INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road Ann Arbor MI 48106-1346 USA
313 761-4700 800 521-0600

Order Number 9304664

A distributed protocol for conferencing

Friedman, Eluzor, Ph.D.

City University of New York, 1992

Copyright ©1992 by Friedman, Eluzor. All rights reserved.

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A Distributed Protocol for Conferencing

by

Eluzor Friedman

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York.

1992

© 1992

ELUZOR FRIEDMAN

All Rights Reserved

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

AUGUST /24/92
Date

Chaim Ziegler
Chair of Examining Committee

September 8, 1992
Date

Stanley Habib
Executive Officer

Professor S. Habib

Professor T. Raphan

Professor T. Sadaawi

Professor D. Vaman

Professor C. Ziegler (Chairman)

Supervisory Committee

The City University of New York

Abstract

A Distributed Protocol for Conferencing

by

Eluzor Friedman

Advisor: Professor Chaim Ziegler

With faster workstations connected to faster networks, distributed computer networks are being used for more than just the sharing of data. They are being viewed as communication systems that allow users to communicate and cooperatively work with each other. Included in these communication systems is the ability to integrate multiple varied information types, such as data, voice and video. More recently, these communication systems allow users to create conference connections to allow multiple users to cooperatively work together simultaneously utilizing a multiplicity of information types. This is referred to as multimedia conferencing. This thesis concentrates on conference connection management and presents a distributed conferencing protocol that sets up, maintains and terminates a conference connection among multiple users.

The principal contribution of the conferencing protocol is the concept of distributed conference connection management. The distributed connection management scheme used by the conferencing protocol views a conference connection as a logical ring of conference participants. A participant of an ongoing conference must only keep track of its predecessor and its successor in the logical ring of conference participants. This enables efficient distributed connection management.

A detailed description of the conferencing protocol is presented. The thesis defines the conferencing protocol services provided to the user, details the CPDUs used by the protocol,

describes the protocol mechanisms, presents a formal protocol specification in terms of a finite state machine and defines the interface to the services provided by the lower layer. An actual implementation the protocol is described.

Acknowledgements

I would like to express my sincere gratitude and thanks to Professor Chaim Ziegler for his confidence, encouragement, guidance and effort provided to me during the course of this research. Without his support, I would not be receiving this degree.

I wish to thank the members of my committee, Professors S. Habib, T. Raphan, T. Sadaawi, D. Vaman, for their time and effort in reading and examining this dissertation.

To my colleagues in suffering, Chuck Schnabolk and Debbie Sturm, thanks for the bull sessions, as well as for the support. Thank you Professor Joe Thurm for being a real friend in my time of need.

I must express my deepest gratitude and thanks to my parents, Mr. and Mrs J. Friedman, for their support, guidance and love they have always given me in all my endeavors. I would also like to thank my in-laws, Mr. and Mrs. M. Newman, for their support and guidance they have given me.

Lastly, and most dearly, I must express my deepest love, gratitude, and indebtedness to my wife, Hadassa, for her support, encouragement, listening abilities, love and, especially, patience that she has given me for the past six years. Without you, Hadassa, I never would have made it.

This research was supported in part by the New York State Science and Technology Foundation through the Center for Advanced Technology in Telecommunications under Grant RF#773689 and in part by PSC-CUNY under Research Award RF#661338.

To my wife Hadassa, the love of my life,

and

my daughter Chana, the light in my life.

Table of Contents

Chapter 1: Introduction 1
1.1 The ISO OSI Reference Model and Terminology 5
1.2 Conferencing Protocol Overview 7

Chapter 2: Experimental Point-to-Point Packet Voice Systems 11
2.1 Packet Voice Issues and LAN Protocols 11
2.2 Token-Passing Ring Experimental System 17
2.2.1 System Configuration 17
2.2.2 System Operation 18
2.2.3 System Performance Analysis 19
2.3 Ethernet Experimental System 25
2.3.1 System Configuration 25
2.3.2 System Operation 27
2.3.3 System Performance Analysis 27
2.4 PC Dependent Implementation Issues 33
2.5 Subsequent Developments 35

Chapter 3: Design of a Point-to-Point Packet Voice Communication Protocol 36
3.1. Comparison of Packet Voice Protocols and Packet Data Protocols 36
3.2. Definitions and Environment 37
3.3. Protocol Services 38
3.4. Voice Protocol Data Units (VPDU) 41
3.5. Protocol Mechanisms 43
3.5.1. Connection Establishment 43
3.5.2. Connection Closing 46
3.5.3. Voice Transport 47
3.5.3.1. Voice Transport Start Procedure - Full-Duplex 48
3.5.3.2. Voice Transport Start Procedure - Half-Duplex 48
3.5.3.3. Voice Transport Procedure 49
3.5.3.4. End Voice Transport Procedure 50
3.5.4. Hold Procedure 51
3.5.5. Reconnection Establishment 51
3.6 Subsequent Developments 52

Chapter 4: Distributed Conferencing Issues 53
4.1 Conference Connection Management 53
4.2 Conference Data Types and Data Delivery 54
4.2.1 Multicast Data 55
4.2.2 Voice Data 55
4.2.2.1 Regulated Speech Control 56
4.2.2.2 Unregulated Speech Control 56
4.2.3 Multimedia Data Delivery 59
4.3 Conference Application Sharing 60

4.3.1 Input Control 60
4.3.2 Execution Control 61

Chapter 5: Distributed Conferencing Protocol 63
5.1 Conferencing Protocol Services 67
5.1.1 Service Primitives 68
5.1.2 Primitive Mechanisms 75
5.2 Conferencing Protocol Data Units 81
5.2.1 CPDU Types and Functions 83
5.2.2 CPDU Format 88
5.3 Conferencing Protocol Mechanisms 90
5.3.1 Conference Connection Management 91
5.3.1.1 Conference Initiation 91
5.3.1.2 Sending a Conference Invitation 92
5.3.1.3 Revoking a Conference Invitation 94
5.3.1.4 Invitation Response 95
5.3.1.5 Set Predecessor 97
5.3.1.6 Set Successor 99
5.3.1.7 Expanding a Conference 99
5.3.1.8 Suspending and Resuming a Conference Connection 99
5.3.1.9 Conference State 99
5.3.1.10 Removing a Participant from the Conference 100
5.3.1.11 Leaving the Conference 100
5.3.1.12 Conference Termination 105
5.3.2 Data Delivery 106
5.3.2.1 Multicast Data 106
5.3.2.2 Successor Bound Data 106
5.3.2.3 Unicast Data 108
5.3.3 Error Detection and Recovery Mechanisms 108
5.3.3.1 Unconfirmed CPDU Error Detection and Recovery 109
5.3.3.2 Successor Recovery 110
5.3.3.3 Predecessor Recovery 114
5.4 Conferencing Protocol Formal Specification 118
5.4.1 Finite State Machine 119
5.4.2 Variable Definitions 142
5.4.3 Event Definitions 148
5.4.4 Condition Definitions 150
5.4.5 Procedure Definitions 153
5.5 Lower Layer Interface 160
5.5.1 t-send Procedure 162
5.5.2 t-receive Procedure 165
5.6 Protocol Implementation 167

Chapter 6: Conclusion 171

Bibliography 174

List of Tables

Table 3.1: Voice Protocol Service Primitives 39
Table 5.1: Conferencing Protocol Service Primitives 68
Table 5.2: Conferencing Protocol Data Units 83
Table 5.3: Actions Performed When Exceeding Retransmission Limit 109
Table 5.4: ISO Transport Protocol Service Primitive Used By Conferencing Protocol . . 161

List of Figures

Figure 2.1: Proteon ProNET token-ring packet format. 18

Figure 2.2: N_{max}, as a function of the number of samples per packet, for the token-ring network. 22

Figure 2.3: The fraction of packets lost, Φ, as a function of N_{active}, in the token-ring network with silence detection. Results are shown for a packet size of 512 samples/packet. 25

Figure 2.4: N_{max}, as a function of the number of samples per packet, for the Ethernet system. 30

Figure 2.5: The fraction of packets lost, Φ, as a function of N_{talk}, in the Ethernet System with no silence detection. Results are shown for packet sizes of 256, 512, 1024 and 2048 samples/packet. 31

Figure 2.6: The fraction of packets lost, Φ, as a function of N_{active}, in the Ethernet System with silence detection. Results are shown for packet sizes of 256, 512, 1024 and 2048 samples/packet. 32

Figure 5.1: A timing diagram depicting a conference connection setup. At the end users A, B and D are conference participants. 77

Figure 5.2: A timing diagram showing the transfer of conference data. 78

Figure 5.3: A timing diagram depicting users being removed from the conference and leaving the conference. A the end, the conference is over. 81

Figure 5.4: CPDU format. The dashed lines indicate a variable length field of the CPDU. 89

Figure 5.5: A list of the CPDU parameter types and their information fields. 90

Figure 5.6: This a timing diagram that shows Entity A inviting entities B, C, and D to participate in a conference. At the end, entities B and C confirm the invitation and entity D rejects the invitation. 94

Figure 5.7: Timing diagram showing pending invited entities responding to the invitation. Entity A is the inviter. Entities B and D accept the invitation an entity C rejects the invitation. At the end, entities A, B and D are participants. 98

Figure 5.8: Timing diagram that shows the ending of a conference. 102

Figure 5.9: A timing diagram that shows all entities leaving the conference at the same time. 105

Figure 5.10: Timing diagram showing the successor recovery process. 114

Figure 5.11: Timing diagram that shows the predecessor recovery process. 118

Chapter 1

Introduction

With faster workstations connected to faster networks, distributed computer networks are being used for more than just the sharing of data. They are being viewed as communication systems that allow users to communicate and cooperatively work with each other. Included in these communication systems is the ability to integrate different types of information, such as data, voice and video, on one communication system. More recently, these communication systems allow the users to create conference connections to allow multiple users to cooperatively work together simultaneously utilizing a multiplicity of information types. This is referred to as multimedia conferencing. This thesis concentrates on the conferencing aspect of these communications systems and presents a distributed conferencing protocol that sets up, maintains and terminates a conference connection among multiple users. The protocol maintains the conference connection as a logical ring of conference participants.

Much research has been done on the feasibility of packet networks to be used for voice and video and the integration of the different types of information over packet networks [Detr83, Muss83, Wein83, Frie86a, Frie86b, IEEE89]. These papers concentrate on the point-to-point transmission of voice, video and/or data over a packet network. They discuss different methods and schemes used in packet voice and video systems and present and analyze different types of experimental systems. Chapter 2 discusses issues concerning packet voice systems and presents two experimental point-to-point packet voice systems over two PC-based LANs.

The above mentioned papers concentrate on point-to-point delivery of voice packets. They do not set up a point-to-point connection that is needed before the voice packets can be delivered. Chapter 3 presents a design for a distributed packet voice communication protocol. The protocol sets up and maintains a point-to-point connection that provides for point-to-point packet voice delivery. Its purpose is to mimic a normal telephone call over a distributed network and allow a telephone conversation to take place between two users. This protocol was a precursor to the conferencing protocol to be presented.

[Forg80] is one of the first papers to discuss packet voice conferencing issues. It describes experimental packet voice conferencing systems implemented over ARPANET and SATNET. The Etherphone system [Swin83, Swin87, Zwel88] is a packet voice system implemented over an Ethernet that includes conferencing capabilities. A novel approach to packet voice conferencing can be found in [Zieg89] and is described later, along with other issues of packet voice conferencing, in section 4.2.2.

Real-time conferencing systems can be traced back to [Sari85]. In that paper two separate applications that can be shared among conference participants in real-time are presented. One application, RTCAL, supports meeting scheduling through an on-line calendar. The other application, MBlink, supports remote bit-mapped graphics output from a remote host to a workstation and graphical input for a remote workstation's pointing device, such as a mouse. The paper discusses the issues involved in implementing applications for real-time conferencing. Many of theses issues are presented in chapter 4.

The Rapport system [Ahuj88, Ensor88, Ahuj88, Ahuj90] is a multimedia conferencing system for voice, video and data that uses a centralized connection management scheme. It is implemented using Sun workstations running the Unix and the X Window system. The data is transmitted over an Ethernet with a separate networks for voice and signal data. This system

allows conventional programs that were developed for a single user environment to execute within a conference. This is in contrast to special programs that are developed for the multiuser environment such as the ones presented in [Sari85]. Additional conferencing systems can be found in [Ford86, Gara86, Leun89, Leun90, Kosi89, Herm87, Saka88, Saka90, Soares89, Tani88].

A conferencing system must have the ability to manage a conference connection among multiple users. The user of the system must have the ability to:

- create a multipoint connection among multiple users;
- add users to an ongoing conference;
- leave an ongoing conference;
- transfer data over the conference connection and
- know when a conference connection has been terminated.

Most work on existing conferencing systems, as cited earlier, do not concentrate on the particulars of conference connection management. Instead they concentrate on data delivery and application programs. Those papers that do address conference connection management have generally opted for a centralized approach. The central controller can either be a separate specialized entity, in charge of managing of all conference connections, or the initiator of the conference, that manages the particular conference it initiates. The main contribution of the conferencing protocol that is to be presented in this thesis has as its central focus the concept of distributed conference connection management. This is the principle contribution of the protocol to be presented in the thesis.

A distributed connection management scheme, proposed in [Zieg89], is to view a conference connection as a logical ring of conference participants (as if the conferees are sitting around a table). This scheme is similar to the logical ring of the IEEE 802.4 token-passing bus

protocol [IEEE85, Stal91]. In the token-passing bus protocol, a logical ring is maintained for the purpose of passing a token from station to station. This token permits a station access to the network channel. The logical ring is implemented by each station maintaining pointers to its logical predecessor and logical successor. Similarly, it is proposed that to maintain a conference connection, a participant of an ongoing conference must only keep track of its predecessor and its successor in the logical ring of conference participants. Many of the concepts used for setting up and maintaining the logical ring of the token-passing bus protocol can then be applied for setup, expansion, contraction and general maintenance of the conference connection. This scheme is used to manage the conference connection in the distributed conferencing protocol that will be presented.

This type of a distributed management scheme has several advantages over central management schemes. The distributed control mechanism has all participants of the conference take part in the management of the conference connection through the logical ring. Under centralized control, failure of the controller causes all conferences managed by that controller to fail. However, with distributed control, the failure of one participant does not necessarily result in the failure of the conference. The other participants can communicate with each other to reestablish the logical ring and bypass the failed station. Another drawback of central control occurs in a system where the initiator of the conference is the conference controller and, for some reason, the initiator wishes to leave the conference, while the other participants wish to continue the conference. In such a case, typically, the conference must be terminated and the remaining participants would have to form a new conference. On the other hand, with distributed control, any participant can leave the conference without effecting the conference connection, as a whole. Finally, a central controller must maintain communication with all the conference participants. Using the logical ring for conference management, each participant

must only communicate with two other participants. This tends to equalize the processing overhead among the participants of the conference, as opposed to overloading one central location.

Section 1.2 presents an overview of the conferencing protocol that is to be presented in its entirety in chapter 5. In presenting the protocol, the terminology defined by the International Standard Organization (ISO) for Open Systems Interconnection (OSI) is used. The next section presents an overview of the OSI reference model and terminology.

1.1 The ISO OSI Reference Model and Terminology

The International Standard Organization (ISO) set out to design an architecture by which standards for distributed systems interconnection can be developed. The result is the Open Systems Interconnection (OSI) reference model. The word *open* refers to the ability of any two systems that conform to the model and its associated standards to communicate with each other.

The ISO chose *layers* as the structure to its model. The communication system is divided into seven layers. Each layer performs its own set of functions required for it to communicate with another system. It uses the lower layer beneath it to perform more primitive functions. Peer entities of two systems communicate with each other via a *protocol*. The following are the seven layers defined by the ISO and a brief description of their services.

Layer 1 - Physical Layer. As the name implies, the function of this layer is the physical transmission of an unstructured bit stream over a physical link. The layer provides the mechanical, electrical, functional and procedural characteristics needed to establish, maintain and deactivate the physical link. The familiar EIA-232-D is an example of a protocol of the physical layer.

Layer 2 - Data Link Layer. The data link layer creates a reliable transmission channel

by sending blocks of data, or *frames*, with the necessary error control and flow control, over the physical link.

Layer 3 - Network Layer. The network layer provides network access and data transfer between stations by sending packets through the network. It is responsible for routing functions, congestion control and internetworking needed to deliver the data from the source node to the destination station.

Layer 4 - Transport Layer. The transport layer provides for the reliable transparent transfer of data between end points independent of the underlying network service. It is responsible for end-to-end error recovery and flow control.

Layer 5 - Session Layer. The session layer provides the control structure for communications between applications. It establish, maintains and terminates a *session* (connection) between cooperating applications.

Layer 6 - Presentation Layer. The presentation layer provides for the independence of data representation between applications. It resolves the data syntax and format differences between applications by negotiating a transfer syntax.

Layer 7 - Application Layer. The application layer provides access to the OSI environment to the user. It contains the service elements to support application processes and network management.

The user of the services implemented by the protocol of layer N is the protocol entity of layer $N + 1$ and is called the *service user*. The *service provider* is the protocol entity that performs the services of layer N. OSI separates the services, provided to the user, from the protocol operation, implemented by the service provider. This allows changes to be made in the protocol operation without effecting the user of the protocol. Services of layer N are available to layer $N + 1$ at a *Service Access Point (SAP)*. A layer N SAP is an abstract address that

defines the interface point for services provided by the layer.

The user of each layer is provided with a group of *service primitives* (operations) and their associated *parameters* that are used to invoke and deliver the services provided by a layer. OSI defines four types of service primitives. The *request* primitive is issued by the service user to request some specified service form the service provider. The service provider informs the service user that an event has occurred through an *indication* primitive. The service user responds to an indication by issuing a *response* primitive. The *confirm* primitive is used by the service provider to confirm a *request* to perform a service. Service provided to a user can either be *confirmed* or *unconfirmed*. A confirmed service uses a request, an indication, a response and a confirm service primitive. A unconfirmed service just uses a request and an indication service primitive.

Corresponding entities of the same layer on different systems (*peer entities*) use *protocol data units* (PDUs) to communicate with each other. There is no direct connection between the peer entities, except on the physical level. The connection between the peer entities is a logical one. The services of the lower layer provide PDU delivery between peer entities.

1.2 Conferencing Protocol Overview

The distributed conferencing protocol, to be presented, establishes and manages a multipoint connection among multiple users. The conferencing protocol principally concentrates on conference connection management. The purpose of the conferencing protocol is as follows:

- The protocol provides the user with the ability to establish and manage a conference connection among multiple users.
- The protocol uses a distributed mechanism for conference connection management by setting up and maintaining a logical ring of conference participants.

- The conference connection is dynamic, in that users can join and leave an ongoing conference.
- The protocol provides the user with multiple methods for conference data delivery.

The protocol provides the user with a set of protocol service primitives for conference connection management. The following is a list of some of the service primitives provided to the user and a brief description of their function.

- **C-INVITE** is used for inviting users to a conference. It can be used to invite remote users to a new conference or to invite additional users to an ongoing conference.
- **C-REVOKE** allows a user to revoke all pending invitations.
- **C-ACCEPT** is used to accept an invitation. The accepting user becomes an active conference participant.
- **C-REJECT** allows a user to reject an invitation request.
- **C-STATE** allows a user to request information about the state of the conference, specifically, which users are participating in the conference.
- **C-SUSPEND** allows the user to suspend the conference connection and become a suspended participant. Suspended participants can not send or receive data.
- **C-RESUME** allows the user to resume a suspended connection and become, once again, an active participant.
- **C-LEAVE** allows a user to leave an ongoing conference.
- **C-REMOVE** is used to remove a remote participating user from the

conference.

- **C-CONF-DATA** allows the user to send data to all participants of the conference.
- **C-SUCC-DATA** allows the user to send data to its successor in the logical ring of conference participants.
- **C-DATA** allows the user to unicast data.

The protocol is responsible for implementing the services provided to the user and in many ways mimics the service primitives that are provided to the user. A protocol entity can:

- invite remote protocol entities to a conference;
- accept a conference invitation from a remote entity;
- reject a conference invitation from a remote entity;
- revoke a pending invitation sent to a remote entity;
- leave an ongoing conference;
- forcibly remove a remote entity from an ongoing conference;
- verify the participants of the conference;
- suspend and reconnect the conference connection;
- multicast conference data to all conference participants;
- send data to its logical successor; and
- unicast data to a single remote participating entity.

This is all accomplished by the peer conferencing protocol entities communicating with each other through Conferencing Protocol Data Units (CPDUs).

As stated previously, the protocol manages the conference connection by setting up a logical ring of conference participants. This is implemented by the protocol maintaining two pointers; one is for its predecessor and the other for its successor . Thus, the protocol must also

be able to:

- set up the logical ring by setting and maintaining its successor and predecessor pointers;
- modify its predecessor and successor pointers, in order to allow protocol entities to added an deleted from the conference; and
- recover from an erroneous break in the logical ring.

The conferencing protocol creates a conference connection among application processes. It assumes the underlying protocol layer, to which it interfaces, provides transparent end-to-end transmission of data. This interface would occur most naturally at the transport layer of the ISO OSI reference model. Since the conferencing protocol creates a multipoint connection between application processes, the conferencing protocol is viewed as a session layer protocol with the ability to create and regulate a multipoint, session layer, conference connection among multiple session layer users. An application process uses the protocol to create a session layer multipoint connection. It should be noted that, at present, ISO only defines point-to-point communication at the session layer. However, defining a conferencing protocol as part of the session layer can be found in [Leun89] and [Leun90].

A detailed description of the conferencing protocol can be found in chapter 5. Section 5.1 defines the conferencing protocol services provided to the user. Sections 5.2 through 5.4 specifies the internal operation of the conferencing protocol. Section 5.2 details the CPDUs used by the protocol. A description of the protocol mechanisms can be found in section 5.3. A formal protocol specification, in terms of a finite state machine, is in section 5.4. Section 5.5 defines the interface to the services provided by the lower layer. The protocol has been implemented as described in section 5.6.

Chapter 2

Experimental Point-to-Point Packet Voice Systems

This chapter presents experimental implementations of real-time, point-to-point packet voice communication systems over two types of PC based LANs. The first is a Proteon proNET token-passing ring network and the second is a 3-Com Ethernet network. These systems implement the point-to-point, bidirectional transmission of voice packets. System configuration, operation and performance is presented for each implementation. A major portion of this chapter can be found in [Frie89].

The performance analysis that is presented for each system is to estimate an upper bound on the number active of voice stations each network can handle while providing acceptable levels of service. Voice is real time data. In order to provide timely delivery, the number of active simultaneous participants must be bounded. Results are derived for scenarios with and without silence detection.

2.1 Packet Voice Issues and LAN Protocols

In order to convert analog speech into a digital form that is acceptable for transmission over a packet-switched network, one must go through a process called speech encoding. There are several, well known algorithms that are used for this purpose. The most basic of these is Pulse Code Modulation (PCM). In its most usual form, PCM will produce

digitized speech at a data rate of 64 kbs. For networks where this data rate is too high, other techniques which yield lower data rates, such as ADPCM, delta modulation (e.g., CVSD) and predictive coding (e.g., LPC), are available. The selection of the type of speech encoding to use is usually dependent on several factors including, network bandwidth, network throughput, and the voice quality needed [Wein83].

There are several general characteristics that are inherent in all point-to-point packet voice systems. Once a call has been established between two sites, they begin to converse. As a conversation proceeds, consecutive speech samples are digitized and the resulting bits are initially stored in a buffer until a complete packet is gathered. Once a complete packet has been accumulated, the packet is encapsulated with the addition of any needed header information and then the packet is scheduled for transmission.

Most data network protocols provide for reliable source-to-destination packet delivery. This is usually accomplished through the use of an error recovery technique which generally involves the retransmission of lost or incorrectly received packets. This type of service is desirable for data transfers. However, when it comes to packet voice, real-time constraints are of greater importance. The system must provide for the ability to have timely playback at the receiver. To accomplish this, a packet voice system might sacrifice some reliability for the sake of timeliness. Indeed, most packet voice systems simply eliminate recovery procedures for lost and incorrectly received packets. This is usually of little consequence to the quality of the received voice because of the robust nature of voice [DeTr83, Wein83].

Data network protocols use traffic control schemes to regulate the flow of traffic between two entities and to control network congestion. Flow control schemes such as the sliding window protocols and credit schemes are used [Stal85]. Delivery of data packets can be delayed, for reasons of flow control, without adversely affecting the delivery of the whole

message. On the other hand, real-time voice packets are useless if they are overly delayed. Thus, traffic control schemes that hinder the timely delivery of packets should be eliminated from any voice protocol. However, some scheme must be used to protect the network resources so that network performance does not degrade to an unacceptable level [DeTr83, Wein83]. This can be accomplished by limiting the number of two-way conversations allowed to be in progress at any one time over the network.

It has been shown [Brad68] that during a typical, point-to-point, voice conversation, actual speech at each station will take place on the average of only forty percent of the time. Consequently, in order to conserve bandwidth it only seems logical to attempt to detect the silent intervals of a conversation and not send the packets that are generated during those intervals. Indeed, most packet voice systems (e.g., [DeTr83, Muss83, Wein83, Zieg80]) incorporate some type of silence detection algorithm.

Another important characteristic of any packet voice system is the amount of speech that is contained in each packet. This is an important parameter because in many packet voice systems, lost voice packets are not retransmitted. In addition, large packets can introduce an unacceptable delay in the playback due to a large packetization time. This indicates that the packets should be made small. In experimental studies [DeTr83, Muss83, Wein83], it has been found that best performance will be obtained with packets containing no more than approximately 50ms of speech. However, one must be careful not to make the voice packet too small in order to avoid adverse effects on throughput due to the increase in actual number of packets transmitted and the packet overhead bits.

As far as the packet header is concerned, most systems [DeTr83, Hober83, Muss83, Wein83] include a time stamp indicating the time origin of the packet. Since delivery time for consecutive packets can vary, it is clear that a receiver must account for this variability in

playing back received voice packets. Otherwise, unacceptable gaps affecting the quality of the received voice will occur. Consequently, the receiver can not play back a voice packet as soon as it arrives, but instead puts the packet into a buffer and will play it back after a certain delay. This delay is called reconstitution delay. Those packets which arrive at the receiver after their respective reconstitution delay time must be discarded. The time stamp is used to facilitate a timely playback at the receiving end and is helpful in the reconstitution delay.

The reconstitution delay time used is dependent on the average packet delay of each network. In ARPANET, 99 percent of all packets experience delays between 200 and 700 milliseconds. Therefore, in the packet voice experiment done over ARPANET, described in [Wein83], a reconstitution delay of 500ms was used. In [DeTr83], an experiment on a CSMA/CD LAN using PCM, a reconstitution delay of 5.75ms was used for a system that had a packet size of 5.75ms of voice. It was observed that only one percent of the packets were lost using this delay.

When the receiver has no packets available for play back, due to packet loss or delay, an interpolation scheme must be used to determine what should be played back. One method is to fill in the missing packet with silence. But, studies have shown that silent gaps are disturbing to the average listener. Several alternatives have been studied in [Muss83, Wein83, Zieg80, Zieg79]. One alternative is to playback the complete previous packet received. A second choice would be to freeze receive the receiver by playing back the last sample. Studies have shown that by using these methods the listener is not bothered by the missing speech.

The most widely used LAN topologies, the bus and the ring, have all their stations connected to a common transmission medium. Therefore, a medium access control mechanism is needed to determine who has access, or who can transmit a packet, over the network medium. There are two widely used medium access control mechanisms. One is the random

access contention scheme, typified by CSMA/CD and the other is the non-contention scheme, typified by the token-passing protocol.

Carrier Sense Multiple Access with Collision Detection (CSMA/CD) is a random access protocol that works in the following way. A station senses the transmission medium and attempts to transmit when the medium is silent. During the transmission, the station monitors the medium to detect a collision, which indicates that more than one station transmitted a packet. If a collision is detected, the station backs-off and attempts to retransmit at a later time according to some rescheduling algorithm. If no collision is detected, the station has gained access to the transmission medium. CSMA/CD works well under light and medium traffic loads but does not guarantee the delivery of packets within a deterministic time period. During high traffic loads CSMA/CD's performance degrades considerably. This is due to the large number of nodes that will transmit, collide, back off, retransmit, and collide again.

In token-passing networks, a token is passed from station to station. When a node receives the token, it has access rights to the transmission medium. Token passing networks guarantee the delivery of the packets within a deterministic time period. Even though the time period between each access to the network grows with the traffic load, the time period between successive acquisitions of the token is upper bounded and deterministic, and is proportional to the network load.

Because of the real-time requirements on voice packet delivery, it would seem that the guaranteed determinism of token-passing networks would be best suited for packet voice. However, if one notes that during a talk spurt voice packets are generated at regular intervals, one realizes that the nondeterministic nature of networks operating under CSMA/CD may be of little consequence in their ability to handle packet voice traffic.

During talk spurts, voice traffic is generated at regular intervals. Thus the instantaneous

load is not the result of a Poisson process. (In fact, the packet generation distribution is close to a deterministic process.) Consequently, the active voice users of the network can be considered as generating their packets uniformly distributed throughout the embedded frame of a packet length [Muss83a]. This allows for the possible use of a nondeterministic protocol, such as CSMA/CD, for voice traffic and it might be just as suitable as token-passing protocols.

In [DeTr83b] and [Muss83] studies of a packet voice communication system over a CSMA/CD network were performed. These studies showed that 60 to 93 two-way conversations can be supported by the network. Additionally, in [Muss83] a study of a packet voice communication system over a GBRAM (group - Broadcast Recognizing Access Method) type network, that uses a virtual token-passing access algorithm, was done. The GBRAM protocol works by assigning the users time slots, equal to the end-to-end propagation delay, in a logical ring fashion. The station senses the carrier and recognizes the source address of the transmitted packets to determine who has the next transmission right. The term *virtual* refers to the fact that no real token is passed around. The study showed that 125 two-way conversations can be supported over that network. The above studies were performed using 64 kbs PCM with silence detection.

Two experimental LANs, called Expressnet [Toba83] and Fasnet [Limb83], have been developed to provide integrated services, including voice and data integration. Both systems have been developed to operate as baseband transmission systems at rates of up to 200 Mbs. In both of these systems, voice transmission is accomplished by specifically allocating (dynamically) a portion of the transmission bandwidth to voice packets. In this way, voice packets can be distinguished from data packets at the network level and given a guaranteed delivery time.

In the experimental systems, to be detailed in succeeding sections, we have addressed

the above issues in the following way:

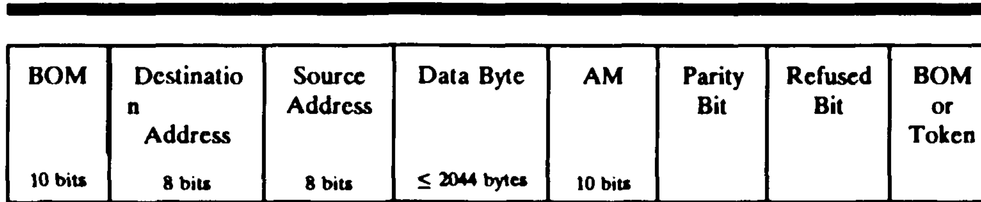
- The system ignores error recovery procedures by not checking if the voice packets were transmitted or received correctly.
- The experiments performed on the system used packets containing from approximately 37 ms to 150 ms of speech.
- For the token-passing ring experimental system, no reconstitution delay was used in the playback procedure of the system. Instead, we rely on the upper bound delay characteristics of token-passing ring networks to guarantee timely delivery of the packets.
- For the CSMA/CD experimental system, a one packet reconstitution delay was used.
- When packet loss occurs, the receiver is frozen; that is, the last sample is played back until the next packet arrives.

2.2 Token-Passing Ring Experimental System

2.2.1 System Configuration

The token-passing ring LAN used in our experimental system is a PC based, Proteon proNET network [Prot84]. The Proteon proNET has a ring network architecture that uses a decentralized token-passing access protocol. The transmission rate is 10 Mbps with twinax cable as the transmission media. The network interface provides for full-duplex operation with separate transmit and receive buffers. The maximum number of stations that a single proNET can support is 255.

Data are transmitted between stations in packets. The fields of a packet are depicted in Figure 2.1. A packet is not transmitted by a station until it receives a free token. Upon the receipt of a free token, the transmitting station changes the free token into a Beginning of



BOM	Destinatio n Address	Source Address	Data Byte	AM	Parity Bit	Refused Bit	BOM or Token
10 bits	8 bits	8 bits	≤ 2044 bytes	10 bits			

Figure 2.1: Proteon ProNET token-ring packet format.

Message delimiter (BOM) and then transmits the rest of the packet. The data field is filled with the user data in the transmit buffer. The End of Message delimiter marks the end of the data field. After the transmission of the packet, the transmitting station transmits a free token. The packet makes its way around the ring back to the transmitting station, which drains the packet off the network.

All stations on the network repeat the packets that pass through them. While a packet is passing through, address matching is being performed on the destination address field of the packet. Those packets that match a stations own address are copied into the receive buffer.

Voice digitization was accomplished using Digital Pathway's Communicard. This PC based voice board converts speech using eight bit PCM at a sampling rate of 7 Khz. Voice input and output is currently done either through a telephone handset or via microphone and speaker. (In later experiments, we switched to the IBM Voice Communication Adapter.)

The workstations used were IBM PCs and compatible. At the time of the experiment the network consisted of three IBM PCs, three IBM PC ATs and 2 Compaq portables.

2.2.2 System Operation

Voice samples are taken at rate of 7 Khz. These samples are digitized, using 8-bit PCM, and put into an internal transmit buffer. When a full packet of samples is collected, the packet is transferred to the network interface transmit buffer and transmitted over the network.

This gives a data rate of 56,000 bits per second for each active voice station. During active speech, these packets are generated periodically with the interpacket generation time dependent on the number of samples in a packet. In our system, if a new packet is generated before an old packet has been transmitted, the old packet is discarded. Packet voice can tolerate some loss of data (up to approximately two percent) without an adverse affect on the quality of the received speech perceived by the listener [DeTr83a, Fine86, Wein83, Muss83, Nutt82].

Voice packets that are received by a station and copied into its network interface receive buffer are then transferred into an internal receive buffer. The digitized voice samples are then taken from the internal receive buffer and played back through a D/A on the Communicard. At present, there is no reconstitution delay introduced to the playback of received voice packets. As soon as the packet is copied into the internal receive buffer it is ready for playback. An interpolation scheme of freezing the receiver is used for lost or overly delayed packets [Zieg80, Muss83].

The system was tested for packet sizes of 256, 512 and 1024 samples per packet. For all the above packet sizes, the system provided a voice quality comparable to that of the regular telephone network.

2.2.3 System Performance Analysis

If we assume no silence detection, the maximum number of conversations the token ring network can support is fundamentally limited by the network transmission rate. In our experimental system, a conversation produces 112,000 bps (full-duplex) and the network transmission rate is 10 Mbps. Therefore the maximum number of allowable two-party conversations due the network transmission rate is

$$\left\lfloor \frac{10,000,000}{112,000} \right\rfloor = 89. \tag{2.1}$$

This figure does not include the propagation delay, nodal delay and packet overhead bits. To include these parameters, we proceed as follows.

Let D_{token} be defined as the interval of time between two consecutive receptions of a free token by a station. Hence, we can write

$$D_{token} = N_{talk} T_{pk} + N_{station} T_{nd} + T_{pr}, \tag{2.2}$$

where

- N_{talk} is the number of voice stations that are currently generating voice packets;
- $N_{station}$ is the number of stations on the network;
- T_{pk} is the time it takes to transmit a packet including the overhead bits;
- T_{nd} is the delay introduced by each node on the token-ring network; and
- T_{pr} is the propagation delay.

If D_{token} is limited to a maximum, D_{max}, (in order to guarantee timely delivery of voice packets) one can compute N_{max}, the maximum allowable number of voice packet generating stations (that is, the maximum allowable value of N_{talk}).

With no silence detection, each active voice station periodically transmits packets with an interpacket generation time of S/R, where S is the number of voice samples per packet and R is the voice sampling rate. Therefore, for timely, loss free delivery of voice packets, a free token must be received by a station within each S/R seconds. Equating D_{max} to S/R, we can solve for N_{max} using equation (2.2) and find,

$$N_{max} = \min\left(\left\lfloor \frac{S/R - N_{station} T_{nd} - T_{pr}}{T_{pk}} \right\rfloor, N_{station}\right) \tag{2.3}$$

The minimum function is used since the maximum number of active voice stations is bounded by the number of stations on the network. The maximum allowable number of simultaneous two-way conversations is simply

$$\left\lfloor \frac{N_{max}}{2} \right\rfloor \tag{2.4}$$

Using equation (2.3), N_{max} can be computed for our experimental system. The network transmission rate is 10 Mbps and the sampling rate is 7 Khz. The Proteon proNET network introduces 38 bits of overhead for each packet. The propagation velocity for the twinax cable used is 2×10^8 meters per second. The node delay, T_{nd}, is a single bit delay, which is 100 nanoseconds.

For a packet size of 256 samples, N_{max}, the maximum number of generating voice stations, is equal to 175. This numerical result is the same for cable lengths of 1 and 0.5 kilometers and $N_{station}$, the number of listening stations, of up to 255 (which is the maximum for the network). The maximum number of two-way conversations is 87. For packet sizes of 512 and 1024 samples, N_{max} is equal to 176 and 177 respectively. Figure 2.2 shows a graph of N_{max} as a function of the number of samples per packet. Note, it is a simple exercise to show that equation (2.3) is upper bounded to 178.

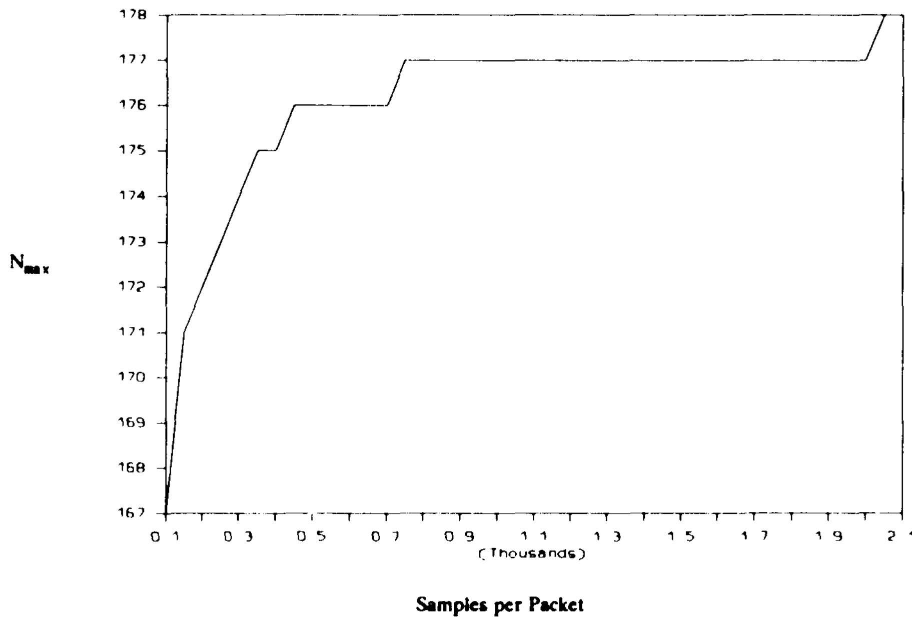


Figure 2.2: N_{max}, as a function of the number of samples per packet, for the token-ring network.

The above maxima are valid for a system that does not contain silence detection. In such a system, all stations that are actively participating in a conversation are periodically generating voice packets. Thus, the number of voice stations that are actively participating in a conversation, N_{active}, is always equal to N_{talk}, the number of stations that are currently generating voice packets.

In a system where silence detection is used, the stations that are active, (those that are participating in a conversation) do not constantly generate voice packets. Voice packets are generated periodically by an active voice station only when in a talkspurt. When a station is in a silent period, no voice packets are generated. Therefore, the system can, in general, allow the

number of active stations, N_{active}, to be greater than N_{max}. However, there is a finite probability that in an interval of D_{max}, the number of stations in a talkspurt, N_{talk}, will be greater than N_{max}. When this occurs, some packets of voice data will be lost (since D_{token} will be greater than D_{max} for some stations). As mentioned before, up to two percent packet loss can be tolerated without adversely affecting the subjective quality of the reconstructed voice.

The following analysis is an estimate of the fraction of speech lost when allowing the number of active voice stations, N_{active}, to be greater than N_{max}, in a system that employs a silence detection algorithm. It is based on analyses methods introduced in [Fine86] and [Wein78].

Of the N_{active} voice stations, N_{talk} are in a talkspurt and generating voice packets periodically. Each active voice station is assumed to be in a talkspurt, independent of all other active stations, with probability p. Thus, N_{talk} is a random variable, such that $N_{talk} \leq N_{active}$, and has the following binomial density function.

$$P[N_{talk} = n] = \binom{N_{active}}{n} p^{n}(1 - p)^{N_{active} - n} \tag{2.5}$$

where

$$\binom{n}{r} = \frac{n!}{r!(n - r)!}. \tag{2.6}$$

In addition,

$$P[N_{talk} \le n] = \sum_{i=0}^{n} \binom{N_{active}}{i} p^{i}(1 - p)^{N_{active} - i}. \tag{2.7}$$

When $N_{talk} \le N_{max}$, there is no loss of packets. When $N_{talk} > N_{max}$, the fraction of packets lost [Fine86], Φ, can be defined as

$$\Phi = \frac{N_{talk} - N_{max}}{N_{talk}}. \tag{2.8}$$

Thus, over a long period of time, the estimate of the fraction of packets lost can be given by:

$$\Phi = \sum_{i=N_{max}+1}^{N_{active}} \frac{i - N_{max}}{i} P[N_{talk} = i]. \tag{2.9}$$

In our experimental system, for a packet size of 512 samples, N_{max} = 176. Figure 2.3 is a graph of the fraction of packets lost, as a function of N_{active}, using equation (2.9) and p = 0.4 [Fine86, Wein78]. The graph shows a negligible loss of data for $N_{active} \le 2N_{max}$. There is a sharp increase in the loss rate as N_{active} increases above this value. These results are similar to the studies reported in [Fine86] and [Wein78]. Thus, when silence detection is used in our experimental system, it is estimated that the number of active voice stations can be increased to $2N_{max}$ with little affect on the quality of the reconstructed speech.

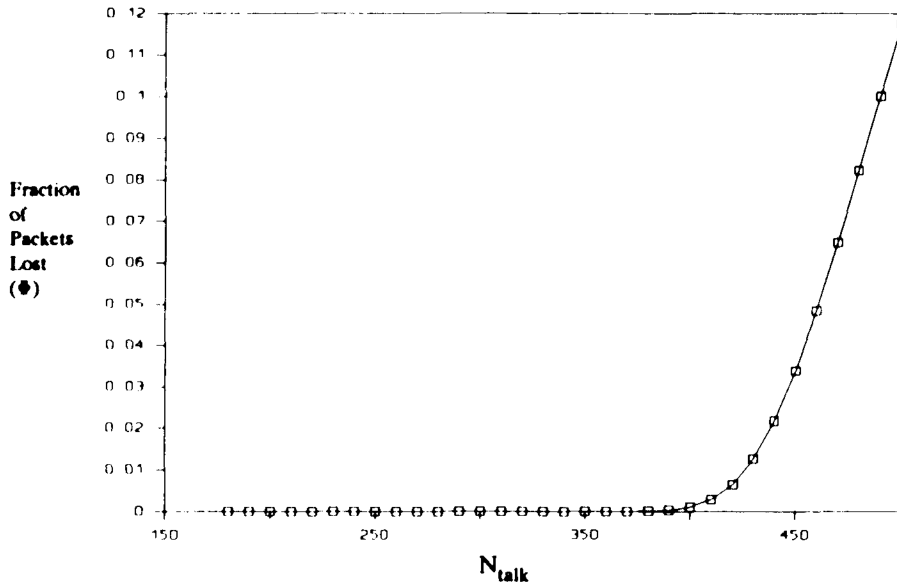


Figure 2.3: The fraction of packets lost, Φ, as a function of N_{active}, in the token-ring network with silence detection. Results are shown for a packet size of 512 samples/packet.

2.3 Ethernet Experimental System

2.3.1 System Configuration

The Ethernet LAN used is based on 3Com's IE Ethernet controller/transceiver for the IBM PC [3Com84]. The IE conforms to the Ethernet specification, version 1.0 [DEC80, Metc76]. The IE board contains a single two kilobyte packet buffer, which is shared by the transmitter and receiver. The transmission media is coaxial cable with a transmission rate of 10 Mbps.

Data are transmitted in packets that have the following fields: The first field of the packet is a 64 bit preamble. This is followed by a source and destination address, each six bytes

long. A two byte type field is next, followed by the data, which can be up to 2048 bytes. A packet check sequence of four bytes is the last field of the packet. The total number of overhead bits for a packet is 208 bits. The data for the packet is taken from the two kilobyte buffer.

A station that wants to transmit defers transmission until the channel is silent. It then waits 9.6 μseconds, the interpacket time, before attempting transmission. While the packet is being transmitted, the channel is being monitored for a possible collision. If no collision occurs within a slot time, the channel is acquired by the station. A slot time must be greater then the maximum possible round trip propagation delay and is equal to 512 bit times for the 3Com based Ethernet.

If a collision occurs, a jam signal of at least 32 bits, but not more then 48, is transmitted. The retransmission of the packet on the 3Com IE is under software control. When the IE receives the command to transmit, it delays transmission. The rescheduling algorithm used is a truncated binary exponential back off. The delay of the transmission is a multiple of the slot time. This multiple is dependent on the number of collisions a packet has already experienced. On the nth collision, the multiple used is a uniformly distributed random integer between 0 and 2^k, where $k = \min(n,10)$. Even though the Ethernet specification limits the number of collision a packet can experience to 16, the 3Com IE allows for the retransmission of packets that experience more than 16 collisions.

A station constantly monitors the channel. When a packet goes by that has a destination address that matches its own, the packet is copied into the buffer.

As in the token-ring implementation, the initial voice board used was the Digital Pathways Communicard. The workstations were IBM PCs and compatible.

2.3.2 System Operation

The system operation is similar to that of the token-ring implementation. Voice samples are digitized at a rate of 7 Khz using eight bit PCM. Samples are gathered into an internal transmit buffer. When a full packet of samples is gathered, the packet is transferred to the LAN buffer and transmitted over the network. All packets are of the same size. If a collision occurs, the packet is continually rescheduled for transmission. A packet that experiences multiple collisions is rescheduled until it is either successfully transmitted or a new packet of samples is collected. When the latter occurs, the old packet is discarded and the new packet is scheduled for transmission.

Packets that are received at a station are transferred from the LAN buffer to an internal receive buffer. The digitized samples are then taken from the internal receive buffer and played back through a D/A converter. A packet of samples is assumed ready for playback as soon as it is copied into the internal interface buffer. When voice samples are not available, due to delayed or lost packets, the receiver is frozen.

The system was tested for packet sizes of 256, 512 and 1024 samples. In all cases the system was found to provide voice quality service comparable to the regular telephone network.

2.3.3 System Performance Analysis

The analysis that follows estimates the system performance in order to calculate an approximation of the maximum number of two-party conversations that can be supported by the network without adversely affecting the speech quality. The analysis procedure is based on previous analyses of Ethernet performance that can be found in [Metc76], [Stal87] and [Tane81]

For our analysis, we assume a heavy traffic approximation. Activity on the network is modeled to alternate between contention and transmission intervals. In a contention interval, stations compete for access to the network. The contention interval is viewed as being split into

slots. It is assumed that all stations attempting to gain access to the network do so at the beginning of a slot. A slot time is the maximum time it takes to detect a collision (that is, the maximum round trip propagation time, plus the time it takes to transmit the jam signal). A contention interval ends and a transmission interval immediately begins when a station successfully gains access to the channel.

Let Q be the number of stations currently queued to transmit a packet. A queued station is assumed to transmit in a given slot with probability P. The probability that any station acquires the network in a given slot, A, is the probability that only one station attempted transmission in that slot. Then, assuming heavy traffic with a constant load of Q stations queued for transmission, we can write

$$A = QP(1 - P)^{Q-1} \qquad Q = 1, 2, \cdots \tag{2.10}$$

The number of contention slots, M, in a given contention interval is given by the following modified geometric distribution:

$$P[M = m] = A(1 - A)^m \qquad m = 0, 1, \cdots \tag{2.11}$$

The mean number of contention slots in a contention interval, $\bar{M}$, is $(1 - A)/A$.

In heavy traffic, A, the acquisition probability, is maximized when $P = 1/Q$. As $Q \to \infty$, $A \to 1/e$ and $\bar{M} \to e - 1$.

We can now determine the maximum utilization of the channel, U, which is just the length of a transmission interval as a proportion of a cycle consisting of a transmission interval and an average contention interval [Metc76, Stal87]. Thus, U can be written as

where T_{pk} is the time it takes to transmit a packet, including all overhead, and T_s is the duration of a contention slot. It is noted that U is a decreasing function of Q. Asymptotically,

$$U = \frac{T_{pk}}{T_{pk} + \bar{M}T_s} \tag{2.12}$$

as $Q \rightarrow \infty$, U is seen to approach

$$\lim_{Q \rightarrow \infty} U = U_{asym} = \frac{T_{pk}}{T_{pk} + (e - 1)T_s}. \tag{2.13}$$

Assuming, for the moment, no silence detection, (that is, $N_{talk} = N_{active}$) each active station in our experimental system will transmit packets periodically at a rate of $\lambda = R/S$ packets per second, where, as above, R is the sampling rate and S is the number of samples per packet. For N_{talk} stations, the total packet rate would be $N_{talk}\lambda$. From equation (2.13) it is seen that the asymptotic bound on the packet service rate for the network can be written as $(U_{asym}C)/B$, where C is the capacity of the network and B is the logical number of bits per packet transmission (that is, data plus overhead plus minimum interpacket time and propagation delay in terms of bits). Since $B = CT_{pk}$, the asymptotic bound on the packet service rate can be rewritten as

$$\frac{U_{asym}C}{B} = \frac{U_{asym}}{T_{pk}} = \frac{1}{T_{pk} + (e - 1)T_s}. \tag{2.14}$$

Consequently, it seems reasonable to limit N_{talk} such that we not exceed the asymptotic packet service rate of the network. Hence, one can write that N_{talk} should be bounded such that

where $D_{max} = 1/\lambda$, the interpacket generation time. Allowing N_{talk} to exceed N_{max} will result in an intolerable loss of voice packets. (This will be expanded upon shortly.)

$$N_{talk} \leq N_{max} = \left\lfloor \frac{1}{(T_{pk} + (e-1)T_s)\lambda} \right\rfloor = \left\lfloor \frac{D_{max}}{T_{pk} + (e-1)T_s} \right\rfloor \tag{2.15}$$

Our experimental system has a network transmission rate, C, of 10 Mbps. The sampling rate, R, is 7 Khz. A cable length of 1 kilometer was used in computing the equation. The propagation velocity of the cable is 2×10^8 meters per second. A maximum jam time of 48 bits was used. The transmission time of the packet, T_{pk}, includes the transmission of the voice samples, the 208 overhead bits, the propagation delay and the interpacket time of 9.6 μseconds.

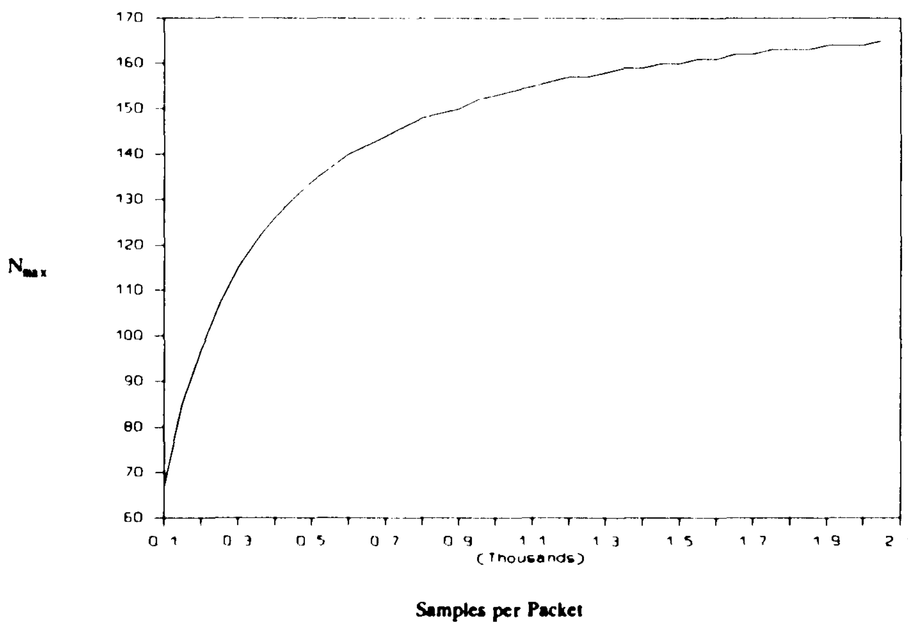


Figure 2.4: N_{max}, as a function of the number of samples per packet, for the Ethernet system.

Figure 2.4 shows a graph of N_{max} as a function of packet size for our experimental

system. With a packet size, S, of 256 voice samples, the maximum number of generating voice stations, N_{max} [equation (2.15)], is 108. With a packet size of 512, 1024 and 2048 samples per packet, N_{max} is equal to 134, 152 and 164 respectively.

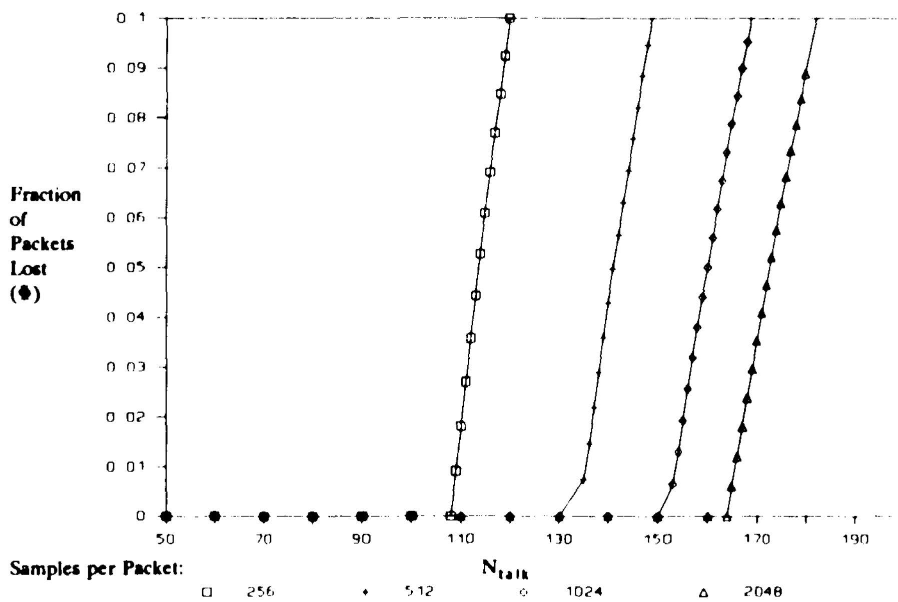


Figure 2.5: The fraction of packets lost, Φ, as a function of N_{talk}, in the Ethernet System with no silence detection. Results are shown for packet sizes of 256, 512, 1024 and 2048 samples/packet.

To now estimate the degradation of voice quality when N_{talk} exceeds N_{max}, one can proceed by estimating the packet loss rate, Φ, by using equation (2.8) where N_{max} will be given by equation (2.15). (It is noted that this is not an exact determination of the packet loss rate since it is clear that, for an Ethernet, there can be packet loss even when N_{talk} is less than N_{max}.

However, this is a useful estimate of the loss rate incurred as N_{talk} exceeds N_{max}.)

Figure 2.5 shows the fraction of packets lost as a function of N_{talk} for packet sizes of 256, 512, 1024 and 2048 samples per packet, for our experimental system assuming no silence detection. The figure shows a sharp increase in the fraction of packets lost as N_{talk} goes above N_{max}. The results compare favorably to the simulation results reported in [Nutt82], [Toba82] and [Gons87].

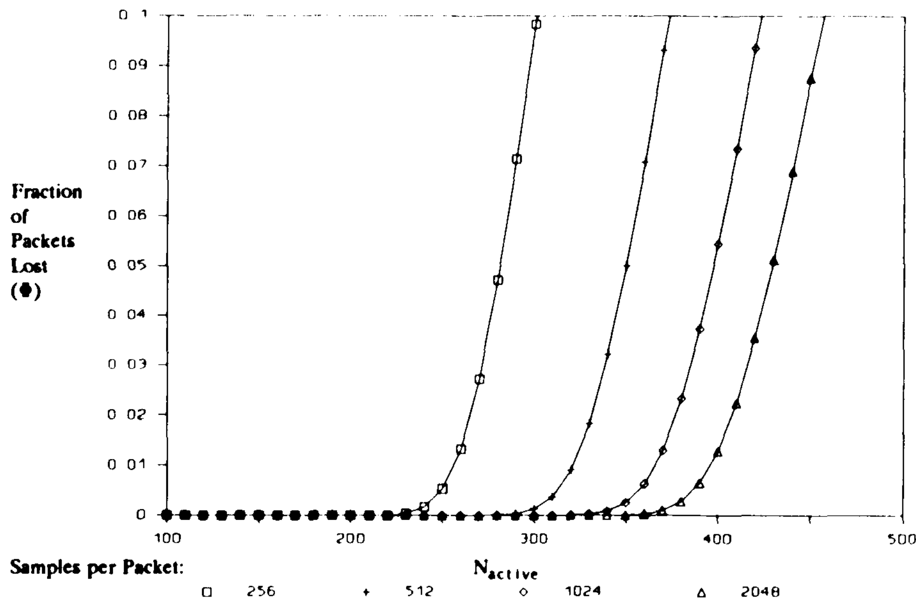


Figure 2.6: The fraction of packets lost, Φ, as a function of N_{active}, in the Ethernet System with silence detection. Results are shown for packet sizes of 256, 512, 1024 and 2048 samples/packet.

If silence detection is now added, N_{talk}, as before, becomes a random variable distributed as given by the density function of equation (2.5). Silence detection, once again,

allows the number of active voice station to exceed N_{max} without necessarily incurring intolerable delay and packet loss. To now calculate the estimate of the packet loss rate, one need only proceed using (2.9) above.

Figure 2.6 depicts the estimate of the packet loss rate as a function of N_{active} for packet sizes of 256, 512, 1024 and 2048 samples per packet, in a scenario utilizing silence detection. Once again, the figure shows an negligible loss of packets for $N_{active} \leq 2N_{max}$. There is a sharp increase in the packet loss rate above this value. Once again, this compares favorably to the simulation results reported in [DeTr83a] and [Gons87].

2.4 PC Dependent Implementation Issues

All transfers between the internal PC buffers and the LAN buffers, for both experimental networks, were done using Direct Memory Access (DMA). The internal transmit and receive buffers are handled by two sets of pointers. The first set of pointers is used for the voice board; one to keep track where the A/D is to put the next sample and one to tell the D/A where to take the next sample from. The other set of pointers is used for the LAN; one to keep track where the next packet is to be transmitted from and one to point to where the next incoming packet is to be stored. The buffers are each 64k bytes long and are circular in nature.

The basic IBM PC has a 20 bit address. The internal DMA chip used, the 8237, only allows for a 16 bit address. The PC provides a page register, for the most significant four bits of the address, which has to be loaded under program control. The PC system is designed so that a DMA transfer continuing over a page boundary will not work. This is because the hardware does not provide for a way to increment the page register during a DMA operation. In our initial design, when a DMA transfer over a page boundary was to occur, the system

detected this and transferred only the data up to the edge of the page and ignored the rest. This introduced errors in the playback. The errors can occur twice in 64k samples, once due to the transmit side and once due to the receive side.

The number of errors introduced each time a page boundary transfer is to occur is approximately uniformly distributed between zero and the number of samples in a packet. Consequently, with separate transmit and receive buffers, the expected error rate is one packet of samples per 64k samples. For 64k size packets the expected error rate would be .016, while for packets of size 256 samples the rate would be .004. Hence, the smaller the packet size, the smaller the error rate due to DMA.

A transmission error rate analysis was done for both systems by comparing the samples output by the transmitting A/D with the samples received by the receiving node. The analysis showed an error rate close to the expected value. In the playback of the packets, a click was periodically heard due to this DMA error. The click was less noticeable with packets of smaller size.

Subsequent analysis traced the root of the problem to the *unwillingness* of DOS to allocate the transmit and receive buffers such that packet boundaries coincided with page boundaries. Additional software, added to override the memory allocation of DOS, has since been written to force a page boundary within a 64k buffer to coincide with a packet boundary. This has removed the DMA problem and eliminated the clicks from the played back speech.

A second implementation issue deals with the inherent system delay on packet playback. As an example, let us consider the delay incurred by the first sample of a random packet. Before this sample is to be played back at a receiving station, it must incur the following delays:

a) Wait for a full packet of samples to be collected,

b) wait for the packet to be loaded from the transmitter's packet buffer onto the network interface board,

c) wait for access to the network,

d) wait for transmission and reception over the network,

e) wait for loading from the receiver's network interface board to the receiver's playback buffer.

Of the above delays, part a) depends on the sampling rate. Parts c) and d) are a function of the network access mechanism and network transmission rate. Parts b) and e). on the other hand, are station dependent. If, as in our two experimental implementations, DMA is used for inter-buffer packet movement, then implementation of parts b) and e) requires the execution of approximately 200 assembly language instructions plus two DMA packet transfers. The exact amount of playback delay introduced by parts b) and e) would be a function of the type of PC used (XT, AT, etc.) and the packet size.

2.5 Subsequent Developments

The above detailed systems were initial implementations. The system was later expanded to a multipoint packet voice system to allow packet voice conferencing. Subsequently, a multipoint packet delivery system was developed for the integration of different types of data. These developments will be presented in Chapter 4, when distributed conferencing will be introduced.

Chapter 3

Design of a Point-to-Point Packet Voice Communication Protocol

The purpose of the system presented in the previous chapter is point-to-point delivery of voice packets. It does not set up a point-to-point connection that is needed before the voice packets can be delivered. The focus of this chapter is to present a design for a distributed packet voice communication protocol. The protocol sets up and maintains a point-to-point connection, and provides for point-to-point voice packet delivery. Its purpose is to mimic a normal telephone call over a distributed network and allow a telephone conversation to take place between two users on the distributed network.

It should be noted, the conferencing protocol, presented in later chapters, differs from this protocol. The conferencing protocol sets up a multipoint connection between multiple users and provides the user with the capabilities for multipoint delivery of different types of data, not just voice data.

A version of this chapter has been published in [Frie87].

3.1. Comparison of Packet Voice Protocols and Packet Data Protocols

As mentioned in section 2.1, error recovery and flow control schemes, that are inherent in packet data protocols, can be detrimental to real-time packet delivery needed for voice packets. There is another interesting difference between a packet voice protocol and other types of protocols in the connection procedure. When a connection request comes in, a protocol entity informs the user of the protocol of the request and waits for a reply. This reply is usually

to accept or reject the connection request. In most protocols, the user is a process of the host computer, such as the operating system. In a packet voice protocol, the ultimate user of the protocol in many instances is a human. The response to a connection request can depend on the human, who may decide to ignore the ring of a *telephone*, answer the telephone or, if the service is provided, deny the telephone connection. The protocol has to take into account the human factor in its connection establishment procedure.

3.2. Definitions and Environment

The term *user* is defined as the combination of the *telephone-terminal*, the system using the protocol and the operator of the *telephone-terminal*. Information is passed between the *user* and the protocol entity through a voice service access point (VSAP) by means of the protocol service primitives (described later).

Communications between two peer voice protocol entities is through Voice Protocol Data Units (VPDU). The protocol described below perform the services and functions associated with the upper layer protocols of the communication system architecture. In terms of the International Standard Organization (ISO) Open Systems Interconnection (OSI) reference model, these protocol layers are the application, presentation and session layers. Section 3.5.3.3 details which services preformed by the protocol are associated with the different layers. The underlying lower layer protocol entity must provide the voice protocol with the services of transporting packets from end-to-end. Due to the time constraints on voice packets, the underlying protocol should provide the end-to-end transport service with minimal flow control and without retransmission of lost or erroneous packets. To best meet these requirements, a datagram type transport service would seem appropriate [Stal90].

In terms of the ISO OSI model, the underlying layer that provides this end-to-end service is the transport layer [ISO82a, ISO82b, ISO84a, Stal85]. The most appropriate type of

service would be a connectionless mode transport service. A connection oriented transport layer could be used as long as it could deliver the voice packets in real-time. The transport service primitives of the connectionless and connection oriented transport services have a *Quality of Service* (QOS) parameter which allows the user to request, amongst other qualities, the desired maximum delay for delivery of data packets. This parameter can be used to guarantee the timely delivery of the voice data. Thus, any service and any protocol class of the transport layer can be used as long as the data can be delivered in real time.

Of course, network layer services must be available to the transport layer that can deliver the data in real time. Such network services have been shown to be available over existing LANs [DeTr83, Muss83, Frie86a, Frie86b] and over certain long haul networks [Wein83]. We note that the network layer service primitives defined by the ISO [ISOa] also contain a QOS parameter which allows you to request the maximum delay for data packet delivery, which can be used to guarantee timely delivery of packets.

In terms of the DoD protocol architecture, the underlying layer would be the host-host layer [Stal85]. Using the DoD Transport Control Protocol (TCP) as the host-host layer may cause problems due to the reliability and flow control schemes present in TCP. The DoD User Datagram Protocol (UDP) would seem more appropriate for voice connections since it provides a datagram service.

3.3. Protocol Services

One of the main purposes of the protocol is to provide the user with the mainstream services of a regular telephone connection. This includes establishing, maintaining and closing connections between two users. Maintaining a connection includes services such as suspending and reconnecting a connection (i.e., hold) and the transferring of actual voice data between the two users. The protocol can also provide services that are usually not found in the mainstream

telephone service. One such service is to allow for different types of protocol connections so as to allow for simplex, half-duplex or full-duplex voice transmission.

Many telephone systems allow for multi-line service. This service is incorporated into the designed protocol. This service will allow a single telephone-terminal to support more than one connection at a time. It allows the user of the protocol to create a new connection, even though one already exists. At present, at any given time only one connection can be active (that is, actually sending voice) while the other existing connections must be suspended (that is, put on hold). The number of separate lines or connections that can be supported is dependent on the resources available and the implementation of the protocol.

Table 3.1: Voice Protocol Service Primitives

Name	Type	Parameters
V-CONNECT	request	calling address, called address, options
	indication	calling address, called address, id, options
	response	calling address, called address, options
	confirm	calling address, called address, id, options
V-PROGRESS-SIGNAL	signal	id
V-DISCONNECT	request	id
	indication	id, cause
V-DATA	request	id, voice
	indication	id, voice
V-DATA-DENIAL	indication	id
V-DATA-END	request	id
	indication	id
V-HOLD	request	id
	indication	id
V-RECONNECT	request	id
	indication	id
	confirm	id

3.3.1. Service Primitives and Description

A list of the service primitives is shown in Table 3.1. The telephone functions mentioned above can be performed with the listed primitives.

The V-CONNECT primitive is used to create a *voice protocol connection* between two *voice protocol users*. Options have to be negotiated at the time the connection is created. Options can include the packet size, that is, the number of voice samples or milliseconds of voice per voice data packet; simplex, half or full duplex voice transmission; and the voice encoding method to be used.

The protocol allows a user to put one active connection on hold and go create a new connection with a different user. This necessitates providing the user with a connection identification number. The identification number is a local reference to the connection that is set by the protocol and provided to the user by the V-CONNECT service primitive. The service primitives need the identification number to inform the protocol to which connection it is referring.

The V-PROGRESS-SIGNAL allows for the monitoring of the voice protocol connection during its creation and throughout the lifetime of the connection. The *signal* parameter informs the user the status of the connection. As an example, the V-PROGRESS-SIGNAL is used as a *ringback signal*, which is the signal a caller hears on the telephone receiver when waiting for the called person to answer the phone.

The V-DISCONNECT primitive is not only used for the normal disconnection procedure, but is also used to refuse a connection request and indicate disconnection due to a protocol error. The reason for breaking a connection is indicated by the *cause* parameter. Since a user can close a suspended connection, the connection identification number is provided as a parameter.

The V-DATA primitive is used to transmit voice to the remote user. A request to transmit voice is denied through the V-DATA-DENIAL primitive. As an example, it can be used to deny a request to transmit voice over a half-duplex connection while the remote user is transmitting or to deny a request to transmit data the wrong way over a simplex connection. The cause parameter indicates the reason for the denial. The V-DATA-END primitive allows the user to stop transmitting voice without breaking or suspending the connection. As an example, the V-DATA-END primitive can be used to indicate the end of a voice transmission over a half-duplex connection and to turn around the line.

A connection between two users can be temporarily suspended (put on hold) by using the V-HOLD primitive. This allows the user to perform other functions, such as making a separate connection to a different user, without breaking the current connection. The connection that has been put on hold can then be reconnected by using the V-RECONNECT primitive. Only the user who issued the hold request can issue the reconnect request. This is similar to when someone is put on hold using a conventional telephone network. Note, the user put on hold is not suspended. The user on hold can issue a V-HOLD.request or even a V-DISCONNECT.request.

3.4. Voice Protocol Data Units (VPDU)

Information is exchanged between voice protocol entities by using VPDUs. The following is a list of VPDUs and a description of their purpose.

CR - The Connection Request VPDU is sent, by a calling entity to a remote entity, to request a voice protocol connection.

CF - the Called entity Free VPDU is sent by the called entity, in response to a CR VPDU. The CF VPDU informs the calling entity that the called entity is free for a connection and that the called user has been informed of the connection

request. The called entity is waiting for a response from the called user.

CC - The Connection Confirmed VPDU is sent by the called entity to confirm the connection request. The called entity is informing the calling entity that the called user responded to and accepted the connection request. The connection is established with the CC VPDU.

DR - The Disconnection Request VPDU is sent to the remote entity to request the breaking of the voice protocol connection. It can also be used to refuse a connection with a remote entity.

DC - The Disconnection Confirm VPDU is sent to the entity requesting a disconnect, confirming the connection is broken.

VD - The Voice Data packet VPDU contains the voice data.

ER - The End voice Request VPDU is sent to a remote entity informing it not to expect any more voice data. It also turns around the line on a half duplex connection.

EC - The End voice Confirm VPDU is sent to confirm an end voice request.

HR - The Hold Request VPDU is sent to the remote entity to request the suspension of the connection.

HC - The Hold Confirm VPDU is sent to confirm the connection suspension.

RR - The Reconnect Request VPDU is sent by the entity that put the connection on hold to request a reconnection of the suspended connection.

RC - The Reconnect Confirm VPDU is sent to confirm the reconnection of the suspended connection.

The services provided by the lower layer, the transport layer [Stal91, Tann89], are used to transfer the VPDUs between the voice protocol entities. If using a connectionless transport

layer, the T-UNIDATA service primitives are used to transfer the VPDUs. If a connection mode transport layer is used, a transport layer connection must be created, through the T-CONNECT service primitives, with the creation of the voice protocol connection. (It may be possible to piggyback the CR and CF VPDUs onto the T-CONNECT service primitives by using the user-data parameter of the primitive.) Upon termination of the voice protocol connection, the transport layer connection must be terminated through the T-DISCONNECT service primitives. (It may be possible to piggyback the CR and CF VPDUs onto the T-DISCONNECT service primitives by using the user-data parameter of the primitive.) All other VPDUs are transferred using the T-DATA service primitives.

3.5. Protocol Mechanisms

3.5.1. Connection Establishment

Before a conversation can proceed, a voice protocol connection must be established. A request to set up a connection is made through the issuance of a V-CONNECT.request. The protocol then attempts to create the voice protocol connection between the two users.

Prior to attempting a connection establishment, the protocol checks to see if it has available the resources needed to create and maintain a voice protocol connection. These resources include that the local host machine, the local network interface and if the network as a whole can handle all the data to be produced by the transmission of a voice conversation. If the resources do not exist at that moment, the connection request is denied through the V-DISCONNECT.indication primitive. If the resources exists, they are allocated and the normal connection procedure continues.

Once network resources have been allocated, the calling voice protocol entity sends a CR VPDU to the called protocol entity, requesting a connection between the two entities. Any options that have to be negotiated (such as the voice message size) are sent along with the CR

VPDU. The calling entity waits for a response from the called entity.

The receipt of a DR VPDU from the called entity indicates that the called entity is currently not accepting the connection for some reason, such as the called entity is busy. The user is informed that the called entity is not accepting any calls through V-DISCONNECT.indication (with the *cause* parameter indicating the reason of the disconnection) and the attempt to establish a voice protocol connection is terminated. The user can then start over again with a new V-CONNECT.request.

The receipt of a CF VPDU from the called entity, in response to the CR VPDU, indicates the called entity is free for a voice protocol connection and is attempting to create one. The user is informed of this thorough the V-PROGRESS-SIGNAL.indication service primitive, with the *signal* parameter indicating a *ringback signal*. The protocol entity then goes into a wait-for-answer mode, where it is waiting for the remote user to answer the connection request. During this waiting period, CF VPDUs are periodically expected to arrive from the called entity. This is used to inform the calling entity that the called entity is still waiting for a response from the called user. If a timeout occurs, meaning, a CF VPDU was not received within the allowed interval, then the connection attempt is terminated. The normal closing procedure (the exchange of DR and DC VPDUs between the entities described in section 3.5.2) is performed. The local user is informed of the termination through V-DISCONNECT.indication. If no timeout occurs, the protocol remains in the wait mode until either there is a response from the called user or a V-DISCONNECT.request is issued by the local user.

If a CC VPDU is received from the called entity while in the wait-for-answer mode, indicating that the called user acknowledges the voice protocol connection, the voice protocol connection is successfully established and the user is so informed through

V-CONNECT.confirm. The user can then request the transmission of voice through the V-DATA.request primitive.

If a DR VPDU is received from the remote entity while in the wait-for-answer mode, indicating that the called user refuses the call, the voice protocol connection attempt is terminated. In fact, the receipt of a DR VPDU at any time during the call establishment phase will result in the connection attempt terminating.

If the user issues a V-DISCONNECT.request while in the wait-for-answer mode, the voice protocol connection attempt is terminated and the normal connection closing protocol procedure is performed. The same process will happen if the user issues a V-DISCONNECT.request at any time during the connection establishment phase.

We now take a look at the connection establishment procedure from the point of view of the called entity.

When the called entity receives a CR VPDU, the protocol checks to see if it has the resources available to make a voice protocol connection. If the resources are not free, a DR VPDU is sent to the calling entity and the voice protocol connection attempt is terminated.

If the resources are available, a CF VPDU is sent to the calling entity and the user is informed that a voice protocol connection is requested through V-CONNECT.indication. The protocol waits for either a response from the user or a disconnection request from the calling entity. During this waiting period, a CF VPDU is periodically sent to the calling entity to inform it that a response from the called user is still anticipated.

If the user responds affirmatively to establish the connection through V-CONNECT.response, a CC VPDU is sent to the calling entity and a voice protocol connection is successfully established.

If the user issues a V-DISCONNECT.request, indicating the user is refusing the

connection, a DR VPDU is sent to the calling entity and the voice protocol connection attempt is terminated.

If a DR VPDU is received from the calling entity, indicating that the calling user issued a V-DISCONNECT.request before the called user issued a response, the connection attempt is terminated and the normal connection closing protocol procedure is performed.

The above procedure can be performed using either a connection mode transport layer or a connectionless mode transport layer. There is only one difference in the connection establishment procedure between the two modes. If the connection mode transport service is used, a transport connection must be set up using the T-CONNECT service primitives. It may be possible to piggyback the CR and CF VPDUs onto the T-CONNECT service primitives by using the user-data parameter of the primitive. If the transport connection is denied, the user is informed using V-DISCONNECT.indication.

3.5.2. Connection Closing

Upon the request from the user, a voice protocol connection is closed in the following manner. Either user issues a V-DISCONNECT.request. This causes the requesting entity to send a DR VPDU to the other entity, which responds with a DC VPDU. The responding entity then informs the user that the connection is closed through V-DISCONNECT.indication. After the connection is closed, the user could then start over again by issuing a new V-CONNECT.request.

As soon as a DR VPDU is sent or received, the connection is considered closed. Any other type of VPDUs that are received are discarded. If voice data are currently being transported, the voice transport procedure (described later) is immediately terminated. An entity that sends a DR VPDU and then receives a DR VPDU should consider the DR VPDU received as the acknowledgement. This takes care of connection closing collisions.

If a connection mode transport service is used the transport connection has to be closed through the T-DISCONNECT service primitives. It may be possible to piggyback the CR and CF VPDUs onto the T-DISCONNECT service primitives by using the user-data parameter of the primitive.

3.5.3. Voice Transport

The primitives used in the transporting of the stream of voice data between the two users are V-DATA and V-DATA-END. The V-DATA primitive is used to initiate the transport of the voice data stream and the V-DATA-END primitive is used to stop the transport of the voice data stream. The transporting of the voice data stream is accomplished through the voice transport procedure (described later).

The voice data stream can be considered as one long *application layer data unit* and V-DATA as the primitive to send and receive this data unit. The V-DATA-END primitive signals the end of the voice stream and can be viewed as the end delimiter of the *application layer data unit*. Alternatively, in terms of files, the stream of voice data can be viewed as a *virtual file* of indeterminate length. The V-DATA primitive is used to send and receive this *virtual file* and the V-DATA-END primitive is used as the end of file delimiter.

The voice transport procedure can be split into two parts; transmitting and receiving of the voice data stream. When a user issues a V-DATA.request, the transmit part of the voice transport procedure begins. The transmit part of the transport procedure takes the stream of voice data (the long *application layer data unit* or *virtual file*), fragments it into VD VPDUs and sends these VD VPDUs over the network. Upon the receipt of the first VD VPDU, the user is notified through V-DATA.indication and the receive part of the voice transport procedure begins. The function of the receive part is to convert the incoming VD VPDUs into a stream of voice data. The sending of voice data can be terminated by issuing a V-DATA-END.request.

3.5.3.1. Voice Transport Start Procedure - Full-Duplex

Upon successful completion of a voice protocol connection or the reconnection of a suspended connection (described later), both users can issue a V-DATA.request. This is a request to transport the voice data stream from the local user to the remote user. In response to the V-DATA.request, the voice protocol checks if there is a reason not to grant the request, such as the user was put on hold. If no reason exists, the request is granted and the transmit part of the voice transport procedure begins. Otherwise, the request is denied and the user is informed through V-DATA-DENIAL.indication with the *cause* parameter indicating the reason of the denial.

When a voice protocol entity receives the first VD VPDU, after its successful connection establishment or reconnection establishment, the user is notified through V-DATA.indication and starts the receive part of the voice transport procedure.

3.5.3.2. Voice Transport Start Procedure - Half-Duplex

The right to transmit data over a half-duplex connection alternates between the users. When a user is willing to give up its right to transmit data, the user issues a V-DATA-END.request. In response to the request, the protocol entity sends an ER VPDU to the remote entity, signaling the end of voice data. The remote entity then informs the user through V-DATA-END.indication, signaling the remote user that it has the right to send data. The remote entity then sends an EC VPDU to the entity, turning around the line. The user that just turned around the line, by issuing the V-DATA-END.request, does not have the right to send data again until it receives a V-DATA-END.indication from the remote user. At present, the user who initiates the connection has the right to send data first. A user who requests to send data when it does not have the right to is denied through the V-DATA-DENIAL.indication primitive.

Once a user has the right to send data, the user can then issue a V-DATA.request and the procedure described above for the full duplex mode takes effect.

3.5.3.3. Voice Transport Procedure

The voice transport procedure can be split up into three hierarchical processes:

1) Data encoding and decoding.

2) Data encapsulation and playback scheduling.

3) Sending and receiving voice packets.

The function of the first process is to periodically take analog voice samples and convert them into digital data and vice versa. Voice samples are taken from the local user, reconverted into digital data and passed to the second process. Incoming digitized voice samples are periodically provided by the second process to the first process to be converted to analog voice.

The second process is the interface between the first process and the third process. Its responsibilities include data encapsulation, time stamping, silence detection, interpolation and playback scheduling.

First, a look at outgoing voice samples. The digitized samples are periodically provided by the first process. These samples are gathered together into packets. When a packet contains a certain number of samples, negotiated at connection time, it is considered full. The full packet is put into a VD VPDU by the second process, time stamped and passed to the third process for transmission. If a silence detection scheme is being used, the full packet is checked to see if contains voice before it is passed to the third process. Silent packets are discarded.

Now, a look at incoming voice samples. The second process takes an incoming packet passed to it by the third process. What is done with incoming packets depends on the time stamp. Those packets which are overly delayed are discarded. Other packets are artificially

delayed and played back in a timely manner. When a packet is scheduled for playback, digitized voice samples are periodically removed from the packet one at a time and passed up to the first process for reconversion. When a new packet is needed and there is none available, an interpolation scheme is used to provide the first process with samples.

The third process is responsible for the sending and receiving of packets. Packets provided by the second process are to be sent out over the transport connection. When a VD VPDU is received from the transport connection, it is passed up to the second process.

In terms of the ISO OSI model, the above procedure can be viewed in terms of the application, presentation and session layers. The application layer passes and accepts a stream of analog voice to and form the presentation layer. The first two processes of the voice transport procedure can be classified as part of the presentation layer, since it is these two processes that make the voice *presentable* for transmission over the network and vice versa. The third process can be classified as a session layer function since it is responsible for sending and receiving voice data between the application processes.

3.5.3.4. End Voice Transport Procedure

The voice transport process continues until it is stopped in one of two ways; by an explicit or implicit request from the user. The user could stop the transmit part of the voice transport procedure by issuing a V-DATA-END.request. This will result in the following procedure. The protocol will stop sending voice data to the remote entity, but will still continue to receive voice data. An ER VPDU is sent to the remote entity, informing it not to expect any more VD VPDUs. Upon receipt of an ER VPDU, the remote entity stops the receive part of the voice transport procedure. The remote entity then responds with an EC VPDU and locally informs the remote user not to expect any more stream voice data by issuing a V-DATA-END.indication. If the entities were in full duplex operation, the remote entity does

not stop the transmit part of its voice transport procedure in response to an ER VPDU.

Issuing a V-HOLD.request (described later) or a V-DISCONNECT.request during the voice transport procedure results in an implicit request to stop the voice transport procedure. The sending of a HR or DR VPDU by the requesting entity stops the transmit and receive parts of the voice transport procedure. The receipt of a HR or DR VPDU by the remote entity results in stopping the transmit and receive parts of the voice transport procedure and the remote entity responds with a HC or DC VPDU.

3.5.4. Hold Procedure

When a V-HOLD.request is issued by the user, the following hold procedure is performed. An HR VPDU is sent to remote entity. The remote entity responds with a HC VPDU and informs the remote user the connection is on hold through V-HOLD.indication. When an entity sends or receives a HR VPDU, the connection is considered on hold and, if currently transporting voice data, the voice transport procedure is immediately terminated. The entity that requested the hold is the only entity that can request a reconnection. The user that requested the hold can now issue a new V-CONNECT.request to set up a different connection. As mentioned before, the user on hold can issue its own V-HOLD.request or even a V-DISCONNECT.request.

3.5.5. Reconnection Establishment

The user which requested the hold can issue a V-RECONNECT.request to initiate the following reconnection procedure. The reconnecting entity sends a RR VPDU to the remote entity. The remote entity responds with a RC VPDU and the suspended connection is considered reconnected. The remote user is then informed of the reconnection through V-RECONNECT.indication. When the reconnecting entity receives a RC VPDU, the suspended connection is considered reconnected and the user issuing the reconnection request

is informed through V-RECONNECT.confirm. After the reconnection procedure, the users can then issue a V-DATA.request.

3.6 Subsequent Developments

The point-to-point voice communication protocol was designed but not fully implemented. Instead, it was decided that it would be preferable to design and implement a distributed conferencing protocol that would provide multipoint communication for multiple types of data. The new protocol would encompass the point-to-point protocol of this chapter. The subsequent chapters present the distributed conferencing protocol that has been developed.

Chapter 4

Distributed Conferencing Issues

In the point-to-point packet voice communication protocol presented in the previous chapter, a connection must be setup between the two users of the protocol. On the other hand, a conferencing protocol must have the ability to setup a connection among many users. This introduces several issues, unique to conferencing systems, that must be addressed. These issues include:

- conference connection management,
- conference data types and data delivery, and
- conference application sharing.

This chapter defines and contains the concepts that address these issues.

4.1 Conference Connection Management

Conference connection management involves setting up and maintaining a multipoint connection among the conference participants. The user must have the ability to:

- create a multipoint connection among multiple users;
- add users to an ongoing conference;
- leave an ongoing conference; and
- know when a conference connection has been terminated.

Most reports of existing conferencing systems do not concentrate on the particulars of conference connection management. A number of them seem to use a central conference

controller to manage the conference connection. The central controller can be a separate specialized entity, that is not a conference participant, in charge of connection management of all conference connections. The Etherphone system [Swin83, Swin87, Zwel88] is an example of this scheme. Alternately, the initiator of the conference can be the central controller of connection management of the conference that it initiated. The Rapport [Enso88a, Enso88b, Ahuj88] system is an example of this scheme.

A distributed connection management scheme, proposed in [Zieg89], is to view a conference connection as a logical ring of conference participants (as if the conferees are sitting around a table). This scheme is similar to the logical ring of the IEEE 802.4 token-passing bus protocol [IEEE85, Stal91]. In the token-passing bus protocol, a logical ring is maintained for the purpose of passing a token from station to station. This token permits a station access to the network channel. The logical ring is implemented by each station maintaining pointers to its logical predecessor and logical successor. Similarly, it is proposed that to maintain a conference connection, a participant of an ongoing conference must only keep track of its predecessor and its successor in the logical ring of conference participants. Many of the concepts used for setting up and maintaining the logical ring of the token-passing bus protocol can then be applied for setup, expansion, contraction and general maintenance of the conference connection. This scheme is used to manage the conference connection in the distributed conferencing protocol that will be presented.

4.2 Conference Data Types and Data Delivery

Typical types of information that can be expected from a multimedia conferencing system has been discussed in [Zieg90]. The types mentioned include:

- Voice Data - which require time constrained, real-time transmission of the data to all conference participants;

- Files and Memos - which require acknowledged, error free transmission of the data to one or more conference participants;
- Screen Data - which require the contents of a user's screen to be dynamically viewed by all conference participants;
- Keystrokes - which require the keyboard strokes of one participant to be piped into an application program of a remote participant; and
- Video Data - which, similar to voice data, requires time constrained, real-time transmission of the data to all conference participants.

The following sections address the issues that arise from the different types of data listed above.

4.2.1 Multicast Data

Many of the types of information listed above require the user to have the ability to multicast the data to all conference participants. If the underlying network has multicast capabilities, it can be used to transmit the data to all the users. If multicast capabilities are not available, most conferencing systems emulate multicasting by sending a separate data packet to each user of the conference connection. However, by maintaining a conference connection as a logical ring of conference participants, multicast transmission of conference data can be attained by passing the data around the logical ring to all participating users.

4.2.2 Voice Data

Conference voice data over a packet network introduces some difficulties in the playback of the voice data. During a conference call on an analog, circuit switched network, when more than one person speaks simultaneously no difficulties are anticipated at the receiver's ends. This is because the signals produced by the speakers can be simply combined at the receiver by the analog system. However, when voice is digitized and sent out in packets,

the receipt of voice packets from more than one source can be a problem. One D/A cannot handle multiple samples from multiple sources without preprocessing. This problem can be addressed through the introduction of a speech control mechanism. Two such mechanisms are presented. The first is called *Regulated Speech Control* and the second is called *Unregulated Speech Control*. The next two sections discuss these two control schemes and how the logical ring of conference participants can be used to implement the schemes.

4.2.2.1 Regulated Speech Control

Regulated speech control allows only one participant to talk at a time with each participant talking in turn. A scheme described in [Forg80] involved the use of a central controller, called the CHAIRMAN, to decide who has the right to speak next. A user would send a request to speak to the CHAIRMAN, who would put the request on a queue. The CHAIRMAN would inform the user when its turn came up.

By viewing the conference connection as a logical ring of conference participants, a similar scheme, of the participants speaking in turn, can be used without the use of a central controller. This can be done by sending a token, which grants speaking privileges, around the logical ring.

This mechanism is a "formal" way of running a conference call, in that each user must wait its turn for the floor (as in being recognized by the chair at a formal meeting). Though formal and simple to implement (compared to unregulated speech control), this is not the ideal way of running a conference call. This is because participants are not always amenable to such formalism.

4.2.2.2 Unregulated Speech Control

In unregulated speech control, all participants can speak at will and do not need explicit permission to speak. This is the preferred and more natural method. In this case, the

packets of digitized voice generated by each participant must be processed and combined before being played back at each receiver. In [Zieg89] two possible implementation schemes for unregulated speech control are presented.

Method 1 involves the multicast transmission of the voice data packets. As soon as a packet is ready to be sent, the multicast capabilities of the network are used to transmit the packet to all conferees. If multicast capabilities do not exist on the network, individual copies of the voice data packets are sent to each participant. On the receiving end, all incoming voice packets are combined for playback. Assuming a periodic generation of voice packets and N participants in the conference, a station transmits one voice packet, and receives and combines $N - 1$ voice packets during each voice packet generation cycle.

Method 2 makes use of the logical ring of conference participants described earlier. In this method, a *shuttle packet* of the combined voice samples of all the participants is shuttled around the logical ring. A participant's voice data packet, that is ready to be sent out, is delayed by the sending station until receipt of the shuttle packet. Upon receipt of the shuttle packet, the station

- removes its previous contribution to shuttle packet,
- schedules the resultant packet for playback,
- "adds" its new voice packet to the shuttle packet and
- sends the shuttle packet to its successor on the logical ring.

Assuming periodic generation of voice packets, the shuttle packet must be able to make its way around the entire logical ring in one voice packet generation cycle. Using this method, a station receives and transmits a single voice packet during each packet generation cycle.

An analysis of the two methods, in terms of the station workload, network load and maximum number of participants in a single conference, was done. The station workload for

method 1 grew substantially as the number of conference participants grew. The workload on the station was constant for method 2 regardless of the number of conference participants. The analysis shows that, in terms of station workload, method 2 is more efficient for any conference with four or more participants. Assuming N conference participants, the network load was N packets per packet generation cycle for both method 1, when multicast capabilities existed, and method 2. Method 1 with no multicast capabilities required $N (N - 1)$ packets per packet generation. Due to the sequential operational nature of method 2, method 1, in general, was shown to support more conference participants in comparison to method 2. But it was noted that method 2 was able to support a substantial number of participants in most cases. More details of the two methods, the analyses and implementations can be found in [Zieg89].

In [Weis88], the unregulated speech control methods, that were just described, were extended for a conference over interconnected networks. The mechanism of the two methods described above remain exactly the same for the workstations participating in the conference. However, the methods had to be extended to include the gateway operation.

For method 1, the gateway receives all the voice packets that are sent by each participating station. The gateway forwards the voice packets it receives form one network onto the other network.

To extend method 2, two possible schemes were described. The differences between the two schemes is how to view the logical ring of conference participants. Method 2a viewed the conference as a one logical ring, spanning the interconnected networks, with one shuttle packet for the whole conference. In constructing the logical ring, care must be taken to minimize network crossovers. In this method, the gateway simply transfers the shuttle packet from one network to the next unchanged.

Method 2b uses a separate logical ring and shuttle packet for each of the

interconnected networks with the gateway being used to articulate among the individual logical rings. The gateway is a participant of each logical ring and receives the shuttle packets of each ring. When the shuttle packet of one logical ring arrives, the gateway

- removes its previous contribution to shuttle packet,
- combines to the shuttle packet with the contents of the shuttle packets of the other rings and
- sends the resultant shuttle packet to its successor on the first ring.

These operations must be done for the shuttle packet of each of the logical rings passing through the gateway. The use of separate logical rings for each interconnected network allows the stations of each network to work in parallel. This allows for a higher number of conference participants.

An analysis of the three schemes, in terms of gateway workload, network load and maximum number of conference participants, has been done. A detailed description of the analysis can be found in [Weis88]. The conclusion derived from this analysis was that method 2b is the preferred scheme. It allows more participants than method 2a with minimal increase in workload and, under certain conditions, the maximum number of conference participants was comparable to method 1.

4.2.3 Multimedia Data Delivery

In [Zieg90], the shuttle packet scheme, described in the previous section, was expanded, to allow its usage for the delivery of both voice and data. In addition to the already present voice data field, two additional data fields, and their associated control fields, were added to the shuttle packet. One is used for acknowledged data delivery (for example, files and memos) and the other field is unacknowledged data delivery (for example, keystrokes and screens). Access control to these additional data fields is via flags which mark the fields either empty or

full. When a station receives a shuttle packet with a data field marked empty, the station can use that field and insert its data into the shuttle packet. The inserted data makes its way around the logical ring and can be copied by all other participants. Upon return, the data field is marked empty. A more detailed presentation is found in [Zieg90].

4.3 Conference Application Sharing

Multimedia conferencing can involve the shared use of application programs and the data they manipulate. An example of this is the case of multiple users that are collaborating on the editing of a technical paper. This brings up two issues:

- input control and
- execution control.

4.3.1 Input Control

When multiple users are sharing an application program, input commands to the program can come from the multiple users. One possible way of dealing with this is to ignore it, resulting in what is termed uncontrolled input. All participants can input commands into the application program without any restrictions. In [Saka90], uncontrolled input was found to be workable for small conferences of four users or less, where the users can communicate easily about the commands they would like to perform. (Of course, this assumes the conference system has voice capabilities.) However, large conferences needed some sort of input control. A conferencing system can impose a input control scheme where only one user at any time has the "floor" and can input commands to the application program. Control of the floor is passed from user to user. [Saka90] describes three schemes for passing input control.

- The chairperson of the conference can decide which user gets the floor next and has the right to input commands to the application program.

- The current holder of the floor can decide which user gets the floor next.
- The floor can be passed in the order the users request the floor. This requires that all requests be sent to a central controller to keep track of the order the request arrived. The central controller informs the users who has the floor next.

By taking advantage of the logical ring of conference participants, a distributed round robin floor passing scheme can be implemented. In this scheme, the holder of the floor passes floor control to its successor in the logical ring.

4.3.2 Execution Control

With the sharing of an application program by multiple users, the question arises how the execution of the shared program take place or, to be more precise, is the program executing on all the participating workstations or just one of therm.

The Rapport system [Enso88a, Enso88b, Ahu90] discusses the two methods. The first method, the multi-site approach, has each station executing its own copy of the application program on its own copy of the data. For each workstation, all input commands destined for the application program are multicast to all workstations. The receiving workstations use it as input to their locally executing program. Using the example of multiple users jointly editing a file, each workstation locally executes the editor on a copy of the file. The keyboard commands of the user (or users, if uncontrolled input is used) currently editing the paper is multicast to all participating workstations and is used as keyboard inputs to their own locally executing editor. The initial implementation of the Rapport system tried this scheme and had difficulty in maintaining consistency among all the workstations.

The Rapport system prefers a single-site approach, in which only one workstation

actually executes the shared application program. Input commands for the application program are sent that executing workstation. The output of the program is multicast to all participating workstations. Using our example, only one workstation executes the editor on one copy of the file. The keyboard commands of the user (or users) editing the paper is sent to that workstation. The output, from the editor to the display, is then multicast to all workstations, to be displayed on their own display screens.

It should be noted, all the mechanisms described in this chapter are supportable by the distributed conferencing protocol presented in the next chapter.

Chapter 5

Distributed Conferencing Protocol

The distributed conferencing protocol, to be presented in this chapter, establishes and manages a multipoint connection among multiple users. The conferencing protocol principally concentrates on conference connection management. However, the protocol also addresses the conference data delivery and application sharing issues, described in the previous chapter, by providing the conference user with different means to deliver conference data. Thus, the purpose of the conferencing protocol is the following:

- The protocol provides the user with the ability to establish and manage a conference connection among multiple users.
- The protocol uses a distributed mechanism for conference connection management by setting up and maintaining a logical ring of conference participants.
- The conference connection is dynamic, in that users can join and leave an ongoing conference.
- The protocol gives the user the ability implement the conference data delivery and application sharing mechanisms of its choice by providing the user with multiple ways to deliver conference data.

In describing the conferencing protocol, care will be taken to use terminology defined by ISO for OSI. In the conferencing protocol, information is passed between the user of the

protocol and the protocol entity through a conference service access point (CSAP) by means of the protocol service primitives (described later). Communications between two peer protocol entities is through Conferencing Protocol Data Units (CPDU).

The protocol provides the user with a set of protocol service primitives for conference connection management. With these service primitives, a user is able to

- invite remote users to a conference (whether for a new conference or to an ongoing conference),
- revoke a pending invitation (that is, an invitation that was neither accepted nor rejected),
- accept an invitation to a conference (to become a conference participant),
- reject a conference invitation,
- inquire about the state of the conference (specifically, which users are participating in the conference),
- suspend the conference connection (to terminate any data transfer and still remain a participant of the conference),
- reconnect a suspended connection (to resume data transfer),
- leave an ongoing conference and
- remove a remote user from an ongoing conference.

Once a conference among two or more users is set up, the user can send and receive data in three different ways. Conference data can be

- multicast to all conference participants,
- sent the user's successor in the logical ring of participants and
- unicast to one conference participant.

Multicasting conference data over a multipoint connection is the principal data delivery service to be provided to the user. The protocol can accomplishes this, as discussed in section 4.1.2.1, by either using the multicast capabilities of the underlying network, if they are available, or by sending the data around the logical ring of conference participants. However, the protocol provides the user various means to deliver data to enable the user to implement all the conference data delivery and application sharing mechanisms, described in the previous chapter. For example, by the protocol providing the user with the ability to send data to its logical ring of conference participants, the user can, if it wishes, implement the shuttle packet method for voice and multimedia data and/or a round-robin regulated speech control, and floor passing mechanism for input control, described earlier. By giving the user the ability to unicast data to only one user of the conference, the user, if it wishes, can implement the centralized floor passing mechanisms for regulated speech control and input control, and/or single-site execution control, described earlier. It also allows the user to send a "memo" directly to one participating conferee, as one might want to do in a conference meeting, without the other participants receiving the memo.

The protocol is responsible for implementing the services provided to the user. The protocol accomplishes this by setting up and maintaining the logical ring of conference participants, described earlier. In order to implement the logical ring, every protocol entity must maintain two pointers. One points to the entity's successor in the logical ring and the other points to its predecessor. These pointers are set up and maintained through the exchange of CPDUs among the peer conferencing protocol entities. A brief explanation of the protocol mechanisms, used to maintain the pointers, follows.

Entity A invites entities B and C to a conference. Entity B is the first to accept the invitation. Entity A initiates the logical ring by making entity B its successor and predecessor.

Entity A informs entity B that its successor is entity A. Entity B assumes entity A to be its predecessor. At this point, the logical ring looks like the following: A → B → A. (The notation used to describe an entity's successor in the logical ring is $x \rightarrow y$. It denotes that the successor of entity x is entity y. Conversely, it also indicates the predecessor of entity y is entity x.) When entity C accepts the invitation, entity A inserts entity C into the logical ring as its successor. Entity A informs entity C that its successor is entity B and entity C assumes that its predecessor is entity A. Entity C then informs entity B that it has a new predecessor. Thus, the logical ring would be: A → C → B → A.

If entity B wants to leave the conference, it informs its predecessor, entity C, that it is leaving. Entity B informs entity C that it should set its successor to point to Entity A. Entity C will then inform entity A that its new predecessor is entity C. This results in a logical ring of A → C → A.

The protocol includes mechanisms that allow an entity to remove a remote entity from an ongoing conference, verify which entities are participating in the conference, suspend and resume a conference connection, and send and receive data. The protocol also contains an error recovery mechanism for breaks in the logical ring. If an entity does not receive a response from it successor, it will try to recover the ring by sending error messages through its predecessor. A detailed description of the protocol mechanisms can be found later in section 5.3.

The user also views a conference as a logical ring of conference participants. This allows the user the ability to implement different types of conference data and application sharing mechanisms, as described previously. However, all the mechanisms involved in maintaining the logical ring, including which remote user is a user's successor, do not have to be known to the user. In other words, the mechanisms used to establish and maintain the logical ring are transparent to the user.

The conferencing protocol creates a conference connection among application processes. It assumes the underlying protocol layer, to which it interfaces, provides transparent end-to-end transmission of data. This interface would occur most naturally at the transport layer of the ISO OSI reference model. Since the conferencing protocol creates a multipoint connection between application processes, the conferencing protocol is viewed as a session layer protocol with the ability to create and regulate a multipoint, session layer, conference connection among multiple session layer users. An application process uses the protocol to create a session layer multipoint connection. It should be noted that, at present, ISO only defines point-to-point communication at the session layer. However, defining a conferencing protocol as part of the session layer can be found in [Leun89] and [Leun90].

[HALS88] describes three parts that are necessary to specify a protocol of any layer of the OSI reference model. Part one is to define the service primitives that are provided to the layer above. Part two is the specification of the internal operation of the protocol. This involves detailed specification of the PDUs; a description of the mechanisms used by the protocol; and a formal protocol specification defining the precise protocol entity operations. Part three defines the services that is expected from the lower layer.

Along these lines, section 5.1 defines the conferencing protocol services provided to the user. Sections 5.2 through 5.4 specifies the internal operation of the conferencing protocol. Section 5.2 details the CPDUs used by the protocol. A description of the protocol mechanisms can be found in section 5.3. A formal protocol specification, in terms of a finite state machine, is in section 5.4. Section 5.5 defines the services provided by the lower layer.

5.1 Conferencing Protocol Services

The conferencing protocol services that are provided to the user, the service primitives, their functions, and their mechanisms are described here. The introduction to the chapter stated

the services a user should expect from the conferencing protocol. The service primitives presented provide the user with those services.

5.1.1 Service Primitives

Table 5.1 lists the protocol service primitives and their parameters. The following is a description of each of the service primitives and their parameters.

Table 5.1: Conferencing Protocol Service Primitives

Service	Type	Parameters
C-INVITE	request	conf_id, inviter, invited list, options
	indication	conf_id, inviter, invited, options
C-INVITE-STATUS	indication	conf_id, inviter, status list
C-ACCEPT	request	conf_id
	indication	conf_id, invited
C-ACCEPT-STATUS	indication	conf_id, status
C-REJECT	request	conf_id, cause
	indication	conf_id, invited, cause
C-REVOKE	request	conf_id
	indication	conf_id
C-LEAVE	request	conf_id
	indication	conf_id, remote user
C-REMOVE	request	conf_id, remote user
	indication	conf_id, source, cause
C-REMOVE-STATUS	indication	conf_id, status
C-STATE	request	conf_id
C-STATE-STATUS	indication	conf_id, status list
C-SUSPEND	request	conf_id
C-SUSPEND-STATUS	indication	conf_id
C-RESUME	request	conf_id

Table 5.1: Conferencing Protocol Service Primitives		
Service	Type	Parameters
C-RESUME-STATUS	indication	conf_id
C-CONF-DATA	request	conf_id, data
	indication	conf_id, source, data
C-SUCC-DATA	request	conf_id, data
	indication	conf_id, data
C-SUCC-DATA-ACK	request	conf_id, data
	indication	conf_id, data
C-DATA	request	conf_id, source, destination, data
	indication	conf_id, source, destination, data
C-DATA-ACK	request	conf_id, source, destination, data
	indication	conf_id, source, destination, data
C-DATA-ACK-STATUS	indication	conf_id, source, destination, status

C-INVITE.request

This service primitive is issued by the service user to invite a list of one or more remote users to a conference. It can be an invitation to form a new conference or to join an ongoing conference. The *invited list* parameter contains the list of invited users. The *conf_id* is a locally defined identification number that is used by the user to identify a conference connection. It is necessary since a user can be a participant of more than one conference at one time. The *options* parameter allows the user to request optional services from the service provider. Such optional services currently include:

- acknowledged unicast data service,
- unacknowledged unicast data service,
- acknowledged successor bound data service and

- unacknowledged successor bound data service.

C-INVITE.indication

This is issued by the service provider to inform the user it has been invited to a conference by the remote user indicated in the *inviter* parameter. The user considers itself a pending invited user with the issuance of this service primitive.

C-INVITE-STATUS.indication

This is issued by the service provider to inform the user of the status of its invitation request. It informs the user of the status of one or more invited remote users that is contained in the *status list*. The *status list* is a list of invited users and the status of the invitation. The status indicates if the invitation was successfully sent to the invited user. The status of the invitation is expected for each invited user.

C-ACCEPT.request

This is issued by an invited user, in response to a C-INVITE.indication. It is a request to accept the invitation and become an conference participant.

C-ACCEPT.indication

This is issued by the service provider to indicate that an invited user, contained in the *invited* parameter, has accepted the invitation and is a conference participant. All participating users are informed, by this service primitive, when an invited user accepts an invitation. The inviting user that initiated the conference considers the conference active, and itself an active participant, with the first invocation of this primitive. It should be noted that a user who becomes an active participant remains so until it either leaves the conference, is removed from the conference or becomes a suspended participant.

C-ACCEPT-STATUS.indication

This is issued by the service provider to inform the invited user of the status of its

request to accept the invitation. The *status* parameter contains the success or failure of the request. With the issuance of this primitive indicating success, the user becomes an active participant. On failure, the user has the option of either trying again, by issuing a C-ACCEPT.request, or rejecting the invitation, by issuing a C-REJECT.request.

C-REJECT.request

This is issued by an invited user, in response to a C-INVITE.indication. It is a rejection of the conference invitation. The reason for the rejection is contained in the *cause* parameter.

C-REJECT.indication

This is issued by the service provider to inform the service user that a conference invitation has been rejected by the invited remote user, contained in the *invited* parameter. The *cause* parameter contains the reason for the rejection. This is issued only to the user that sent the invitation.

C-REVOKE.request

This is issued by the user to revoke all pending conference invitations, that is, invitations to which the invited users have not responded. Currently, the service does not allow the user to revoke the invitation of one or some of the pending invited users. It is all or nothing.

C-REVOKE.indication

This is issued by the service provider to inform the local user that its conference invitation has been revoked and the user is no longer invited to the conference.

C-LEAVE.request

This is issued by a participating user who wishes to leave an ongoing conference. At the issuance of the primitive, the user no longer considers itself a conference participant.

C-LEAVE.indication

This is issued by the service provider to inform the user that a participating user has left the conference. All conference participants are notified when a user leaves the conference.

C-REMOVE.request

This is issued by a participating user to request that a remote participating user be forcibly removed from the conference. The protocol allows, with no restrictions, the user to remove any remote participating user at any time. The user does not consider the remote user removed from the conference until the user receives a confirmation from the service provider.

C-REMOVE.indication

This is issued by the service provider to inform the user that it has been removed from the conference and is no longer a participant. The user can be forcibly removed from the conference at the request of a remote user, specified in the *source* parameter, who has issued a C-REMOVE.request. The user can, also, be removed from the conference as a result of the conference being over. A conference is considered over when there is only one conference participant left in an active conference. The service provider recognizes this and informs the user that the conference is over. The *cause* parameter specifies whether the removal was due to a remote user request or as a result of the conference being over. (The other participating users will be informed that the removed user is no longer a participant through a C-LEAVE.indication service primitive, which will be issued by each local protocol entity.)

C-REMOVE-STATUS.indication

This is issued by the service provider to inform the service user of the status of its request to remove a remote user from the conference. The *status* parameter specifies if the request was successfully fulfilled.

C-STATE.request

This is issued by the user to request information about the state of the conference, specifically, which users are participating in the conference.

C-STATE-STATUS.indication

This is issued by the service provider to inform the user of the status of its conference state request. The *status list* contains the list of users participating in the conference and their status in the conference. The status can be either indicate the user is active or suspended.

C-SUSPEND.request

This is issued by the user to suspend the conference connection and become a suspended participant of the conference. Suspended users neither send nor accept any conference data over the conference connection. Any data the service provider receives is not passed up to the suspended user.

C-SUSPEND-STATUS.indication

This issued by the service provider to inform the user of the status of its suspend connection request.

C-RESUME.request

This issued by the service user to request the resumption of the suspended connection and, once again, become an active participant.

C-RESUME-STATUS.indication

This is issued by the service provider to inform the user of the status of its request to resume the suspended connection. The user, once again, becomes an active participant with the issuance of this service primitive.

C-CONF-DATA.request

This is issued by the user to multicast conference data to all conference participants.

The service provider attempts to deliver the data to all conference participants. However, the service provider does not guarantee that all (or any) of the participants receive the data. The *data* parameter contains the conference data.

C-CONF-DATA.indication

This is issued by the service provider to inform the user that it has received multicast conference data. The user that sent the data is specified in the *source* parameter. The *data* parameter contains the conference data.

C-SUCC-DATA-ACK.request and C-SUCC-DATA.request

These service primitives are issued by the user to send conference data to the user's successor in the logical ring. The user does not, necessarily, have to know which remote user is its successor. The user relies on the service provider to maintain that information and to deliver the data to the proper user. With the C-SUCC-DATA-ACK.request service primitive, the user is provided an acknowledged data delivery service. It should be noted that if the user's successor is a suspended participant, the data will be delivered, by the service provider, through the logical ring to the next active participating user. The conference data is specified in the *data* parameter.

C-SUCC-DATA-ACK.indication and C-SUCC-DATA.indication

These service primitives are issued by the service provider to inform the user that it received data from its predecessor in the logical ring of participating users. The *data* parameter contains the conference data.

C-DATA-ACK.request and C-DATA.request

This is issued by the user to unicast conference data to only one specific remote participant of the conference, who is not, necessarily, the sender's successor. There is no point-to-point connection between the two users. This a datagram service provided to the user so it

can send data, such as a "memo", to only one user of the conference. With the C-DATA-ACK.request, the protocol provides acknowledgement of data delivery; that is, an acknowledged datagram service. The *destination* parameter specifies to which participant the data is to be sent. The *data* parameter contains the conference data.

C-DATA-ACK.indication and C-DATA.indication

These are issued by the service provider to inform the user that it received data from a participant of the conference, who is not, necessarily, the receiver's predecessor. The user that sent the data is specified in the *source* parameter. The *data* parameter contains the data.

C-DATA-ACK-STATUS.indication

This is issued by the service provider to inform the user of the status of its request to unicast data to the *destination* user. The *status* parameter indicates whether the service provider was successful in delivering the data.

5.1.2 Primitive Mechanisms

This section describes how a service user would use the above specified protocol service primitives to setup and maintain a conference. The description includes timing diagrams to depict how the service users communicate with each other.

A user, who initiates the conference, invites one or more remote users to a conference by issuing a C-INVITE.request service primitive. The inviting user specifies the list of invited users in the *invited list* parameter. The inviting user expects to be informed of the status of the invitation request by the service provider through the C-INVITE-STATUS.indication service primitive. The status indicates, in the *status* parameter, whether the invitation been successfully sent to the invited users. A status report is expected for each of the invited users. Each one of the invited users receives the invitation through the C-INVITE.indication service primitive. An invited user that receives an invitation becomes a pending invited user. The inviting user expects

an answer from each of the pending invited users, either accepting or rejecting the invitation. All the pending invitation can be revoked by the inviting user through the C-REVOKE.request service primitive.

The invited user can accept the invitation by issuing a C-ACCEPT.request service primitive. The invited user is not considered a conference participant until the service provider issues a C-ACCEPT-STATUS.indication primitive with the *status* parameter indicating a successful connection. All remote participating users, including the inviting user, receive the invited user's acceptance through the C-ACCEPT.indication. The protocol provides for the asynchronous acceptance of invitations. Thus, at the issuance of the first C-ACCEPT.indication, the conference is considered active. All additional C-ACCEPT.indication primitives are additional users joining an active conference.

Alternately, the invited user can reject the invitation by issuing a C-REJECT.request service primitive and can use the *cause* parameter to convey the reason for the rejection. The receipt of a C-REVOKE.indication by an invited user revokes its invitation and the user is no longer invited to the conference.

Figure 5.1 is a timing diagram depicting the setup of a conference connection. The figure depicts the interaction that might occur between four users, users A, B, C and D. In the figure, each user is represented by a column. The vertical lines represent the service access point between the user and the service provider. The entries in the column represent the issuance of protocol service primitives, with the arrows indicating the source of service primitive. An arrow pointing from a service primitive towards the vertical line indicates a primitive issued by the user. An arrow pointing from the vertical line towards a service primitive indicates a primitive issued by the service provider. The figure starts with user A inviting user B, C and D to a conference. At the end, user C rejects the invitation and users A, B, and D

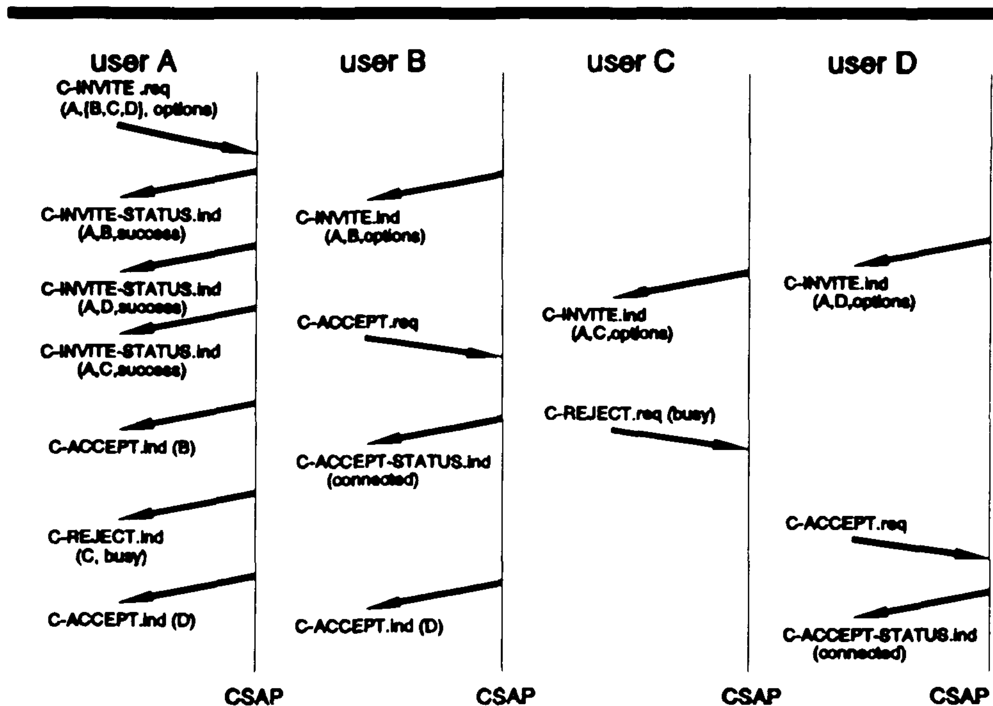


Figure 5.1: A timing diagram depicting a conference connection setup. At the end users A, B and D are conference participants.

are participants of the conference.

A participating user can send conference data in three ways. Data can be multicast to all conference participants by issuing a C-CONF-DATA.request service primitive. The conference users receive the data through the C-CONF-DATA.indication. The user can also issue a C-SUCC-DATA(-ACK).request to send data to its successor in the logical ring of participants maintained by the service provider. The successor receives this data through the C-SUCC-DATA(-ACK).indication. Finally, a user can send data to a one specific conference participant by issuing a C-DATA(-ACK).request primitive. The specified remote user receives the data through the C-DATA(-ACK).indication service primitive.

Application programs can implement shuttle packet data delivery by using the

C-SUCC-DATA or C-SUCC-DATA-ACK service primitives. Shuttle packet transmission is transparent to the conferencing protocol entity. The fact that it is a shuttle packet is only known to the application program. A participating user sends a shuttle packet to its successor by issuing a C-SUCC-DATA(-ACK).request. A user receives the shuttle packet through the C-SUCC-DATA(-ACK).indication service primitive. The user performs the necessary functions related to the shuttle packet, and sends the data as a shuttle packet to its successor by issuing a C-SUCC-DATA(-ACK).request service primitive.

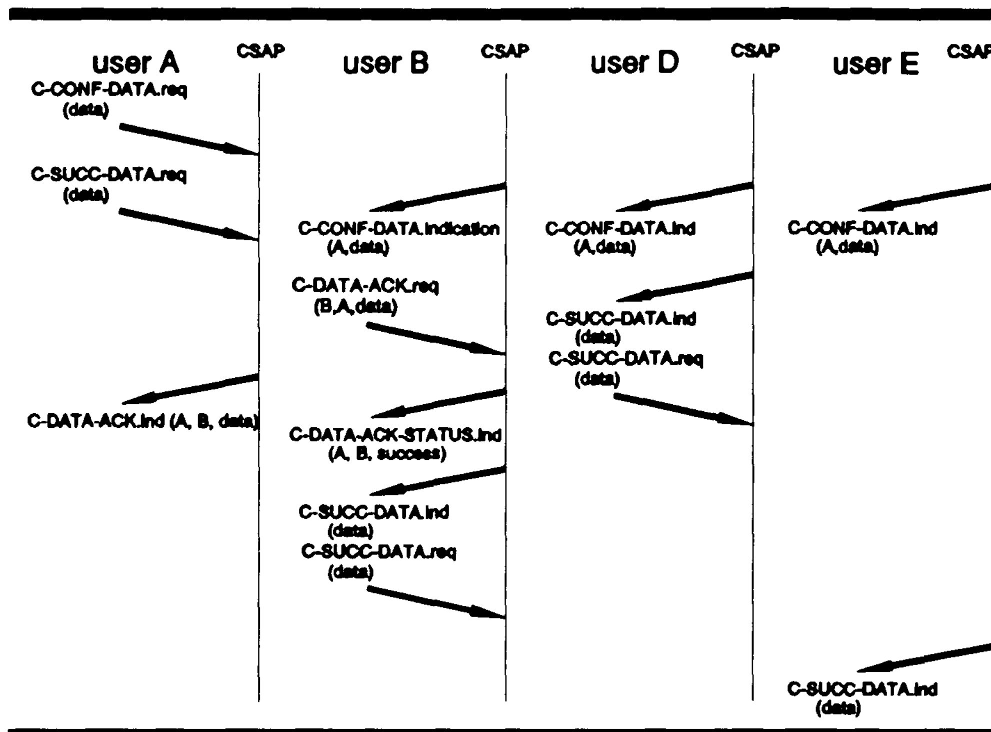


Figure 5.2: A timing diagram showing the transfer of conference data.

Figure 5.2 depicts the use of the data transfer service primitives. In this figure it is assumed that users A, B, D and E are active participants of the conference. The underlying logical ring, set up by the protocol, looks like the following: A → D → B → E → A. This figure

first shows user A sending data to all conference participants by issuing a C-CONF-DATA.request. This results in users B, D and E receiving the data from user A through the C-CONF-DATA.indication service primitive.

The figure then shows user A sending data to its logical successor by issuing a the C-SUCC-DATA.request service primitive. User D, user A's successor, receives the data through the C-SUCC-DATA.indication primitive. User D then sends the data to its successor by issuing a C-SUCC-DATA.indication. This results in user B receiving the data and then sending the data to its successor, user E.

The figure also shows user B sending unicast data to one specific remote participant, user A. This occurs at about the same time user D receives data from its predecessor. The figure shows user B issuing a C-DATA-ACK.request service primitive. The *destination* specifying user A as the destination for the data. This results in user A receiving the data through the C-DATA-ACK.indication primitive, with the *source* parameter specifying user B as the source of the data. User B's request is confirmed through the C-DATA-STATUS-ACK.indication.

The service primitives used to inquire of the state of the conference, suspend the conference connection and resume a suspended connection are straightforward. A user issues a C-STATE.request and expects a response from the service provider through the C-STATE-STATUS.request service primitive with the *status list* containing a list of the participating users. A conference connection can be suspended by the user issuing a C-SUSPEND.request and the service provider responding with a C-SUSPEND-STATUS.indication. A suspended connection can be reconnected by the user issuing a C-RESUME.request and the service provider responding with a C-RESUME-STATUS.indication. Currently, a suspended participant can not send or receive

data. All other services can be used, without restriction, by the suspended user.

A user can voluntarily leave an ongoing conference by issuing a C-LEAVE.request. The user no longer considers itself a conference participant with the issuance of this service primitive. All conference participants are informed, through C-LEAVE.indication, that a remote user has left the conference.

A user can request the removal of a remote user from the conference by issuing a C-REMOVE.request service primitive. Currently, there is no restriction on the use of this service. Any user can forcibly remove any remote user from the conference. The remote user receives this request through the C-REMOVE.indication and considers itself removed from the conference. The user, requesting the removal of the remote user, receives a confirmation from the service provider through the C-REMOVE-STATUS.indication service primitive. The *status* parameter indicates success. All other users will be informed the removed user is no longer a conference participant through the C-LEAVE.indication service primitive.

Figure 5.3 is a timing diagram that depicts the ending of the conference by the use of the C-REMOVE and C-LEAVE service primitives. As before, it is assumed that users A, B, D and E are active participants and the underlying logical ring looks like the following: A → D → B → E → A. Initially, user A removes user B from the conference by issuing a C-REMOVE.request. User B receives the removal request through C-REMOVE.indication. User A receives a confirmation through the C-REMOVE-STATUS.indication service primitive. Alt the other user are issued a C-LEAVE.request.

The diagram also shows, during the exchange between users A and B, user D leaves the conference by issuing a C-LEAVE.request service primitive. This then results in user A and E receiving a C-LEAVE.indication. Then User A leaves the conference and, for all practical purposes, the conference is over, since user E is the only participating user of the conference.

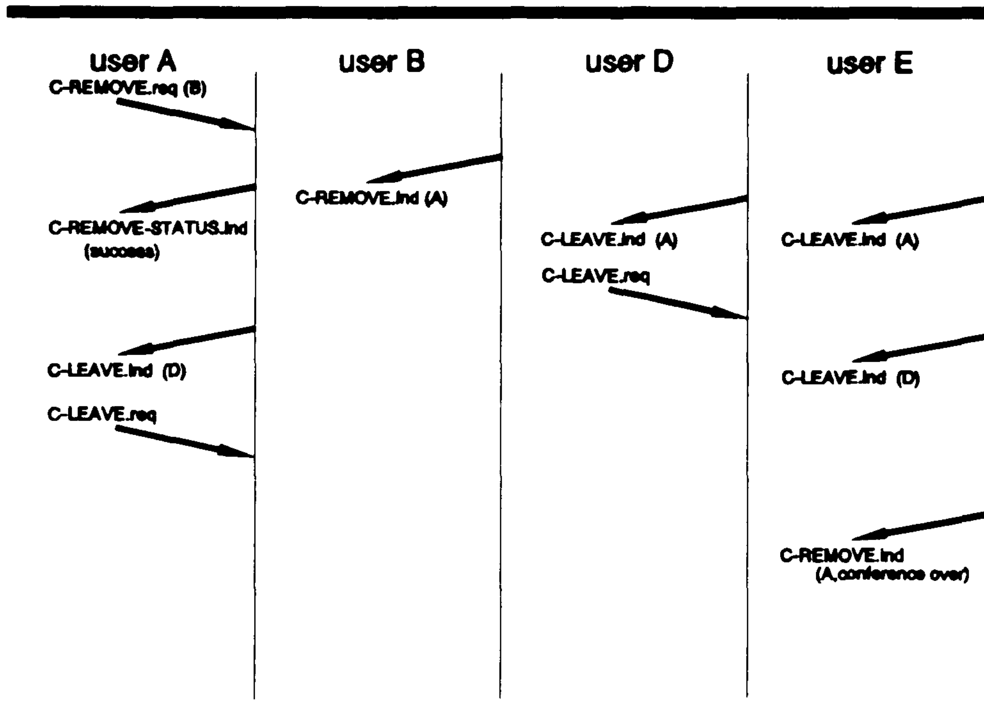


Figure 5.3: A timing diagram depicting users being removed from the conference and leaving the conference. A the end, the conference is over.

Therefore, user E receives a C-REMOVE.indication with the parameters indicating that the conference is over.

5.2 Conferencing Protocol Data Units

Peer conferencing protocol entities communicate with each other through Conferencing Protocol Data Units (CPDUs). The CPDUs are used by the protocol entities to establish and manage a conference connection. This is accomplished by setting up and maintaining a logical ring of conference participants. The CPDUs are also used to transfer data among the participating entities. This section specifies the structure and functions of these CPDUs.

The protocol in many ways mimics the service primitives that are provided to the user. A protocol entity can:

- invite remote protocol entities to a conference;
- accept a conference invitation from a remote entity;
- reject a conference invitation from a remote entity;
- revoke a pending invitation sent to a remote entity;
- leave an ongoing conference;
- forcibly remove a remote entity from an ongoing conference;
- verify the participants of the conference;
- suspend and reconnect the conference connection;
- multicast conference data to all conference participants;
- send data to its logical successor; and
- unicast data to a single remote participating entity.

As stated in the introduction to the chapter, the protocol manages the conference connection by setting up a logical ring of conference participants. This is implemented by the protocol maintaining two pointers; one is for its predecessor and the other for its successor . Thus, the protocol must also be able to:

- set up the logical ring by setting and maintaining its successor and predecessor pointers;
- modify its predecessor and successor pointers, in order to allow protocol entities to added an deleted from the conference; and
- recover from an erroneous break in the logical ring.

The above functions are accomplished through the exchange of CPDUs between the peer protocol entities.

5.2.1 CPDU Types and Functions

Table 5.2: Conferencing Protocol Data Units					
Type	Parameters	Code	Type	Parameters	Code
AC	STATUS, SET_SUCC	00H	ACC		01H
AR		02H	DC		03H
DCR	DATA	04H	DR	DATA	05H
DR-ACK	DATA	06H	DSC	SEQ#	07H
DSR	DATA	08H	DSR-ACK	SEQ#, DATA	09H
IC		0AH	IR	CONF_ID, OPTIONS	0BH
LC	LEAVING	0CH	LR	SET_SUCC, PASS, ORIG	0DH
PRC		0EH	PRR	ORIG, NR_PRED	0FH
RJR	CAUSE	10H	RMC		11H
RMR		12H	RVR		13H
SPC		14H	SPR		15H
SRC		16H	SRR	ORIG, NR_SUCC	17H
SSC		18H	SSR		19H
STR	ORIG, LIST	1AH			

Table 5.2 lists the CPDUs, their parameters and their codes. The following defines the functions of each of the CPDU types listed in the table. In addition to the parameters shown, all CPDUs contain a source and destination field (shown later in figure 5.4).

AC - Accept Confirm

The AC CPDU confirms the AR CPDU sent by an invited entity. The reply to the invited entity's request to be inserted into the logical ring of participants is in the STATUS parameter. If at the time the inviting entity received the request it is unable to insert the invited entity into the logical ring, the STATUS parameter information field contains a WAIT. This indicates to the invited entity to send another AR CPDU sometime later.

The STATUS parameter information field contains SUCCESS and the CPDU

contains the SET_SUCC parameter, if the invited entity can be inserted into the logical ring of participants. The SET_SUCC parameter information field contains the remote protocol entity, which is now the inviting entity's current successor, that is to become the invited entity's successor. The inviting entity assumes its new successor to be the invited entity and the invited entity assumes its predecessor to be the inviting entity.

ACC - AC Confirm

The ACC CPDU is multicast by the invited entity to all conference participants. It confirms the linkage information that is sent by the inviting entity in the AC CPDU and informs all participants of a new participating entity.

AR - Accept Request

The AR CPDU is sent by an invited entity to accept the conference invitation and request to be inserted into the logical ring of participants.

DC - Data Confirm

The DC CPDU confirms the data sent in the DR-ACK CPDU.

DCR - Data Conference Request

The DCR CPDU contains conference data that is multicast to all participating entities in the conference.

DR - Data Request

The DR CPDU contains conference data that is being unicast to one participating entity of the conference.

DR-ACK - Data Request ACK

The DR-ACK CPDU contains conference data that is being unicast to one participating entity of the conference and requires an acknowledgement from the remote protocol entity to which the DR-ACK CPDU is being sent.

DSC - Data Successor Confirm

The DSC CPDU confirms the data sent in the DSR-ACK CPDU. The SEQ# identifies which DSR-ACK is being confirmed.

DSR - Data Successor Request

The DSR-CPDU contains the conference data that is being sent to the entities successor in the logical ring of participating entities.

DSR-ACK - Data Successor Request ACK

The DSR-ACK CPDU contains the conference data that is being sent to the entities successor in the logical ring of participating entities and requires an acknowledgement from the successor. The SEQ# identifies the DSR-ACK CPDU and is used to avoid duplication.

IC - Invitation Confirm

The IC CPDU is sent to the inviting entity to confirm the invitation request.

IR - Invitation Request

The IR CPDU is used to invite a remote protocol entity to participate in a conference. The CONF_ID parameter information field contains a conference identification number which is set by the initiator of the conference. The OPTIONS parameter information field contains the options that will be in effect for this conference. The OPTIONS information field values mimic the options described for the protocol service.

- UNACKED_DATA - indicates unacknowledged unicast and successor bound data.
- ACKED_SUCC_DATA - indicates unacknowledged unicast data and acknowledged successor bound data.
- ACKED_UNI_DATA - indicates acknowledged unicast data and unacknowledged successor bound data.

- ACKED_DATA - indicates acknowledged unicast data and successor bound data.

LC - Leave Confirm

The LC CPDU is multicast to all participating entities to inform them that a protocol entity, contained in the LEAVING parameter information field, has left the conference. It also confirms the leaving entity's LR CPDU.

LR - Leave Request

The LR CPDU is a request to leave the conference. It is sent to the entity's predecessor in the logical ring of conference participants. The SET_SUCC parameter information field contains the remote protocol entity, which is now the leaving entity's current successor, that is to become the new successor of the entity that receives the leave request. The PASS and ORIG parameter is included if the entity is not the originator of the leave request. (This happens when two or more successive entities in the logical ring leave at the same time. A leaving entity that receives a LR CPDU, passes the CPDU to its predecessor.) The ORIG parameter information field then contains the originating entity of the leave request.

PRC - Predecessor Recovery Confirm

The PRC CPDU is sent to confirm the PRR CPDU.

PRR - Predecessor Recovery Request

The PRR CPDU is sent to the entity's successor to indicate the logical ring has been broken. A participating entity, contained in the ORIG parameter information field, has lost contact with its predecessor, contained in the NR_PRED parameter information field. This is an attempt to reestablish the integrity of the logical ring.

RJR - ReJect Request

The RJR CPDU rejects the invitation request.

RMC - ReMove Confirm

The RMC CPDU confirms the RMR CPDU and that the entity is removing itself from the conference.

RMR - ReMove Request

The RMR CPDU is sent to a remote entity to remove it from the conference.

RVR - ReVoke Request

The RVR CPDU is sent to revoke a pending invitation from an entity that has not responded with an AR or RJR CPDU.

SPC - Set Predecessor Confirm

The SPC CPDU is sent to confirm the SPR CPDU.

SPR - Set Predecessor Request

The SPR CPDU is sent to inform a remote participating entity of its new predecessor. The receiving entity assumes its new predecessor to be the entity that sent the SPR CPDU.

SRC - Successor Recovery Confirm

The SRC CPDU is sent to confirm the SRR CPDU.

SRR - Successor Recovery Request

The SRR CPDU is sent to the entity's predecessor to indicate the logical ring has been broken. A participating entity, contained in the ORIG parameter information field, has lost contact with its successor, contained in the NR_SUCC parameter information field. This is an attempt to reestablish the integrity of the logical ring.

SSC - Set Successor Confirm

The SSC CPDU is sent to confirm the SSR CPDU.

SSR - Set Successor Request

The SSR CPDU is sent to inform a remote participating entity of its new logical

successor. The receiving entity assumes its new successor to be the entity that sent the SSR CPDU.

STR - STate Request

The STR CPDU is used to verify the current entities participating in the conference. It is originally sent by the entity contained in the ORIG parameter information field and is passed around the logical ring from entity to entity. When the originating entity sends the STR CPDU, it only contains the ORIG parameter. Eventually, the STR CPDU makes its way back to the originating entity and contains one or more LIST parameters, in addition to the ORIG parameter. Each LIST parameter represents a participating entity and its status (active or suspended).

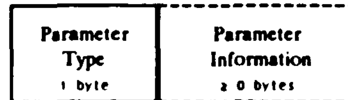
5.2.2 CPDU Format

Figure 5.4 shows the format of CPDUs. The figure shows the fields, and their lengths in bytes, that make up a CPDU. The figure depicts four types of data units. Figure 5.4(a) shows the fields of the control CPDU. Figure 5.4(c) through figure 5.4(e) depict the format of the data CPDUs. Figure 5.4(c) shows the format of the DCR and DSR CPDUs. The format of the DSR-ACK CPDU is shown in figure 5.4(d). The format of the DR and DR-ACK CPDUs is shown in figure 5.4(e).

The first three fields of all types of CPDUs are the same. The first field is the CPDU type. Table 5.2 lists all valid CPDU types and their codes. The second field uniquely defines the source entity, the entity that sends the CPDU. The third field uniquely defines the destination of the CPDU, the entity that should receive the data unit. If the CPDU is being multicast to all participating entities, the destination field contains the conference identification number.

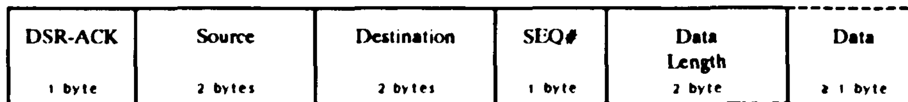
The CPDUs start to differentiate at the fourth field. The fourth field of the DR and DR-ACK CPDU, figure 5.4(e), is the conference identification number. The fourth field of the

(a) Control CPDU

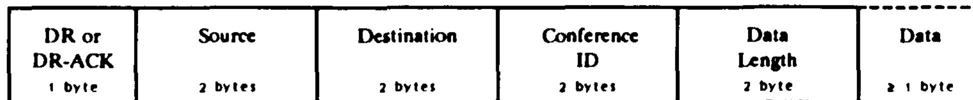


(b) Parameter Field of Control CPDU

(c) Multicast and Unacknowledged Successor Bound Data CPDU



(d) Acknowledged Successor Bound Data CPDU



(e) Unicast Data CPDU

Figure 5.4: CPDU format. The dashed lines indicate a variable length field of the CPDU.

DSR-ACK CPDU of figure 5.4(d) is a SEQ# used to detect duplicate CPDUs. The fifth field of the DR, DR-ACK CPDU and DSR-ACK CPDUs, and the fourth field of the DCR and DSR CPDUs of figure 5.4(c) is an unsigned integer indicating the length of the data field. This is followed by the data field, which contains the actual conference data.

The control CPDU can include parameter fields that contain supplementary information associated with the CPDU type. A control CPDU can contain zero or more parameter fields. The fourth field of a control CPDU, figure 5.4(a), is an unsigned integer

Parameter Type	Cause Type Information Field	Status Type Information Field	Option Type Information Field	Code
NR_PRED	BUSY	FAILED	UNACKED_DATA	0
NR_SUCC	LINK_BUSY	SUCCESS	ACKED_SUCC_DATA	1
SET_SUCC	LEAVING	WAIT	ACKED_UNI_DATA	2
ORIG	REJECTED		ACKED_DATA	3
LEAVING				4
LIST				5
STATUS				6
OPTIONS				7
CAUSE				8
PASS				9
CONF_ID				10
SEQ#				11

NOTE: The CONF_ID parameter is associated with a 2 byte information field containing the conference identification number. The CAUSE, STATUS and OPTIONS parameters include a 1 byte information field specified in the above three middle columns. The PASS parameter has no information field. The SEQ# parameter information field is a 1 byte number. The LIST information field is three bytes. The first two bytes identify a conference protocol entity. The third byte indicates if the entity is ACTIVE (0) or SUSPENDED (1). The other parameter types include a two byte information field that uniquely identifies a conference protocol entity.

Figure 5.5: A list of the CPDU parameter types and their information fields.

indicating the number of parameter fields in the data unit. This is followed by zero or more parameter fields.

A parameter field, shown in Figure 5.4(b), is made up of two subfields. The first subfield is the parameter type. A parameter type can be associated with a parameter information field and is the second subfield of the parameter field. The far left column of figure 5.5 lists all valid parameter types, with its code specification in the far right column of the figure. The middle three columns of figure 5.5 lists the valid information fields associated with the STATUS, CAUSE and OPTIONS parameter types. The specified codes for the information fields are in the far right column of the figure. The note that is associated with Figure 5.5 specifies the valid information fields of the other parameter types.

5.3 Conferencing Protocol Mechanisms

In the previous section, the structure and functions of the CPDUs were described.

Here, the mechanisms used to implement the services provided to the user, utilizing the CPDUs, are presented. These include the mechanisms that are used to setup and maintain the logical ring of conference participants, as well as those used for data delivery. Timing diagrams are included to show the exchange of CPDUs between the peer conferencing protocol entities.

The section is split into three parts. The first presents the mechanisms necessary for conference connection management. This includes conference initiation, sending, rejecting and accepting invitations, adding protocol entities to and deleting protocol entities from the logical ring of conference participants, and conference termination. The second part describes conference data delivery. This includes multicasting data to all conference participants, sending data to the entity's successor and unicasting data to one participant. The third part describes the mechanisms used to detect and recover protocol errors.

5.3.1 Conference Connection Management

The purpose of the mechanisms described in this section is to establish and manage the conference connection. As mentioned previously, the protocol views the conference connection as a logical ring of conference participants. Every participating protocol entity maintains two pointers. One points to its successor in the logical ring and the other to its predecessor. The mechanisms described in this section are used by the protocol to set the successor and predecessor pointers when becoming a conference participant. The mechanisms can also be used to change the pointers to allow other entities to be added to or leave the conference.

5.3.1.1 Conference Initiation

A conference is initiated by a user, the *inviter*, issuing a C-INVITE.request. The *invited list* parameter contains a list of one or more remote CSAPs that are being invited to the conference. If the protocol entity can not create a connection, the protocol issues a

C-REMOVE.indication. Otherwise, the protocol sends conference invitations, as described in the next section.

5.3.1.2 Sending a Conference Invitation

The inviting user can request to send a conference invitation by issuing a C-INVITE.request. The protocol entity performs the following:

- selects a conference identification number, if the inviting entity is initiating the conference;
- sends an IR CPDU to each entity on the *invited list*, inviting them to participate in the conference;
- waits to receive an IC or RJR CPDU from each of the invited entities.

At this point, the invited entities are *unconfirmed* invited entities.

An entity that receives an IR CPDU and can not create a conference connection responds by sending a RJR CPDU to the inviting entity. Otherwise, the entity

- sends an IC CPDU to the inviting entity;
- informs the local user of the invitation by issuing a C-INVITE.indication service primitive.
- waits for the local user to either accept or reject the invitation.

With the receipt of an IC CPDU by the inviting entity from an *unconfirmed* invited entity, the *unconfirmed* invited entity becomes a *pending* invited entity. The protocol informs the local user that a remote invited user received the invitation by issuing a C-INVITE-STATUS.indication service primitive. The entity then waits for a either an acceptance or rejection of the invitation from the remote protocol entity.

The receipt, by an inviting entity, of a RJR CPDU, from the remote entity, indicates

the remote entity is not able to or does not wish to participate in the conference call. The local user is informed of this through the C-REJECT.indication primitive. If at this point there are neither any entities participating in the conference, any *pending* invited entities, nor any *unconfirmed* invited entities, the connection attempt is terminated. The local protocol user is informed that the connection attempt failed through the C-REMOVE.indication service primitive.

Figure 5.6 is a timing diagram that depicts the exchange of CPDUs that occurs when an entity sends out conference invitations. The figure shows the exchange of the CPDUs between entities A, B, C and D. Each entity is represented by two column. The left column shows the exchange of protocol service primitives between the protocol user and the protocol entity. The single vertical line represents the interface between the protocol user and the protocol entity at the CSAP. The direction of the arrow indicates whether the protocol user or protocol entity issued the service primitive. A user issued service primitive results in sending the CPDU, shown in the right column, to a peer protocol entity. A protocol issued service primitive is the result of receiving the CPDU, shown in the right column, from a peer protocol entity.

The right column shows the exchange of CPDUs between the entity and other peer protocol entities. The double line represents the interface to the underlying layer protocol services that are used to send and receive data between the entity and other peer protocol entities. The direction of the arrow indicates whether the CPDU is being sent or received by the entity.

In figure 5.6, entity A, at the protocol user's request, invites entities B, C, and D to participate in a conference by sending each of the an IR CPDU. The figure then shows entities B and C confirming the invitation with an IC CPDU and entity D rejecting the invitation with

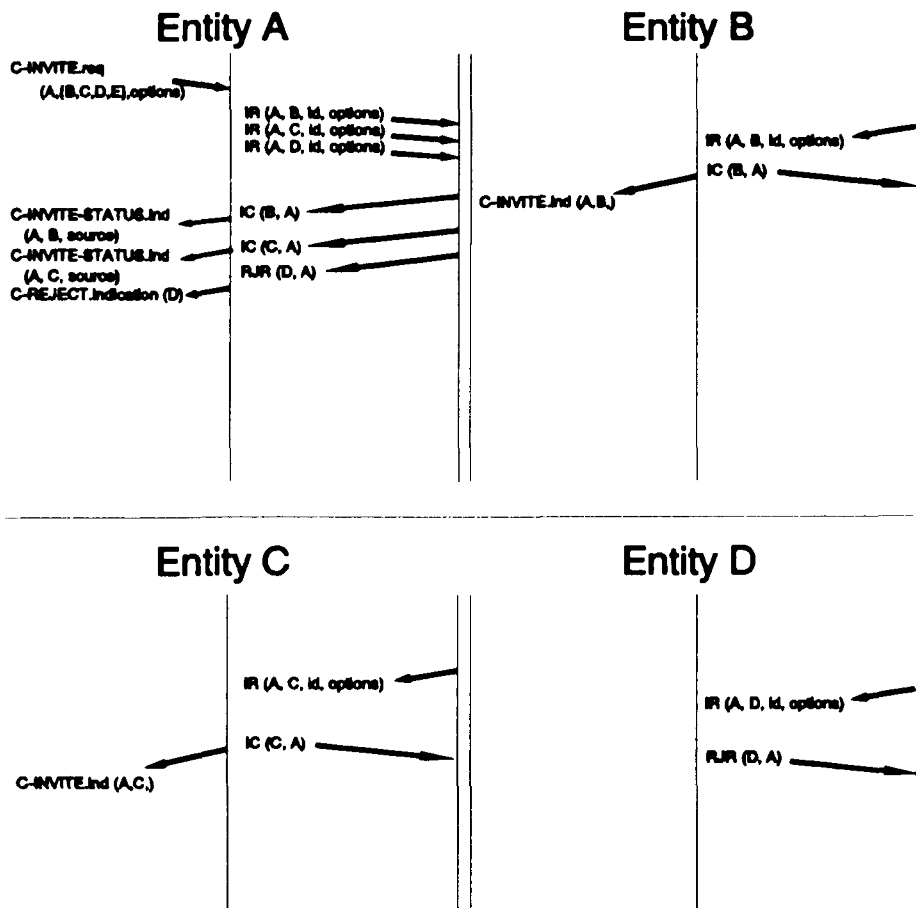


Figure 5.6: This a timing diagram that shows Entity A inviting entities B, C, and D to participate in a conference. At the end, entities B and C confirm the invitation and entity D rejects the invitation.

a RJR CPDU.

5.3.1.3 Revoking a Conference Invitation

An inviting entity can revoke all pending invitation by sending a RVR CPDU to all

pending invited entities. A *pending* invited entity, that receives a RVR CPDU, no longer considers itself an invited entity and informs the local user of this by issuing a C-REVOKE.indication service primitive. An inviting entity revokes the *pending* invitations at the request of the local user, that issues a C-REVOKE.request, or when leaving the conference.

5.3.1.4 Invitation Response

An invited user can reject an invitation by issuing a C-REJECT.request service primitive. The invited entity will then send an RJR CPDU to the inviting entity. An inviting entity that receives a RJR CPDU from an invited entity informs the local user, by issuing a C-REJECT.indication service primitive, that the invited user rejected the invitation.

An invited user can accept an invitation request by issuing a C-ACCEPT.request service primitive. The invited entity then sends an AR CPDU to the inviting entity and then wait for an AC CPDU from the inviting entity. The invited entity becomes an *accepting* invited entity.

An inviting entity that receives of an AR CPDU from a *pending* invited entity performs the following:

- It changes its successor pointer to point to the *accepting* invited entity, thus, inserting the invited entity into the logical ring of participating entities;
- The entity sends an AC CPDU to the invited entity with following parameters:
    - A STATUS parameter with its information field containing SUCCESS.
    - A SET_SUCC parameter with its information field containing the inviting entity's previous successor,

which will become the invited entity's successor. If this is the first AR CPDU to be received from an invited entity, the SET_SUCC parameter will contain the inviting entity, since the inviting entity has no successor.

- It then waits for an ACC CPDU from the invited entity, confirming the receipt of the linkage information.

It should be noted that the conference connection becomes active with sending of the first AC CPDU.

It is possible, that at the time the invited entity receives the AR CPDU, the invited entity can not be inserted into the logical ring. This can happen, for example, when the inviting entity is involved with inserting another invited entity into the ring. The inviting entity then sends an AC CPDU with the STATUS parameter information field containing WAIT.

An *accepting* invited entity that receives an AC CPDU with a STATUS parameter information field containing WAIT, waits a short amount of time and retransmits the AR CPDU. If the STATUS parameter information field contains SUCCESS, the *accepting* entity considers itself an active participant and performs the following:

- sets its predecessor to point to the inviting entity;
- sets its successor to point to the entity which is in the SET_SUCC parameter;
- multicasts an ACC CPDU to all conference participants, to inform them of a new conference participant and to confirm the linkage information sent by the inviting entity in the AC CPDU;
- informs the protocol user that it is a conference participant by issuing

a C-ACCEPT-STATUS.indication service primitive; and

- perform the set predecessor mechanisms of the next section (5.3.1.5) in order to inform its successor of its new predecessor. If the entity is the first one to accept an invitation, then the entity's predecessor and successor are equal and the entity's successor already knows its predecessor. Thus, there is no need to perform the set predecessor mechanism.

An entity that receives an ACC CPDU issues a C-ACCEPT.indication informing the local user of the new conference participant.

Figure 5.7 displays the exchange of CPDUs that occurs when a *pending* user either accepts or rejects a conference invitation. The figure assumes entity A to be the inviter and entities B, C and D to be *pending* invited entities. Entities B and D accept the invitation and entity C rejects the invitation. In the figure, the "ID" used in the ACC CPDU indicates that the destination field of the CPDU is the conference identification number and the CPDU is being multicast to all conference participants. At the end, entities A, B and D are conference participants with the logical ring looking like the following: A → D → B → A.

5.3.1.5 Set Predecessor

An entity that sets its logical successor, by receiving an AC CPDU, or resets its logical successor, by receiving a LR CPDU, must inform its new successor of its new predecessor. This is done by sending a SPR CPDU to its new successor. An entity that receives an SPR CPDU updates its predecessor pointer to point to the source of the data unit and sends a SPC CPDU to its new predecessor. Figure 5.7, described in the previous section, shows the exchange of the SPR and SPC CPDUs between entities B and D. This occurs after entity B accepts the invitation and receives its successor information from the inviter, entity A, in the AC CPDU.

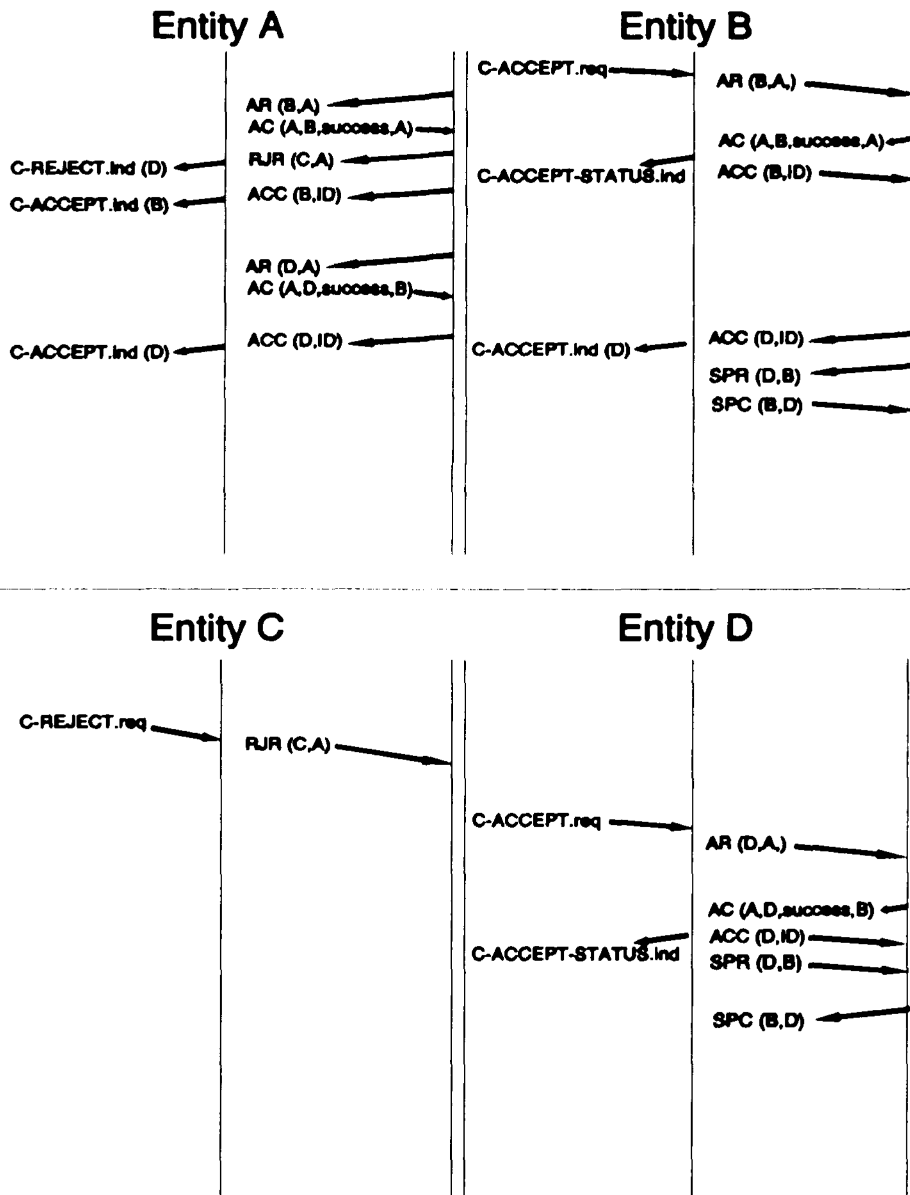


Figure 5.7: Timing diagram showing pending invited entities responding to the invitation. Entity A is the inviter. Entities B and D accept the invitation an entity C rejects the invitation. At the end, entities A, B and D are participants.

5.3.1.6 Set Successor

An entity informs its predecessor of its new successor by sending a SSR CPDU to its new successor. An entity would perform this function as a result of a recovering from a break in the logical ring. More details of this can be found in section 5.3.3.2. An entity that receives an SSR CPDU updates its successor pointer to point to the source of the data unit and sends a SSC CPDU to its new predecessor.

5.3.1.7 Expanding a Conference

At any time, a participating user can invite additional users to participate in an ongoing conference by issuing a C-INVITE.request. The same process that is described above for sending a conference invitation would be performed.

5.3.1.8 Suspending and Resuming a Conference Connection

A user's request to suspend a conference connection is handled locally by the protocol entity and involves no exchange of CPDUs with any other protocol entity. A user issues a C-SUSPEND.request service primitive. The protocol flags the connection as suspended and responds by issuing a C-SUSPEND-STATUS.indication. A suspended connection means that any data that the entity receives is not passed up to the local user. Section 5.3.2.2 explains in detail how a suspended entity that receives a DSR CPDU (the CPDU that contains data being sent to an entity's successor) from its predecessor passes it to its successor.

When a suspended user issues a C-RESUME.request, the protocol flags the connection active and responds with a C-RESUME-STATUS.indication service primitive.

5.3.1.9 Conference State

A user requests the state of the conference by issuing a C-STATE.request service primitive. The entity then sends, to its successor, a STR CPDU with the ORIG parameter field containing itself. An entity that receives a STR CPDU, and is not the originator of the STR

CPDU, adds a LIST parameter with the information field containing itself and the state of the entity. The state indicates if the entity is *active* or *suspended*. It then sends the newly formed STR CPDU to its successor. Eventually, the STR CPDU makes its way around the logical ring back to the originating entity of the STR CPDU. The originating entity compiles a status list of the remote protocol entities from the LIST parameters in the STR CPDU and passes it to the local user by issuing a C-STATE-STATUS.indication.

5.3.1.10 Removing a Participant from the Conference

At a user's request, by issuing a C-REMOVE.request service primitive, an entity can forcibly remove a remote entity from the conference by sending a RMR CPDU to the remote entity indicated in the service primitive. This is a request sent to the remote entity to remove itself from the conference. The entity then waits for a RMC CPDU from the remote entity. Upon the receipt of a RMC CPDU from the remote entity, the entity issues a C-REMOVE-STATUS.indication informing the local user that the remote user has been removed from the conference.

An entity that receives a RMR CPDU from a remote participating entity,

- sends a RMC CPDU to the entity that sent the RMR CPDU, confirming the request to remove itself from the from the conference;
- issues a C-REMOVE.indication service primitive, informing the local user that it has been removed from the conference;
- leaves the conference, as described in the next section.

5.3.1.11 Leaving the Conference

An entity leaves the conference either at the request of the user that issues a C-LEAVE.request or due to the receipt of a RMR CPDU, as described in the previous section. An entity leaves the conference by sending a LR CPDU to its predecessor with the SET_SUCC

parameter information field containing the *leaving* entity's successor. It then waits for a LC CPDU from its predecessor.

An entity, that receives a LR CPDU from its successor, determines whether there are only two participating entities left in the conference, which occurs when its successor is equal to its predecessor. This indicates that the active conference has ended. If the conference has ended, the entity sends a LC CPDU to the *leaving* entity, to confirm the LR CPDU. The protocol entity then informs the user the conference has ended by issuing a C-REMOVE.indication service primitive, if there are no *pending* invited entities. If there are *pending* invited entities, it will then inform the local user that a remote user left the conference by issuing a C-LEAVE.indication and wait for a response from the *pending* invited entities.

If there are more than two participating entities, indicating the conference is still active, the entity

- updates its successor to contain the entity provided in the SET_SUCC parameter information field of the LR CPDU;
- multicasts a LC CPDU, with the LEAVING parameter information field containing the *leaving* entity, to all participating entities to inform them that an entity is leaving the conference and to confirm the LR CPDU sent by the *leaving* entity;
- informs the local user that the remote user left the conference by issuing a C-LEAVE.indication service primitive; and
- performs the set predecessor linkage procedure described, previously, in section 5.3.1.5.

A participating entity that receives a LC CPDU informs the local user that a remote user has left the conference by issuing a C-LEAVE.indication.

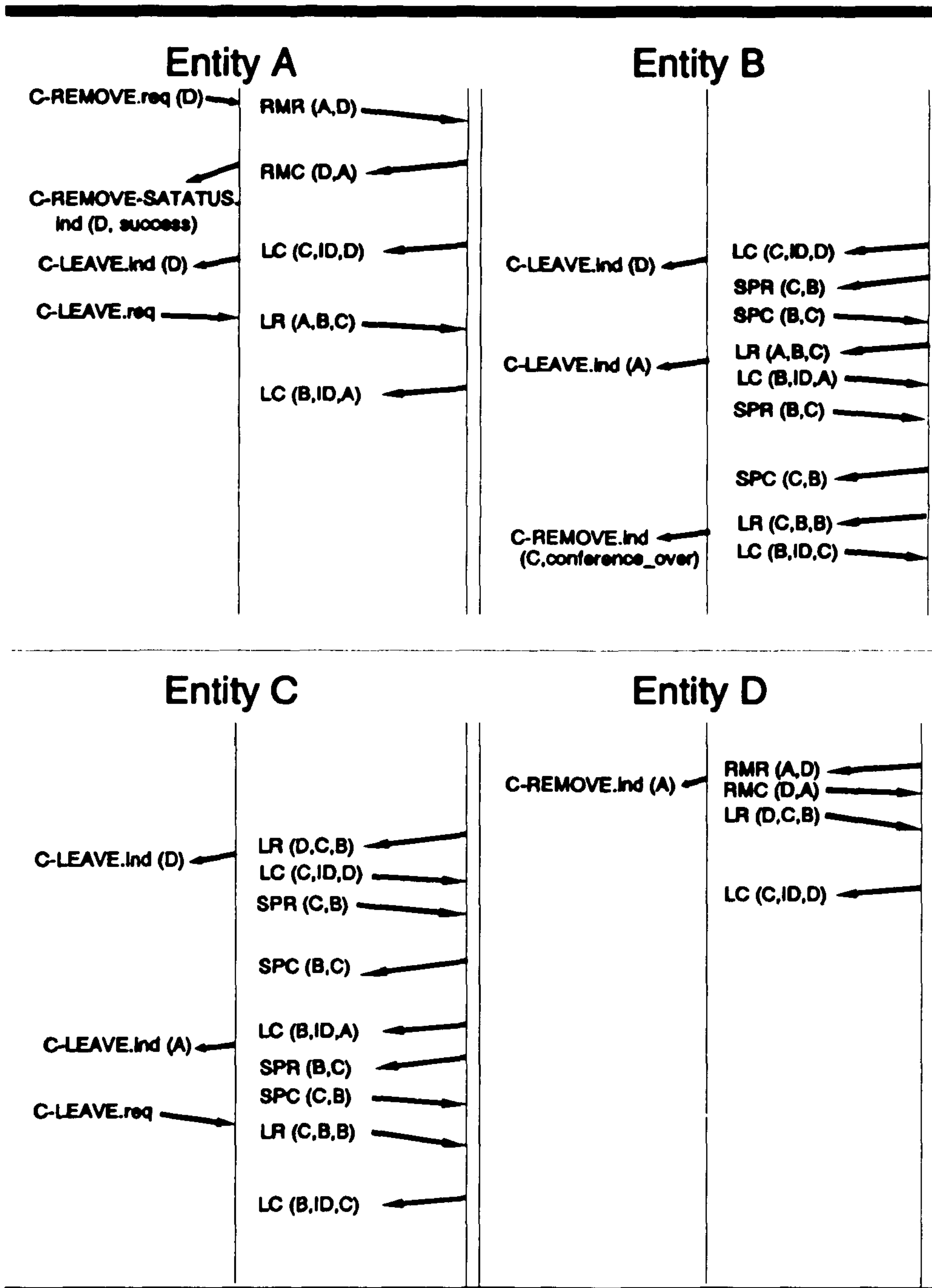


Figure 5.8: Timing diagram that shows the ending of a conference.

Figure 5.8 is a timing diagram displaying the ending of a conference. The figure assumes the logical ring to look like the following: A → C → D → B → A. Entity A forcibly removes entity D from the conference. This results in this logical ring: A → C → B → A. Then entity A leaves the conference resulting in the logical ring like this: C → B → C. Entity C leaves the conference and the conference has ended.

It is possible for two or more consecutive entities of the logical ring to send a LR CPDU at the same time. A *leaving* entity that receives a LR CPDU from its successor does not respond by sending a LC CPDU. If it did, the information sent in the SET_SUCC parameter of the LR CPDU that the *leaving* entity sent to its predecessor would be invalid. Instead, it waits for the *leaving* entity's predecessor to inform the *leaving* entity's successor of its new predecessor. Then, the *leaving* entity's successor will send a new LR CPDU to its new predecessor.

However, it is also possible for all participating entities to send a LR CPDU at the same time. This can result in all entities waiting for an LC CPDU and not receiving it. To allow an entity to detect this situation, the PASS parameter is added to the LR CPDU and is used in the following way:

- A *leaving* entity that receives a LR CPDU from its predecessor "passes" the LR CPDU to its predecessor by sending an LR CPDU with a PASS parameter and an ORIG parameter containing the originating entity of the LR CPDU.
- An entity, not leaving the conference, that receives a LR CPDU with a PASS parameter takes no action on this CPDU. (It will issue a SPR CPDU to its new successor.)
- A *leaving* entity that receives a LR CPDU with a PASS parameter

and is not the originator of the LR CPDU passes the CPDU to its predecessor. This is done by forming and sending, to its predecessor, a LR CPDU with a PASS parameter and a ORIG parameter containing the remote entity that was in the ORIG parameter information field of the LR CPDU it received.

- A *leaving* entity that receives a LR CPDU, with a PASS parameter and ORIG parameter containing itself, is an indication that all entities are leaving the conference at the same time. The *leaving* entity considers the conference ended and takes no action.

Figure 5.9 is a timing diagram that shows what occurs when all entities leave at the same time. The figure assumes the logical ring to be the following: A → C → D → B → A. The following occurs at each entity:

- it sends a LR CPDU to its predecessor;
- it receives an LR CPDU from its successor;
- it forms a LR CPDU with a PASS and ORIG parameter and send it to its predecessor;
- it receives, from its successor, two LR CPDUs with a PASS parameter and the ORIG parameter information field containing remote entities;
- it "passes" the two received LR CPDUs to its predecessor; and
- it receives an LR CPDU with a PASS parameter and an ORIG parameter information field containing itself, indicating the end of the conference.

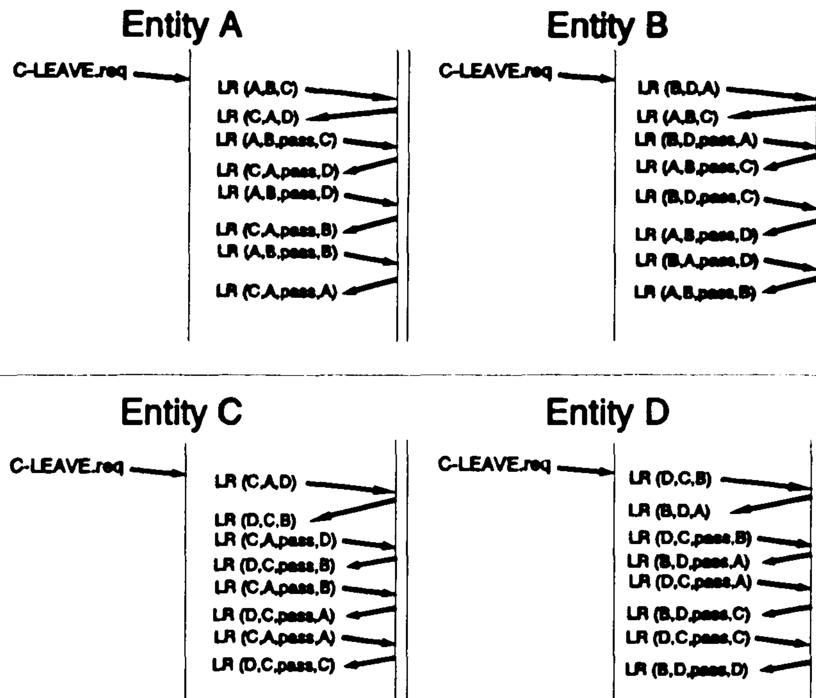


Figure 5.9: A timing diagram that shows all entities leaving the conference at the same time.

5.3.1.12 Conference Termination

An entity detects the conference has been terminated in the following ways:

- An entity that receives a LR CPDU when there are only two participants in the conference is an indication that the active conference is being terminated. As stated previously, when the entity's successor is equal to its predecessor the entity knows there are only two entities left in the conference.
- An entity leaving the conference receives an LR CPDU, with a PASS parameter and is the originator of the "passed" LR CPDU, is an

indication the conference has terminated.

5.3.2 Data Delivery

The protocol provides the mechanism to deliver data in three ways:

- Multicast data to all participating entities.
- Send data to the entity's successor in the logical ring.
- Unicast data to a remote participating entity.

The multicast data can be sent using the multicast capabilities of the underlying layer, if they exist. Otherwise, the logical ring can be used to multicast the data to all entities. For multicast data, only unacknowledged data delivery is provided. On the hand, successor data and the unicast data can either be acknowledged or unacknowledged, depending on the options requested by the user at the time of conference initiation. The sections that follow describe the mechanism used to implement the three types of data delivery.

5.3.2.1 Multicast Data

To send data to all conference participants, a user issues a C-CONF-DATA.request service primitive. The entity then encapsulates the data into a DCR CPDU and multicasts the CPDU to all participating entities. An entity that receives a DCR CPDU passes the data to the user by issuing a C-CONF-DATA.indication service primitive. (At present, it is assumed that the underlying layer provides multicast data delivery services.)

5.3.2.2 Successor Bound Data

A user issues a C-SUCC-DATA.request service primitive to send unacknowledged data. The entity encapsulates the data into a DSR CPDU and sends the CPDU to its successor. An active entity that receives a DSR CPDU from its predecessor responds by passing the data to the local user by issuing a C-SUCC-DATA.indication service primitive. A suspended entity that receives a DSR CPDU, passes the data to its successor by forming and sending a DSR CPDU,

containing the data it received, to its successor.

A user can issue a C-SUCC-DATA-ACK.request service primitive to send acknowledged data. The protocol encapsulates the data into a DSR-ACK CPDU and sends the CPDU to its successor. The entity then waits for a DSC CPDU from its successor. An active entity that receives a DSR-ACK CPDU from its predecessor responds by sending a DSC CPDU to its predecessor and passes the data to the local user by issuing a C-SUCC-DATA-ACK.indication service primitive. A suspended entity that receives a DSR-ACK CPDU, passes the data to its successor by forming and sending a DSR-ACK CPDU, containing the data it received, to its successor and then waits for a DSC CPDU. At the receipt of the DSC CPDU, the suspended entity sends a DSC CPDU to its predecessor. The protocol uses a *stop-and-wait* data delivery mechanism. A new DSR-ACK CPDU is not transmitted until the old DSR-ACK CPDU is confirmed with a DSC CPDU.

The DSR-ACK and DSC CPDUs contain a sequence number which eliminates duplicate data packets. Each protocol entity maintains two counters. One contains the send sequence number which indicates the next packet to be transmitted to its successor, XSEQ#, and the other contains the receive sequence number which indicates the next packet to be received from its predecessor, RSEQ#. An entity that transmits a DSR-ACK CPDU uses the XSEQ# for the SEQ# field of the data unit. XSEQ# is incremented at the receipt of a DSC CPDU with a sequence number equal to XSEQ#. XSEQ# is reset every time the successor pointer is reset.

An entity that receives a DSR-ACK CPDU from its predecessor sends a DSC CPDU with a SEQ# parameter information field containing the SEQ# of the DSR-ACK it received from its predecessor. The entity compares the SEQ# field with the RSEQ#. If SEQ# is less then RSEQ#, the DSR-ACK CPDU is a duplicate and the data is not passed up to the user.

RSEQ# is incremented at the receipt of a DSR-ACK CPDU with a sequence number equal to RSEQ#. RSEQ# is reset every time the predecessor pointer is reset.

5.3.2.3 Unicast Data

The protocol can unicast data to any remote entity participating in the conference. Since there is no point-to-point connection between the two entities, unicast data delivery is similar to a datagram service. A user requests to send unicast unacknowledged data to a remote user by issuing a C-DATA.request service primitive. The entity encapsulates the data into a DR CPDU and sends the CPDU to the remote entity provided by the local user. An entity that receives a DR CPDU from a remote participating entity responds by passing the data to the local user by issuing a C-DATA.indication service primitive.

A user requests to send unicast acknowledged data to a remote user by issuing a C-DATA-ACK.request service primitive. The entity encapsulates the data into a DR-ACK CPDU and sends the CPDU to the remote entity provided by the local user. The entity then waits for a DC CPDU from the remote entity. Upon the receipt of the DC CPDU, the entity issues a C-DATA-ACK-STATUS.indication indicating the successful delivery of the data. An entity that receives a DR-ACK CPDU from a remote participating entity responds by sending a DC CPDU to the source entity and passes the data to the local user by issuing a C-DATA.indication service primitive.

5.3.3 Error Detection and Recovery Mechanisms

There are three categories of errors that are detected by the protocol. They are:

- An entity that sent a request CPDU (such as, AR, IR, DSR-ACK, etc.), did not receive a matching confirm CPDU (such as, AC, IC, DSC, etc.) that is expected.
- An entity's successor is not responding to a request, indicating the

inability to communicate with its successor and, thus, a break in the logical ring.

- An entity's predecessor is not responding a request, indicating the inability to communicate with its predecessor and, thus, a break in the logical ring.

When an error is detected, the protocol attempts to recover from the error. The following sections present the mechanisms that are used to detect and recover from these errors.

5.3.3.1 Unconfirmed CPDU Error Detection and Recovery

When an entity sends a CPDU that requires a confirming CPDU, a timer is started. If a timeout occurs, that is, a confirm CPDU was not received within a set time limit, an error has occurred. The protocol first attempts to recover by retransmitting the CPDU. Of course, another timeout can occur and the protocol can retransmit the CPDU again. A limit is put on the number of retransmissions that are allowed. When the number of retransmissions exceed the limit, the protocol takes the appropriate actions described in table 5.3.

Table 5.3 lists pairs of CPDUs. Each pair consists of a requesting CPDU and its confirming CPDU for which the above described timeout and retransmission mechanisms are performed. For each pair, the actions that are taken when the retransmission limit is exceeded are listed.

Table 5.3: Actions Performed When Exceeding Retransmission Limit

Requesting CPDU	Confirming CPDU	Exceeded Retransmission Limit Actions
AR	AC	Issue a C-ACCEPT-STATUS.indication with the STATUS parameter indicating failure.

Table 5.3: Actions Performed When Exceeding Retransmission Limit		
Requesting CPDU	Confirming CPDU	Exceeded Retransmission Limit Actions
AC	ACC	Set the successor pointer to the entity's previous successor, if one existed. If one does not exist, which occurs with an inactive conference that is in the initiation stage, the conference remains inactive and the entity waits for an AR CPDU from an invited entity.
DR-ACK	DC	Issue a C-DATA-STATUS.indication with the STATUS parameter indicating failure.
DSR-ACK	DSC	Indicates a break in the logical ring. Perform successor recovery described in section 5.3.3.2.
IR	IC	Issue a C-INVITE-STATUS.indication with the STATUS LIST parameter indicating failure.
LR	LC	Indicates a break in the logical ring. Perform predecessor recovery described in section 5.3.3.3.
PRR	PRC	Either indicates where the break in the logical ring occurred, or a fatal error has occurred. Take the actions as described in section 5.3.3.3.
RMR	RMC	Issue a C-REMOVE-STATUS.indication with the STATUS parameter indicating failure.
SPR	SPC	Indicates a break in the logical ring. Perform successor recovery described in section 5.3.3.2.
SRR	SRC	Either indicates where the break in the logical ring occurred or a fatal error has occurred. Take the actions as described in section 5.3.3.2.
SSR	SSC	Indicates a break in the logical ring. Perform predecessor recovery described in section 5.3.3.3.

5.3.3.2 Successor Recovery

As indicated in table 5.3, an entity detects a loss of communications with its successor when its numerous retransmissions of a DSR-ACK or SPR CPDU are not confirmed with a DSC or SPC CPDU, respectively. This indicates a break in the logical ring. The entity attempts

to reestablish communications with its successor in the following way:

- The recovering entity sends a SRR CPDU, with the ORIG parameter containing itself and the NR_SUCC parameter containing the entity's successor in the logical ring, to its predecessor. The recovering entity then waits for a SRC CPDU.
- Three possible events can occur at the recovering entity:
  - If it receives a SRC CPDU from its predecessor, it enters a recovery wait state.
  - If it times out and has not exceeded its retransmission limit, then the entity retransmits the SRR CPDU, as above.
  - If it times out and has exceeded its retransmission limit, a fatal error has occurred and the entity considers the conference terminated in error.
- Three possible events can occur at the entity in a recovery wait state:
  - If it receives a SSR CPDU from an entity, the entity performs the set successor mechanisms described in section 5.3.1.6. The ring is whole and the recovery process has terminated in success. The entity then retransmits the DSR-ACK CPDU, if it was that CPDU that detected the successor error.
  - If it times out and has not exceeded its restart recovery limit, then the entity restarts the recovery process by retransmitting the SRR CPDU.

- If it times out and has exceeded its restart recovery limit, a fatal error has occurred and the entity considers the conference terminated in error.

An entity that receives a SRR CPDU from its successor performs the following:

- It sends a SRC CPDU to its successor.
- The entity checks the NR_SUCC of to see if it is the entity that is not responding to the originator of the recovery process.
    - If it is the entity, it performs the set successor mechanisms, described in section 5.3.1.6, by sending a SSR CPDU to its predecessor, the originator of the recovery process. The ring is whole and the recovery process has terminated in success.
    - If it is not the entity, it passes the successor recovery request to its predecessor by sending a SRR CPDU to its predecessor. The SRR CPDU's ORIG and NR_SUCC parameters contain the information the entity received in the ORIG and NR_SUCC parameters of the SRR CPDU it received from its successor. The entity then waits for a SRC CPDU.
- Three possible events can occur to the entity passing the successor recovery request to its predecessor.
    - If it receives a SRC CPDU from its predecessor, the entity terminates its participation in the recovery

process.

- If it times out and has not exceeded its retransmission limit, then the entity retransmits the SRR CPDU, as above.
- If it times out and has exceeded its retransmission limit, the entity assumes that the break in the logical ring is with its predecessor and attempts to close the ring by making the originator of the recovery process its predecessor. It does this by setting its predecessor pointer to the originator of the recovery process and performs the set successor mechanisms described in section 5.3.1.6. The ring is whole and the recovery process has terminated in success.

Figure 5.10 is an example of a successor recovery process. The figure assumes the logical ring to be: A → C → E → D → B → A. Entity E is not shown in the figure because it does not participate in the recovery process. Entity C detects that it lost contact with its successor, entity E. The figure ia a timing diagram that shows how the entities interact to correct the break in the logical ring. (This figure is a bit different than the ones used in this section so far. Since the successor recovery procedure does not involve the issuance of a service primitive, the figure only shows the interaction between the entities.)

The figure shows the recovery process is started by entity C sending a SRR CPDU to entity A, its predecessor. Entity A receives the CPDU, confirms it with a SRC CPDU, and then passes the request by forming and sending a SRR CPDU to its predecessor, entity B. Entity B receives it, confirms and passes it to entity D. Upon the receipt of the SRR CPDU from

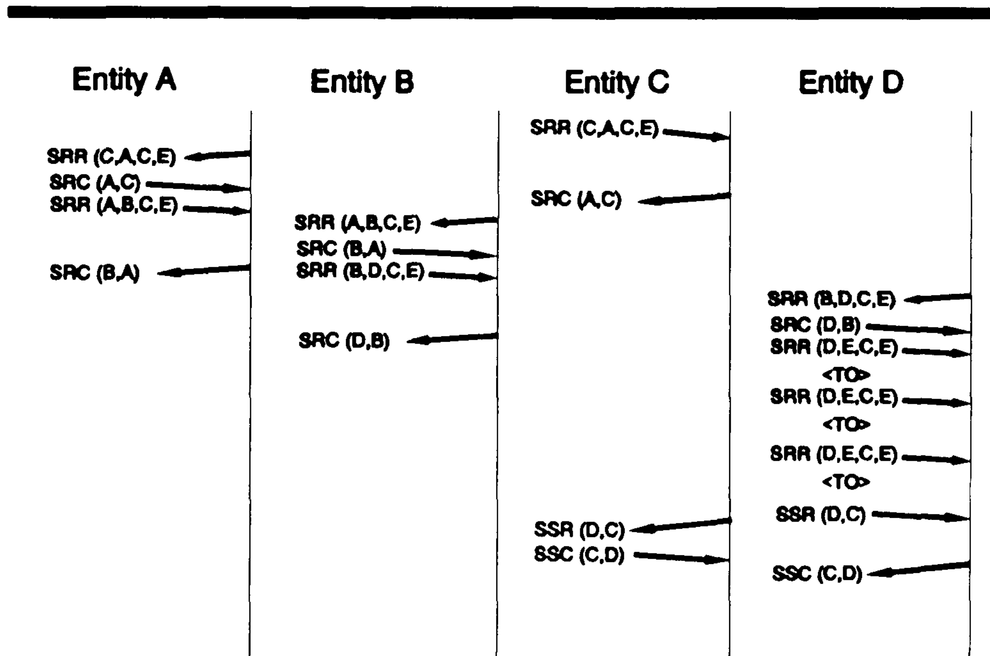


Figure 5.10: Timing diagram showing the successor recovery process.

entity B, entity D confirms the CPDU and passes the request to its predecessor, entity E. However, entity E does not respond with a SRC CPDU. Entity D times out and retransmits the SRR CPDU, twice. When it times out a third time, entity D assumes the break in the logical ring occurred with its predecessor and sends a SSR to entity C. Entity C responds with a SSC CPDU and the successor recovery process is successfully terminated.

5.3.3.3 Predecessor Recovery

As indicated in table 5.3, an entity detects a loss of communications with its predecessor when its numerous retransmissions of a LR or SSR CPDU are not confirmed with a LC or SSC CPDU, respectively. This indicates a break in the logical ring. The entity attempts to reestablish communications with its predecessor in the following way:

- The recovering entity sends a PRR CPDU, with the ORIG parameter

containing itself and the NR_PRED parameter containing the entity's predecessor in the logical ring, to its successor. The recovering entity then waits for a PRC CPDU.

- Three possible events can occur at the recovering entity:
  - If it receives a PRC CPDU from its successor, the entity enters a recovery wait state.
  - If it times out and has not exceeded its retransmission limit, then the entity retransmits the PRR CPDU, as above.
  - If it times out and has exceeded its retransmission limit, a fatal error has occurred and the entity considers the conference terminated in error.
- Three possible events can occur at the entity in a recovery wait state:
  - If it receives a SPR CPDU from an entity, the entity performs the set predecessor mechanisms described in section 5.3.1.5. The ring is whole and the recovery process has terminated in success. The entity then retransmits the LR or SSR CPDU, whichever was the CPDU that detected the predecessor error.
  - If it times out and has not exceeded its restart recovery limit, then the entity restarts the recovery process by retransmitting the PRR CPDU.
  - If it times out and has exceeded its restart recovery

limit, then a fatal error has occurred and the entity considers the conference terminated in error.

An entity that receives a PRR CPDU from its predecessor performs the following:

- It sends a PRC CPDU to its predecessor.
- The entity checks the NR_PRED of to see if it is the entity that is not responding to the originator of the recovery process.
  - If it is the entity, it performs the set predecessor mechanisms, described in section 5.3.1.5, by sending a SPR CPDU to its successor, the originator of the recovery process. The ring is whole and the recovery process has terminated in success.
  - If it is not the entity, it passes the predecessor recovery request to its successor by sending a PRR CPDU to its successor. The PRR CPDU's ORIG and NR_PRED parameters contain the information the entity received in the ORIG and NR_PRED parameters of the PRR CPDU it received from its predecessor. The entity then waits for a PRC CPDU.
- Three possible events can occur to the entity passing the predecessor recovery request to its successor.
  - If it receives a PRC CPDU from its successor, the entity terminates its participation in the recovery process.

- If it times out and has not exceeded its retransmission limit, then the entity retransmits the PRR CPDU, as above.
- If it times out and has exceeded its retransmission limit then the entity assumes that the break in the logical ring is with its successor and attempts to close the ring by making the originator of the recovery process its successor. It does this by setting its successor pointer to the originator of the recovery process and performs the set predecessor mechanisms described in section 5.3.1.5. The ring is whole and the recovery process has terminated in success.

Figure 5.11 is an example of a predecessor recovery process. The figure assumes the logical ring to be: A → C → E → D → B → A. As before, entity E is not shown in the figure because it does not participate in the recovery process. Entity D detects that it lost contact with its predecessor, entity E. The figure is a timing diagram that shows how the entities interact to correct the break in the logical ring.

The figure shows the recovery process is started by entity D sending a PRR CPDU to entity B, its successor. Entity B receives the CPDU, confirms it with a PRC CPDU, and then passes the request by forming and sending a PRR CPDU to its successor, entity A. Entity A receives it, confirms and passes it to entity C. Upon the receipt of the PRR CPDU from entity A, entity C confirms the CPDU and passes the request to its successor, entity E. However, entity E does not respond with a PRC CPDU. Entity C times out and retransmits the PRR

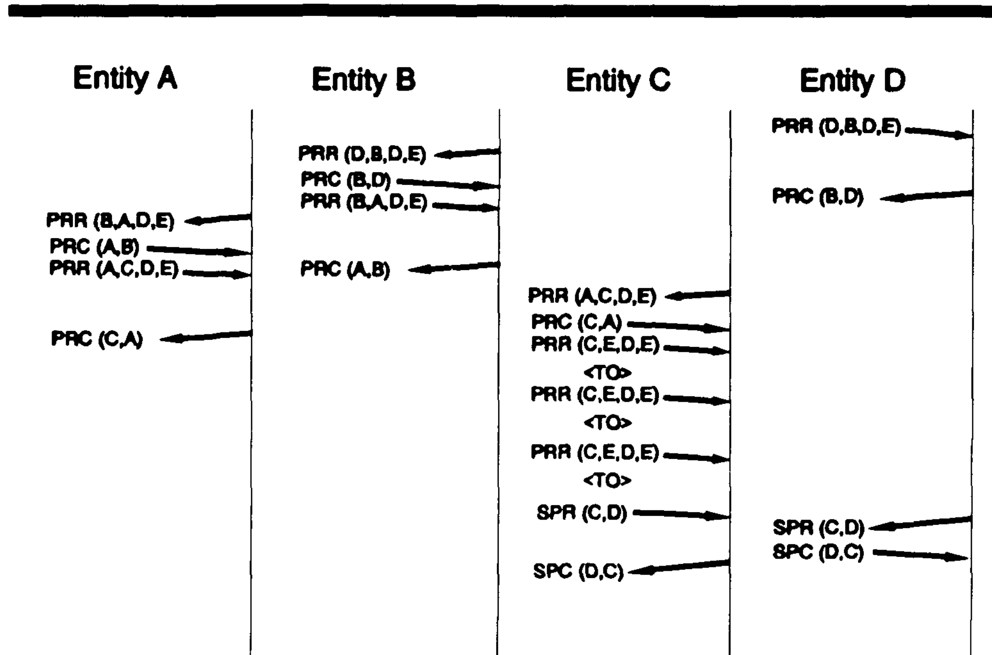


Figure 5.11: Timing diagram that shows the predecessor recovery process.

CPDU, twice. When it times out a third time, entity C assumes the break in the logical ring occurred with its successor and sends a SPR to entity D. Entity D responds with a SPC CPDU and the predecessor recovery process is successfully terminated.

5.4 Conferencing Protocol Formal Specification

This section presents the formal specification of the conferencing protocol operation. The standard method for formal protocol specification is to model the protocol as a finite state machine [TANN88, HALS88]. A finite state machine is in a defined state at every instant of time. A state consists of all the values of variables. Defined events drive the machine. For each state, there are zero or more transition rules that depend on the events that occur. The transition rules consist of actions the machine should take and the state to which the machine should switch. When an event occurs in a particular state, the machine takes the actions and

switches to the state defined in the transition rule for that event and state.

However, the normal finite state machine turns out to be too restrictive for practical use. Since the states of the *protocol machine* are dependant on all the values of all its variables, any realistic protocol specification involves an immense number of states. The extended finite state machine was developed for protocol specification. It allows states to maintain their own variables. The machine can then react based on the values retained in these variables, in addition to the event and the state of the machine [TANN88].

The finite state machine can be represented by either a finite state diagram, or a programming type language, or a transition table that defines the events, conditions, actions and state transitions of the finite state machine. The next section presents the finite state machine, using a state transition table, that models the conferencing protocol operation. The subsequent sections define the variables, events, conditions and procedures used by the finite state machine.

5.4.1 Finite State Machine

The conferencing protocol will be formally specified using a state transition table to represent the finite state machine that models the protocol. A separate table will be presented for each state of the machine. The events that drive the machine are in the first column of the table and are defined in section 5.4.3. The column also contains the conditions, defined in section 5.4.4, that must be satisfied for the actions of the second column to be taken. Each state contains only those events and conditions that result in an action and/or state transition. All other events and conditions are erroneous for that state and no action or state transition takes place. The actions of the second column are assignments to variables and calls to procedures. The procedures are defined section 5.4.5. The third column contains the transition states.

The states of the machine are numbered, with the numbers running between 0 and 8. State 7 indicates the protocol entity is a conference participant. The are substates of state 7 that

are represented by 7.xx, where *xx* are numbers running from 00 to 41. States 7.xx indicate the protocol entity is a conference participant, however, the protocol entity is also involved in some other process. As an example, states 7.01 through 7.03 are involved with maintaining the linkage information for the logical ring of conference participants. Those events and conditions that have the same actions and state transitions for all the states numbered 7.xx are put into a separate table numbered state 7.

The following is the finite state machine that models the conferencing protocol:

State 0: Idle		
Event & Conditions	Action	Next State
C-INVITE.request(INVITED_LIST, OPTIONS) & not_busy	add_to_nonconfirmed set_CONF_ID ID = CONF_ID send_IR (INVITED_LIST, ID, OPTIONS) start_IR_timer	1.Invitation Sent
C-INVITE.request & busy	CAUSE = BUSY C-REMOVE.indication (CAUSE)	0.Idle
rec_IR (SOURCE,ID,OPTIONS) & not_busy	INVITER = SOURCE send_IC (INVITER) C-INVITE.indication (INVITER, OPTIONS) CONF_ID = ID	2.Invited
rec_IR (SOURCE) & busy	CAUSE = BUSY send_RJR (SOURCE, CAUSE)	0.Idle

State 1: Invitation Sent		
Event & Conditions	Action	Next State
rec_IC (SOURCE) & in_nonconfirmed	remove_from_nonconfirmed add_to_pending STATUS = SUCCESS C-INVITE-STATUS.indication (SOURCE, STATUS)	1.Invitation Sent

State 1: Invitation Sent		
Event & Conditions	**Action**	**Next State**
rec_AR (SOURCE) & in_pending	SET_SUCC = SELF STATUS = SUCCESS send_AC (SOURCE, STATUS, SET_SUCC) start_AC_timer remove_from_pending SUCC = SOURCE PRED = SOURCE	7.03.Acceptance Setup
rec_AR (SOURCE) & in_nonconfirmed*	SET_SUCC = SELF STATUS = SUCCESS send_AC (SOURCE, STATUS, SET_SUCC) start_AC_timer remove_from_nonconfirmed SUCC = SOURCE PRED = SOURCE	7.03.Acceptance Setup
rec_RJR (SOURCE, CAUSE) & invited_=_1	C-REMOVE.indication (CAUSE)	0.Idle
rec_RJR (SOURCE, CAUSE) & in_nonconfirmed & invited_>_1	remove_from_nonconfirmed C-REJECT.indication (SOURCE, CAUSE)	1.Invitation Sent
rec_RJR (SOURCE,CAUSE) & in_pending & invited_>_1	remove_from_pending C-REJECT.indication (SOURCE, CAUSE)	1.Invitation Sent
C-LEAVE.request	send_RVR (PENDING)	0.Idle
IR_to & under_IR_limit	send_IR (NONCONFIRMED, ID, OPTIONS) start_IR_timer increment_IR	1.Invitation Sent
IR_to & over_IR_limit & pending_>_0	C-INVITE-STATUS.indication (NONCONFIRMED, FAILED) clear_nonconfirmed	1.Invitation Sent
IR_to & over_IR_limit & pending_=_0	C-INVITE.indication (NONCONFIRMED, FAILED)	0.Idle

State 2: Invited		
Event & Conditions	**Action**	**Next State**
C-ACCEPT.request	send_AR (INVITER) start_AR_timer	4.Acceptance Sent
C-REJECT.request (CAUSE)	send_RJR (INVITER, CAUSE)	0.Idle

State 2: Invited		
Event & Conditions	Action	Next State
rec_RVR (SOURCE) & (SOURCE = INVITER)	C-REVOKE.indication	0.Idle

State 4: Acceptance Sent		
Event & Conditions	Action	Next State
rec_AC (SOURCE,STATUS,SET_SUCC) & SOURCE = INVITER & STATUS = SUCCESS	stop_AR_timer PRED = SOURCE SUCC = SET_SUCC C-ACCEPT-STATUS.indication (CONNECTED) RET_STATE = 7.00 send_ACC (CONF_ID) send_SPR (SUCC) start_SPR_timer	7.01.Participant/ Set Predecessor
rec_AC (SOURCE, STATUS) & SOURCE = INVITER & STATUS = WAIT	stop_AR_timer start_wait_timer	4.Acceptance Sent
rec_RVR (SOURCE)* & SOURCE = INVITER	C-REVOKE.indication	0.Idle
wait_TO	send_AR (INVITER) start_AR_timer	4.Acceptance Sent
AR_TO & under_AR_limit	send_AR (INVITER) start_AR_timer increment_AR	4.Acceptance Sent
AR_TO & over_AR_limit	C-ACCEPT-STATUS.indication (FAILED)	2.Invited
Notes: * cross of RVR and AR		

State 7: Participant		
Event & Conditions	Action	Next State
C-INVITE.request (INVITED_LIST, OPTIONS)	add_to_nonconfirmed ID = CONF_ID send_IR (INVITED_LIST, ID, OPTIONS) start_IR_timer	7.Participant
rec_IC (SOURCE) & in_nonconfirmed	remove_from_nonconfirmed add_to_pending C-INVITE-STATUS.indication (SOURCE, SUCCESS)	7.Participant

State 7: Participant

Event & Conditions	Action	Next State
rec_RJR (SOURCE, CAUSE) & in_nonconfirmed	remove_from_nonconfirmed C-REJECT.indication (SOURCE, CAUSE)	7.Participant
rec_RJR (SOURCE, CAUSE) & in_pending	remove_from_pending C-REJECT.indication (SOURCE, CAUSE)	7.Participant
IR_to & under_IR_limit & nonconfirmed_>_0	send_IR (NONCONFIRMED, ID, OPTIONS) start_IR_timer increment_IR	7.Participant
IR_to & over_IR_limit	C-INVITE-STATUS.indication (NONCONFIRMED, FAILED) clear_nonconfirmed	7.Participant
rec_AC (SOURCE) & SOURCE = PRED	send_ACC (CONF_ID)	7.Participant
rec_ACC (SOURCE, ID) & ID = CONF_ID & SOURCE ≠ SUCC	C-ACCEPT.indication (SOURCE)	7.Participant
rec_LC (SOURCE, ID, LEAVING) & ID = CONF_ID	C-LEAVE.indication (LEAVING)	7.Participant
C-CONF-DATA.request (DATA)	SOURCE = SELF ID = CONF_ID send_DCR (ID, SOURCE, DATA)	7.Participant
C-DATA.request (DEST, DATA)	SOURCE = SELF ID = CONF_ID send_DR (DEST, SOURCE, ID, DATA)	7.Participant
C-SUCC-DATA.request (DATA)	send_DSR (SUCC, DATA)	7.Participant
rec_DSR-ACK (SOURCE, DATA, SEQ#) & SOURCE = PRED & SEQ# = RSEQ# & not_suspended	send_DSC (PRED, SEQ#) incr_RSEQ# C-SUCC-DATA-ACK.indication (DATA)	7.Participant
rec_DSR-ACK (SOURCE, DATA) & SOURCE = PRED & SEQ# < RSEQ#	send_DSC (PRED, SEQ#)	7.Participant
rec_DSR (SOURCE, DATA) & SOURCE = PRED & not_suspended	C-SUCC-DATA.indication (DATA)	7.Participant
rec_DSR (SOURCE, DATA) & SOURCE = PRED & suspended	send_DSR (SUCC,DATA)	7.Participant

State 7: Participant		
Event & Conditions	Action	Next State
rec_DR (SOURCE, ID, DATA) & ID = CONF_ID & not_suspended	C-DATA.indication (SOURCE, DATA)	7.Participant
C-DATA-ACK.request (DEST, DATA)	SOURCE = SELF ID = CONF_ID DATA_DEST = DEST send_DR-ACK (DEST, SOURCE, ID, DATA) start_DR_timer	7.Participant
rec_DC (SOURCE) & SOURCE = DATA_DEST	stop_DR_timer C-DATA-STATUS.indication (SUCCESS)	7.Participant
DR_to & under_DR_limit	SOURCE = SELF ID = CONF_ID DEST = DATA_DEST send_DR (DEST, SOURCE, ID, DATA) start_DR_timer incr_DR	7.Participant
DR_to & over_DR_limit	C-DATA-STATUS.indication (FAILED)	7.Participant
rec_DR-ACK (SOURCE, ID, DATA) & ID = CONF_ID & not_suspended	C-DATA-ACK.indication (SOURCE, DATA) send_DC (SOURCE)	7.Participant
rec_DCR (ID, SOURCE, DATA) & ID = CONF_ID & not_suspended	C-CONF-DATA.indication (SOURCE, DATA)	7.Participant
C-REVOKE.request	send_RVR (PENDING) clear_nonconfirmed clear_pending	7.Participant
C-REMOVE.request (RMT_ENTITY)	REMOVED_ENTITY = RMT_ENTITY send_RMR (REMOVED_ENTITY) start_RMR_timer	7.Participant
rec_RMC (SOURCE) & SOURCE = REMOVED_ENTITY	stop_RMR_timer C-REMOVE-STATUS.indication (SUCCESS)	7.Participant
RMR_to & under_RMR_limit	send_RMR (REMOVED_ENTITY) start_RMR_timer increment_RMR	7.Participant
RMR_to & over_RMR_limit	C-REMOVE-STATUS.indication (FAILED)	7.Participant
C-STATE.request	ORIG = SELF send_STR (CONF_ID, ORIG)	7.Participant

State 7: Participant		
Event & Conditions	Action	Next State
rec_STR (ID, ORIG, LIST) & ID = CONF_ID & ORIG ≠ SELF	add_list send_STR (ID, ORIG, LIST)	7.Participant
rec_STR (ID, ORIG, LIST) & ID = CONF_ID & ORIG = SELF	compile_status_list C-STATE-STATUS (STATUS_LIST)	7.Participant
C-SUSPEND.request	set_suspend C-SUSPEND-STATUS.indication	7.Participant
C-RECONNECT.request	reset_suspend C-RECONNECT-STATUS.indication	7.Participant

State 7.00: Participant		
Event & Conditions	Action	Next State
rec_AR (SOURCE) & in_pending	SET_SUCC = SUCC STATUS = SUCCESS send_AC (SOURCE, STATUS, SET_SUCC) start_AC_timer PREV_SUCC = SUCC SUCC = SOURCE reset_XSEQ# remove_from_pending	7.03.Acceptance Setup
rec_AR (SOURCE) & in_nonconfirmed	SET_SUCC = SUCC STATUS = SUCCESS send_AC (SOURCE, STATUS, SET_SUCC) start_AC_timer PREV_SUCC = SUCC SUCC = SOURCE reset_XSEQ# remove_from_nonconfirmed	7.03.Acceptance Setup
rec_SPR (SOURCE)	PRED = SOURCE reset_RSEQ# send_SPC (PRED)	7.Participant
rec_LR (SOURCE, SET_SUCC) & SOURCE = SUCC & 3_or_more_participants	LEAVING = SOURCE send_LC (ID, LEAVING) SUCC = SET_SUCC reset_XSEQ# send_SPR (SUCC) start_SPR_timer C-LEAVE.indication (SOURCE) RET_STATE = 7.00	7.01.Set Predecessor

State 7.00: Participant		
Event & Conditions	Action	Next State
rec_LR (SOURCE) & SOURCE = SUCC & less_than_3_participants & invited_ > _0	LEAVING = SOURCE send_LC (ID, LEAVING) C-LEAVE.indication (SUCC)	2.Invitation Sent
rec_LR (SOURCE) & SOURCE = SUCC & less_than_3_participants & invited_ = _0	LEAVING = SOURCE send_LC (ID, LEAVING) C-REMOVE.indication (CONFERENCE_ENDED)	0.Idle
C-LEAVE.request	SET_SUCC = SUCC send_LR (PRED, SET_SUCC) start_LR_timer	8.Leaving
rec_PRR (SOURCE,ORIG,NR_PRED) & SOURCE = PRED & NR_PRED ≠ SELF & ORIG ≠ SELF*	send_PRC (PRED) send_PRR (SUCC, ORIG, NR_PRED) start_PRR_timer RCVR_ORIG = ORIG	7.12 Pass Predecessor Recovery
rec_PRR (SOURCE,ORIG,NR_PRED) & SOURCE = PRED & NR_PRED ≠ SELF & ORIG = SELF*		7.00 Participant
rec_PRR (SOURCE,ORIG,NR_PRED) & SOURCE = PRED & NR_PRED = SELF	send_PRC (PRED) SUCC = ORIG reset_XSEQ# send_SPR (SUCC) start_SPR_timer RET_STATE = 7.00	7.01.Set Predecessor
rec_SRR (SOURCE,ORIG,NR_SUCC) & SOURCE = SUCC & NR_SUCC ≠ SELF	send_SRC (SUCC) send_SRR (PRED, ORIG, NR_SUCC) start_SRR_timer RCVR_ORIG = ORIG	7.22 Pass Successor Recovery
rec_SRR (SOURCE,ORIG,NR_SUCC) & SOURCE = SUCC & NR_SUCC = SELF	send_SRC (SUCC) PRED = ORIG reset_RSEQ# send_SSR (PRED) start_SSR_timer	7.02.Set Successor
C-SUCC-DATA-ACK.request (DATA)	send_DSR-ACK (SUCC, DATA, XSEQ#) start_DSR_timer	7.41.Data Sent
rec_DSR-ACK (SOURCE, DATA, SEQ#) & SOURCE = PRED & SEQ# = RSEQ# & suspended	send_DSR-ACK (SUCC, DATA, XSEQ#) start_DSR_timer	7.41.Data Sent

State 7.00: Participant		
Event & Conditions	Action	Next State
rec_RMR (SOURCE, CAUSE)	C-REMOVE.indication (SOURCE) send_RMC (SOURCE) SET_SUCC = SUCC send_LR (PRED, SET_SUCC) start_LR_timer	8.Leaving
Notes: * Ring integrity may have recovered before the recovery request reached the entity that was not responding.		

State 7.01: Set Predecessor		
Event & Conditions	Action	Next State
rec_SPC (SOURCE) & SOURCE = SUCC & RET_STATE = 7.00	stop_SPR_timer check_event_queue	7.00.Participant
rec_SPC (SOURCE) & SOURCE = SUCC & RET_STATE = 7.11	stop_SPR_timer send_PRR (SUCC, ORIG, NR_PRED) start_PRR_timer	7.11 Predecessor Recovery Start
rec_SPC (SOURCE) & SOURCE = SUCC & RET_STATE = 7.12	stop_SPR_timer send_PRR (SUCC, ORIG, NR_PRED) start_PRR_timer	7.12 Pass Predecessor Recovery
rec_SPC (SOURCE) & SOURCE = SUCC & RET_STATE = 7.41	stop_SPR_timer send_DSR-ACK (SUCC, DATA) start_DSR_timer	7.41.Data Sent
rec_SPC (SOURCE) & SOURCE = SUCC & RET_STATE = 8	stop_SPR_timer SET_SUCC = SUCC send_LR (PRED, SET_SUCC) start_LR_timer	8. Leaving
SPR_to & under_SPR_limit	send_SPR (SUCC) start_SPR_timer increment_SPR	7.01.Set Predecessor
SPR_to & over_SPR_limit & RET_STATE ≠ 7.11	NR_SUCC = SUCC ORIG = SELF send_SRR (PRED, ORIG, NR_SUCC) start_SRR_timer RCVR_STATE = RET_STATE	7.21.Successor Recovery Start
SPR_to & over_SPR_limit & RET_STATE = 7.11	fatal_error	0.Idle
rec_LR (PASS)*		7.01.Set Predecessor

State 7.01: Set Predecessor		
Event & Conditions	Action	Next State
rec_LR (SOURCE, SET_SUCC) & SOURCE = SUCC & 3_or_more_participants	stop_SPR_timer LEAVING = SOURCE send_LC (ID, LEAVING) SUCC = SET_SUCC reset_XSEQ# send_SPR (SUCC) set_SPR_timer C-LEAVE.indication (SOURCE) RET_STATE = 7.00	7.01.Set Predecessor
rec_LR (SOURCE) & SOURCE = SUCC & less_than_3_participants & invited_>_0	stop_SPR_timer LEAVING = SOURCE send_LC (ID, LEAVING) C-LEAVE.indication (SUCC)	2.Invitation Sent
rec_LR (SOURCE) & SOURCE = SUCC & less_than_3_participants & invited_=_0	LEAVING = SOURCE send_LC (ID, LEAVING) C-REMOVE.indication (CONFERENCE_ENDED)	0.Idle
C-LEAVE.request	queue_event	7.01.Set Predecessor
C-SUCC-DATA-ACK.request (DATA)	queue_event	7.01.Set Predecessor
rec_DSR-ACK (SOURCE, DATA, SEQ#) & SEQ# = RSEQ# & suspended	queue_event	7.01.Set Predecessor
rec_PRR (SOURCE,ORIG,NR_PRED)	queue_event	7.01 Set Predecessor
rec_SRR (SOURCE,ORIG,NR_SUCC)	queue_event	7.01 Set Predecessor
rec_RMR (SOURCE)	queue_event	7.01 Set Predecessor
rec_SPR (SOURCE) & RET_STATE ≠ 7.11	PRED = SOURCE reset_RSEQ# send_SPC (PRED)	7.01 Set Predecessor
rec_SPR (SOURCE) & RET_STATE = 7.11	PRED = SOURCE reset_RSEQ# send_SPC (PRED) RET_STATE = RCVR_STATE	7.01 Set Predecessor
rec_AR (SOURCE) &(in_nonconfirmed | in_pending)	STATUS = WAIT send_AC (SOURCE, STATUS)	7.01.Set Predecessor

Notes:
* received a passed along LR CPDU (look at state 8).

State 7.02: Set Successor		
Event & Conditions	Action	Next State
rec_SSC (SOURCE) & SOURCE = PRED	stop_SSR_timer check_event_queue	7.00.Participant
rec_DSR-ACK (SOURCE, DATA, SEQ#) & SEQ# = RSEQ# & suspended	queue_event	7.02.Set Successor
SSR_to & under_SSR_limit	send_SSR (PRED) start_SSR_timer increment_SSR	7.02.Set Successor
SSR_to & over_SSR_limit	NR_PRED = PRED ORIG = SELF PRED = NULL send_PRR (SUCC, ORIG, NR_PRED) start_PRR_timer RCVR_STATE = 7.00	7.11.Predecessor Recovery Start
rec_LR (SOURCE, SET_SUCC)	queue_event	7.02.Set Successor
C-LEAVE.request	queue_event	7.02.Set Successor
C-SUCC-DATA-ACK.request (DATA)	queue_event	7.02.Set Successor
rec_PRR (SOURCE,ORIG,NR_PRED)	queue_event	7.02.Set Successor
rec_SPR (SOURCE)	PRED = SOURCE reset_RSEQ# send_SPC (PRED)	7.00.Participant
rec_SRR (SOURCE,ORIG,NR_SUCC)	queue_event	7.02.Set Successor
rec_RMR (SOURCE)	queue_event	7.02.Set Successor
rec_AR (SOURCE) &(in_nonconfirmed | in_pending)	STATUS = WAIT send_AC (SOURCE, STATUS)	7.02.Set Successor

State 7.03: Acceptance Setup		
Event & Condition	Action	Next State
rec_ACC (SOURCE, ID) & ID = CONF_ID & SOURCE = SUCC	stop_AC_timer C-ACCEPT.indication (SOURCE) check_event_queue	7.00.Participant
AC_to & under_AC_limit	send_AC (SUCC, STATUS, SET_SUCC) increment_AC	7.03.Acceptance Setup

State 7.03: Acceptance Setup

Event & Condition	Action	Next State
AC_to & over_AC_limit & 3_or_more_participants	SUCC = PREV_SUCC reset_XSEQ# send_SPR (SUCC) start_SPR_timer RET_STATE = 7.00	7.01.Set Predecessor
AC_to & over_AC_limit & less_than_3_participants & invited_>_0	check_event_queue	1.Invitation Sent
AC_to & over_AC_limit & less_than_3_participants & invited_=_0		0.Idle
C-LEAVE.request	queue_event	7.03.Acceptance Setup
C-SUCC-DATA-ACK.request (DATA)	queue_event	7.03.Acceptance Setup
rec_DSR-ACK (SOURCE, DATA, SEQ#) & SEQ# = RSEQ# & suspended	queue_event	7.03.Acceptance Setup
rec_PRR (SOURCE,ORIG,NR_PRED)	queue_event	7.03.Acceptance Setup
rec_SPR (SOURCE)	PRED = SOURCE reset_RSEQ# send_SPC (PRED)	7.03.Acceptance Setup
rec_SRR (SOURCE,ORIG,NR_SUCC)	queue_event	7.03.Acceptance Setup
rec_RMR (SOURCE)	queue_event	7.03.Acceptance Setup
rec_AR (SOURCE) &(in_nonconfirmed | in_pending)	STATUS = WAIT send_AC (SOURCE, STATUS)	7.03.Acceptance Setup

Notes:
* received a passed along LR CPDU (look at state 8).

State 7.11: Predecessor Recovery Start

Event & Condition	Action	Next State
rec_PRC (SOURCE) & SOURCE = SUCC	stop_PRR_timer start_rcvr_timer	7.13.Predecessor Recovery Wait

State 7.11: Predecessor Recovery Start		
Event & Condition	**Action**	**Next State**
PRR_to & under_PRR_limit	send_PRR (SUCC, ORIG, NR_PRED) start_PRR_timer increment_PRR	7.11.Predecessor Recovery Start
PRR_to & over_PRR_limit	fatal_error	0.Idle
rec_SPR (SOURCE) & RCVR_STATE ≠ 8	stop_PRR_timer PRED = SOURCE reset_RSEQ# send_SPC (PRED) check_event_queue	7.00.Participant
rec_SPR (SOURCE) & RCVR_STATE = 8	stop_PRR_timer PRED = SOURCE send_SPC (PRED) SET_SUCC = SUCC send_LR (PRED, SET_SUCC) start_LR_timer	8.Leaving
rec_DSR-ACK (SOURCE, DATA, SEQ#) & SEQ# = RSEQ# & suspended	queue_event	7.11.Predecessor Recovery Start
rec_LR (SOURCE, SET_SUCC) & SOURCE = SUCC & 3_or_more_participants	stop_PRR_timer LEAVING = SOURCE send_LC (ID, LEAVING) SUCC = SET_SUCC reset_XSEQ# send_SPR (SUCC) start_SPR_timer C-LEAVE.indication (SOURCE) RET_STATE = 7.11	7.01.Set Predecessor
rec_LR (SOURCE) & SOURCE = SUCC & less_than_3_participants & RCVR_STATE ≠ 8 & invited_>_0	stop_PRR_timer LEAVING = SOURCE send_LC (ID, LEAVING) C-LEAVE.indication (SUCC)	1.Invitation sent
rec_LR (SOURCE) & SOURCE = SUCC & less_than_3_participants & RCVR_STATE ≠ 8 & invited_=_0	LEAVING = SOURCE send_LC (ID, LEAVING) C-REMOVE.indication (CONFERENCE_ENDED)	0.Idle
rec_LR (SOURCE) & SOURCE = SUCC & less_than_3_participants & RCVR_STATE = 8 & pending_>_0	LEAVING = SOURCE send_LC (ID, LEAVING) send_RVR (PENDING)	0.Idle

State 7.11: Predecessor Recovery Start		
Event & Condition	**Action**	**Next State**
rec_LR (SOURCE) & SOURCE = SUCC & less_than_3_participants & RCVR_STATE = 8 & pending_ = _0	LEAVING = SOURCE send_LC (ID, LEAVING)	0.Idle
C-LEAVE.request	queue_event	7.11.Predecessor Recovery Start
C-SUCC-DATA-ACK.request (DATA)	queue_event	7.11.Predecessor Recovery Start
rec_PRR (SOURCE,ORIG,NR_PRED)	queue_event	7.11 Predecessor Recovery Start
rec_SRR (SOURCE,ORIG,NR_SUCC)	queue_event	7.11 Predecessor Recovery Start
rec_RMR (SOURCE)	queue_event	7.11 Predecessor Recovery Start
rec_AR (SOURCE)* &(in_nonconfirmed | in_pending)	STATUS = WAIT send_AC (SOURCE, STATUS)	7.11.Predecessor Recovery Start
Notes: * Since communication in progress with SUCC, inserting a new entity in the ring is currently not possible.		

State 7.12: Pass Predecessor Recovery		
Event & Condition	**Action**	**Next State**
rec_PRC (SOURCE) & SOURCE = SUCC	stop_PRR_timer check_event_queue	7.00.Participant
PRR_to & under_PRR_limit	send_PRR (SUCC, ORIG, NR_PRED) start_PRR_timer increment_PRR	7.12.Pass Predecessor Recovery
PRR_to & over_PRR_limit	SUCC = ORIG reset_XSEQ# send_SPR (SUCC) start_SPR_timer RET_STATE = 7.00	7.01.Set Predecessor
C-LEAVE.request	queue_event	7.12.Pass Predecessor Recovery
C-SUCC-DATA-ACK.request (DATA)	queue_event	7.12.Pass Predecessor Recovery

State 7.12: Pass Predecessor Recovery		
Event & Condition	**Action**	**Next State**
rec_DSR-ACK (SOURCE, DATA, SEQ#) & SEQ# = RSEQ# & suspended	queue_event	7.12.Pass Predecessor Recovery
rec_LR (SOURCE, SET_SUCC) & SOURCE = SUCC & 3_or_more_participants	stop_PRR_timer LEAVING = SOURCE send_LC (ID, LEAVING) SUCC = SET_SUCC reset_XSEQ# send_SPR (SUCC) start_SPR_timer C-LEAVE.indication (SOURCE) RET_STATE = 7.12	7.01.Set Predecessor
rec_LR (SOURCE) & SOURCE = SUCC & less_than_3_participants & invited_>_0	stop_PRR_timer LEAVING = SOURCE send_LC (ID, LEAVING) C-LEAVE.indication (SUCC)	2.Invitation sent
rec_LR (SOURCE) & SOURCE = SUCC & less_than_3_participants & invited_=_0	LEAVING = SOURCE send_LC (ID, LEAVING) C-REMOVE.indication (CONFERENCE_ENDED)	0.Idle
rec_SPR (SOURCE)	PRED = SOURCE reset_RSEQ# send_SPC (PRED)	7.12.Pass Predecessor Recovery
rec_PRR (SOURCE,ORIG,NR_PRED) & SOURCE = PRED & RCVR_ORIG = ORIG	send_PRC (PRED)	7.12 Pass Predecessor Recovery
rec_PRR (SOURCE,ORIG,NR_PRED) & SOURCE = PRED & RCVR_ORIG ≠ ORIG	queue_event	7.12 Pass Predecessor Recovery
rec_SRR (SOURCE,ORIG,NR_SUCC)	queue_event	7.12 Pass Predecessor Recovery
rec_RMR (SOURCE)	queue_event	7.12 Pass Predecessor Recovery
rec_AR (SOURCE) &(in_pending | in_nonconfirmed)	STATUS = WAIT send_AC (SOURCE, STATUS)	7.12.Pass Predecessor Recovery

State 7.13: Predecessor Recovery Wait

Event & Condition	Action	Next State
rec_SPR (SOURCE) & RCVR_STATE ≠ 8	stop_rcvr_timer PRED = SOURCE reset_RSEQ# send_SPC (PRED) check_event_queue	7.00.Participant
rec_SPR (SOURCE) & RCVR_STATE = 8	stop_rcvr_timer PRED = SOURCE reset_RSEQ# send_SPC (PRED) SET_SUCC = SUCC send_LR (PRED, SET_SUCC) start_LR_timer	8.Leaving
rec_DSR-ACK (SOURCE, DATA, SEQ#) & SEQ# = RSEQ# & suspended	queue_event	7.13.Predecessor Recovery Wait
rcvr_TO & under_RCVR_limit	send_PRR (SUCC, ORIG, NR_PRED) start_PRR_timer increment_RCVR	7.11.Predecessor Recovery Start
rcvr_TO & over_RCVR_limit	fatal_error	0.Idle
rec_LR (SOURCE, SET_SUCC) & SOURCE = SUCC & 3_or_more_participants	LEAVING = SOURCE send_LC (ID, LEAVING) SUCC = SET_SUCC reset_XSEQ# send_SPR (SUCC) start_SPR_timer C-LEAVE.indication (SOURCE)	7.15.Predecessor Recovery/Set Predecessor
rec_LR (SOURCE) & SOURCE = SUCC & less_than_3_participants & RCVR_STATE ≠ 8 & invited_>_0	stop_rcvr_timer LEAVING = SOURCE send_LC (ID, LEAVING) C-LEAVE.indication (SUCC)	2.Invitation Sent
rec_LR (SOURCE) & SOURCE = SUCC & less_than_3_participants & RCVR_STATE ≠ 8 & invited_=_0	LEAVING = SOURCE send_LC (ID, LEAVING) C-REMOVE.indication (CONFERENCE_ENDED)	0.Idle
rec_LR (SOURCE) & SOURCE = SUCC & less_than_3_participants & RCVR_STATE = 8 & pending_>_0	LEAVING = SOURCE send_LC (ID, LEAVING) send_RVR (PENDING)	0.Idle

State 7.13: Predecessor Recovery Wait		
Event & Condition	Action	Next State
rec_LR (SOURCE) & SOURCE = SUCC & less_than_3_participants & RCVR_STATE = 8 & pending_ = 0	LEAVING = SOURCE send_LC (ID, LEAVING)	0.Idle
C-LEAVE.request	queue_event	7.13.Predecessor Recovery Wait
C-SUCC-DATA-ACK.request (DATA)	queue_event	7.13.Predecessor Recovery Wait
rec_PRR (SOURCE,ORIG,NR_PRED)	queue_event	7.13 Predecessor Recovery Wait
rec_SRR (SOURCE,ORIG,NR_SUCC)	queue_event	7.13 Predecessor Recovery Wait
rec_RMR (SOURCE)	queue_event	7.13 Predecessor Recovery Wait
rec_AR (SOURCE) &(in_nonconfirmed | in_pending)	STATUS = WAIT send_AC (SOURCE, STATUS)	7.13.Predecessor Recovery Wait

State 7.15: Predecessor Recovery/Set Predecessor		
Event & Condition	Action	Next State
rec_SPC (SOURCE) & SOURCE = SUCC	stop_SPR_timer	7.13.Predecessor Recovery Wait
SPR_to & under_SPR_limit	send_SPR (SUCC) start_SPR_timer increment_SPR	7.15.Predecessor Recovery/Set Predecessor
SPR_to & over_SPR_limit	fatal_error	0.Idle
rec_SPR (SOURCE)	PRED = SOURCE reset_RSEQ# send_SPC (PRED) RET_STATE = RCVR_STATE	7.01.Set Predecessor
rec_DSR-ACK (SOURCE, DATA, SEQ#) & SEQ# = RSEQ# & suspended	queue_event	7.15.Predecessor Recovery/Set Predecessor
rcvr_TO & under_RCVR_limit	RET_STATE = 7.11 increment_RCVR	7.01.Set Predecessor
rcvr_TO & over_RCVR_limit	fatal_error	0.Idle

State 7.15: Predecessor Recovery/Set Predecessor		
Event & Condition	Action	Next State
rec_LR (PASS)*		7.15.Predecessor Recovery/Set Predecessor
C-LEAVE.request	queue_event	7.15.Predecessor Recovery/Set Predecessor
C-SUCC-DATA-ACK.request (DATA)	queue_event	7.15.Predecessor Recovery/Set Predecessor
rec_PRR (SOURCE,ORIG,NR_PRED)	queue_event	7.15 Predecessor Recovery/Set Predecessor
rec_SRR (SOURCE,ORIG,NR_SUCC)	queue_event	7.15 Predecessor Recovery/Set Predecessor
rec_RMR (SOURCE)	queue_event	7.15 Predecessor Recovery/Set Predecessor
rec_AR (SOURCE) &(in_nonconfirmed | in_pending)	STATUS = WAIT send_AC (SOURCE, STATUS)	7.15.Predecessor Recovery/Set Predecessor

Notes:
* received a passed along LR CPDU (look at state 8).

State 7.21: Successor Recovery Start		
Event & Condition	Action	Next State
rec_SRC (SOURCE) & SOURCE = PRED	stop_SRR_timer start_rcvr_timer	7.23.Successor Recovery Wait
SRR_to & under_SRR_limit	send_SRR (PRED, ORIG, NR_SUCC) start_SRR_timer increment_SRR	7.21.Successor Recovery Start
SRR_to & over_SRR_limit	fatal_error	0.Idle
rec_SSR (SOURCE) & RCVR_STATE = 7.00	stop_SRR_timer SUCC = SOURCE reset_XSEQ# send_SSC (SUCC) check_event_queue	7.00.Participant

State 7.21: Successor Recovery Start		
Event & Condition	**Action**	**Next State**
rec_SSR (SOURCE) & RCVR_STATE = 7.41	stop_SRR_timer SUCC = SOURCE reset_XSEQ# send_SSC (SUCC) send_DSR-ACK (SUCC, DATA, XSEQ#) start_DSR_timer	7.41.Data Sent
rec_AR (SOURCE) &(in_nonconfirmed | in_pending)	STATUS = WAIT send_AC (SOURCE, STATUS)	7.21.Successor Recovery Start
rec_LR (SOURCE, SET_SUCC) & 3_or_more_participants	stop_SRR_timer LEAVING = SOURCE send_LC (ID, LEAVING) SUCC = SET_SUCC reset_XSEQ# send_SPR (SUCC) start_SPR_timer C-LEAVE.indication (SOURCE) RET_STATE = RCVR_STATE	7.01.Set Predecessor
rec_LR (SOURCE) & less_than_3_participants & invited_>_0	stop_SRR_timer LEAVING = SOURCE send_LC (ID, LEAVING) C-LEAVE.indication (SUCC)	2.Invitation sent
rec_LR (SOURCE) & less_than_3_participants & invited_=_0	LEAVING = SOURCE send_LC (ID, LEAVING) C-REMOVE.indication (CONFERENCE_ENDED)	0.Idle
C-LEAVE.request	queue_event	7.21.Successor Recovery Start
C-SUCC-DATA-ACK.request (DATA)	queue_event	7.21.Successor Recovery Start
rec_DSR-ACK (SOURCE, DATA, SEQ#) & SEQ# = RSEQ# & suspended	queue_event	7.21.Successor Recovery Start
rec_SPR (SOURCE)	PRED = SOURCE reset_RSEQ# send_SPC (PRED) send_SRR (PRED, ORIG, NR_SUCC) start_SRR_timer	7.21.Successor Recovery Start
rec_PRR (SOURCE,ORIG,NR_PRED)	queue_event	7.21.Successor Recovery Start
rec_SRR (SOURCE,ORIG,NR_SUCC)	queue_event	7.21.Successor Recovery Start
rec_RMR (SOURCE)	queue_event	7.21.Successor Recovery Start

State 7.22: Pass Successor Recovery

Event & Condition	Action	Next State
rec_SRC (SOURCE) & SOURCE = PRED	stop_SRR_timer check_event_queue	7.00.Participant
SRR_to & under_SRR_limit	send_SRR (PRED, ORIG, NR_SUCC) start_SRR_timer increment_SRR	7.22.Pass Successor Recovery
SRR_to & over_SRR_limit	PRED = ORIG reset_RSEQ# send_SSR (PRED) start_SSR_timer	7.02.Set Successor
rec_LR (SOURCE, SET_SUCC)	queue_event	7.22.Pass Successor Recovery
C-LEAVE.request	queue_event	7.22.Pass Successor Recovery
C-SUCC-DATA-ACK.request (DATA)	queue_event	7.22.Pass Successor Recovery
rec_DSR-ACK (SOURCE, DATA, SEQ#) & SEQ# = RSEQ# & suspended	queue_event	7.22.Pass Successor Recovery
rec_SPR (SOURCE)	PRED = SOURCE reset_RSEQ# send_SPC (PRED) send_SRR (PRED, ORIG, NR_SUCC) start_SRR_timer	7.22.Pass Successor Recovery
rec_SRR (SOURCE,ORIG,NR_SUCC) & SOURCE = SUCC & RCVR_ORIG = ORIG	send_SRC (SUCC)	7.22 Pass Successor Recovery
rec_SRR (SOURCE,ORIG,NR_SUCC) & SOURCE = SUCC & RCVR_ORIG ≠ ORIG	queue_event	7.22 Pass Successor Recovery
rec_PRR (SOURCE,ORIG,NR_PRED)	queue_event	7.22 Pass Successor Recovery
rec_RMR (SOURCE)	queue_event	7.22 Pass Successor Recovery
rec_AR (SOURCE) &(in_nonconfirmed | in_pending)	STATUS = WAIT send_AC (SOURCE, STATUS)	7.22.Pass Successor Recovery

State 7.23: Successor Recovery Wait		
Event & Condition	Action	Next State
rec_SSR (SOURCE) & RCVR_STATE = 7.00	stop_rcvr_timer SUCC = SOURCE reset_XSEQ# send_SSC (SUCC) check_event_queue	7.00.Participant
rec_SSR (SOURCE) & RCVR_STATE = 7.41	stop_rcvr_timer SUCC = SOURCE reset_XSEQ# send_SSC (SUCC) send_DSR-ACK (SUCC, DATA, SEQ#) start_DSR_timer	7.41.Data Sent
rcvr_TO & under_RCVR_limit	send_SRR (PRED, ORIG, NR_SUCC) start_SRR_timer increment_RCVR	7.21.Successor Recovery Start
rcvr_TO & over_RCVR_limit	fatal_error	0.Idle
rec_AR (SOURCE) &(in_nonconfirmed | in_pending)	STATUS = WAIT send_AC (SOURCE, STATUS)	7.23.Successor Recovery Wait
rec_LR (SOURCE,SET_SUCC) & 3_or_more_participants	stop_rcvr_timer LEAVING = SOURCE send_LC (ID, LEAVING) SUCC = SET_SUCC reset_XSEQ# send_SPR (SUCC) start_SPR_timer C-LEAVE.indication (SOURCE) RET_STATE = RCVR_STATE	7.01.Set Predecessor
rec_LR (SOURCE) & less_than_3_participants & invited_>_0	stop_rcvr_timer LEAVING = SOURCE send_LC (ID, LEAVING) C-LEAVE.indication (SUCC)	2.Invitation sent
rec_LR (SOURCE) & less_than_3_participants & invited_=_0	LEAVING = SOURCE send_LC (ID, LEAVING) C-REMOVE.indication (CONFERENCE_ENDED)	0.Idle
C-LEAVE.request	queue_event	7.23.Successor Recovery Wait
C-SUCC-DATA-ACK.request (DATA)	queue_event	7.23.Successor Recovery Wait
rec_DSR-ACK (SOURCE, DATA) & SEQ# = RSEQ# & suspended	queue_event	7.23.Successor Recovery Wait

State 7.23: Successor Recovery Wait		
Event & Condition	**Action**	**Next State**
rec_SPR (SOURCE)	PRED = SOURCE reset_RSEQ# send_SPC (PRED)	7.23.Successor Recovery Wait
rec_PRR (SOURCE,ORIG,NR_PRED)	queue_event	7.23.Successor Recovery Wait
rec_SRR (SOURCE,ORIG,NR_SUCC)	queue_event	7.23.Successor Recovery Wait
rec_RMR (SOURCE)	queue_event	7.23.Successor Recovery Wait

State 7.41: Data Sent		
Event & Condition	**Action**	**Next State**
rec_DSC (SOURCE, SEQ#) & SEQ# = XSEQ# & SOURCE = SUCC & not_suspended	stop_DSR_timer incr_XSEQ# check_event_queue	7.00.Participant
rec_DSC (SOURCE, SEQ#) & SEQ# = XSEQ# & SOURCE = SUCC & suspended	stop_DSR_timer incr_XSEQ# send_DSC (PRED, RSEQ#) incr_RSEQ# check_event_queue	7.00.Participant
DSR_to & under_DSR_limit	send_DSR (SUCC, DATA, XSEQ#) start_DSR_timer increment_DSR	7.41.Data Sent
DSR_to & over_DSR_limit	NR_SUCC = SUCC ORIG = SELF RCVR_STATE = 7.41 send_SRR (PRED,NR_SUCC,ORIG) start_SRR_timer	7.21.Successor Recovery Start
rec_AR (SOURCE) &(in_nonconfirmed | in_pending)	queue_event	7.41.Data Sent
rec_LR (SOURCE, SET_SUCC) & SOURCE = SUCC & 3_or_more_participants	stop_DSR_timer LEAVING = SOURCE send_LC (ID, LEAVING) SUCC = SET_SUCC reset_XSEQ# send_DSR (SUCC, DATA) start_DSR_timer	7.41.Data Sent
rec_LR (SOURCE) & SOURCE = SUCC & less_than_3_participants & invited_>_0	stop_DSR_timer LEAVING = SOURCE send_LC (ID, LEAVING) C-LEAVE.indication (SUCC)	2.Invitation Sent

State 7.41: Data Sent

Event & Condition	Action	Next State
rec_LR (SOURCE) & SOURCE = SUCC & less_than_3_participants & invited_=_0	LEAVING = SOURCE send_LC (ID, LEAVING) C-REMOVE.indication (CONFERENCE_ENDED)	0.Idle
C-SUCC-DATA-ACK.request (DATA)	queue_event	7.41.Data Sent
rec_SPR (SOURCE)	PRED = SOURCE reset_RSEQ# send_SPC (PRED)	7.41.Data Sent
rec_PRR (SOURCE,ORIG,NR_PRED)	queue_event	7.41.Data Sent
rec_SRR (SOURCE,ORIG,NR_SUCC)	queue_event	7.41.Data Sent
rec_RMR (SOURCE)	queue_event	7.41.Data Sent

State 8: Leaving

Event & Condition	Action	Next State
rec_LC (SOURCE, ID, LEAVING) & SOURCE = PRED & pending_>_0	stop_LR_timer send_RVR (PENDING)	0.Idle
rec_LC (SOURCE, ID, LEAVING) & SOURCE = PRED & pending_=_0	stop_LR_timer	0.Idle
LR_to & under_LR_limit	send_LR (PRED, SET_SUCC) start_LR_timer increment_LR	8.Leaving
LR_to & over_LR_limit	ORIG = SELF NR_PRED = PRED PRED = NULL send_PRR (SUCC, ORIG, NR_PRED) start_PRR_timer RCVR_STATE = 8	7.11.Predecessor Recovery Start
rec_SPR (SOURCE)	PRED = SOURCE send_SPC (PRED) send_LR (PRED,SUCC) start_LR_timer	8.Leaving
rec_LR (SOURCE)* & SOURCE = SUCC	ORIG = SOURCE send_LR (PRED, PASS, ORIG)	8.Leaving
rec_LR (SOURCE, PASS, ORIG) & SOURCE = SUCC & ORIG ≠ SELF*	send_LR (PRED, PASS, ORIG)	8.Leaving

State 8: Leaving		
Event & Condition	Action	Next State
rec_LR (SOURCE, PASS, ORIG) & SOURCE = SUCC & ORIG = SELF** & pending_>_0	send_RVR (PENDING)	0.Idle
rec_LR (SOURCE, PASS, ORIG) & SOURCE = SUCC & ORIG = SELF** & pending_=_0		0.Idle
Notes: * consecutive entities in the logical ring leave at the same time. ** all entities leave at the same time.		

5.4.2 Variable Definitions

This section defines the variables, counters and constants that are used in the state table. These variables are referred to directly by the state table and/or by the events, conditions and procedures defined later in the chapter.

AC# is a counter that keeps track of the number of times an invited entity has retransmitted the same AC CPDU to the inviting entity.

AC_LIMIT is a limit on the number of times the same AC CPDU can be retransmitted.

AR# is a counter that keeps track of the number of times an inviting entity has retransmitted the same AR CPDU to an invited entity.

AR_LIMIT is a limit on the number of times the same AR CPDU can be retransmitted.

BUSY is a value of the CAUSE parameter.

CAUSE is a parameter of the RJR CPDU and C-REMOVE.indication service primitives. It gives the reason an event or action has occurred. The possible values of the parameter are BUSY, LINK_BUSY, LEAVING, NO_ANSWER, REJECTED and CONFERENCE_ENDED.

CONF_ID is an identification number that uniquely identifies the conference connection.

CONFERENCE_ENDED
is a value of the CAUSE parameter.

DATA is a parameter of the DR(-ACK), DCR, and DSR(-ACK) CPDUs, and the C-DATA(-ACK).request, C-DATA(-ACK).indication, C-CONF-DATA.request, C-CONF-DATA.indication, C-SUCC-DATA(-ACK).request and C-SUCC-DATA(-ACK).indication service primitives. It contains conference data.

DATA_DEST is a CSAP that identifies the entity to which the last DR CDPU was sent.

DEST is a CSAP that identifies the destination of the CPDU.

DR# is a counter that keeps track of the number of times an entity has retransmitted the same DR-ACK CPDU.

DR_LIMIT is a limit on the number of times the same DR-ACK CPDU can be retransmitted.

DSR# is a counter that keeps track of the number of times an entity has retransmitted the same DSR-ACK CPDU to its successor.

DSR_LIMIT is a limit on the number of times the same DSR-ACK CPDU can be retransmitted.

ENTITY_LIST is a list containing CSAPs that identify peer protocol entities.

EVENT_QUEUE
is a list that contains events that the protocol was unable to process at the time the event occurred. The protocol will attempt to process these events when it is able.

FAILED is a value of the STATUS parameter.

ID is a parameter of the IR CPDU. It contains the CONF_ID.

INVITED_LIST is a parameter of the C-INVITE.request and C-INVITE.indication service primitives. It is a list of one or more CSAPs that identify the remote entities that are invited to a conference.

INVITED is a parameter of the C-INVITE.indication service primitive. It is a CSAP that identifies a remote entity that is invited to a conference.

INVITER is a parameter of all three C-INVITE service primitives. It is a CSAP that identifies the entity which sent out the conference invitation.

IR# is a counter that keeps track of the number of times an inviting entity has retransmitted the same IR CPDU to an invited entity.

IR_LIMIT is a limit on the number of times the same IR CPDU can be retransmitted.

LEAVING is a parameter of the LC CPDU. It is a CSAP that identifies the entity that is leaving the conference.

LINK_BUSY is a value of the CAUSE parameter.

LIST is a parameter of the STR CPDU that contains a CSAP that identify a peer protocol entities participating in the conference.

LR# is a counter that keeps track of the number of times an entity, requesting to leave an ongoing conference, has retransmitted the same LR CPDU to a remote entity.

LR_LIMIT is a limit on the number of times the same LR CPDU can be retransmitted.

NULL is an identifier of an invalid CSAP. This is used to equate the SUCC and PRED to an invalid CSAP when the successor or predecessor is not known.

NO_ANSWER is a value of the CAUSE parameter.

NONCONFIRMED

is a list of CSAPs that identify the invited entities that have not confirmed the

invitation request. That is, the remote entities have not responded to the IR CPDU with a IC CPDU.

NR_PRED is a parameter of the PRR CPDU. It is a CSAP that identifies the remote entity that is not responding to its successor.

NR_SUCC is a parameter of the SRR CPDU. It is a CSAP that identifies the remote entity that is not responding to its predecessor.

OPTIONS is a parameter of the IR CPDU, and the C-INVITE.request and C-INVITE.indication service primitives.

ORIG is a parameter of the LR, PRR and SRR CPDUs. It is the originating entity of a predecessor or successor recovery procedure, or leave request.

PASS is a parameter of the LR CPDU. It indicates that the LR CPDU did not originate from the sending entity, but originated from or either was passed by the sending entity's successor.

PENDING is a list of CSAPs that identify the invited entities that confirmed the invitation request, but have not accepted or rejected the invitation. That is, the entities has not responded with an AR or RJR CPDU.

PRR# is a counter that keeps track of the number of times an entity has retransmitted the same PRR CPDU to its successor.

PRR_LIMIT is a limit on the number of times the same PRR CPDU can be retransmitted.

PRED is a CSAP that identifies the remote entity that is the entity's predecessor in the logical ring of conference participants.

PREV_SUCC is a CSAP that identifies the entity's previous successor in the logical ring. The entity's successor has been changed due to the receipt of a AR CPDU from an invited entity.

RCVR# is a counter that keeps track of the number of times an entity, who has lost contact with its predecessor or successor, has restarted the logical ring recovery procedure.

RCVR_LIMIT is a limit on the number of times the logical ring recovery procedure can be restarted.

RCVR_ORIG is a CSAP that keeps track of the entity that is the originator of a predecessor or successor recovery procedure.

RCVR_STATE is the state in which the protocol entity should end up after a predecessor or successor recovery procedure.

REMOVED_ENTITY

is a CSAP that identifies the entity to which a RMR CPDU has been sent.

REJECT is a value of the STATUS parameter.

REJECTED is a value of the CAUSE parameter.

RET_STATE is the state to which the protocol entity should return after a specific event has occurred.

RMR# is a counter that keeps track of the number of times an entity has retransmitted the same RMR CPDU to its successor.

RMR_LIMIT is a limit on the number of times the same RMR CPDU can be retransmitted.

RMT_ENTITY is a CSAP that identifies a remote peer protocol entity.

RSEQ# is the expected sequence number of the next DSR-ACK CPDU to be sent by an entity's predecessor.

SELF is the CSAP that identifies the local protocol entity.

SET_SUCC is a parameter of the AC and LR CPDUs. It informs the receiving entity of its new successor in the logical ring of conference participants.

SEQ# is a field of the DSR-ACK and a parameter of the DSC CPDU. It indicates the sequence number of the DSR-ACK CPDU that is being sent or confirmed.

SOURCE is a CSAP that identifies the remote peer protocol entity which is the source of the CPDU for the current event that is being processed.

SPR# is a counter that keeps track of the number of times an entity has retransmitted the same SPR CPDU to its successor.

SPR_LIMIT is a limit on the number of times the same SPR CPDU can be retransmitted.

SRR# is a counter that keeps track of the number of times an entity has retransmitted the same SRR CPDU to its predecessor.

SRR_LIMIT is a limit on the number of times the same SRR CPDU can be retransmitted.

SSR# is a counter that keeps track of the number of times an entity has retransmitted the same SSR CPDU to its predecessor.

SSR_LIMIT is a limit on the number of times the same SSR CPDU can be retransmitted.

STATUS is a parameter of the AC CPDUs, and the C-INVITE-STATUS.indication and C-ACCEPT-STATUS.indication service primitives. It contains the answer to either an invitation, accept or join request. The possible values of the parameter are SUCCESS, FAILED, CONNECTED, WAIT and REJECT.

SUCC is a CSAP that identifies the remote entity that is the entity's successor in the logical ring of conference participants.

SUCCESS is a value of the STATUS parameter.

SUSPEND indicates whether the conference connection is active or suspended.

WAIT is a value of the STATUS parameter.

XSEQ# is the sequence number of the DSR-ACK CPDU to be sent next to an entity's

successor.

5.4.3 Event Definitions

This section defines the meaning of events used in the first column of the state table.

AR_to indicates the timer that was started when the AR CPDU was sent timed out.

AC_to indicates the timer that was started when the AC CPDU was sent timed out.

DR_to indicates the timer that was started when the DR-ACK CPDU was sent timed out.

DSR_to indicates the timer that was started when the DSR-ACK CPDU was sent timed out.

IR_to indicates the timer that was started when the IR CPDU was sent timed out.

LR_to indicates the timer that was started when the LR CPDU was sent timed out.

PRR_to indicates the timer that was started when the PRR CPDU was sent timed out.

rcvr_to indicates the timer that was started when the recovery process was initiated timed out.

RMR_to indicates the timer that was started when the RMR CPDU was sent timed out.

SPR_to indicates the timer that was started when the SPR CPDU was sent timed out.

SRR_to indicates the timer that was started when the SRR CPDU was sent timed out.

SSR_to indicates the timer that was started when the SSR CPDU was sent timed out.

wait_to indicates the timer that indicates the amount of time that has elapsed waiting to resend the AR CPDU has timed out.

rec_AC (SOURCE, STATUS, SET_SUCC),
rec_ACC (SOURCE, ID),
rec_AR (SOURCE),
rec_DC (SOURCE),
rec_DCR (SOURCE,ID, DATA),

rec_DR (SOURCE, ID, DATA),
rec_DR-ACK (SOURCE, ID, DATA),
rec_DSC (SOURCE, SEQ#),
rec_DSR (SOURCE, DATA),
rec_DSR-ACK (SOURCE, DATA, SEQ#),
rec_IC (SOURCE),
rec_IR (SOURCE, ID, OPTIONS),
rec_LC (SOURCE, LEAVING),
rec_LR (SOURCE, SET_SUCC, PASS, ORIG),
rec_PRC (SOURCE),
rec_PRR (SOURCE, ORIG, NR_PRED),
rec_RJR (SOURCE, CAUSE),
rec_RMC (SOURCE),
rec_RMR (SOURCE, CAUSE),
rec_RVR (SOURCE),
rec_SPC (SOURCE),
rec_SPR (SOURCE),
rec_SRC (SOURCE),
rec_SRR (SOURCE, ORIG, NR_SUCC),
rec_SSC (SOURCE),
rec_SSR (SOURCE) and
rec_STR (SOURCE, ORIG, STATUS_LIST)

indicate the receipt of the indicated CPDU from the SOURCE. With each of these event definitions is a list of all possible parameters that can be part of the CPDU. However, not every CPDU must contain all the parameters listed with the event definitions. Additionally, the state table only lists those parameters that are pertinent to the particular event, conditions and actions taken by the state table.

C-INVITE.request (INVITED_LIST,OPTIONS),
C-ACCEPT.request,
C-REJECT.request (CAUSE),
C-REVOKE.request,
C-LEAVE.request,
C-REMOVE.request (SOURCE),
C-STATE.request
C-SUSPEND.request
C-RESUME.request
C-CONF-DATA.request (DATA),
C-SUCC-DATA.request (DATA)
C-SUCC-DATA-ACK.request (DATA)

C-DATA.request (DATA) and
C-DATA-ACK.request (DATA)

indicate that the protocol user has issued the indicated conferencing protocol service primitive. These service primitives are the service primitives that have already been defined in chapter 4.

5.4.4 Condition Definitions

This section defines the conditions that are used in the first column of the finite state machine.

busy indicates whether the entity is *too busy* and can not create a conference connection.

in_nonconfirmed

indicates whether SOURCE is in the NONCONFIRMED list.

in_pending indicates whether SOURCE is in the PENDING list.

invited_=_0 indicates whether the total number of entries in the PENDING and NONCONFIRMED lists combined is equal to zero. This occurs when both lists are empty.

invited_>_0 indicates whether the total number of entries in the PENDING and NONCONFIRMED lists combined is greater than zero. There is at least one entry in the PENDING and/or NONCONFIRMED lists.

invited_=_1 indicates whether the total number of entries in the PENDING and NONCONFIRMED lists combined is equal to one. There is only one entry in either the PENDING or NONCONFIRMED lists, but not both.

invited_>_1 indicates whether the total number of entries in the PENDING and NONCONFIRMED lists combined is greater than one. There is at least two

entries in the PENDING and/or NONCONFIRMED lists, or one entry in the PENDING list and one entry in the NONCONFIRMED list.

less_then_3_participants

indicates whether there are less then three entities that are participating in the conference. This occurs when SUCC = PRED.

nonconfirmed_>_0

indicates whether the number of entries in the NONCONFIRMED list is greater than zero. There is at least one entry in the NONCONFIRMED list.

not_busy indicates whether the entity is *not* too busy and can create a conference connection.

not_suspended indicates whether the connection is active, that is, the SUSPEND variable indicates the connection is active.

over_AR_limit indicates whether AR# > AR_LIMIT.

over_AC_limit indicates whether AC# > AC_LIMIT.

over_DR_limit indicates whether DR# > DR_LIMIT.

over_DSR_limit indicates whether DSR# > DSR_LIMIT.

over_IR_limit indicates whether IR# > IR_LIMIT.

over_LR_limit indicates whether LR# > LR_LIMIT.

over_SPR_limit indicates whether SPR# > SPR_LIMIT.

over_PRR_limit indicates whether PRR# > PRR_LIMIT.

over_RCVR_limit

indicates whether RCVR# > RCVR_LIMIT.

over_RMR_limit

indicates whether RMR# > RMR_LIMIT.

over_SRR_limit indicates whether SRR# > SRR_LIMIT.

over_SSR_limit indicates whether SSR# > SSR_LIMIT.

3_or_more_participants

indicates whether there are three or more entities that are participating in the conference. This occurs when SUCC ≠ PRED.

pending_=_0 indicates whether the number of entries in the PENDING list is equal to zero. This occurs when the list is empty.

pending_>_0 indicates whether the number of entries in the PENDING list is greater than zero. There is at least one entry in the PENDING list.

suspended indicates whether the connection is suspended, that is, the SUSPEND variable indicates the connection is suspended.

under_AR_limit indicates whether AR# ≤ AR_LIMIT.

under_AC_limit indicates whether AC# ≤ AC_LIMIT.

under_DR_limit

indicates whether DR# ≤ DR_LIMIT.

under_DSR_limit

indicates whether DSR# ≤ DSR_LIMIT.

under_IR_limit indicates whether IR# ≤ IR_LIMIT.

under_LR_limit indicates whether LR# ≤ LR_LIMIT.

under_SPR_limit

indicates whether SPR# ≤ SPR_LIMIT.

under_PRR_limit

indicates whether PRR# ≤ PRR_LIMIT.

under_RCVR_limit

indicates whether RCVR# ≤ RCVR_LIMIT.

under_RMR_limit

indicates whether RMR# ≤ RMR_LIMIT.

under_SSR_limit

indicates whether SSR# ≤ SSR_LIMIT.

under_SRR_limit

indicates whether SRR# ≤ SRR_LIMIT.

5.4.5 Procedure Definitions

This section defines the actions of the procedures that are used in the second column of the state table. The procedures are divided into two groups. The first group defines all procedures except for those that deal with forming and sending the CPDUs to the remote peer protocol entities. Those procedures are defined in the second group.

add_list adds a LIST parameter containing the entity's CSAP and connection status (active or suspended) to the LIST parameters of the STR CPDU the entity just received.

add_to_nonconfirmed

adds the CSAPs in the INVITED_LIST to the NONCONFIRMED list.

add_to_pending adds the SOURCE to the PENDING list.

C-INVITE.indication (INVITER, OPTIONS),
C-INVITE-STATUS.indication (INVITED, STATUS_LIST),
C-ACCEPT.indication (INVITED),
C-ACCEPT-STATUS.indication (STATUS),
C-REJECT.indication (INVITED, CAUSE),

C-REVOKE.indication,
C-LEAVE.indication (RMT_ENTITY),
C-REMOVE.indication (SOURCE, CAUSE),
C-REMOVE-STATUS.indication (STATUS),
C-STATE-STATUS.indication (STATUS_LIST),
C-SUSPEND-STATUS.indication,
C-RESUME-STATUS.indication
C-CONF-DATA.indication (SOURCE, DATA),
C-SUCC-DATA.indication (DATA),
C-SUCC-DATA-ACK.indication (DATA),
C-DATA.indication (SOURCE, DESTINATION, DATA),
C-DATA-ACK.indication (SOURCE, DESTINATION, DATA) and
C-DATA-ACK-STATUS.indication (SOURCE, DESTINATION, STATUS)

are conferencing protocol service primitives that are used by the protocol to communicate with the protocol user. These service primitives are the service primitives that have already been defined in chapter 4.

check_event_queue

checks the EVENT_QUEUE and processes all the events that are queued.

clear_nonconfirmed

clears the NONCONFIRMED list from all its entries.

clear_pending clears the PENDING list from all its entries.

fatal_error a fatal error has occurred. The protocol entity is dropping out of the conference.

increment_AR increments AR# by one.

increment_AC increments AC# by one.

increment_DR increments DR# by one.

increment_DSR increments DSR# by one.

increment_IR increments IR# by one.

increment_LR increments LR# by one.

increment_SPR increments SPR# by one.

increment_PRR increments PRR# by one.

increment_RCVR

increments RCVR# by one.

increment_RMR

increments RMR# by one.

increment_RSEQ#

increments RSEQ# by one.

increment_SRR increments SRR# by one.

increment_SSR increments SSR# by one.

increment_XSEQ#

increments XSEQ# by one.

queue_event queues an event on to the EVENT_QUEUE. The event can not be processed at the time it occurs. It will be processed later.

remove_from_nonconfirmed

removes the SOURCE from the NONCONFIRMED list.

remove_from_pending

removes the SOURCE from the PENDING list.

reset_suspend resets the SUSPEND variable to indicate the connection is active.

set_CONF_ID sets the conference identification number for the conference.

set_suspend sets the SUSPEND variable to indicate the connection is suspended.

start_AC_timer starts the timer that indicates the amount of time that has elapsed since the AC CPDU has been sent to a remote protocol entity.

start_AR_timer starts the timer that indicates the amount of time that has elapsed since the

AR CPDU has been sent to a remote protocol entity.

start_DR_timer

starts the timer that indicates the amount of time that has elapsed since the DR-ACK CPDU has been sent to a remote protocol entity.

start_DSR_timer

starts the timer that indicates the amount of time that has elapsed since the DSR-ACK CPDU has been sent to a remote protocol entity.

start_IR_timer starts the timer that indicates the amount of time that has elapsed since the IR CPDU has been sent to a remote protocol entity.

start_LR_timer starts the timer that indicates the amount of time that has elapsed since the LR CPDU has been sent to a remote protocol entity.

start_PRR_timer

starts the timer that indicates the amount of time that has elapsed since the PRR CPDU has been sent to a remote protocol entity.

start_rcvr_timer

starts the timer that indicates the amount of time that has elapsed since the predecessor or successor recovery procedure has started.

start_RMR_timer

starts the timer that indicates the amount of time that has elapsed since the RMR CPDU has been sent to a remote protocol entity.

start_SPR_timer

starts the timer that indicates the amount of time that has elapsed since the SPR CPDU has been sent to a remote protocol entity.

start_SRR_timer

starts the timer that indicates the amount of time that has elapsed since the SRR CPDU has been sent to a remote protocol entity.

start_SSR_timer

starts the timer that indicates the amount of time that has elapsed since the SSR CPDU has been sent to a remote protocol entity.

start_wait_timer

starts the timer that indicates the amount of time that has elapsed waiting to resend the AR CPDU.

stop_AC_timer stops the timer that indicates the amount of time that has elapsed since the AC CPDU has been sent to a remote protocol entity.

stop_AR_timer stops the timer that indicates the amount of time that has elapsed since the AR CPDU has been sent to a remote protocol entity.

stop_DR_timer

stops the timer that indicates the amount of time that has elapsed since the DR-ACK CPDU has been sent to a remote protocol entity.

stop_DSR_timer

stops the timer that indicates the amount of time that has elapsed since the DSR-ACK CPDU has been sent to a remote protocol entity.

stop_LR_timer stops the timer that indicates the amount of time that has elapsed since the LR CPDU has been sent to a remote protocol entity.

stop_PRR_timer

stops the timer that indicates the amount of time that has elapsed since the PRR CPDU has been sent to a remote protocol entity.

stop_rcvr_timer stops the timer that indicates the amount of time that has elapsed since the predecessor or successor recovery procedure has started.

stop_RMR_timer

stops the timer that indicates the amount of time that has elapsed since the RMR CPDU has been sent to a remote protocol entity.

stop_SPR_timer

stops the timer that indicates the amount of time that has elapsed since the SPR CPDU has been sent to a remote protocol entity.

stop_SRR_timer

stops the timer that indicates the amount of time that has elapsed since the SRR CPDU has been sent to a remote protocol entity.

stop_SSR_timer

stops the timer that indicates the amount of time that has elapsed since the SSR CPDU has been sent to a remote protocol entity.

The procedures described in remainder of this section form and send the CPDUs to remote peer protocol entity. The operation of the procedures will be described in a pseudo-code and use the following routines.

for_each_entity (ENTITY_LIST) *procedure*

traverses through the ENTITY_LIST and the *procedure* that follows is performed for each entity in the list.

form_CPDU (CPDU_type, PARAMETERS)

forms the CPDU specified by CPDU_type with the specified PARAMETERS.

t-send (DEST, CPDU_type, CPDU)

sends the CPDU to the DEST entity using the services provided by the lower layer. It is this routine that interfaces with the services of the transport layer. A more detailed description of the routine's operation and interface with the transport layer is described in the next section.

There is a separate procedure for each CPDU type. With each procedure definition is a list of all possible parameters that can be part of the CPDU. However, not every CPDU must contain all the parameters listed with the procedure definitions. The state table lists only those parameters that should be included in the CPDU.

The procedure definitions are grouped together since all the procedures in each group are similar in their operation. The differences between the procedures are the CPDU type and the parameters to be included in the CPDU. Thus, in the pseudo-code of each group, *XXX* is to be replaces by the appropriate CPDU type and *PARAMETERS* is to be replaced by the appropriate parameters.

send_AC (DEST, STATUS, SET_SUCC),
send_ACC (ID),
send_AR (DEST),
send_DC (DEST),
send_DCR (ID, DATA),
send_DR (DEST, ID, DATA),
send_DR-ACK (DEST, ID, DATA),
send_DSC (DEST, SEQ#),
send_DSR (DEST, DATA),
send_DSR-ACK (DEST, DATA, SEQ#),
send_IC (DEST),
send_single_IR (DEST, ID, OPTIONS),
send_LC (DEST),
send_LR (DEST, SET_SUCC),
send_PRC (DEST),
send_PRR (DEST, ORIG, NR_PRED),
send_RJR (DEST, CAUSE),

send_RMC (DEST),
send_RMR (DEST, CAUSE),
send_single_RVR (DEST),
send_SPC (DEST),
send_SPR (DEST),
send_SRC (DEST),
send_SRR (DEST, ORIG, NR_SUCC),
send_SSC (DEST),
send_SSR (DEST) and
send_STR (DEST)

```
            form_CPDU (XXX, PARAMETERS);
            t_send (DEST, XXX, CPDU);
```

send_IR (ENTITY_LIST, ID, OPTIONS) and
send_RVR (ENTITY_LIST)

```
            for_each_entity (ENTITY_LIST)
                  send_single_XXX (DEST, PARAMETERS);
```

5.5 Lower Layer Interface

As stated previously, the conferencing protocol creates a conference connection among application processes. It assumes the underlying protocol layer, to which it interfaces, provides transparent end-to-end transmission of data. This interface would occur most naturally at the transport layer of the ISO OSI reference model. Since the conferencing protocol creates a multipoint connection between application processes, the conferencing protocol is viewed as a session layer protocol with the ability to create and regulate a multipoint, session layer, conference connection among multiple session layer users. An application process uses the protocol to create a session layer multipoint connection. It should be noted that, at present, ISO only defines point-to-point communication at the session layer. However, defining a conferencing protocol as part of the session layer can be found in [Leun89] and [Leun90].

Table 5.4: ISO Transport Protocol Service Primitive Used By Conferencing Protocol

Service Primitive	Function
connection-oriented	
T-CONNECT.request (calling TSAP, called TSAP) T-CONNECT.indication (calling TSAP, called TSAP) T-CONNECT.response (calling TSAP, called TSAP) T-CONNECT.confirm (calling TSAP, called TSAP)	Creates a transport layer connection between the calling TSAP and the called TSAP.
T-DISCONNECT.request T-DISCONNECT.indication	Terminates the transport layer connection.
T-DATA.request (user data) T-DATA.indication (user date)	Sends data over the transport layer connection.
connectionless	
T-UNIDATA.request (source TSAP, destination TSAP, user data) T-UNIDATA.indication (source TSAP, destination TSAP, user data)	Connectionless transport layer services.

The interface to the transport layer is described in the terms of the transport protocol service primitives defined by ISO. It is assumed that both connection-oriented and connectionless transport protocol services are available to the conferencing protocol. The connection-oriented services are used to create transport layer connections between the conferencing protocol entity and its predecessor and successor in the logical ring of conference participants. The connectionless services are used to transport multicast CPDUs and unicast CPDUs to other conference participants. (It should be noted, the protocol can be implemented over a transport layer that only provides either a connection-oriented or connectionless service.) Table 5.4 lists the ISO transport protocol service primitives that are used by the interface to the transport layer. They are a subset of the service primitives defined for the ISO transport protocol [STALL91, TANN89]. These services are available to the conferencing protocol entity at defined TSAPs. It is also assumed that the T-UNIDATA service primitive provides multicast capabilities by using a multicast TSAP for the *destination TSAP* parameter.

ISO, at present, only defines a one-to-one connection between a CSAP and a TSAP.

However, a conferencing protocol entity must communicate with its successor, its predecessor and other participating entities. The entity must also create two transport layer connections; one with its successor and the other with its predecessor. Thus, the conferencing protocol entity must interface with two connection endpoints (CEPs) [Toma87] of the TSAP providing the connection-oriented transport layer services, called the C-TSAP. One CEP is used to create a transport layer connection with the entity's successor, called the S-CEP. The other CEP is used to create a transport layer connection with the entity's predecessor, called the P-CEP. In addition, the protocol interfaces with the TSAP that provides the connectionless oriented transport layer service, called the U-TSAP, to send unicast and multicast CPDUs to remote conferencing protocol entities.

The remainder of the section details the interface to the transport layer. Specifically, it details

- creating and terminating the transport layer connections between the entity and its predecessor and successor,
- which service primitive is used to send and receive each CPDU, and
- over which TSAP and/or CEP is each CPDU sent and received.

The interface is described in two procedures called *t-send* and *t-receive*. Section 5.5.1 describes the actions of the *t-send* procedure and section 5.5.2 describes the *t-receive* procedure. The actions of these procedures are described in pseudocode.

5.5.1 t-send Procedure

The *t-send* procedure was introduced previously in section 5.4.5 as part of the definitions describing the "actions" within the finite state table. The procedure was defined then as to send CPDUs to remote protocol entities by using the services provided by the lower layer.

More specifically, the *t-send* procedure

- requests the creation and waits for confirmation of transport layer connections,
- requests the termination of transport layer connections, and
- sends CPDUs to remote entities using the appropriate service primitives.

The *t-send* procedure uses the transport protocol service primitives listed in table 5.4.

In describing the actions of *t-send* procedure, the following procedures are used by the pseudocode.

connection_exists (cep)

is a function that returns TRUE if the a connection exists over the CEP specified in *cep*.

p-cep (service_primitive, parameters),
s-cep (service_primitive, parameters) and
u-tsap (service_primitive, parameters)

are procedures that issue transport layer services primitive at the CEP or TSAP specified in the procedure name. The *service_primitive* parameter contains the service primitive to be issued. The parameters to be used with the primitive are specified in the *parameters* parameter.

remote_p-tsap (dest),
remote_s-tsap (dest) and
remote_u-tsap (dest)

are functions that return the indicated TSAP of the remote CSAP contained in the *dest* parameter. If *dest* of the *remote_u-tsap* procedure contains the conference identification number, the *remote_u-tsap* function returns a

multicast TSAP.

wait_for_confirmation

is a procedure that waits for a signal indicating the receipt of a T-CONNECT.confirm service primitive.

The following is the pseudocode describing the actions of the *t-send* procedure:

```
t-send (dest, cpdu_type, cpdu)
{
   switch (cpdu_type) {
      case ACC:
      case AR:
      case DC:
      case DCR:
      case DR:
      case DR-ACK:
      case IC:
      case IR:
      case RJR:
      case RMR:
      case RVR:
         u-tsap (T-UNIDATA.request, U-TSAP,
                remote_u-tsap(dest), cpdu);
         break;


      case AC:
      case SPR:
         if connection_exists (S-CEP)
            s-cep (T-DISCONNECT.request);
         s-cep (T-CONNECT.request, C-TSAP,
                remote_p-tsap(dest));
         wait_for_confirmation;
         s-cep (T-DATA.request, cpdu);
         break;

      case SSR:
         if connection_exists (P-CEP)
            p-cep (T-DISCONNECT.request);
         p-cep (T-CONNECT.request, C-TSAP,
                remote_s-tsap(dest));
         wait_for_confirmation;
         p-cep (T-DATA.request, cpdu);
         break;
```

```
        case LC:
           s-cep (T-DISCONNECT.request);
           u-tsap (T-UNIDATA.request, U-TSAP,
                  remote_u-tsap(dest), cpdu);
           break;


        case DSR:
        case DSR-ACK:
        case PRR:
        case SRC:
        case SSC:
        case STR:
           s-cep (T-DATA.request, cpdu);
           break;

        case DSC:
        case LR:
        case PRC:
        case SPC:
        case SRR:
           p-cep (T-DATA.request, cpdu);
           break;
     } /* switch */
  } /* t-send */
```

5.5.2 t-receive Procedure

The *t-receive* procedure interfaces with the transport protocol in the opposite direction.

The procedure

- responds to the request to create a transport layer connection and
- receives the CPDUs from remote entities over the different CEPs and TSAPS.

The *t-receive* procedure uses and reacts to the transport layer service primitives listed in table 5.4.

The procedures used by *t-send* is also available to *t-receive*. The following procedures

are used by t-receive in addition to the previously described procedures.

signal_CPDU_event (cpdu)

disassembles the CPDU into its field and parameter. It then signals the protocol entity of the receipt of a CPDU, the type of CPDU received and the parameters.

signal_confirmation

signals the *wait_for_confirmation* procedure, described earlier, that a T-CONNECT.confirm service primitive has been received.

wait_for_primitive (primitive, parameters)

waits for the transport protocol to issue a service primitive. This procedure handles the service primitives issued by the transport protocol at all CEPs and TSAPs. Upon return from the procedure, the service primitive that was issued and its parameters are contained in *primitive* and *parameters*, respectively.

The following is the pseudocode describing the actions of the *t-receive* procedure:

```
t-receive ()
{
   while (1) {
      wait_for_primitive (primitive, parameters);
      switch (primitive) {

         case T-CONNECT.indication:
            if called_TSAP == P-CEP
               p-cep (T-CONNECT.response, calling_TSAP,
                     P-CEP);
            else if called_TSAP == S-CEP
               s-cep (T-CONNECT.response, calling_TSAP,
                     S-CEP);
            break;

         case T-CONNECT.response:
            signal_confirmation;
            break;
```

```
            case T-DATA.indication:
            case T-UNIDATA.indication:
               signal_CPDU_event (user-data);
               break;

         } /* switch */
      } /* while */
   } /* t-receive */
```

5.6 Protocol Implementation

The conferencing protocol was implemented in the Brooklyn College Local Area Network Laboratory (BrookLAN). The laboratory currently consists of an IBM PC AT file server and seven remote workstations (three Compaq 386/20s, two IBM PC ATs, and two IBM PCs). Five portable PCs, (Three Toshiba 3200s and two Compaq PCs) are connected to the network as needed. The workstations and file server are networked with three different networks: an Ethernet, a Proteon Pronet 10 mbs token ring and an IBM token ring.

The program implementing the conferencing protocol is written in the C programming language. The protocol is implemented over a lower layer that provides a connectionless service. It implements the services and mechanisms described for the protocol. They include:[1]

- Initiating a conference connection among the workstations including sending, accepting, rejecting and revoking conference invitations.
- Expanding a conference to include additional workstations.
- Contracting a conference including leaving an ongoing conference and removing a remote workstation from the conference.
- Setting up and maintaining a logical ring among the participating workstations by setting and modifying the successor and predecessor

[1] At present, the program does not implement the services to suspend and reconnect a conference connection (C-SUSPEND and C-RECONNECT), request the state of the conference (C-STATE), unicast data (C-DATA and C-DATA-ACK) and send unacknowledged successor data (C-SUCC-DATA).

pointers, and detecting and recovering from erroneous breaks in the logical ring.

- Multicasting unacknowledged conference data to all workstations.
- Sending acknowledged data to the workstation's successor in the logical ring.

The program implements the finite state machine that specifies the conferencing protocol as described in section 5.4. It implements the variables, events, conditions and actions (sections 5.4.2 - 5.4.5) defined for the transition table that represents the finite state machine. The program mimics the state transition table (section 5.4.1) by executing the actions and state transitions stated in the transition table for all the events and conditions defined for each state of the transition table.

The protocol was initially implemented on a single machine to test that it executed all operations defined for the transition table. This was done by simulating the exchange of CPDUs between protocol entities. The sending of CPDUs was simulated by writing to the workstation's display all CPDUs that are to be sent to remote protocol entities. The receipt of CPDUs was simulated by keying in from the keyboard the events indicating the receipt of a CPDU from a remote protocol entity. All events and conditions were tested to verify that the program executes all actions and state transitions described in the transition table for the finite state machine.

The complete protocol was then implemented to allow for the direct exchange of CPDUs between the workstations running over a live Ethernet. The program was tested between three workstations and did establish, maintain, expand, contract, terminate, and send unacknowledged multicast and acknowledged successor data over a conference connection. It established the conference connection as a logical ring of conference participants, and sets and

modifies the successor and predecessor pointers correctly. The error recovery mechanisms was tested by shutting down the program of a participating workstation before it left an ongoing conference. The remaining workstations did reestablish the logical ring. It should be noted the timeout mechanisms for the program was simulated by keying in from the keyboard all timeout events.

The following is an analysis of the number of CPDUs that are required to establish and terminate a conference among the conference participants. The analysis assumes N conference participants, $N - 1$ entities were invited by an inviting entity, $N - 1$ entities accepted the invitation and no timeouts occur throughout the whole process. The following is the number of CPDUs that are required to establish the conference:

- $N - 1$ IR and IC CPDUs for the inviting entity to send out the conference invitation to each invited entity.
- $N - 1$ AR, AC and ACC CPDUs for each invited entity to accept the conference invitation and be inserted into the logical ring.
- $N - 2$ SPR and SPC CPDUs for each invited entity accepting an invitation, except for the first accepting entity as explained in section 5.3.1.4, to update the predecessor pointer of its successor.

For conference termination it is assumed that all conference participants leave in turn. Thus, there are no "passed" LR CPDUs. The following is the number of CPDUs required to terminate the conference:

- $N - 1$ LR and LC CPDUs for each participating entity, except for the last remaining entity, to leave the conference and set the successor of its predecessor.
- $N - 2$ SPR and SPC CPDUs for each participating entity that receives

an LR CPDU, except for the last remaining entity, to set the predecessor of its new successor.

Thus, a conference with N participants requires a minimal of $11N - 15$ CPDUs to establish and terminate a conference. Of course, this does not include any recovery mechanisms including the possible retransmission of lost CPDUs and possible breaks in the logical ring.

Chapter 6

Conclusion

A distributed conferencing protocol has been presented. Its function is to setup, maintain, transfer data over and terminate a conference connection among multiple users. The protocol uses a distributed conference connection control mechanism that views a conference as a logical ring of conference participants. It allows for expansion and contraction of ongoing conferences. The conferencing protocol is viewed as a session layer protocol.

The conference protocol service provides have been presented and provide the user with the ability to:

- invite remote users to a conference (whether for a new conference or to an ongoing conference),
- revoke a pending invitation (that is, an invitation that was neither accepted nor rejected),
- accept an invitation to a conference (to become a conference participant),
- reject a conference invitation,
- inquire about the state of the conference (specifically, which users are participating in the conference),
- suspend the conference connection (to terminate any data transfer and still remain a participant of the conference),

- reconnect a suspended connection (to resume data transfer),
- leave an ongoing conference,
- remove a remote user from an ongoing conference,
- multicast conference data to all conference participants,
- send data to the user's successor in the logical ring of participants

  and
- unicast data to a conference participant.

The CPDUs and mechanisms that are used by the peer protocol entities to establish and maintain a conference connection and establish a logical ring of participating entities were described. A formal specification of the protocol as a finite state machine was presented. The protocol was specified to interface with the ISO OSI transport layer protocol. An implementation of the protocol over a PC-based Ethernet was described.

As stated previously, the conferencing protocol is a session layer protocol. However, many of the functions that ISO defined for the session layer are not currently included in the protocol. These functions include the use and management of tokens, synchronization points and activities. Further development of the protocol would incorporate these features into the conferencing protocol. Additionally, the application protocols must be developed to interface with the conferencing protocol.

The protocol was implemented over an Ethernet. Due to the broadcast nature of Ethernet and the relatively small sizes of LANs, in general, this type of environment is amicable to any type of conferencing scheme. On the other hand, large, long-haul, packet-switched networks do not have this advantage and are not as affable to conferencing. It is in this long-haul environment that the full advantage of the distributed conferencing protocol might be realized. Further research would include implementations over long-haul, packet-switched

networks.

Additional studies of the protocol would include a quantitative analysis of the protocol performance. A method must be developed to measure the efficiency of the protocol as compared with other conferencing schemes and over different types of networks; that is, broadcast versus non-broadcast networks.

Additional research would include the development of a full service multimedia conferencing system that will use the conferencing protocol that has been presented. This will show the full power of the conferencing protocol.

Bibliography

Agui86 Aguilar, L., Garcia-Luna-Aveces, J.J., Moran, D., Craighill, E.J. and Brungardt, R., "Architecture for a Multimedia Teleconferencing System," *Proceedings of SIGCOM '86*, pp. 126-138, Aug. 1986.

Ahuj88a Ahuja, S.R., Horn, D.N. and Ensor, J.R., "Networking Requirements of the Rapport Multimedia Conferencing System," *Proceedings of INFOCOM '88*, pp. 746-751, March 1988.

Ahuj88b Ahuja, S.R., Horn, D.N. and Ensor, J.R., "The Rapport Multimedia Conferencing System," *Proceedings of the Conference on Office Information Systems*, pp. 1-8, March 1988.

Ahuj88b Ahuja, S.R., Ensor, J.R. and Lucco S.E., "A Comparison of Application Sharing Mechanisms in Real-Time Desktop Conferencing Systems," *Proceedings of the Conference on Office Information Systems*, pp. 238-248, April 1990.

Brad68 Brady, P.T. "A Statistical Analysis of On-off Patterns In Sixteen Conversations," *Bell System Technical Journal* Vol. 47, January, 1968.

DeTr83 DeTreville, J. and Sincoskie, W.D. "A Distributed Experimental Communication System," *IEEE Journal On Selected Areas in Communications*, Vol. SAC-1, No. 6 December 1983.

DEC80 DEC, Intel and Xerox, *The Ethernet: A Local Area Network, Data Link and Physical Layer Specification*, Version 1.0, September, 1980.

Enso88 Ensor, J.R., Ahuja, S.R., Horn, D.N. and Lucco, S.E., "The Rapport Multimedia Conferencing System - A Software Overview," *Proceedings of IEEE 2nd Conference on Computer Workstations*, pp. 52 - 58, Mar. 1988.

Forg80 Forgie, J.W., "Voice Conferencing in Packet Networks," *Conference Record of the International Conference on Communications*, June 1980.

Fors85 Forsdick, H., "Explorations into Real-Time Multimedia Conferencing," *Proceedings of The 2nd International Symposium in Computer Message Systems*, pp. 61-75, Sept. 1985.

Frie86 Friedman, E. and Ziegler, C., "Real-time Voice Communications Over a Token-Passing Ring Local Area Network," *Conference Proceedings of SIGCOM'86 Symposium*, August 1986, pp. 52-57.

Frie87 Friedman, E. and Ziegler, C., "An Initial Design of a Distributed Packet Voice Communications Protocol," *Conference Proceedings of the Sixth Annual Phoenix Conference on Computers and Communications*, February 1987, pp. 412-416.

Frie89 Friedman, E. and Ziegler, C., "Packet Voice Communications Over PC-Based Local Area Networks," *IEEE Journal On Selected Areas in Communications*, Vol. 7, No. 2, pp. 211-218, February 1989.

Garc85 Garcia-Luna, J. "Towards Computer-Based Multimedia Information Systems," *Proceedings of The 2nd International Symposium in Computer Message Systems*, pp. 61-75, Sept. 1985.

Hals88 Halsall, F., *Data Communications, Computer Networks and OSI*, Second Edition, Addison-Wesley, New York, 1988.

Hobe83 Hoberecht, W.L., "A Layered Network Protocol for Packet Voice and Data Integration" *IEEE Journal On Selected Areas in Communications*, Vol.SAC-1, No. 6 December 1983.

IEEE85 Institute of Electrical and Electronics Engineers, *ANSI/IEEE Standards for LANs: Token-Passing Bus Access Method and Physical Layer Specifications*, Wiley-Interscience, 1985.

ISOa International Standard Organization, *Network Service definition*, DIS 8348.

ISOb International Standard Organization, *Addendum to the Network Service Definition Covering Connectionless Mode Transmission*, DIS 8348/DAD1.

ISO82a International Standard Organization, *Transport Protocol Services*, ISO/TC 97/SC 16 N1162, June 1982.

ISO82b International Standard Organization, *Transport Protocol Specification*, ISO/TC 97/SC 16 N1169, June 1982.

ISO84a International Standard Organization, *Addendum to the Transport Service Definition Covering Connectionless Mode Transmission*, ISO/TC 97/SC 16 N2008, October 1984.

ISO84b International Standard Organization, *Protocol for Providing the Connectionless-Mode Transport Service Utilizing the Connectionless-Mode Network Service or the Connection Oriented Network Services*, ISO/TC 97/SC 16 N2008, October 1984.

John81 Johnson, D. H. and O'leary, G. C., "A Local Access Network for Packetized Digital Voice Communication," *IEEE Transactions on Communications*, Vol.Com-29, No. 5, May 1981.

Koba78 Kobayashi, H. *Modeling and Analysis: An Introduction to System Performance Methodology*, Addison Wesley, New York, 1978.

Kosi89 Kositpaiboon, R., Tsingotiidis, P., Barbosa, L.O., and Georganas, N.D., "Packetized Radiographic Image Transfers over Local Area Networks for Diagnostics and Conferencing," *IEEE Journal on Selected Areas in Communications*, pp. 842-855, Vol. 7, No. 5, Jun. 1989.

Leun89 Leung, W.H., Morgan, L.F., Morgan, M.J. and Wong, B.F., "The Connector and Active Devices Mechanisms for Constructing Multimedia Applications," *Proceedings of the 2nd Workshop on Workstation Operating Systems*, pp. 68-72, 1989.

Leun90 Leung, W.F., Baumgartner, T.J., Hwang, Y.H., Morgan, M.J. and TU, S, "A Software Architecture for Workstation Supporting Multimedia Conferencing in Packet Switching Networks," *IEEE Journal on Selected Areas in Communications*, pp. 380-390, Vol. 8, No. 3, Apr. 1990.

Limb83 Limb, J.O. and Flamm, L.E., "A Distributed Local Area Network Packet Protocol for Combined Voice and Data Transmission," *IEEE Journal on Selected Areas in Communication*, Vol. SAC-1 No. 5, November 1983.

Litt61 Little, J. D. C., "A Proof of the Queuing Formula $L^ = ^lambda W$," *Operations Research*, Vol. 9, No. 3, pp. 383-387, 1961.

Metc76 Metcalfe, R. M. and Boggs, D. R., "Ethernet: Distributed Packet Switching for Local Computer Networks," *Communications of the ACM*, Vol. 19, No. 7, July 1976.

Muss83 Musser, J.M., Liu, T.T., Li L. and Boggs G.J. "A Local Area Network as a Telephone Local Subscriber Loop," *IEEE Journal On Selected Areas in Communications*, Vol. SAC-1, No. 6, December 1983.

Prot84 Proteon proNET, *IBM PC Local Network System User's Guide*, Draft 2, June 1984.

Saka88 Sakata, S. and Udea, T., "Multiparty Desktop Conference System Based on Integrated Group Communication Architecture," *Proceedings of 1988 International Zurich Seminar on Digital Communications*, pp. A2.1-A2.7, 1988.

Saka90 Sakata, S., "Development and Evaluation of an In-House Multimedia Desktop Conference System," *IEEE Journal on Selected Areas in Communications*, pp. 380-390, Vol. 8, No. 3, Apr. 1990.

Sari85 Sarin, S. and Greif, I., "Computer-Based Real-Time Conferencing Systems," *IEEE Computer*, pp. 33-45, Vol. 18, No. 10, Oct. 1985.

Stal90 Stallings, W., *Local Networks*, Third Edition, Macmillan, New York, 1990.

Stal91 Stallings, W., *Data and Computer Communications*, Third Edition, Macmillan, New York, 1991.

Swin83 Swinehart, D.C., "Telephone Management in the Etherphone System," *Conference Record of IEEE Globecom '87*, pp. 30.3.1-30.3.5, Nov. 1987.

Swin83 Swinehart, D.C., Stewart, L.C. and Ornstein, S. M., "Adding Voice to an Office Computer Network," *Conference Record of IEEE Globecom '83*, pp. 392-398, Nov. 1983.

Tane88 Tanenbaum, A. S., *Computer Networks*, Second Edition, Prentice Hall, New Jersey, 1988.

Tani88 Tanigawa, H., Hayashi, Y. and Matsumoto, M., "Multipoint Communication Control For Document-Oriented Teleconferencing", *Proceedings of the 1988 International Zurich Seminar on Digital Communications*, pp. A3.1-A3.7, 1988.

3Com84 3Com Corporation, *IE Ethernet Controller/Transceiver External Reference Specification*, March 1984.

Toba82 Tobagi, F. A., "Distributions of Packet Delay and Interdeparture Time in Slotted Aloha and Carrier Sense Multiple Access," *Journal of the ACM*, Vol. 29 No. 4, October 1982.

Toba83 Tobagi, F.A., Flaminio, B. and Frata, L., "Expressnet: A High-performance Integrated-services Local Area Network," *IEEE Journal on Selected Areas in Communication*, Vol. SAC-1 No. 5, November 1983.

Toba83 Tobagi, F.A. and Fine M., "Performance of Unidirectional Broadcast Local Area Networks: Expressnet and Fasnet," *IEEE Journal on Selected Areas in Communication*, Vol. SAC-1 No. 5, November 1983.

Toma87 Tomas, J.G., Pavon, J. and Pereda, O., "OSI Service Specification: SAP and CEP Modelling," *Computer Communications Review*, pp. 48-70, Vol. 17, No. 1 and 2, Jan./April 1987.

Wein83 Weinstein, C.J. and Forgie, J.W., "Experience with Speech Communication in Packet Networks," *IEEE Journal On Selected Areas in Communications*, Vol. SAC-1, No. 6 December 1983.

Weis88 Weiss, G. and Ziegler, C., "Packet-Switched Voice Conferencing Across Interconnected Networks," *Proceedings of the 13th Conference on Local Computer Networks*, pp. 114-124, October 1988.

Weis90 Weiss, G. and Ziegler, C., "A Comparative Analysis of Implementation Mechanisms for Packet Voice Conferencing," *Proceedings of Infocom '90*, pp. 1,062-1070, June 1990.

Zell88 Zellweger, P.T., Terry, D.B. and Swinhchart, D.C., "An Overview of the Etherphone System and Its Applications," *Proceeding of the 2nd International Conference on Computer Workstations*, pp. 160 - 168, 1988.

Zeig79 Ziegler, C., Dhadesugoor, V.R., and Schilling, D.L., "Digital Silence Detection in Delta Modulation Packet Voice Networks," *Conference Record of ICC '79*, pp. 24.7.1-24.7.5, June 1979.

Zieg79 Ziegler, C., Dhadesugoor, V.R., and Schilling, D.L., "Source Encoding Using Delta Modulation in Packet Voice Networks," *Conference Record of NTC '79*, pp. 4.4.1-4.4.5, Nov. 1979.

Zeig79 Ziegler, C., Dhadesugoor, V.R., Chakraverthy, C.B., and Schilling,D.L., "Delta Modulation Packet Voice Network with a Variable Packet Size Algorithm," *Conference Record of NTC '79*, pp. 13.2.1-13.2.5, Nov. 1979.

Zieg80 Ziegler, C., Dhadesugoor, V.R., and Schilling, D.L., "Delta Modulators in Packet Voice Networks," *IEEE Transaction on Communications*, vol. COM-28, pp. 33-51, Jan. 1980.

Zieg89 Ziegler, C., Weiss, G. and Friedman, E., "Implementation Mechanisms for Packet Switched Voice Conferencing," *IEEE Journal On Selected Areas in Communications*, Vol. 7, No. 5, pp. 698-706, June 1989.

Zieg90 Ziegler, C. and Weiss G., "Multimedia Conferencing on Local Area Networks," *IEEE Computer* pp. 52-61, Vol. 23, No. 9, Sept. 1990.

Zieg91 Ziegler, C. and Weiss G., "A Hierarchial Architecture for Multimedia Conferencing on LANs," *Proceedings of the Fourth ISMM/IASTED International Conference on Parallel and Distributed Computing and Systems*, pp. 276-280, Oct. 1991.