Cluster Computing

8–11 October 2001

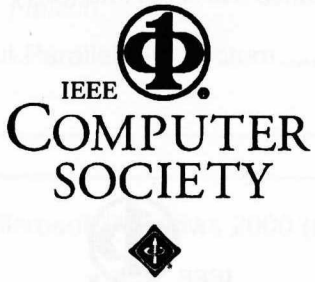
Newport Beach, California, USA

Edited by

Daniel S. Katz, Thomas Sterling, Mark Baker, Larry Bergman, Marcin Paprzycki, and Rajkumar Buyya

Sponsored by the

IEEE Computer Society Task Force on Cluster Computing



Los Alamitos, California

Washington · Brussels · Tokyo

Amazon
Exhibit 1005
Page 377

Proceedings



2001 IEEE International Conference on

Cluster Computing

Editors: Daniel S. Katz, Thomas Sterling, Mark Baker, Larry Bergman, Marcin Paprzycki, and Rajkumar Buyya

8–11 October 2001
California, USA

IEEE COMPUTER SOCIETY

IEEE

http://www.ieeetfcc.org

Sponsored by
IEEE
IEEE Computer Society and
IEEE Computer Society Task Force on Cluster Computing

TFCC
IEEE Task Force on Cluster Computing

Amazon
Exhibit 1005
Page 378

QA76.58
.I132
2001

Copyright © 2001 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved

Copyright and Reprint Permissions: Abstracting is permitted with credit to the source. Libraries may photocopy beyond the limits of US copyright law, for private use of patrons, those articles in this volume that carry a code at the bottom of the first page, provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923.

Other copying, reprint, or republication requests should be addressed to: IEEE Copyrights Manager, IEEE Service Center, 445 Hoes Lane, P.O. Box 133, Piscataway, NJ 08855-1331.

The papers in this book comprise the proceedings of the meeting mentioned on the cover and title page. They reflect the authors' opinions and, in the interests of timely dissemination, are published as presented and without change. Their inclusion in this publication does not necessarily constitute endorsement by the editors, the IEEE Computer Society, or the Institute of Electrical and Electronics Engineers, Inc.

IEEE Computer Society Order Number PR01116
ISBN 0-7695-1116-3
ISBN 0-7695-1118-X (microfiche)
Library of Congress Number 2001093692

Additional copies may be ordered from:

IEEE Computer Society
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1314
Tel: + 1 714 821 8380
Fax: + 1 714 821 4641
http://computer.org/
csbooks@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: + 1 732 981 0060
Fax: + 1 732 981 9667
http://shop.ieee.org/store/
customer-service@ieee.org

IEEE Computer Society
Asia/Pacific Office
Watanabe Bldg., 1-4-2
Minami-Aoyama
Minato-ku, Tokyo 107-0062
JAPAN
Tel: + 81 3 3408 3118
Fax: + 81 3 3408 3553
tokyo.ofc@computer.org

Editorial production by Anne Jacobs

Cover art production by Joe Daigle/Studio Productions

Printed in the United States of America by The Printing House



Amazon
Exhibit 1005
Page 379

CLUSTER2001

Table of Contents

Message from the Task Force Co-Chairs xi

Message from the Conference General Co-Chairs xiii

Message from the Program Chairs xv

IEEE Cluster 2001 Conference Organization Committee xvii

Cluster 2001 Technical Program Committee xix

Keynote

"Architecture Recapitulates Phylogeny": How Scalability Requires Specialization 3
S. Oberlin

Networking I

Early Experiences with the Myricom 2000 Switch on an SMP Beowulf-Class Cluster for Unstructured Adaptive Meshing 7
C. D. Norton and T. A. Cwik

Using Multirail Networks in High-Performance Clusters 15
S. Coll, E. Frachtenberg, F. Petrini, A. Hoisie, and L. Gurvits

I/O

Next Generation Parallel Virtual File System (invited) 27
W. Ligon

Experiences with Oasis+: A Fault Tolerant Storage System 29
D. Watson, Y. Luo, and B. D. Fleisch

Clusterfile: A Flexible Physical Layout Parallel File System 37
F. Isailă and W. F. Tichy

Cluster Management I

High Performance Computing with Microsoft Windows 2000 (invited) 47
D. A. Lifka

GulfStream—A System for Dynamic Topology Management in Multi-Domain Server Farms 55
S. A. Fakhouri, G. Goldszmidt, M. Kalantar, J. A. Pershing, and I. Gupta

Cluster Rolling Upgrade Using Multiple Version Support 63
E. T. Roush

LINDA HALL LIBRARY
Kansas City, Mo

Amazon
Exhibit 1005
Page 380

Alternative Hardware

Dense Computing with Transmeta's Crusoe (invited) 73
D. Gaudet

PVL: An Object Oriented Software Library for Parallel Signal Processing (invited) 74
E. Rutledge and J. Kepner

Numerically-Intensive "Plug-and-Play" Parallel Computing 75
D. E. Dauger and V. K. Decyk

Applications I

Parallel Standard Cell Placement on a Cluster of Workstations 85
F. H. Khundakjie, P. H. Madden, N. B. Abu-Ghazaleh, and M. C. Yildiz

Stingray: Cone Tracing Using a Software DSM for SCI Clusters 95
A. Meyer and E. Cecchet

Parallel and Adaptive Reduction of Hyperspectral Data to Intrinsic Dimensionality 102
T. El-Ghazawi, S. Kaewpijit, and J. Le Moigne

Scheduling

Workload Management: More Than Just Job Scheduling (invited) 113
J. P. Jones, B. Nitzberg, and B. Henderson

Scheduling Parallel Adaptive Applications in Networks of Workstations and Clusters of Processors 116
D. Kebbal, E.-G. Talbi, and J.-M. Geib

Sharing Partitionable Workloads in Heterogeneous NOWs: Greedier Is Not Better 124
A. L. Rosenberg

Plenary Talks

High Performance Computing and Trends: Connected Computational Requirements with Computing Resources 135
J. Dongarra

Future Directions for the Scyld Advanced Beowulf Management System 136
D. Becker

OSCAR and the Beowulf Arms Race for the "Cluster Standard" 137
S. L. Scott

Posters

CPPvm—Parallel Programming in C++ 141
S. Görzig

Amazon
Exhibit 1005
Page 381

Dynamic Cluster Configuration and Management Using JavaSpaces 145
K. A. Hawick and H. A. James

Communication with Threads in Software DSMs 149
W. Hu, G. Shi, and F. Zhang

IEEE 1394 and RFC 2734: A Viable HSI for Hypercubes 155
K. L. Keville

Hierarchical Cluster for Scalable Web Servers 156
J. Hong and D. Kim

A Low-Cost, High-Density Mounting System for Computer Clusters 157
J. F. Kolen and T. Hutcheson

Table-Comparison Prefetching in VIA-Based Parallel File System 163
Y.-Y. Lee, C.-Y. Kim, and D.-W. Seo

Cluster Trend: Data Speaking 168
P. Liu and Z. Du

Performance Evaluation of Parallel File Systems for PC Clusters and ASCI Red 172
S. Garg and J. Mache

A High-Availability Clustering Architecture with Data Integrity Guarantees 178
R. Rabbat, T. McNeal, and T. Burke

PVM Support for Clusters 183
P. L. Springer

A Distributed Object Model for Solving Irregularly Structured Problems on Cluster 187
Y. Sun and C.-L. Wang

Keynote

Towards High-Level Programming Support for Scientific Computing on Clusters 193
H. P. Zima

Networking II

Performance and Analysis of the PCI-DDC Remote-Write Implementation 197
É. Renault and P. David

GMSOCKS—A Direct Sockets Implementation on Myrinet 204
M. Fischer

A Reconfigurable Extension to the Network Interface of Beowulf Clusters 212
K. D. Underwood, R. R. Sass, and W. B. Ligon, III

Middleware I

Intelligence in High Performance Network Interface Card:
Hardware vs. Software (invited) 225
B. Tourancheau and R. Westrelin

Amazon
Exhibit 1005
Page 382

An SCI-Based Software VIA System for PC Clustering 227
J. Shin, S.-W. Chung, and W. Hahn

A Parallel Programming Environment for a V-Bus Based PC-Cluster 235
S. S. Lim, Y. Paek, K. H. Park, and J. Hoeflinger

Cluster Management II

Cluster Software Tools: Beauty is in Simplicity (invited) 247
P. Uthayopas

Dynamic Allocation of Nodes on a Large Space-Shared Cluster 250
L. Kennicott and L. A. Fisk

NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters 258
P. M. Papadopoulos, M. J. Katz, and G. Bruno

Algorithms I

Graph Partitioning for Dynamic, Adaptive and Multi-Phase Scientific Simulations (invited) 271
K. Schloegel, G. Karypis, and V. Kumar

Approximation Algorithms for Data Distribution with Load Balancing of Web Servers 274
L.-C. Chen and H.-A. Choi

A Class of Loop Self-Scheduling for Heterogeneous Clusters 282
A. T. Chronopoulos, R. Andonie, M. Benche, and D. Grosu

Applications II

A Visualization Tool for Analyzing Cluster Performance Data 295
R. Haynes, P. Crossno, and E. Russell

Semi-Continuous Transmission in Cluster-Based Video Servers 303
S. Irani and N. Venkatasubramanian

Building a Scalable Web Server with Global Object Space Support on Heterogeneous Clusters 313
G. Chen, C.-L. Wang, and F. C. M. Lau

Programming Models

Adaptive Cluster Computing Using JavaSpaces 323
J. Batheja and M. Parashar

Programmability and Performance of M++ Self-Migrating Threads 331
M. Fukuda, N. Suzuki, L. M. Campos, and S. Kobayashi

Cluster-Based Computing with Active, Persistent Objects on the Web 341
F. Sommers, S. Ghandeharizadeh, and S. Gao

Amazon
Exhibit 1005
Page 383

Plenary Talks

Cluster and Grid Superservers: The Dawning Experiences in China 351
Z. Xu, N. Sun, D. Meng, and W. Li

Teragrids: The Future of Terascale Cluster Computing 359
D. A. Reed

Programming with MPI on Clusters 360
E. Lusk

Keynote

Recent Advances in Cluster Networks 365
C. L. Seitz

Networking III

Aspects of the InfiniBand™ Architecture (invited) 369
G. F. Pfister

DECK-SCI: High-Performance Communication and Multithreading for SCI Clusters 372
F. A. D. de Oliveira, R. B. Ávila, M. E. Barreto, P. O. A. Navaux, and C. A. F. De Rose

Dynamic Reconfiguration in High-Speed Computer Clusters 380
D. Avresky, N. Natchev, and V. Shurbanov

Middleware II

The Lightweight Protocol CLIC: Performance of an MPI Implementation on CLIC 391
A. F. Díaz, J. Ortega, A. Cañas, F. J. Fernández, and A. Prieto

SOVIA: A User-Level Sockets Layer over Virtual Interface Architecture 399
J.-S. Kim, K. Kim, and S.-I. Jung

Reliably Locking System V Shared Memory for User Level Communication in Linux 409
F. Seifert and W. Rehm

Algorithms II

The Master-Slave Paradigm with Heterogeneous Processors 419
O. Beaumont, A. Legrand, and Y. Robert

An Efficiency and Scalability Model for Heterogeneous Clusters 427
L. Pastor and J. L. Bosque Orero

Adaptive Runtime Partitioning of AMR Applications on Heterogeneous Clusters 435
S. Sinha and M. Parashar

Amazon
Exhibit 1005
Page 384

Plenary Talks

The Grid, Virtual Organizations, and Problem-Solving Environments 445
D. W. Walker

Scalable Cluster Computing—Opportunities and Challenges 448
W. J. Camp

SCore: An Integrated Cluster System Software Package for High Performance Cluster Computing 449
A. Hori

Tutorials

Advanced Cluster Programming with MPI 453
B. Gropp

Design and Implementation of Highly Available Linux Clusters 454
I. Haddad and F. Rossi

The Scyld Beowulf System 455
D. Becker

The Parallel Virtual File System for Commodity Clusters 456
W. Ligon and R. Ross

Emerging Technologies for MultiCluster/Grid Computing 457
R. Buyya and M. Baker

Author Index 459

Amazon
Exhibit 1005
Page 385

APPENDIX 1012C



CONFERENCE PROCEEDING

2001 IEEE International Conference on Cluster Computing : 8-11 October 2001, Newport Beach, California, USA : proceedings / edited by Daniel S. Katz ... [et al.] ; sponsored by IEEE Computer Society Task Force on Cluster Computing

IEEE International Conference on Cluster Computing (3rd : 2001 : Newport Beach, Calif.); Katz, Daniel S. (Daniel Steven), 1966-; IEEE Computer Society; IEEE Computer Society. Task Force on Cluster Computing

c2001

Available at Linda Hall Library Books - 2nd Floor (QA76.58 .I132 2001) ›

Top
Send to
Get It
Details
Virtual Browse
Links

Send to

Email | Permalink | Citation | EasyBib | Export RIS | Export to Excel

Get It

‹ Back to Locations

LOCATION ITEMS

Linda Hall Library
Available , Books - 2nd Floor QA76.58 .I132 2001
(1 copy, 1 available, 0 requests)

Item in place
Loanable

Amazon
Exhibit 1005
Page 387

Details

Title	2001 IEEE International Conference on Cluster Computing : 8-11 October 2001, Newport Beach, California, USA : proceedings / edited by Daniel S. Katz ... [et al.] ; sponsored by IEEE Computer Society Task Force on Cluster Computing
Other title	IEEE International Conference on Cluster Computing Cluster 2001 Proceedings, IEEE International Conference on Cluster Computing
Creator/Contributor	IEEE International Conference on Cluster Computing (3rd : 2001 : Newport Beach, Calif.) Katz, Daniel S. (Daniel Steven), 1966- IEEE Computer Society IEEE Computer Society. Task Force on Cluster Computing
Subjects	Beowulf clusters (Computer systems) -- Congresses High performance computing -- Congresses Parallel computers -- Congresses Electronic data processing -- Distributed processing -- Congresses
Identifier	ISBN : 9780769511160 ISBN : 0769511163 ISBN : 9780769511184 (microfiche) ISBN : 076951118X (microfiche) OCLC : (OCoLC)48264589
Publisher	Los Alamitos, Calif. : IEEE Computer Society
Creation date	c2001
Format	xxi, 460 p. : ill. ; 28 cm.
General notes	"IEEE Computer Society order number PR01116"--T.p. verso. Includes bibliographic references and index.

```
leader  01705cam a2200397Ma 4500
001     993633983405961
005     20081026224000.0
008     001211s2001    caua     b    101 0 eng c
020     ##$a9780769511160
020     ##$a0769511163
020     ##$a9780769511184 (microfiche)
020     ##$a076951118X (microfiche)
035     ##$a(OCoLC)48264589
035     ##$9500643
035     ##$a(MoKL)363398-lhalldb
035     ##$a(lhalldb)363398-lhalldb
040     ##$aZCU $cZCU $dCOD $dLHL
049     ##$aLHLA
050     #4$aQA76.58 $b.I132 2001
111     2#$aIEEE International Conference on Cluster Computing $n(3rd : $d2001 : $cNewport Beach, Calif.)
245     10$a2001 IEEE International Conference on Cluster Computing : $b8-11 October 2001, Newport Beach, California, USA : proceedings / $cedited by Daniel S.
246     30$aIEEE International Conference on Cluster Computing
246     15$aCluster 2001
246     3#$aProceedings, IEEE International Conference on Cluster Computing
260     ##$aLos Alamitos, Calif. : $bIEEE Computer Society, $cc2001.
300     ##$axxi, 460 p. : $bill. ; $c28 cm.
500     ##$a"IEEE Computer Society order number PR01116"--T.p. verso.
504     ##$aIncludes bibliographic references and index.
650     #0$aBeowulf clusters (Computer systems) $vCongresses.
650     #0$aHigh performance computing $vCongresses
650     #0$aParallel computers $vCongresses
650     #0$aElectronic data processing $xDistributed processing $vCongresses
700     1#$aKatz, Daniel S. $q(Daniel Steven), $d1966-
710     2#$aIEEE Computer Society
710     2#$aIEEE Computer Society. $bTask Force on Cluster Computing
948     ##$aLTI 09/01/2008
948     ##$aLTI 04/14/2009
948     ##$aLTI 10/04/2009
```

APPENDIX 1012D

APPENDIX 1012D

Selected Citations to Batheja

Source Document

Batheja, J., & Parashar, M. (2001, October). Adaptive cluster computing using javaspaces. In *Proceedings 2001 IEEE International Conference on Cluster Computing* (pp. 323-330). IEEE.

Selected Citations 2003 – 2006 (in chronological order)

Kim, H. G., Song, H. Y., Park, J., & Lee, K. C. (2003, February). Dynamic configuration and management of clustered system with JMX. In *International Conference on Information Networking* (pp. 858-867). Springer, Berlin, Heidelberg.

Purusothaman, T., Annadurai, S., Ganesh, H. V., Chockalingam, C. T., & Kumar, B. U. (2004, September). Dynamically Scalable, Heterogeneous and Generic Architecture for a Grid of Workstations. *Journal of Grid Computing*, *2*(3), 239-246.

Takeda, K., Ono, S., Ashley, C. J., & Nakayama, S. (2005, December). JSGrid: An environment for heterogenous cluster computing. In *Sixth International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT'05)* (pp. 507-512). IEEE.

Galtier, V. (2006, October). Distributing a n-body problem algorithm at largescale over a multi-sites grid using javaspace. In *Cracow Grid Workshop'06 (CGW'06)*.