ANSI/IEEE Std 802.1D, 1998 Edition

[Incorporates ANSI/IEEE Std 802.1D, 1993 Edition, IEEE P802.1p,
IEEE Std 802.1j-1996, IEEE Std 802.6k-1992,
IEEE Std 802.11c-1998, and IEEE P802.12e]

(Adopted by ISO/IEC and redesignated as
ISO/IEC 15802-3:1998)

IEEE Standard for Information technology—

Telecommunications and information exchange between systems—

Local and metropolitan area networks—

Common specifications

Part 3: Media Access Control (MAC) Bridges

**Adopted by the ISO/IEC and redesignated as
ISO/IEC 15802-3:1998**

Sponsor

LAN/MAN Standards Committee
of the
IEEE Computer Society

ANSI/IEEE Std 802.1D, 1998 Edition

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

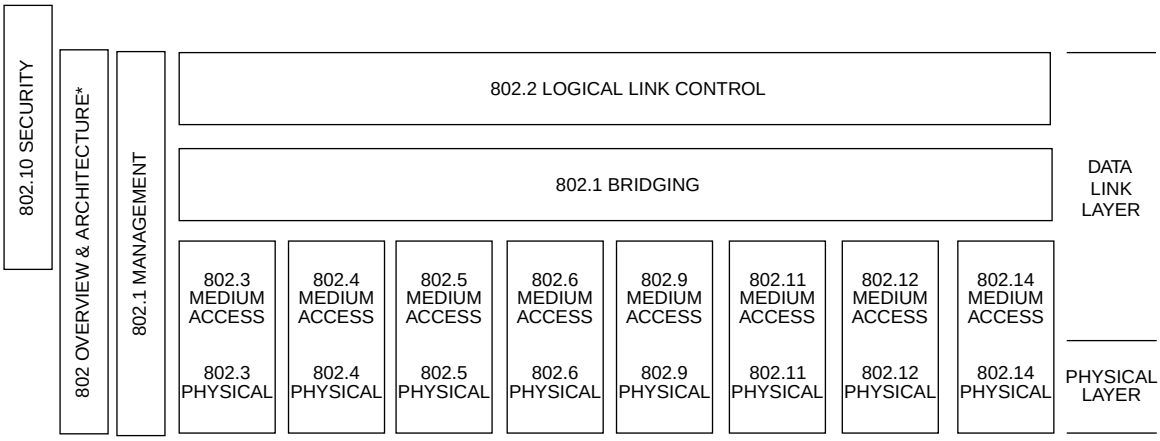
Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention. A patent holder has filed a statement of assurance that it will grant licenses under these rights without compensation or under reasonable rates and nondiscriminatory, reasonable terms and conditions to all applicants desiring to obtain such licenses. The IEEE makes no representation as to the reasonableness of rates and/or terms and conditions of the license agreements offered by patent holders. Further information may be obtained from the IEEE Standards Department.

Introduction to ANSI/IEEE Std 802.1D, 1998 Edition

[This introduction is not part of ANSI/IEEE Std 802.1D, 1998 Edition, Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Common specifications—Part 3: Media Access Control (MAC) Bridges.]

This standard is part of a family of standards for local and metropolitan area networks. The relationship between the standard and other members of the family is shown below. (The numbers in the figure refer to IEEE standard numbers.)



* Formerly IEEE Std 802.1A.

This family of standards deals with the Physical and Data Link layers as defined by the International Organization for Standardization (ISO) Open Systems Interconnection (OSI) Basic Reference Model (ISO/IEC 7498-1 : 1994). The access standards define seven types of medium access technologies and associated physical media, each appropriate for particular applications or system objectives. Other types are under investigation.

The standards defining the technologies noted above are as follows:

- IEEE Std 802 — *Overview and Architecture.* This standard provides an overview to the family of IEEE 802 Standards.
- ANSI/IEEE Std 802.1B and 802.1k [ISO/IEC 15802-2] — *LAN/MAN Management.* Defines an OSI management-compatible architecture, and services and protocol elements for use in a LAN/MAN environment for performing remote management.
- ANSI/IEEE Std 802.1D [ISO/IEC 15802-3] — *Media Access Control (MAC) Bridges.* Specifies an architecture and protocol for the interconnection of IEEE 802 LANs below the MAC service boundary.
- ANSI/IEEE Std 802.1E [ISO/IEC 15802-4] — *System Load Protocol.* Specifies a set of services and protocol for those aspects of management concerned with the loading of systems on IEEE 802 LANs.
- ANSI/IEEE Std 802.1F — *Common Definitions and Procedures for IEEE 802 Management Information*
- ANSI/IEEE Std 802.1G [ISO/IEC 15802-5] — *Remote Media Access Control (MAC) bridging.* Specifies extensions for the interconnection, using non-LAN communication technologies, of geographically separated IEEE 802 LANs below the level of the logical link control protocol.
- ANSI/IEEE Std 802.2 [ISO/IEC 8802-2] — *Logical link control*

- ANSI/IEEE Std 802.3 [ISO/IEC 8802-3] — *CSMA/CD access method and physical layer specifications*
- ANSI/IEEE Std 802.4 [ISO/IEC 8802-4] — *Token passing bus access method and physical layer specifications*
- ANSI/IEEE Std 802.5 [ISO/IEC 8802-5] — *Token ring access method and physical layer specifications*
- ANSI/IEEE Std 802.6 [ISO/IEC 8802-6] — *Distributed Queue Dual Bus (DQDB) access method and physical layer specifications*
- ANSI/IEEE Std 802.9 [ISO/IEC 8802-9] — *Integrated Services (IS) LAN Interface at the Medium Access Control (MAC) and Physical (PHY) Layers*
- ANSI/IEEE Std 802.10 — *Interoperable LAN/MAN Security*
- ANSI/IEEE Std 802.11 [ISO/IEC DIS 8802-11] — *Wireless LAN Medium Access Control (MAC) and physical layer specifications*
- ANSI/IEEE Std 802.12 [ISO/IEC 8802-12] — *Demand-priority access method, physical layer and repeater specifications*

In addition to the family of standards, the following is a recommended practice for a common Physical Layer technology:

- IEEE Std 802.7 — *IEEE Recommended Practice for Broadband Local Area Networks*

The following additional working group has authorized standards projects under development:

- IEEE 802.14 — *Standard Protocol for Cable-TV Based Broadband Communication Network*

Conformance test methodology

An additional standards series, identified by the number 1802, has been established to identify the conformance test methodology documents for the 802 family of standards. Thus the conformance test documents for 802.3 are numbered 1802.3.

ANSI/IEEE Std 802.1D, 1998 Edition

The MAC Bridge standardization activities that resulted in the development of IEEE Std 802.1D-1990 (subsequently republished as ISO/IEC 10038:1993 [IEEE Std 802.1D, 1993 Edition]) specified an architecture and protocol for the interconnection of IEEE 802 LANs below the MAC Service boundary. IEEE Std 802.1D-1990 also introduced the concept of filtering services in Bridged LANs, and mechanisms whereby filtering information in such LANs may be acquired and held in a Filtering Database. This revision of ISO/IEC 10038: 1993 extends this concept of filtering services in order to define additional capabilities in Bridged LANs aimed at the following:

a) The provision of expedited traffic capabilities, to support the transmission of time-critical information in a LAN environment;
b) The provision of filtering services that support the dynamic definition and establishment of Groups in a LAN environment, and the filtering of frames by Bridges such that frames addressed to a given

Group are forwarded only on those LAN segments that are required in order to reach the members of that Group.

To this end, this document incorporates a set of changes and additions to ISO/IEC 10038: 1993 that define the following:

a) The nature of Filtering Services in Bridged LANs;
b) The concept of Traffic Classes and the effect on the operation of the Forwarding Process of supporting multiple Traffic Classes in Bridges;
c) The structure of the Filtering Database that is needed in order to support Dynamic Multicast Filtering services;
d) The registration protocol that is required in order to provide Dynamic Multicast Filtering Services;
e) The management services and operations that are required in order to support administration of Dynamic Multicast Filtering Services.

Relationship between IEEE Std 802.1D and IEEE P802.1Q

A further IEEE standard under development, IEEE P802.1Q, extends the concepts of filtering services and MAC Bridging in order to provide a set of capabilities that allow MAC Bridges to support the definition and management of Virtual LANs (VLANs).

The capabilities defined in IEEE P802.1Q include the definition of a VLAN frame format that is able to carry VLAN identification and user priority information over LAN technologies, such as CSMA/CD, that have no inherent capability to signal priority information. This information is carried in an additional header field, known as the *Tag Header*, which is inserted immediately following the Destination MAC Address, and Source MAC Address (and Routing Information field, if present) of the original frame. IEEE P802.1Q extends the priority handling aspects of this standard to make use of the ability of the VLAN frame format to carry user priority information end to end across any set of concatenated underlying MACs.

The VLAN Bridging specification contained in IEEE 802.1Q is independent of this standard, in the sense that IEEE 802.1Q makes a separate and distinct statement of the conformance requirements for VLAN Bridges from the conformance requirements for MAC Bridges defined in this standard. However, IEEE 802.1Q makes use of many of the elements of the specification contained in this standard, in particular

a) The Bridge architecture;
b) The Internal Sublayer Service, and the specification of its provision by IEEE 802 LAN MACs;
c) The major features of the operation of the Forwarding Process;
d) The Spanning Tree Algorithm and Protocol;
e) The Generic Attribute Registration Protocol (GARP); and
f) The GARP Multicast Registration Protocol (GMRP).

Participants

The following is a list of participants in the Interworking activities of the IEEE 802.1 Working Group. Voting members at the time of publication are marked with an asterisk (*).

William P. Lidinsky, *Chair**
Mick Seaman, *Chair, Interworking Task Group**
Tony Jeffree*, *Editor*

Steve Adams*
Stephen Ades
Ken Alonge
Floyd Backes*
John Bartlett*
Les Bell*
Avner Ben-Dor
Michael Berger*
James S. Binder*
David Brady
Martin Brewer
Bill Bunch*
Bob Cardinal
Paul Carroll*
Jeffrey Catlin*
Dennis Cave
Alan Chambers*
Steve Chan
David W. Chang*
Ken Chapman
Hon Wah Chin*
Chi Chong
Chris Christ*
Paul Congdon*
Glenn Connery*
David Cullerot*
Ted Davies*
Andy Davis
David Delaney*
Prakash Desai
Jeffrey Dietz*
Kurt Dobbins
Peter Ecclesine*
J. J. Ekstrom*
Norman W. Finn*
Yishai Fraenkel
Paul Frantz
Lars Henrik Frederiksen*
Anoop Ghanwani*
John Grinham
Steve Haddock
Sharam Hakimi*
John Hart*
Scott Harvell
Wayne Hathaway
Richard Hausman*
Vic Hayes
David Head*
Gaby Hecht
Deepak Hegde*
Ariel Hendel
John Hickey
David Hollender
Steve Horowitz*
Michelle Hsiung
Rita Hunt
David Husak
Altaf Hussain*
Vipin K. Jain*
Neil Jarvis
Tony Jeffree*
Allen Kasey
Toyoyuki Kato*
Hal Keen*
Kevin Ketchum*
Keith Klamm*
Bruce Kling*
Walter Knitl
Dan Krent*
Paul Kummer
Paul Lachapelle*
Bill Lane
Paul Langille*
Bill Lidinsky*
Johann Lindmeyr*
Gary Littleton
Robert D. Love
Andy Luque
Peter Martini
Keith McCloghrie
Martin McNealis
Milan Merhar*
John Messenger*
Colin Mick
Amol Mitra
Yaron Nachman*
Krishna Narayanaswamy*
Paul Nikolich
Lawrence Ng*
Henry Ngai*
Eugene O'Neil
Satoshi Obara*
Toshio Ooka*
Jorg Ottensmeyer*
Luc Pariseau*
Yonadav Perry
John Pickens*
Gideon Prat
Kirk Preiss
Steve Ramberg*
Shlomo Reches*
Dick Reohr
James Richmond*
Anil Rijsinghani*
Doug Ruby
Ray Samora
Ayman Sayed*
Mick Seaman*
Rich Seifert
Lee Sendelbach*
Himanshu Shah*
Phil Simmons*
K. Karl Shimada
Fred Shu
Paramjeet Singh*
Rosemary V. Slager*
Alexander Smith*
Andrew Smith*
Larry Stefani*
Stuart Soloway*
Sundar Subramaniam*
Richard Sweatt
Robin Tasker*
Fouad Tobagi
Naoki Tsukutari
Dhadesugoor Vaman
Steve Van Seters*
Dono van-Mierop*
John Wakerly*
Peter Wang*
Philip Wang
Y. C. Wang*
Trevor Warwick*
Bob Watson
Alan Weissberger
Glenn Wenig
Keith Willette*
Michael Witkowski*
Edward Wong*
Michael D. Wright*
Michele Wright*
Allen Yu*
Wayne Zakowski*

The following persons were on the balloting committee of IEEE Std 802.1D:

William B. Adams
Kit Athul
William E. Ayen
Thomas W. Bailey
Brad J. Booth
Peter K. Campbell
James T. Carlo
David E. Carlson
Alan M. Chambers
Frederick N. Chase
Robert S. Crowder
Thomas J. Dineen
Peter Ecclesine
John W. Fendrich
Michael A. Fischer
Harvey A. Freeman
Patrick S. Gonia
Julio Gonzalez-Sanz
Robert M. Grow
Chris G. Guy
Stephen R. Haddock
Allen W. Hathaway
J. Scott Haugdahl
Kenneth C. Heck
Henry Hoyt
Raj Jain
Neil A. Jarvis
Tony Jeffree
Edward R. Kelly
Peter M. Kelly
Yongbum Kim
Thaddeus Kobylarz
Daniel R. Krent
Stephen Barton Kruger
Kenneth C. Kung
William G. Lane
David J. Law
Lanse M. Leach
William Lidinsky
Randolph S. Little
Joseph G. Maley
Peter Martini
Chris McDonald
Milan Merhar
John L. Messenger
Bennett Meyer
Colin K. Mick
Gene E. Milligan
David S. Millman
Warren Monroe
John E. Montague
Wayne D. Moyers
Shimon Muller
Paul Nikolich
Robert O'Hara
Charles Oestereicher
Joerg Ottensmeyer
Roger Pandanda
Lucy W. Person
John R. Pickens
Vikram Punj
Andris Putnins
Edouard Y. Rocher
James W. Romlein
Floyd E. Ross
Christoph Ruland
Norman Schneidewind
Mick Seaman
Rich Seifert
Michael A. Smith
William R. Smith
Patricia Thaler
Geoffrey O. Thompson
Mark-Rene Uchida
John Viaplana
Barry M. Vornbrock
Donald F. Weir
Earl J. Whitaker
Qian-li Yang
Oren Yuen
Jonathan M. Zweig

When the IEEE-SA Standards Board approved IEEE Std 802.1D on 25 June 1998, it had the following membership:

Richard J. Holleman, *Chair* **Donald N. Heirman,** *Vice Chair*
Judith Gorman, *Secretary*

Satish K. Aggarwal
Clyde R. Camp
James T. Carlo
Gary R. Engmann
Harold E. Epstein
Jay Forster*
Thomas F. Garrity
Ruben D. Garzon
James H. Gurney
Jim D. Isaak
Lowell G. Johnson
Robert Kennelly
E. G. "Al" Kiener
Joseph L. Koepfinger*
Stephen R. Lambert
Jim Logothetis
Donald C. Loughry
L. Bruce McClung
Louis-François Pau
Ronald C. Petersen
Gerald H. Peterson
John B. Posey
Gary S. Robinson
Hans E. Weinrich
Donald W. Zipse

*Member Emeritus

Kristin M. Dittmann
IEEE Standards Project Editor

IEEE Std 802.11c-1998

IEEE Std 802.11c-1998 adds the necessary information to map the IEEE 802.11 MAC parameters onto ISO/IEC 15802-3 (IEEE Std 802.1D) parameters.

Participants

At the time the draft of IEEE Std 802.11c was sent to sponsor ballot, the IEEE 802.11 working group had the following voting members:

Victor Hayes, *Chair*
Victoria M. Poncini, *Task Group Chair*

Jeff Abramowitz
Keith B. Amundsen
Carl F. Andren
Kazuhiro Aoyagi
David Bagby
Phil Belanger
John Biddick
Simon Black
Jan Boer
Ronald Brockmann
Wesley Brodsky
John H. Cafarella
Naftali Chayat
Ken Clements
Wim Diepstraten
Darrol Draper
Peter Ecclesine
Darwin Engwer
John Fakatselis
Jeff Fischer
Matthew Fischer
Michael Fischer
George Fishel
John Fisher
Motohiro Gochi
Tim Godfrey
Jan Haagh
Karl Hannestad
Robert Heile
Maarten Hoeben
Duane Hurne
Masayuki Ikeda
Richard Jai
Donald C. Johnson
Nobuo Karaki
Dean M. Kawaguchi
Stuart J. Kerry
Isao Masaki
Jim McDonald
Gene Miller
Akira Miura
Masaharu Mori
Masahiro Morikura
Ravi P. Nalamati
Colin Nayler
Richard van Nee
Bob O'Hara
Tomoki Ohsawa
Kazuhiro Okanoue
Richard H. Paine
Al Petrick
Bob Pham
Stanley A. Reible
William Roberts
Kent G. Rollins
Oren Rosenfeld
Michael Rothenberg
Clemens C.W. Ruppel
Chandos Rypinski
Anil K. Sanwalka
Roy Sebring
Mike Shiba
Thomas Siep
Donald I. Sloan
Hitoshi Takanashi
Satoru Toguchi
Cherry Tom
Mike Trompower
Tom Tsoulogiannis
Sarosh Vesuna
Nien C. Wei
Harry Worstell
Timothy M. Zimmerman
Johnny Zweig
Jim Zyren

Major contributions were received from Henri Moelard.

The following persons were on the balloting committee of 802.11c:

Kit Athul
Thomas W. Bailey
Peter K. Campbell
James T. Carlo
David E. Carlson
Brian J. Casey
Naftali Chayat
Robert S. Crowder
Wim Diepstraten
Thomas J. Dineen
Christos Douligeris
Paul S. Eastman
John E. Emrich
Philip H. Enslow
Changxin Fan
John W. Fendrich
Michael A. Fischer
Harvey A. Freeman
Robert J. Gagliano
Gautam Garai
Patrick S. Gonia
Julio Gonzalez-Sanz
Chris G. Guy
J. Scott Haugdahl
Vic Hayes
Donald N. Heirman
Henry Hoyt
Raj Jain
Tony Jeffree
A. Kamerman
Dean M. Kawaguchi
Edward R. Kelly
Peter M. Kelly
Gary C. Kessler
Yongbum Kim
Stephen Barton Kruger
Joseph Kubler
Jan Kurys
Lanse M. Leach
Jai Yong Lee
Randolph S. Little
Ronald Mahany
Peter Martini
Bennett Meyer
Gene E. Milligan
David S. Millman
Warren Monroe
John E. Montague
Wayne D. Moyers
Shimon Muller
Ken Naganuma
Paul Nikolich
Robert O'Hara
Donal O'Mahony
Charles Oestereicher
Young Oh
John M. Osepchuk
Roger Pandanda
Ronald C. Petersen
John R. Pickens
Alberto Profumo
Vikram Punj
Fernando Ramos
James A. Renfro
Everett Rigsbee III
Edouard Y. Rocher
James W. Romlein
Floyd E. Ross
Michael Rothenberg
Christoph Ruland
Anil K. Sanwalka
Norman Schneidewind
James E. Schuessler
Rich Seifert
Leo Sintonen
Patricia Thaler
Mike Trompower
Mark-Rene Uchida
Sarosh N. Vesuna
John Viaplana
James Vorhies
Barry M. Vornbrock
Qian-li Yang
Oren Yuen
Chris Zegelin
Jonathan M. Zweig

When the IEEE-SA Standards Board approved IEEE Std 802.11c on 16 September 1998, it had the following membership:

Richard J. Holleman, *Chair* **Donald N. Heirman,** *Vice Chair*

Judith Gorman, *Secretary*

Satish K. Aggarwal
Clyde R. Camp
James T. Carlo
Gary R. Engmann
Harold E. Epstein
Jay Forster*
Thomas F. Garrity
Ruben D. Garzon
James H. Gurney
Jim D. Isaak
Lowell G. Johnson
Robert Kennelly
E. G. "Al" Kiener
Joseph L. Koepfinger*
Stephen R. Lambert
Jim Logothetis
Donald C. Loughry
L. Bruce McClung
Louis-François Pau
Ronald C. Petersen
Gerald H. Peterson
John B. Posey
Gary S. Robinson
Hans E. Weinrich
Donald W. Zipse

*Member Emeritus

Kristin M. Dittmann
IEEE Standards Project Editor

Abstract: The concept of Media Access Control (MAC) Bridging, introduced in the 1993 edition of this standard, has been expanded to define additional capabilities in Bridged LANs aimed at providing for expedited traffic capabilities, to support the transmission of time-critical information in a LAN environment; and providing filtering services that support the dynamic use of Group MAC Addresses in a LAN environment.
Keywords: local area networks, MAC Bridge management, MAC bridges, media access control (MAC) bridges, multicast address filtering, traffic class expediting

---

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

Print: ISBN 0-7381-0329-2 SH94651
PDF: ISBN 0-7381-1416-2 SS94651

Contents

Figures

Tables

Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Common specifications—

Part 3: Media Access Control (MAC) Bridges

1. Overview

1.1 Introduction

IEEE 802 Local Area Networks (or LANs; see 3.4) of all types can be connected together using MAC Bridges. Each individual LAN has its own independent MAC. The Bridged LAN created allows the interconnection of stations attached to separate LANs as if they were attached to a single LAN, although they are in fact attached to separate LANs each with its own MAC. A MAC Bridge operates below the MAC Service Boundary, and is transparent to protocols operating above this boundary, in the Logical Link Control (LLC) sublayer or Network Layer (ISO/IEC 7498-1: 1994[1]). The presence of one or more MAC Bridges can lead to differences in the Quality of Service provided by the MAC sublayer; it is only because of such differences that MAC Bridge operation is not fully transparent.

A Bridged LAN can provide for

a) The interconnection of stations attached to LANs of different MAC types;
b) An effective increase in the physical extent, the number of permissible attachments, or the total performance of a LAN;
c) Partitioning of the physical LAN for administrative or maintenance reasons.

NOTE—Scope, definitions, references, and conformance requirements relating to the operation of Source-Routing Transparent Bridge operation can be found in Annex C.1.

1.2 Scope

For the purpose of compatible interconnection of data processing equipment using the IEEE 802 MAC Service supported by interconnected IEEE 802 LANs (see 3.4) using different or identical Media Access Control methods, this standard specifies a general method for the operation of MAC Bridges. To this end it

a) Positions the bridging function within an architectural description of the MAC Sublayer.
b) Defines the principles of operation of the MAC Bridge in terms of the support and preservation of the MAC Service, and the maintenance of Quality of Service.
c) Specifies the MAC Internal Sublayer Service provided by individual LANs to the Media Access Method Independent Functions that provide frame relay in the Bridge.
d) Identifies the functions to be performed by Bridges, and provides an architectural model of the internal operation of a Bridge in terms of Processes and Entities that provide those functions.
e) Establishes the requirements for a protocol between the Bridges in a Bridged LAN to configure the network, and specifies the distributed computation of a Spanning Tree active topology.
f) Specifies the encoding of the Bridge Protocol Data Units (BPDUs).
g) Establishes the requirements for Bridge Management in the Bridged LAN, identifying the managed objects and defining the management operations.

[1]Information about references can be found in Clause 2.

h) Specifies how the management operations are made available to a remote manager using the protocol and architectural description provided by ISO/IEC 15802-2: 1995.
i) Specifies performance requirements and recommends default values and applicable ranges for the operational parameters of a Bridge.
j) Specifies the requirements to be satisfied by equipment claiming conformance to this standard.
k) Specifies criteria for the use of MAC-specific bridging methods.

This standard specifies the operation of MAC Bridges that attach directly to IEEE 802 LANs, as specified in the relevant MAC standards for the MAC technology or technologies implemented.

The specification of Remote Bridges, which interconnect LANs using Wide Area Network (WAN) media for the transmission of frames between Bridges, is outside the scope of this standard.

NOTE—Remote MAC Bridging is specified in ISO/IEC 15802-5: 1997 [ANSI/IEEE Std 802.1G, 1997 Edition].

2. References

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 15802. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 15802 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of ISO and IEC maintain registers of currently valid International Standards.

ANSI X3.159-1989, American National Standards for Information Systems—Programming Language—C.[2]

IEEE Std 802-1990, IEEE Standards for Local and Metropolitan Area Networks: Overview and Architecture.[3]

IEEE Std 802.1F-1993, IEEE Standards for Local and Metropolitan Area Networks: Common Definitions and Procedures for IEEE 802 Management Information.

IEEE Std 802.3, 1998 Edition, Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications.

IEEE Std 802.9a-1995, IEEE Standards for Local and Metropolitan Area Networks: Supplement to Integrated Services (IS) LAN Interface at the Medium Access Control (MAC) and Physical (PHY) Layers: Specification of ISLAN 16-T.

IEEE Std 802.11-1997, Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications.[4]

IETF INTERNET-DRAFT, Internet Group Management Protocol (IGMP), Version 2, January 20th 1997[5].

IETF RFC 1493, Decker, Langille, Rijsinghani and McCloughrie, Definitions of Managed Objects for Bridges, July 1993[6].

ISO 6937-2: 1983, Information processing—Coded character sets for text communication—Part 2: Latin alphabetic and non-alphabetic graphic characters.[7]

ISO/IEC 7498-1: 1994, Information processing systems—Open Systems Interconnection—Basic Reference Model—Part 1: The Basic Model.

---

[2]ANSI publications are available from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA.

[3]IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

[4]A draft International Standard (ISO/IEC DIS 8802-11) is under way for this standard. For information about the status of this DIS, contact the ISO Central Secretariat, 1 rue de Varembé, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse; or the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA.

[5]Internet Drafts are retrievable at the IETF Web site, http://www.ietf.cnri.reston.va.us/home.html, or call InterNIC at 1-800-444-4345 for information about receiving copies through the mail.

[6]Internet RFCs are retrievable by FTP at ds.internic.net/rfc/rfcnnnn.txt (where nnnn is a standard's publication number such as 1493), or call InterNIC at 1-800-444-4345 for information about receiving copies through the mail.

[7]ISO and ISO/IEC documents are available from the ISO Central Secretariat, 1 rue de Varembé, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse; and from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA.

ISO/IEC 8802-2: 1998 [ANSI/IEEE Std 802.2, 1998 Edition], Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 2: Logical link control.[8]

ISO/IEC 8802-4: 1990 [ANSI/IEEE Std 802.4-1990], Information processing systems—Local area networks—Part 4: Token-passing bus access method and physical layer specifications.

ISO/IEC 8802-5: 1998 [ANSI/IEEE Std 802.5, 1998 Edition], Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 5: Token ring access method and physical layer specifications.

ISO/IEC 8802-6: 1994 [ANSI/IEEE Std 802.6, 1994 Edition], Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 6: Distributed Queue Dual Bus (DQDB) access method and physical layer specifications.

ISO/IEC 8802-9: 1996 [ANSI/IEEE Std 802.9, 1996 Edition], Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 9: Integrated Services (IS) LAN Interface at the Medium Access Control (MAC) and Physical (PHY) Layers.

ISO/IEC 8802-12: 1998 [ANSI/IEEE Std 802.12, 1998 Edition], Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 12: Demand-Priority access method, physical layer and repeater specifications.

ISO/IEC 8824: 1990, Information technology—Open Systems Interconnection—Specification of Abstract Syntax Notation One (ASN.1) (Provisionally retained edition).

ISO/IEC 8825: 1990, Information technology—Open Systems Interconnection—Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1) (Provisionally retained edition).

ISO/IEC 9595: 1998, Information technology—Open Systems Interconnection—Common management information service.

ISO 9314-2: 1989, Information processing systems—Fibre Distributed Data Interface—Part 2: FDDI Token Ring Media Access Control (MAC).

ISO/IEC 9596-1: 1991, Information technology—Open Systems Interconnection—Common management information protocol—Part 1: Specification.

ISO/IEC TR 11802-2: 1997, Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Technical reports and guidelines—Part 2: Standard Group MAC addresses.

ISO/IEC 11802-5: 1997 [ANSI/IEEE Std 802.1H, 1997 Edition], Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Technical reports and guidelines—Part 5: Media Access Control (MAC) Bridging of Ethernet V2.0 in Local Area Networks.

---

[8]ISO [IEEE] and ISO/IEC [IEEE] documents are available from ISO Central Secretariat, 1 rue de Varembé, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse; and from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

ISO/IEC 15802-1: 1995, Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Common specifications—Part 1: Medium Access Control (MAC) service definition.

ISO/IEC 15802-2: 1995 [ANSI/IEEE Std 802.1B, 1995 Edition], Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Common specifications—Part 2: LAN/MAN management.

ISO/IEC 15802-5: 1998 [ANSI/IEEE Std 802.1G, 1998 Edition], Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Common specifications—Part 5: Remote Medium Access Control (MAC) bridging.

3. Definitions

For the purposes of this standard, the following terms and definitions apply:

3.1 Bridged Local Area Network

A concatenation of individual IEEE 802 LANs interconnected by MAC Bridges.

3.2 Expedited traffic

Traffic that requires preferential treatment as a consequence of jitter, latency, or throughput constraints, or as a consequence of management policy.

3.3 Group

A Group associates

a) A group MAC address; and
b) A set of properties that define membership characteristics; and
c) A set of properties that define the forwarding/filtering behavior of a Bridge with respect to frames destined for members of that group MAC address;

with a set of end stations that all wish to receive information destined for that group MAC address. Members of such a set of end stations are said to be *Group members.*

A Group is said to *exist* if the properties associated with that Group are visible in an entry in the Filtering Database of a Bridge, or in the GARP state machines that characterize the state of the Group; a Group is said to *have members* if the properties of the Group indicate that members of the Group can be reached through specific Ports of the Bridge.

NOTE—An example of the information that Group members might wish to receive is a multicast video data stream.

3.4 IEEE 802 Local Area Network (LAN)

IEEE 802 LANs (also referred to in the text simply as LANs) are LAN technologies that provide a MAC Service equivalent to the MAC Service defined in ISO/IEC 15802-1. IEEE 802 LANs include IEEE Std 802.3 (CSMA/CD), ISO/IEC 8802-4 (Token Bus), ISO/IEC 8802-5 (Token Ring), ISO/IEC 8802-6 (DQDB), ISO/IEC 8802-9 (IS-LAN), IEEE Std 802.11 (Wireless), ISO/IEC 8802-12 (Demand Priority), and ISO 9314-2 (FDDI) LANs.

NOTE—The connectionless service part of ISO/IEC 8802-6 provides an equivalent MAC Service. This standard covers Bridging of 48-bit addressed ISO/IEC 8802-6 PDUs only. The Bridging of 60-bit addressed ISO/IEC 8802-6 PDUs is outside the scope of this standard.

4. Abbreviations

The following abbreviations are used in this standard:

BPDU	Bridge Protocol Data Unit
CRC	Cyclic Redundancy Check
FCS	Frame Check Sequence
GARP	Generic Attribute Registration Protocol
GARP PDU	GARP Protocol Data Unit
GID	GARP Information Declaration
GIP	GARP Information Propagation
GMRP	GARP Multicast Registration Protocol
IETF	Internet Engineering Task Force
IGMP	Internet Group Management Protocol
MAC	Media Access Control

5. Conformance

5.1 Static conformance requirements

A MAC Bridge for which conformance to this standard is claimed shall

a) Conform to the relevant MAC standards for the MAC technology or technologies implemented at its Ports, as described in 6.5;
b) Conform to ISO/IEC 8802-2 for the implementation of a class of LLC supporting Type 1 operation as required by 7.3, and 7.1.2;
c) Relay and filter frames as described in 7.1 and specified in 7.5, 7.6, and 7.7 for support of Basic Filtering Services, and for a single traffic class associated with each outbound Port;
d) Maintain the information required to make frame-filtering decisions as described in 7.1 and specified in 7.8 and 7.9, for the support of Basic Filtering Services.
e) Use stated values of the following parameters of the Filtering Database (7.9):
    1) Filtering Database Size, the maximum number of entries that can be held in the Filtering Database.
    2) Permanent Database Size, the maximum number of entries that can be held in the Permanent Database.
f) Conform to the provisions for addressing specified in 7.12.
g) Provide
    1) A means of assigning a group MAC Address to identify the Bridge Protocol Entity, if 48-bit Universally Administered Addresses are not used;
    2) A distinct Port identifier for each Port of the Bridge, as specified in 8.5, as required by 8.2 for operation of the Spanning Tree Algorithm and Protocol.
h) Implement the Spanning Tree Algorithm and Protocol described in Clause 8, as specified in 8.7.
i) Not exceed the values given in 8.10.2 for the following parameters:
    1) Maximum bridge transit delay
    2) Maximum Message Age increment overestimate
    3) Maximum BPDU transmission delay
j) Use the value given in Table 8-3 for the following parameter:
    1) Hold Time
k) Encode transmitted BPDUs and validate received BPDUs as specified in Clause 9.
l) Specify a Guaranteed Port Filtering Rate for each Bridge Port, a Guaranteed Bridge Relaying Rate for the Bridge, and the related time intervals *TF* and *TR* as specified in Clause 16. Operation of the Bridge within the specified parameters shall not violate any of the other conformance provisions of this standard.

5.2 Options

A MAC Bridge for which conformance to this standard is claimed may

a) Provide the capability to control the mapping of the priority of forwarded frames as specified in 6.4, 7.5.1, and 7.7.3.
b) Provide the capability to read and update the Filtering and Permanent Databases as specified in 7.9.
c) Provide the capability to set the Ageing Time as specified in 7.9. A Bridge that provides this capability shall implement the full range of values specified in Table 7-4.
d) Implement the optional provisions for addressing Bridge Management as specified in 7.12.4, for associating the Bridge Address with a Bridge Port as specified in 7.12.5, and for preconditioning group addresses in the Permanent Database as specified in 7.9.6.
e) Provide the capability to assign values to the following parameters to allow configuration of the Spanning Tree active topology:

1) Bridge Priority
2) Port Priority
3) Path Cost for each Port

A Bridge that provides this capability shall implement the range of values specified in 8.10.2 and Table 8-4 and Table 8-5.

f) Provide the capability to set the values of the following parameters of the Spanning Tree Algorithm and Protocol:
   1) Bridge Max Age
   2) Bridge Hello Time
   3) Bridge Forward Delay

   A Bridge that provides this capability shall implement the range of values specified in 8.10.2 and Table 8-3.
g) Support management of the Bridge. Bridges claiming to support management shall support all the management objects and operations defined in Clause 14.
h) Support the use of a remote management protocol. Bridges claiming to support remote management shall
   1) State which remote management protocol standard(s) or specification(s) are supported.
   2) State which standard(s) or specification(s) for managed object definitions and encodings are supported for use by the remote management protocol.
i) Support the ability to disable topology change detection, as described in 8.5.5.10.
j) Provide multiple traffic classes, as described in 7.7, associated with one or more outbound Ports;
k) Provide the capability for Static Filtering Entries for individual MAC addresses to specify that forwarding or filtering behavior should be based on dynamic filtering information, as specified in 7.9.1;
l) Relay and filter frames as described in 7.1 and specified in 7.5, 7.6, 7.7, and 7.9 for support of Extended Filtering Services, and implement the GARP Multicast Registration Protocol (GMRP), as detailed in the conformance requirements defined in 10.4.1;
m) Provide the ability to manage the priority of relayed frames, as specified in 6.4, 7.5.1, and 7.7.3.

5.3 Protocol Implementation Conformance Statement (PICS) proforma

The supplier of an implementation that is claimed to conform to this standard shall complete a copy of the PICS proforma provided in Annex A and shall provide the information necessary to identify both the supplier and the implementation.

5.4 Recommendations

5.4.1 Management

Support of the managed objects and management operations specified in Clause 14 is highly recommended.

5.5 MAC-specific bridging methods

MAC-specific bridging methods may exist. Use of a MAC-specific bridging method and the method specified in this standard on the same LAN shall

a) Not prevent communication between stations in a Bridged LAN.
b) Preserve the MAC Service.
c) Preserve the characteristics of each bridging method within its own domain.
d) Provide for the ability of both bridging techniques to coexist simultaneously on a LAN without adverse interaction.

6. Support of the MAC Service

MAC Bridges interconnect the separate IEEE 802 LANs that comprise a Bridged LAN by relaying and filtering frames between the separate MACs of the Bridged LAN. The position of the bridging function within the MAC Sublayer is shown in Figure 6-1.

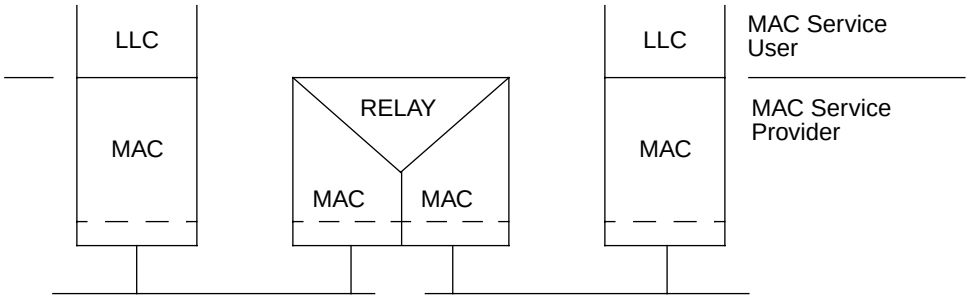


Figure 6-1—Internal organization of the MAC Sublayer

This clause discusses the following aspects of service provision in Bridged LANs:

a) Provision of the MAC Service to end stations;
b) Preservation of the MAC Service;
c) Maintenance of Quality of Service;
d) Provision of the internal sublayer service within the MAC Bridge;
e) Support of the Internal Sublayer Service by specific MAC procedures;
f) Filtering services.

6.1 Support of the MAC Service

The MAC Service provided to end stations attached to a Bridged LAN is the (unconfirmed) connectionless-mode MAC Service defined in ISO/IEC 15802-1. The MAC Service is defined as an abstraction of the features common to a number of specific MAC Services; it describes the transfer of user data between source and destination end stations, via MA-UNITDATA request primitives and corresponding MA-UNITDATA indication primitives issued at MAC Service access points. Each MA-UNITDATA request and indication primitive has four parameters: Destination Address, Source Address, MAC Service data unit (MSDU), and Priority.

The style of Bridge operation maximizes the availability of the MAC Service to end stations and assists in the maintenance of the Bridged LAN. It is therefore desirable that Bridges be capable of being configured in the Bridged LAN:

a) So as to provide redundant paths between end stations to enable the Bridged LAN to continue to provide the Service in the event of component failure (of Bridge or LAN).
b) So that the paths supported between end stations are predictable and configurable given the availability of Bridged LAN components.

6.2 Preservation of the MAC Service

The MAC Service offered by a Bridged LAN consisting of LANs interconnected by MAC Bridges is similar to that offered by a single LAN (6.3). In consequence,

a) A Bridge is not directly addressed by communicating end stations, except as an end station for management purposes: frames transmitted between end stations carry the MAC Address of the peer-end station in their Destination Address field, not the MAC Address, if any, of the Bridge.
b) All MAC Addresses must be unique within the Bridged LAN.
c) The MAC Addresses of end stations are not restricted by the topology and configuration of the Bridged LAN.

6.3 Quality of service maintenance

The MAC Sublayer provides the MAC Service to end stations attached to a LAN or a Bridged LAN. The quality of the MAC Service supported by a Bridge should not be significantly inferior to that provided by a single LAN. The Quality of Service parameters to be considered are those that relate to

a) Service availability
b) Frame loss
c) Frame misordering
d) Frame duplication
e) The transit delay experienced by frames
f) Frame lifetime
g) The undetected frame error rate
h) Maximum service data unit size supported
i) User priority
j) Throughput

6.3.1 Service availability

Service availability is measured as that fraction of some total time during which the MAC Service is provided. The operation of a Bridge can increase or lower the service availability.

The service availability can be increased by automatic reconfiguration of the Bridged LAN in order to avoid the use of a failed component (e.g., repeater, cable, or connector) in the data path. The service availability can be lowered by failure of a Bridge itself, through denial of service by the Bridge, or through frame filtering by the Bridge.

A Bridge may deny service and discard frames (6.3.2) in order to preserve other aspects of the MAC Service (6.3.3 an 6.3.4) when automatic reconfiguration takes place. Service may be denied to end stations that do not benefit from the reconfiguration; hence, the service availability is lowered for those end stations. Bridges may filter frames in order to localize traffic in the Bridged LAN. Should an end station move, it may then be unable to receive frames from other end stations until the filtering information held by the Bridges is updated.

To maximize the service availability, no loss of service or delay in service provision should be caused by Bridges, except as a consequence of a failure, removal, or insertion of a Bridged LAN component, or as a consequence of the movement of an end station. These are regarded as extraordinary events. The operation of any additional protocol necessary to maintain the quality of the MAC Service is thus limited to the configuration of the Bridged LAN, and is independent of individual instances of service provision.

NOTE—This is true only in circumstances where admission control mechanisms are not present, i.e., where the Bridges provide a “best effort” service. The specification and applicability of admission control mechanisms in Bridges is outside the scope of this standard.

6.3.2 Frame loss

The MAC Service does not guarantee the delivery of Service Data Units. Frames transmitted by a source station arrive, uncorrupted, at the destination station with high probability. The operation of a Bridge introduces minimal additional frame loss.

A frame transmitted by a source station can fail to reach its destination station as a result of

a) Frame corruption during physical layer transmission or reception.
b) Frame discard by a Bridge because
    1) It is unable to transmit the frame within some maximum period of time and, hence, must discard the frame to prevent the maximum frame lifetime (6.3.6) from being exceeded.
    2) It is unable to continue to store the frame due to exhaustion of internal buffering capacity as frames continue to arrive at a rate in excess of that at which they can be transmitted.
    3) The size of the service data unit carried by the frame exceeds the maximum supported by the MAC procedures employed on the LAN to which the frame is to be relayed.
    4) Changes in the connected topology of the Bridged LAN necessitate frame discard for a limited period of time to maintain other aspects of Quality of Service.

6.3.3 Frame misordering

The MAC Service does not permit the reordering of frames with a given user priority for a given combination of destination address and source address. MA_UNITDATA.indication service primitives corresponding to MA_UNITDATA.request primitives, with the same requested priority and for the same combination of destination and source addresses, are received in the same order as the request primitives were processed.

NOTE—The operation of the Forwarding Process in Bridges (7.7) is such that the frame-ordering characteristics of the MAC Service are preserved.

Where Bridges in a Bridged LAN are capable of connecting the individual MACs in such a way that multiple paths between any source station–destination station pairs exist, the operation of a protocol is required to ensure that a single path is used.

6.3.4 Frame duplication

The MAC Service does not permit the duplication of frames. The operation of Bridges does not introduce duplication of user data frames.

The potential for frame duplication in a Bridged LAN arises through the possibility of duplication of received frames on subsequent transmission within a Bridge, or through the possibility of multiple paths between source and destination end stations.

A Bridge shall not duplicate user data frames. Where Bridges in a Bridged LAN are capable of connecting the individual MACs in such a way that multiple paths between any source station–destination station pairs exist, the operation of a protocol is required to ensure that a single path is used.

6.3.5 Transit delay

The MAC Service introduces a frame transit delay that is dependent on the particular media and MAC method employed. Frame transit delay is the elapsed time between an MA_UNITDATA.request primitive and the corresponding MA_UNITDATA.indication primitive. Elapsed time values are calculated only on Service Data Units that are successfully transferred.

Since the MAC Service is provided at an abstract interface within an end station, it is not possible to specify precisely the total frame transit delay. It is, however, possible to measure those components of delay associated with media access and with transmission and reception; and the transit delay introduced by an intermediate system, in this case a Bridge, can be measured.

The minimum additional transit delay introduced by a Bridge is the time taken to receive a frame plus that taken to access the media onto which the frame is to be relayed. Note that the frame is completely received before it is relayed as the Frame Check Sequence (FCS) is to be calculated and the frame discarded if in error.

6.3.6 Frame lifetime

The MAC Service ensures that there is an upper bound to the transit delay experienced for a particular instance of communication. This maximum frame lifetime is necessary to ensure the correct operation of higher layer protocols. The additional transit delay introduced by a Bridge is discussed in 6.3.5.

To enforce the maximum frame lifetime, a Bridge may be required to discard frames. Since the information provided by the MAC Sublayer to a Bridge does not include the transit delay already experienced by any particular frame, Bridges must discard frames to enforce a maximum delay in each Bridge.

The value of the maximum bridge transit delay is based on both the maximum delays imposed by all the Bridges in the Bridged LAN and the desired maximum frame lifetime. A recommended and an absolute maximum value are specified in Table 8-2.

6.3.7 Undetected frame error rate

The MAC Service introduces a very low undetected frame error rate in transmitted frames. Undetected errors are protected against by the use of an FCS that is appended to the frame by the MAC Sublayer of the source station prior to transmission, and checked by the destination station on reception.

The FCS calculated for a given service data unit is dependent on the MAC method employed. It is therefore necessary to recalculate the FCS within a Bridge providing a relay function between IEEE 802 MACs of dissimilar types if differences in the method of calculation and/or the coverage of the FCS, or changes to the data that is within the coverage of the FCS, would lead to a different FCS being calculated for the service data unit by the two MAC methods. This introduces the possibility of additional undetected errors arising from the operation of a Bridge. For frames relayed between LANs of the same MAC type, the Bridge shall not introduce an undetected frame error rate greater than that which would have been achieved by preserving the FCS.

NOTE—Application of the techniques described in Annex G allow an implementation to achieve an arbitrarily small increase in undetected frame error rate, even in cases where the data that is within the coverage of the FCS is changed. As a maintenance activity on this standard, revision of the wording of this requirement will be initiated, with a view to placing a quantitative limit on the increase in undetected frame error rate that is acceptable in a conformant implementation.

6.3.8 Maximum Service Data Unit Size

The Maximum Service Data Unit Size that can be supported by an IEEE 802 LAN varies with the MAC method and its associated parameters (speed, electrical characteristics, etc.). It may be constrained by the owner of the LAN. The Maximum Service Data Unit Size supported by a Bridge between two LANs is the smaller of that supported by the LANs. No attempt is made by a Bridge to relay a frame to a LAN that does not support the size of Service Data Unit conveyed by that frame.

6.3.9 Priority

The MAC Service includes user priority as a Quality of Service parameter. MA_UNITDATA.request primitives with a high priority may be given precedence over other request primitives made at the same station, or at other stations attached to the same LAN, and can give rise to earlier MA_UNITDATA.indication primitives.

The MAC Sublayer maps the requested user priorities onto the access priorities supported by the individual MAC method. The requested user priority may be conveyed to the destination station.

The transmission delay experienced by a frame in a bridge can be managed by associating a user_priority with the frame.

The transmission delay comprises

a) A queuing delay until the frame becomes first in line for transmission on the Port, in accordance with the procedure for selecting frames for transmission described in 7.7.4;
b) The access delay for transmission of the frame.

Queueing delays can be managed using user_priority. Access delays can be managed using user_priority in MAC methods that support more than one access priority.

The user priority associated with a frame can be signaled by means of the priority signaling mechanisms inherent in some IEEE 802 LAN MAC types. Since not all IEEE 802 LAN MAC types are able to signal the user priority associated with a frame, MAC Bridges regenerate user priority based upon a combination of signaled information and configuration information held in the Bridge.

The Bridge maps the user priority onto one or more traffic classes; Bridges that support more than one traffic class are able to support expedited classes of traffic. The Forwarding Process, 7.7, describes the use of user priority and traffic classes in MAC Bridges. Given the constraints placed upon frame misordering in a Bridge, as expressed in 6.3.3, the mappings of priority and traffic class are static.

NOTE 1—The term Traffic Class, as used in this standard, is used only in the context of the operation of the priority handling and queueing functions of the Forwarding Process, as described in 7.7. Any other meanings attached to this term in other contexts do not apply to the use of the term in this standard.

The ability to signal user_priority in IEEE 802 LANs allows user_priority to be carried with end-to-end significance across a Bridged LAN. This, coupled with a consistent approach to the mapping of user_priority to traffic classes, and of user_priority to access_priority, allows consistent use of user_priority information to be made, according to the capabilities of the Bridges and MAC methods that are involved in the transmission path.

NOTE 2—IEEE P802.1Q[8] defines a frame format and associated procedures that can be used to carry user priority information across LAN MAC types that are not able to signal user priority. Use of the P802.1Q frame format allows the end-to-end significance of user_priority to be maintained regardless of the ability of individual LAN MAC types to signal priority.

Under normal circumstances, user_priority is not modified in transit through the relay function of a Bridge; however, there may be some circumstances where it is desirable for management purposes to control how user_priority is propagated. The User Priority Regeneration Table (Table 7-1) provides the ability to map incoming user_priority values on a per-Port basis, under management control. In its default state, this table

[8]IEEE P802.1Q/D11 (July 30th 1998), is an authorized IEEE Standards Project that was not approved by the IEEE-SA at the time this standard went to press. For information on obtaining the draft, contact the IEEE.

provides an identity mapping from user_priority values to Regenerated user_priority values; i.e., by default, the Regenerated user_priority is identical to the incoming user_priority.

6.3.10 Throughput

The total throughput provided by a Bridged LAN can be significantly greater than that provided by an equivalent single LAN. Bridges may localize traffic within the Bridged LAN by filtering frames. Filtering services available in Bridged LANs are described in 6.6.

The throughput between end stations on individual LANs, communicating through a Bridge, can be lowered by frame discard in the Bridge due to the inability to transmit at the required rate on the LAN forming the path to the destination for an extended period.

6.4 Internal Sublayer Service provided within the MAC Bridge

The Internal Sublayer Service provided by a MAC entity to the MAC Relay Entity within a Bridge is that provided by the individual MAC for the LAN Port. This observes the appropriate MAC procedures and protocol for the LAN to which it attaches. No control frames, i.e., frames that do not convey MAC user data, are forwarded on any LAN other than that on which they originated.

The Internal Sublayer Service is derived from the MAC Service defined by ISO/IEC 15802-1 by augmenting that specification with elements necessary to the performance of the relay function. Within the attached end station, these additional elements can be considered to be either below the MAC Service boundary, and pertinent only to the operation of the service provider; or local matters not forming part of the peer-to-peer nature of the MAC Service. Three parameters are added to the list of parameters associated with the MA_UNITDATA.request and MA_UNITDATA.indication primitives defined by ISO/IEC 15802-1. These are frame_type, MAC_action, and frame_check_sequence. The definition of the Internal Sublayer Service does not add any new service primitives to those defined by the LAN MAC Service Definition.

The Internal Sublayer Service excludes MAC-specific features and procedures whose operation is confined to that of the individual LANs. The unit-data primitives that describe this service are

M_UNITDATA.indication (
frame_type,
mac_action,
destination_address,
source_address,
mac_service_data_unit,
user_ priority,
frame_check_sequence
)

Each M_UNITDATA indication primitive corresponds to the receipt of an error-free MAC frame from an individual LAN.

NOTE—Detailed specifications of error conditions in received frames are contained in the relevant MAC standards; for example, FCS errors, length errors, nonintegral number of octets.

The **frame_type** parameter indicates the class of frame. The value of this parameter is one of user_data_ frame, mac_specific_frame, or reserved_frame.

The **mac_action** parameter indicates the action requested of a MAC entity receiving the indication. If the value of the frame_type parameter is user_data_frame, then the mac_action parameter is one of

request_with_response, request_with_no_response, or response. For mac_specific_frames and reserved_frames, this parameter does not apply.

The **destination_address** parameter is either the address of an individual MAC entity or a group of MAC entities.

The **source_address** parameter is the individual address of the source MAC entity.

The **mac_service_data_unit** parameter is the service user data.

The **user_priority** parameter is the priority requested by the originating service user. The value of this parameter is in the range 0 through 7.

NOTE 1—The default user_priority value is 0. Values 1 through 7 form an ordered sequence of user_priorities, with 1 being the lowest value and 7 the highest. See 7.7.3 and H.2 for further explanation of the use of user_priority values and how they map to traffic classes.

Bridges have the capability to regenerate user priority from the user priority information contained in the data indication and the User Priority Regeneration Table for the reception Port, as specified in 7.5.1. The user_priority parameter takes the value of the Regenerated user_priority. In the case of IEEE 802 LAN MAC types that are not able to signal priority, the user_priority parameter takes the value of the Default User Priority for the Port on which the indication was received. The default value of Default User Priority is 0 for all Ports of the Bridge; the value for a given Port may be modified by means of the management functionality described in Clause 14 if such management functionality is supported.

The **frame_check_sequence** parameter is explicitly provided as a parameter of the primitive so that it can be used as a parameter to a related request primitive without recalculation.

The identification of the LAN from which particular frames are received is a local matter and is not expressed as a parameter of the service primitive.

M_UNITDATA.request (
frame_type,
mac_ action,
destination_address,
source_address,
mac_service_data_unit,
user_priority,
access_priority,
frame_check_sequence
)

A data request primitive is invoked to transmit a frame to an individual LAN.

The **frame_type** parameter indicates the class of frame.

The **mac_action** parameter indicates the action requested of the destination MAC entity.

The **destination_address** parameter is either the address of an individual MAC entity, or a group of MAC entities.

The **source_address** parameter is the individual address of the source MAC entity.

The **mac_service_data_unit** parameter is the service user data.

The **user_priority** parameter is the priority requested by the originating service user. The value of this parameter is in the range 0 through 7.

NOTE 2—The default user_priority value is 0. Values 1 through 7 form an ordered sequence of user_priorities, with 1 being the lowest value and 7 the highest. See 7.7.3 and H.2 for further explanation of the use of user_priority values and how they map to traffic classes.

The **access_priority** parameter is the priority used by the local service provider to convey the request. It can be used to determine the priority attached to the transmission of frames queued by the local MAC Entity, both locally and among other stations attached to the same individual LAN—if the MAC method permits. The value of this parameter, if specified, is in the range 0 (lowest) through 7 (highest).

The **frame_check_sequence** parameter is explicitly provided as a parameter of the primitive so that it can be used without recalculation.

The identification of the LAN to which a frame is to be transmitted is a local matter and is not expressed as a parameter of the service primitive.

6.5 Support of the Internal Sublayer Service by specific MAC procedures

This subclause specifies the mapping of the Internal Sublayer Service to the MAC Protocol and Procedures of each individual IEEE 802 MAC type, and the encoding of the parameters of the service in MAC frames. The mapping is specified by reference to the IEEE 802 standards that specify the individual MAC methods. The mapping draws attention to any special responsibilities of Bridges attached to LANs of that MAC type.

6.5.1 Support by IEEE Std 802.3 (CSMA/CD)

The CSMA/CD access method is specified in IEEE Std 802.3. Clause 3 of that standard specifies the MAC frame structure, and Clause 4 specifies the MAC method.

On receipt of an M_UNITDATA.request primitive, the local MAC Entity performs Transmit Data Encapsulation, assembling a frame using the parameters supplied as specified below. It prepends a preamble and a Start Frame Delimiter before handing the frame to the Transmit Media Access Management Component in the MAC Sublayer for transmission (IEEE Std 802.3, 4.2.3).

On receipt of a MAC frame by Receive Media Access Management, the MAC frame is passed to Receive Data Decapsulation, which validates the FCS and disassembles the frame, as specified below, into the parameters that are supplied with an M_UNITDATA.indication primitive (IEEE Std 802.3, 4.2.4).

The **frame_type** parameter takes only the value user_data_frame and is not explicitly encoded in MAC frames.

The **mac_action** parameter takes only the value request_with_no_response and is not explicitly encoded in MAC frames.

The **destination_address** parameter is encoded in the destination address field of the MAC frame (IEEE Std 802.3, 3.2.3).

The source_address parameter is encoded in the source address field of the MAC frame (IEEE Std 802.3, 3.2.3).

The number of octets in the mac_service_data_unit parameter is encoded in the length field of the MAC frame (IEEE Std 802.3, 3.2.6), and the octets of data are encoded in the data field (IEEE Std 802.3, 3.2.7).

The **user_priority** parameter provided in a data request primitive is not encoded in MAC frames. The user_priority parameter provided in a data indication primitive takes the value of the Default User Priority parameter for the Port through which the MAC frame was received (see 6.4).

The **frame_check_sequence** parameter is encoded in the FCS field of the MAC frame (IEEE Std 802.3, 3.2.8). The FCS is computed as a function of the destination address, source address, length, data, and PAD fields. If an M_UNITDATA.request primitive is not accompanied by this parameter, it is calculated in accordance with IEEE Std 802.3, 3.2.8.

NOTE 1—Since the PAD field, if present, contributes to the FCS, this parameter needs to include at least the contribution of the PAD field to the FCS in order for the original FCS to be preserved (See Annex G).

No special action, above that specified for the support of use of the MAC Service by LLC, is required for the support of the MAC Internal Sublayer Service by the CSMA/CD access method.

NOTE 2—The support by IEEE Std 802.3 is described only in terms of the operation of a Bridge when relaying frames that result from the use of LLC services over an 802.3 MAC. ISO/IEC 11802-5 defines the recommended practice for bridging Ethernet V2.0 frames.

NOTE 3—IEEE Std 802.3, 1998 Edition, describes the use of either a Length or an Ethernet protocol type in its frame format; however, the text of this subclause has yet to be revised to describe the use of Ethernet protocol types.

6.5.2 Support by ISO/IEC 8802-4 (token-passing bus)

The token-passing bus access method is specified in ISO/IEC 8802-4. In that standard, Section 4 specifies frame formats, Section 5 discusses the basic concepts of the access protocols, and Section 7 provides the definitive specification of the token-passing bus MAC operation.

On receipt of an M_UNITDATA.request primitive the local MAC Entity Interface Machine (IFM) queues a frame for transmission by the Access Control Machine (ACM) (ISO/IEC 8802-4, Section 7). On transmission the frame fields are set using the parameters supplied as specified below, and the Preamble, Start Delimiter, and End Delimiter are added (ISO/IEC 8802-4, Section 4).

On receipt of a valid MAC frame by the Receive Machine (RxM) (ISO/IEC 8802-4, 7.1.5), an M_UNITDATA.indication primitive is generated, with parameters derived from the frame fields as specified below.

The **frame_type** parameter is encoded in the FF bits (bit positions 1 and 2) of the frame control (FC) field (ISO/IEC 8802-4, 4.1.3.1, 4.1.3.2). A bit pattern of 0 1 denotes a user_data_frame, a bit pattern of 0 0 or 1 0 denotes a mac_specific_frame, and a bit pattern of 1 1 denotes a reserved_frame.

The **mac_action** parameter is encoded in the MMM bits (bit positions 3, 4, and 5) of the FC field (ISO/IEC 8802-4, 4.1.3.2). For user_data_frames these take the values 0 0 0 for request_with_no_response, 0 0 1 for request_with_response, and 0 1 0 for response.

The **destination_address** parameter is encoded in the destination address field of the MAC frame (ISO/IEC 8802-4, 4.1.4.1).

The **source_address** parameter is encoded in the source address field of the MAC frame (ISO/IEC 8802-4, 4.1.4.2).

The **mac_service_data_unit** parameter is encoded in the MAC Data Unit field (ISO/IEC 8802-4, 4.1.5).

The **user_priority** parameter is encoded in the PPP bits (bit positions 6, 7, and 8) of the FC field (ISO/IEC 8802-4, 4.1.3.2; 5.1.7).

The **frame_check_sequence** parameter is encoded in the FCS field of the MAC frame (ISO/IEC 8802-4, 4.1.6). The FCS is computed as a function of the frame control, destination address, source address, and data fields. If an M_UNITDATA.request primitive is not accompanied by this parameter, it is calculated in accordance with ISO/IEC 8802-4, 4.1.6.

No special action, above that specified for the support of use of the MAC Service by LLC, is required for the support of the MAC Internal Sublayer Service by the token-passing bus access method.

6.5.3 Support by ISO/IEC 8802-5 (token-passing ring)

The token-passing ring access method is specified in ISO/IEC 8802-5. Clause 3 of that standard specifies formats and facilities, and Clause 4 specifies token-passing ring protocols.

On receipt of an M_UNITDATA.request primitive the local MAC Entity composes a frame using the parameters supplied as specified below, appending the frame control, destination address, source address, and FCS fields to the user data, and enqueuing the frame for transmission. On transmission, the starting delimiter, access control field, ending delimiter, and frame status fields are added.

On receipt of a valid MAC frame (ISO/IEC 8802-5, 4.1.4) that was not transmitted by the Bridge Port's local MAC Entity, with the Routing Information Indicator bit (which occupies the same position in the source address field as does the Group Address bit in the destination address field) set to zero, an M_UNITDATA.indication primitive is generated, with parameters derived from the frame fields as specified below.

The **frame_type** parameter is encoded in the frame_type bits (FF bits) of the frame control field (ISO/IEC 8802-5, 3.2.3.1). A bit pattern of 0 1 denotes a user_data_frame, a bit pattern of 0 0 denotes a mac_specific_frame, and a bit pattern of 1 0 or 1 1 denotes a reserved_frame.

The **mac_action** parameter only takes the value request_with_no_response and is not explicitly encoded in MAC frames.

The **destination_address** parameter is encoded in the destination address field of the MAC frame (ISO/IEC 8802-5, 3.2.4.1).

The **source_address** parameter is encoded in the source address field of the MAC frame (ISO/IEC 8802-5, 3.2.4.2).

The **mac_service_data_unit** parameter is encoded in the information field (ISO/IEC 8802-5, 3.2.6).

The **user_priority** parameter associated with user_data_frames is encoded in the YYY bits of the frame control field (ISO/IEC 8802-5, 3.2.3).

The **frame_check_sequence** parameter is encoded in the FCS field of the MAC frame (ISO/IEC 8802-5, 3.2.7). The FCS is computed as a function of the frame control, destination address, source address, and information fields. If an M_UNITDATA.request primitive is not accompanied by this parameter, it is calculated in accordance with ISO/IEC 8802-5, 3.2.7.

The Address Recognized (A) bits in the Frame Status field of a frame ISO/IEC 8802-5, 3.2.9) may be set to 1 if an M_ UNITDATA.indication primitive with frame_type and mac_action parameter values of user_data_frame and request_with_no_response respectively is generated, or if such an indication would be generated if buffering had been available; otherwise the A bits shall not be set except as required by ISO/IEC 8802-5.

If the A bits are set to 1, the Frame Copied (C) bits (ISO/IEC 8802-5, 3.2.9) may be set to 1 to reflect the availability of receive buffering; otherwise the C bits shall not be set.

In order to support the MAC Internal Sublayer Service, a Token Ring Bridge must be capable of recognizing and removing frames transmitted by itself, even though they can carry a source address different from that of the Bridge Port that transmitted them.

6.5.4 Support by fibre distributed data interface (FDDI)

The FDDI access method is specified in ISO 9314-2. Clause 6 of that standard specifies Services, and Clauses 7 and 8 specify Facilities and Operation, respectively.

On receipt of a valid frame (ISO 9314-2, 8.3.1) that was not transmitted by the Bridge Port's local MAC entity, with the first bit of the source address equal to zero, an M_UNITDATA.indication primitive is generated. The parameters associated with the primitive are derived from the frame fields as specified shortly below.

The Address Recognized (A) indicator in the Frame Status field of the frame (ISO 9314-2, 7.3.8) on the ring from which it was received shall not be set except as required by ISO 9314-2. The Frame Copied (C) indicator (ISO 9314-2, 7.3.8) may be set if an M_UNITDATA.indication primitive with frame_type and mac_action parameter values of user_data_frame and request_with_no_response, respectively, is generated, if the frame is to be forwarded, and if receive buffering is available. Otherwise, the C indicator shall not be altered except as required by ISO 9314-2.

NOTE—This specification of the setting of the C indicator by ISO 9314-2 MAC Bridges enhances that given in ISO 9314-2. A Bridge can be required by ISO 9314-2 to alter the A and/or C indicators when receiving a frame addressed to the Bridge as an FDDI end station, or when receiving a frame associated with the operation of the FDDI MAC

The parameters associated with the M_UNITDATA.indication that is generated on receipt of a frame now follow:

The **frame_type** parameter is encoded in the frame format bits (CL, FF, and ZZZZ bits) of the Frame Control field (ISO 9314-2, 7.3.3). The bit pattern 0L01rXXX denotes a user_data_frame (asynchronous LLC frame, where L represents the address length and can be 0 or 1, r is reserved and can be received as 0 or 1, and XXX can range from 000 to 111). All other bit patterns yield a frame_type parameter value of not_user_data_frame.

The **mac_action** parameter takes only the value request_with_no_response and is not explicitly encoded in MAC frames.

The **destination_address** parameter is enclosed in the destination address field of the MAC frame (ISO 9314-2, 7.3.4–7.3.4.1).

The **source_address** parameter is encoded in the source address field of the MAC frame (ISO 9314-2, 7.3.4–7.3.4.2).

The **mac_service_data_unit** parameter is encoded in the information field (ISO 9314-2, 7.3.5).

The **user_priority** parameter associated with user_data_frames is encoded in the PPP bits of the frame control field (ISO 9314-2, 7.3.3.4) when the frame is an asynchronously transmitted LLC frame whose frame control field value is 0L010PPP, where L represents the address length (ISO 9314-2, 7.3.3.2).

The **frame_check_sequence** parameter is encoded in the Frame Check Sequence Field of the MAC frame (ISO 9314-2, 7.3.6). The frame_check_sequence is computed as a function of the frame control, destination address, source address, and information fields.

On receipt of an M_UNITDATA.request primitive, the local MAC entity composes a frame using the parameters supplied as specified above, appending the frame control, destination address, source address, and frame check sequence to the user data, and enqueuing the frame for transmission on reception of a suitable token (ISO 9314-2, 8.3.1). On transmission, the preamble, starting delimiter, ending delimiter, and frame status fields are added.

If an M_UNITDATA.request primitive is not accompanied by a frame check sequence, one is calculated in accordance with ISO 9314-2, 7.3.6.

The bit pattern of the frame control field shall be 0L01rPPP, indicating an asynchronous LLC frame with L representing the address length (ISO 9314-2, 7.3.3.2), r being reserved and set to zero, and PPP indicating the frame's priority (ISO 9314-2, 7.3.3.4).

If the **user_priority** parameter value is specified, it is encoded in the PPP bits and the access priority (ISO 9314-2, 8.1.4) is derived from the token-holding timer (THT) or otherwise by implementor option. If the user_priority parameter value is unspecified, the PPP bits shall be set to zero and the access priority derived from the THT.

In order to support the MAC Internal Layer Service, an FDDI bridge removes frames transmitted by itself as required by ISO 9314-2, even though they can carry a source address different from that of the bridge port that transmitted them.

6.5.5 Support by ISO/IEC 8802-6 (Distributed Queue Dual Bus)

The ISO/IEC 8802-6 access method is specified in ISO/IEC 8802-6. Subclause 5.1 of that standard specifies the provision of the MAC Service to LLC, and Section 6 specifies DQDB Layer Protocol Data Unit (PDU) formats.

On receipt of an M_UNITDATA.request primitive, the local MAC entity (the MAC Convergence Function [MCF], ISO/IEC 8802-6, 5.1.1) shall compose an Initial MAC Protocol Data Unit (IMPDU) using the parameters supplied, by encoding and concatenating a Common PDU Header (ISO/IEC 8802-6, 6.5.1.1), a MAC Convergence Protocol (MCP) Header (ISO/IEC 8802-6, 6.5.1.2), a Header Extension (ISO/IEC 8802-6, 6.5.1.3), an INFO field (ISO/IEC 8802-6, 6.5.1.4), a PAD field (ISO/IEC 8802-6, 6.5.1.5), an optional CRC32 field (ISO/IEC 8802-6, 6.5.1.6) and a Common PDU Trailer (ISO/IEC 8802-6, 6.5.1.7). The values set within these headers and trailers are as specified and are in conformance with ISO/IEC 8802-6, 5.1.1.1.1 (Creation of the IMPDU). The IMPDU shall then be presented to the IMPDU Segmentation machine for transmission (ISO/IEC 8802-6, 5.1.1.1.2).

The **frame_type** parameter shall take the value user_data_frame and is not explicitly encoded in IMPDUs.

The **mac_action** parameter shall take the value request_with_no_response and is not explicitly encoded in IMPDUs.

The Common PDU Header shall be encoded according to ISO/IEC 8802-6, 6.5.1.1.

The **destination_address** parameter shall be encoded right-justified in the Destination Address (DA) subfield of the MCP Header, with the Address_Type subfield set to 48_bit_address (binary 1000), and padded with 12 zeroes to left-fill the 60-bit address field (ISO/IEC 8802-6, 6.5.1.2.1).

The **source_address** parameter shall be encoded right-justified in the Source Address (SA) subfield of the MCP Header, with the Address_Type subfield set to 48_bit_address (binary 1000), and padded with 12 zeroes to left-fill the 60-bit address field (ISO/IEC 8802-6, 6.5.1.2.1).

The Protocol Identifier (PI) subfield of the MCP header shall be encoded to decimal 1 (to indicate LLC) (ISO/IEC 8802-6, 6.5.1.2.4.1).

The PAD Length (PL) subfield of the MCP header shall be encoded according to ISO/IEC 8802-6, 6.5.1.2.4.2.

The Bridging field of the MCP Header shall be set to all zeroes (ISO/IEC 8802-6, 6.5.1.2.6). This field is reserved for the bridging of 60-bit ISO/IEC 8802-6 PDUs only.

If a Header Extension field was present in the corresponding data indication, then the Header Extension field shall be encoded using the received value. Otherwise, the Header Extension field shall be encoded according to ISO/IEC 8802-6, 5.1.1.1.1, steps 11) and 13).

NOTE—The Header Extension field in ISO/IEC 8802-6 is a MAC-specific facility that is not relevant to the operation of the MAC Bridge; hence, its value is not represented in the Internal Sublayer service. However, if the Header Extension is present in received MAC frames, its value is preserved in corresponding transmitted frames.

The **mac_service_data_unit** parameter shall be encoded into the INFO field of the IMPDU (ISO/IEC 8802-6, 6.5.1.4).

The **user_priority** parameter shall be encoded in the QOS_DELAY subfield of the MCP Header (ISO/IEC 8802-6, 6.5.1.2.5.1). The QOS_LOSS subfield of the MCP Header shall be encoded to zero (ISO/IEC 8802-6, 6.5.1.2.5.2). The CRC32 Indicator Bit (CIB) subfield of the MCP Header shall be set to one if the value of the frame_check_sequence parameter is specified (ISO/IEC 8802-6, 6.5.1.2.5.3), or if the frame_check_sequence parameter is unspecified but the FCS is calculated and included in the CRC32 field. The length of the header_extension parameter shall be encoded in the Header Extension Length (HEL) subfield of the MCP Header (ISO/IEC 8802-6, 6.5.1.2.5.4).

The PAD field shall be encoded according to ISO/IEC 8802-6, 6.5.1.5.

The **frame_check_sequence** parameter, if specified, shall be encoded in the CRC32 field (ISO/IEC 8802-6, 6.5.1.6). If the frame_check_sequence parameter is unspecified, then the implementor has the choice of either

a) Calculating and including the CRC32 field and setting the CIB to one, or
b) Not including the CRC32 field and setting the CIB to zero.

The Common PDU Trailer shall be encoded according to ISO/IEC 8802-6, 6.5.1.1.

On receipt of a valid IMPDU by the MCF Receive Block (ISO/IEC 8802-6, 5.1.1.2), and if it is verified to contain 48-bit addresses by inspecting the Address_Type subfield in the destination and source address fields (ISO/IEC 8802-6, 6.5.1.2.1), an M_UNITDATA.indication primitive shall be generated, with the parameters derived from the IMPDU fields as indicated below. In the event that the Address_Type does not indicate a 48-bit address, the IMPDU shall be discarded, and no M_UNITDATA.indication primitive shall be generated.

The **frame_type** parameter shall take the value user_data_frame since it is not explicitly encoded in IMPDUs.

The **mac_action** parameter shall take the value request with_no_response since it is not explicitly encoded in IMPDUs.

The **destination_address** parameter shall be given the value of the MSAP address of the Destination Address (DA) subfield of the MCP Header (ISO/IEC 8802-6, 6.5.1.2.1).

The **source_address** parameter shall be given the value of the MSAP address of the Source Address (SA) subfield of the MCP Header (ISO/IEC 8802-6, 6.5.1.2.1).

The **mac_service_data_unit** parameter shall be given the value of the INFO field of the IMPDU (ISO/IEC 8802-6, 6.5.1.4).

The **user_priority** parameter shall be given the value of the QOS_DELAY subfield of the MCP Header (ISO/IEC 8802-6, 6.5.1.2.5.1).

The **frame_check_sequence** parameter shall be given the value of the CRC 32 field (ISO/IEC 8802-6, 6.5.1.6) if the CRC32 Indicator Bit (CIB) (ISO/IEC 8802-6, 6.5.1.2.5.3) is set to 1; else it shall be unspecified.

No special action, above that specified for the support of use of the MAC Service by LLC, is required for the support of the MAC Internal Sublayer Service by the DQDB Access Method.

6.5.6 Support by IEEE Std 802.11 (Wireless LANs)

The wireless LAN access method is specified in IEEE Std 802.11. Clause 7 of that standard specifies frame formats, Clause 9 specifies the MAC sublayer function, and Clause 11 specifies the mandatory MAC sublayer management function.

A Bridge to an 8802-11 LAN shall connect to an 8802-11 Portal, which in turn connects to an 8802-11 Distribution System. For the purposes of bridging, the service interface presented at the Portal is identical to the service interface presented at the 8802-11 MAC SAP. An instance of an 8802-11 Distribution System can be implemented from 802 LAN components. 8802-11 STAs attach to the Distribution System via an 8802-11 Access Point. A bridge shall not connect to an 8802-11 Independent BSS. For a description of the 8802-11 architecture, see Clause 5 of IEEE Std 802.11.

On receipt of an M_UNITDATA.request primitive the portal constructs a MAC Service Data Unit and passes it to the MAC Data service for transmission, in accordance with the frame formats and procedures specified in IEEE Std 802.11, Clauses 6, 7, 9, and Annex C, using the parameters supplied as specified below.

On receipt of a valid MAC Service Data Unit (see IEEE Std 802.11, Clauses 6, 7, 9, and Annex C), the portal generates an M_UNITDATA.indication primitive with parameter values derived from the frame fields as specified below.

The frame_type parameter only takes the value user_data_frame. When processing MSDU_from_LLC, the frame_type of user_data_frame shall be translated according to parameters specified in 7.1.3.1 of IEEE Std 802.11 and is explicitly encoded in MAC frames.

The mac_action parameter only takes the value request_with_no_response and is not explicitly encoded.

The destination_address parameter is encoded in MAC frames as the DA described in Table 4 of 7.2.2 of IEEE Std 802.11.

The source_address parameter is encoded in MAC frames as the SA described in Table 4 of 7.2.2 of IEEE Std 802.11.

The mac_service_data_unit parameter is encoded in the Frame Body field (IEEE Std 802.11, 7.1.3.5) of MAC frames. The length of the MSDU shall be ≤ 2304 octets. The length is not encoded in MAC frames, but is conveyed in the PHY headers.

The user_priority parameter is not encoded in MAC frames. The user_priority parameter provided in an M_UNITDATA.indication primitive shall take the value of the Default_User_Priority parameter for the port through which the MAC Service Data Unit was received (see 6.4).

The frame_check_sequence parameter is encoded in the Frame Check Sequence (FCS) field of MAC frames in accordance with IEEE Std 802.11, 7.1.3.6 FCS.

The access_priority parameter is not encoded in MAC frames.

No special action, above that specified in IEEE Std 802.11, is required for the support of the MAC Internal Sublayer Service by the wireless LAN access method.

6.5.7 Support by ISO/IEC 8802-12 (Demand Priority)

The demand priority access method is specified in ISO/IEC 8802-12. Clause 10 of that standard specifies frame formats, and Clause 11 specifies the MAC protocol. Two formats of MAC frame are specified, one compatible with the IEEE Std 802.3 frame format, and one compatible with the ISO/IEC 8802-5 frame format (a given demand-priority LAN operates using only one of these formats throughout).

Following receipt of an M_UNITDATA.request primitive the local MAC entity constructs and transmits the corresponding MAC frame as specified in ISO/IEC 8802-12, 11.5.7 (FUNCTION Build_Frame) and 11.6.6 (MAC6_TRANSMIT_FRAME).

On receipt of a MAC frame (ISO/IEC 8802-12, 11.6.5, MAC_READ_FRAME), the local MAC entity generates an M_UNITDATA.indication primitive as specified in ISO/IEC 8802-12, 11.5.6 (PROCEDURE Process_Received_MAC_Frame).

The **frame_type** parameter only takes the value user_data_frame and is not explicitly encoded in MAC frames.

The **mac_action** parameter only takes the value request_with_no_response and is not explicitly encoded in MAC frames.

The **destination_address** parameter is encoded in the Destination Address (DA) field of the MAC frame (ISO/IEC 8802-12, 10.2.1 and 10.3.3).

The **source_address** parameter is encoded in the Source Address (SA) field of the MAC frame (ISO/IEC 8802-12, 10.2.1 and 10.3.3).

The **mac_service_data_unit** parameter is encoded in the Data field (IEEE Std 802.3 frame format, ISO/IEC 8802-12, 10.2.3) or Information field (ISO/IEC 8802-5 frame format, ISO/IEC 8802-12, 10.3.5) of the MAC frame.

For the IEEE Std 802.3 frame format, the **user_priority** parameter is not encoded in the MAC frame, but corresponds to the ISO/IEC 8802-12 priority value “normal” or “high.” On frame reception, the value “normal” maps to user_priority 0 and the value “high” maps to user_priority 4. On frame transmission, user_priority values 0 through 3 map to “normal” and values 4 through 7 map to “high.”

For the ISO/IEC 8802-5 frame format, the **user_priority** parameter is encoded in the YYY bits of the Frame Control field (ISO/IEC 8802-12, 10.3.2.2).

The **access_priority** parameter in an M_UNITDATA.request primitive is mapped to the ISO/IEC 8802-12 priority value “normal” or “high”: access_priority values 0 through 3 map to “normal,” access_priority values 4 through 7 map to “high.”

The **frame_check_sequence** parameter is encoded in the Frame Check Sequence (FCS) field of the MAC frame (ISO/IEC 8802-12, 10.2.4 and 10.3.6).

No special action, above that specified in ISO/IEC 8802-12, is required for the support of the MAC Internal Sublayer Service by the demand-priority access method.

6.6 Filtering services in Bridged LANs

MAC Bridges provide filtering services in Bridged LANs that support some aspects of the maintenance of Quality of Service; in particular, transit delay, priority, and throughput. In addition, these services provide for a degree of administrative control over the propagation of particular MAC Addresses in the Bridged LAN.

The services described are services in the most general sense; i.e., they are descriptions of the functionality that are made available to the MAC Service user or an administrator in order to control and access filtering capabilities in Bridged LANs. The description of each service makes no assumptions in terms of how the service might be realized. There are at least the following possibilities:

a) Use of existing protocols and mechanisms, defined in IEEE 802 standards and elsewhere;
b) Use of management functionality, either locally defined or implemented via remote management protocols;
c) Other means, standardized or otherwise.

6.6.1 Purpose(s) of filtering service provision

Filtering services are provided in Bridged LANs for the purposes described in the following subclauses.

6.6.1.1 Administrative control

Filtering services provide for administrative control over the use of particular source and destination addresses in designated parts of the network. Such control allows network managers and administrators to limit the extent of operation of network layer and other protocols that make use of individual and group MAC Addresses by establishing administrative boundaries across which specific MAC Addresses are not forwarded.

6.6.1.2 Throughput and end station load

Filtering services increase the overall throughput of the network, and reduce the load placed on end stations caused by the reception of frames that are destined for other end stations. They achieve this end by

a) Limiting frames destined for specific MAC Addresses to parts of the network which, to a high probability, lie along a path between the source MAC Address and the destination MAC Address;
b) Reducing the extent of group addressed frames to those parts of the network which contain end stations that are legitimate recipients of that traffic.

NOTE—Some aspects of the filtering services described in this standard are dependent upon the active participation of end stations. Where such participation is not possible, those aspects of the filtering services will be unavailable.

6.6.2 Goals of filtering service provision

The filtering services provided in Bridged LANs offer a set of capabilities that may be used in order to

a) Allow the MAC Service provider to dynamically learn where the recipients of frames addressed to individual MAC Addresses are located;
b) Allow end stations that are the potential recipients of MAC frames destined for group MAC Addresses to dynamically indicate to the MAC Service provider which destination MAC Address(es) they wish to receive;
c) Exercise administrative control over the extent of propagation of specific MAC Addresses.

6.6.3 Users of filtering services

The filtering services provided in Bridged LANs are available to the following users:

a) Network management and administration, for the purposes of applying administrative control. Interactions between administrators of the network and the filtering service provider may be achieved by local means or by means of explicit management mechanisms;
b) End stations, for the purposes of controlling the destination addresses that they will receive. Interactions between end stations and the filtering service provider may be implicit, as is the case with the filtering services provided by the Learning Process (7.8), or by explicit use of filtering service primitives.

6.6.4 Basis of service

All filtering services in Bridged LANs rely on the establishment of filtering rules, and subsequent filtering decisions, that are based on the value(s) contained in the Source or Destination MAC Address fields in MAC frames propagated in the Bridged LAN.

NOTE—The filtering services defined by this standard are based on source address learning and filtering on destination address.

6.6.5 Categories of service

Filtering services in Bridged LANs fall into the following categories:

a) *Basic Filtering Services*. These services are supported by the Forwarding Process (7.7) and by Static Filtering Entries (7.9.1) and Dynamic Filtering Entries (7.9.2) in the Filtering Database. The information contained in the Dynamic Filtering Entries is maintained through the operation of the Learning Process (7.8).
b) *Extended Filtering Services*. These services are supported by the Forwarding Process (7.7), and the Static Filtering Entries (7.9.1) and Group Registration Entries (7.9.3) in the Filtering Database. The information contained in the Group Registration Entries is maintained through the operation of GMRP (10). The categories of Extended Filtering Service are as follows:
    1) Support of dynamic Group forwarding and filtering behavior;
    2) The ability for static filtering information for individual MAC Addresses to specify a subset of Ports for which forwarding or filtering decisions are taken on the basis of dynamic filtering information.

NOTE—Basic Filtering Services as defined in this standard correspond exactly to the filtering capabilities provided by the MAC Bridges standard in its previously published form, ISO/IEC 10038: 1993 [IEEE Std 802.1D, 1993 Edition].

All Bridges shall support Basic Filtering Services. Support of either category of Extended Filtering Services by a Bridge is optional.

6.6.6 Service configuration

In the absence of explicit information in the Filtering Database, the behavior of the Forwarding Process with respect to the forwarding or filtering of frames destined for group MAC Addresses depends upon the categories of service supported by the Bridge.

Basic Filtering Services support the filtering behavior required for regions of a Bridged LAN in which potential recipients of multicast frames exist, but where either the recipients or the Bridges are either unable to support the dynamic configuration of filtering information for those group MAC Addresses, or the recipients have a requirement to receive all traffic destined for group MAC Addresses.

Extended Filtering Services support the filtering behavior required for regions of a Bridged LAN in which potential recipients of multicast frames exist, and where both the potential recipients of frames and the Bridges are able to support dynamic configuration of filtering information for group MAC Addresses. In order to integrate this extended filtering behavior with the needs of regions of the network that support only Basic Filtering Services, Bridges that support Extended Filtering Services can be statically and dynamically configured to modify their filtering behavior on a per-group MAC Address basis, and also on the basis of the overall filtering service provided by each outbound Port with regard to multicast frames. The latter capability permits configuration of the Port's default forwarding or filtering behavior with regard to group MAC Addresses for which no specific static or dynamic filtering information has been configured.

Service configuration provides the ability to configure the overall filtering for the following cases:

a) Bridges that only implement Basic Filtering Services;
b) Bridges that support Extended Filtering Services in a heterogeneous environment, where some equipment is unable to participate in Dynamic Multicast Filtering, or where some equipment (e.g., routers) have specific needs to see unfiltered traffic; and
c) Bridges that support Extended Filtering Services in a homogeneous environment, where all equipment is able to participate in Dynamic Multicast Filtering.

6.6.7 Service definition for Extended Filtering Services

The Filtering Services are described by means of service primitives that define particular types of interaction between MAC Service users and the MAC Service provider across the MAC Service boundary. As these interactions are not defined between peer entities, they are described simply in terms of service requests sent from the MAC Service user to the MAC Service provider.

6.6.7.1 Dynamic registration and de-registration services

These services allow MAC Service users dynamic control over the set of destination Group MAC Addresses that they will receive from the MAC Service provider, by

a) Registering/de-registering membership of specific Groups associated with those addresses;
b) Registering/de-registering their service requirements with regard to the overall forwarding/filtering behavior for Groups.

Provision of these services is achieved by means of GMRP and its associated procedures, as described in Clause 10.

NOTE—The intent of these services is to provide the MAC Service user with dynamic control over access to multicast data streams, for example, multiple video channels made available by a server using a different group MAC Address for each channel. The ability to both register and de-register Group membership, coupled with the filtering action associated with the Group membership, limits the impact of such services on the bandwidth available in the Bridged LAN. These services can be used to control the reception of other categories of multicast traffic, for similar reasons.

REGISTER_GROUP_MEMBER (MAC_ADDRESS)

Indicates to the MAC Service provider that the MAC Service user wishes to receive frames containing the group MAC Address indicated in the MAC_ADDRESS parameter as the destination address. The MAC Addresses that can be carried by this parameter do not include

a) Any individual address;
b) Any of the Reserved Addresses identified in Table 7-9;
c) Any of the GARP Application addresses, as defined in Table 12-1.

DEREGISTER_GROUP_MEMBER (MAC_ADDRESS)

Indicates to the MAC Service provider that the end station no longer wishes to receive frames containing the group MAC Address indicated in the MAC_ADDRESS parameter as the destination address.

REGISTER_SERVICE_REQUIREMENT (REQUIREMENT_SPECIFICATION)

Indicates to the MAC Service provider that the MAC Service user has a requirement for any devices that support Extended Filtering Services to forward frames in the direction of the Mac Service User in accordance with the definition of the service requirement defined by the REQUIREMENT_SPECIFICATION parameter. The values that can be carried by this parameter are

a) Forward All Groups;
b) Forward Unregistered Groups.

DEREGISTER_SERVICE_REQUIREMENT (REQUIREMENT_SPECIFICATION)

Indicates to the MAC Service provider that the MAC Service user no longer has a requirement for any devices that support Extended Filtering Services to forward frames in the direction of the Mac Service User in accordance with the definition of the service requirement defined by the REQUIREMENT_SPECIFICATION parameter. The values that can be carried by this parameter are

a) Forward All Groups;
b) Forward Unregistered Groups.

The use of these services can result in the propagation of group MAC Address and service requirement information across the Spanning Tree, affecting the contents of Group Registration Entries (7.9.3) in Bridges and end stations in the Bridged LAN, and thereby affecting the frame forwarding behavior of the Bridges and end stations with regard to multicast frames.

7. Principles of operation

This clause establishes the principles and a model of the operation of a Bridge as follows:

a) Explains the principal elements of Bridge operation and lists the functions that support these.
b) Establishes an architectural model for a Bridge that governs the provision of these functions.
c) Provides a model of the operation of a Bridge in terms of Processes and Entities that support the functions.
d) Details the addressing requirements in a Bridged LAN and specifies the addressing of Entities in a Bridge.

7.1 Bridge operation

The principal elements of Bridge operation are

a) Relay and filtering of frames.
b) Maintenance of the information required to make frame filtering and relaying decisions.
c) Management of the above.

7.1.1 Relay

A MAC Bridge relays individual MAC user data frames between the separate MACs of the Bridged LANs connected to its Ports. The order of frames shall be preserved as defined in 7.7.3.

The functions that support the relaying of frames and maintain the Quality of Service supported by the Bridge are

a) Frame reception.
b) Discard on received frame in error (6.3.2).
c) Frame discard if the frame_type is not user_data_frame, or if its mac_action parameter is not request_with_no_response (6.4).
d) Regeneration of user priority, if required (6.4).
e) Frame discard following the application of filtering information.
f) Frame discard on transmittable service data unit size exceeded (6.3.8).
g) Forwarding of received frames to other Bridge Ports.
h) Selection of traffic class, following the application of filtering information.
i) Queuing of frames by traffic class.
j) Frame discard to ensure that a maximum bridge transit delay is not exceeded (6.3.6).
k) Selection of queued frames for transmission.
l) Selection of outbound access priority (6.3.9).
m) Mapping of service data units and recalculation of Frame Check Sequence, if required (6.3.7, 7.7.6).
n) Frame transmission.

7.1.2 Filtering and relaying information

A Bridge filters frames, i.e., does not relay frames received by a Bridge Port to other Ports on that Bridge, in order to prevent the duplication of frames (6.3.4). The function that supports the use and maintenance of information for this purpose is

a) Calculation and configuration of Bridged LAN topology.

A Bridge also filters frames in order to reduce traffic in parts of the Bridged LAN that do not lie in the path between the source and destination of that traffic. The functions that support the use and maintenance of information for this purpose are:

b) Permanent configuration of reserved addresses.
c) Explicit configuration of static filtering information.
d) Automatic learning of dynamic filtering information for unicast destination addresses through observation of source addresses of Bridged LAN traffic.
e) Ageing out of dynamic filtering information that has been learned.
f) Automatic addition and removal of dynamic filtering information as a result of GMRP protocol exchanges.

A Bridge classifies frames into traffic classes in order to expedite transmission of frames generated by critical or time-sensitive services. The function that supports the use and maintenance of information for this purpose is

g) Explicit configuration of traffic class information associated with the Ports of the Bridge.

7.1.3 Bridge Management

The functions that support Bridge Management control and monitor the provision of the above functions. They are specified in Clause 14.

7.2 Bridge architecture

7.2.1 Architectural model of a Bridge

Figure 7-1 gives an example of the physical topology of a Bridged LAN. The component LANs are interconnected by means of MAC Bridges; each Port of a MAC Bridge connects to a single LAN. Figure 7-2 illustrates a Bridge with two Ports, and Figure 7-3 illustrates the architecture of such a Bridge.

A Bridge is modeled as consisting of

a) A MAC Relay Entity that interconnects the Bridge's Ports;
b) At least two Ports;
c) Higher layer entities, including at least a Bridge Protocol Entity.

7.2.2 MAC Relay Entity

The MAC Relay Entity handles the MAC method independent functions of relaying frames between Bridge Ports, filtering frames, and learning filtering information. It uses the Internal Sublayer Service provided by the separate MAC Entities for each Port. (The Internal Sublayer Service and its support are described in 6.4 and 6.5.) Frames are relayed between Ports attached to different LANs.

7.2.3 Ports

Each Bridge Port transmits and receives frames to and from the LAN to which it is attached. An individual MAC Entity permanently associated with the Port provides the Internal Sublayer Service used for frame transmission and reception. The MAC Entity handles all the MAC method dependent functions (MAC protocol and procedures) as specified in the relevant standard for that IEEE 802 LAN MAC technology.

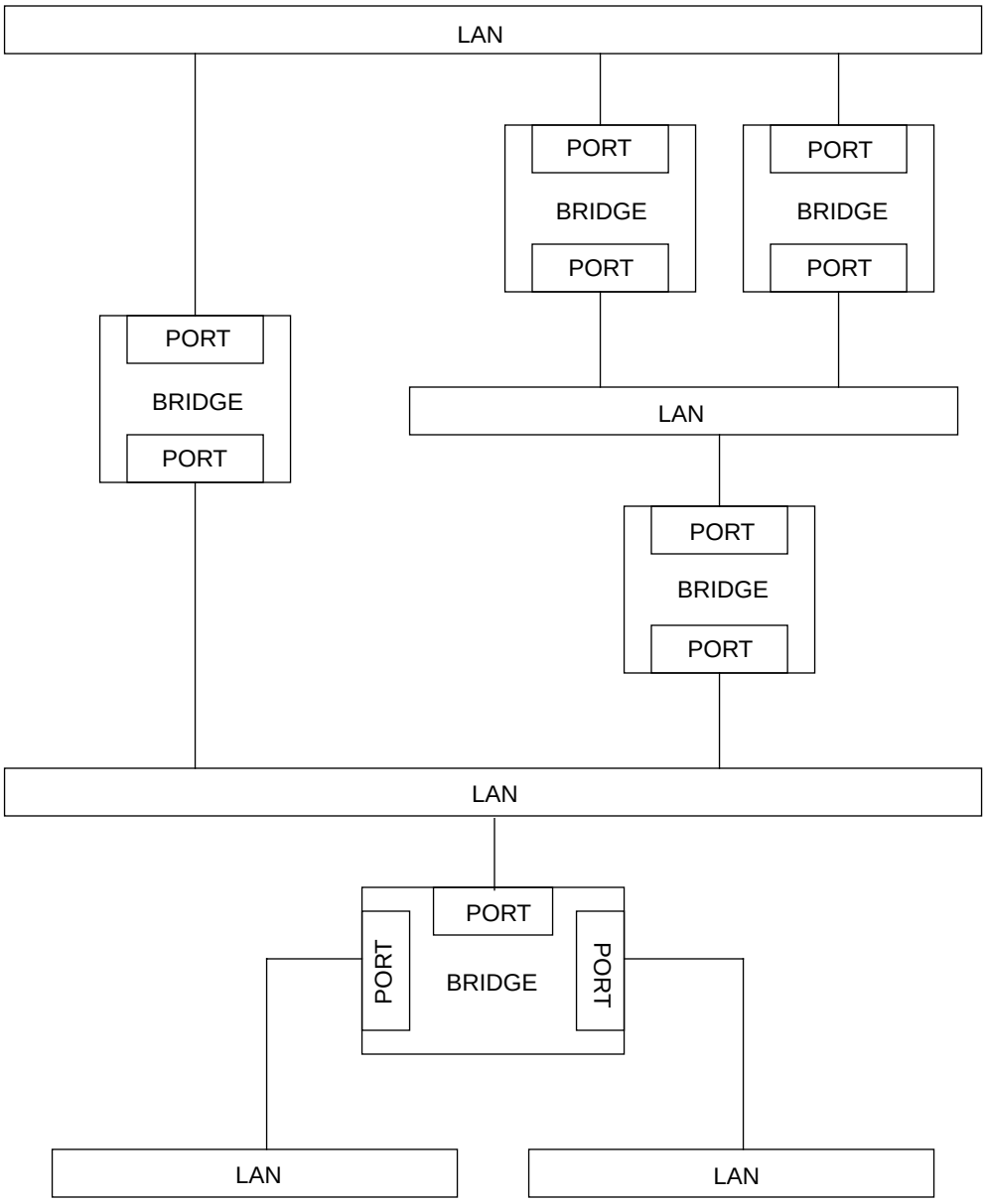


Figure 7-1—A Bridged local area network

7.2.4 Higher Layer Entities

The Bridge Protocol Entity handles calculation and configuration of Bridged LAN topology.

The Bridge Protocol Entity and other higher layer protocol users, such as Bridge Management (7.1.3) and GARP application entities including GARP Participants (Clause 12), make use of Logical Link Control procedures. These procedures are provided separately for each Port, and use the MAC Service provided by the individual MAC Entities.

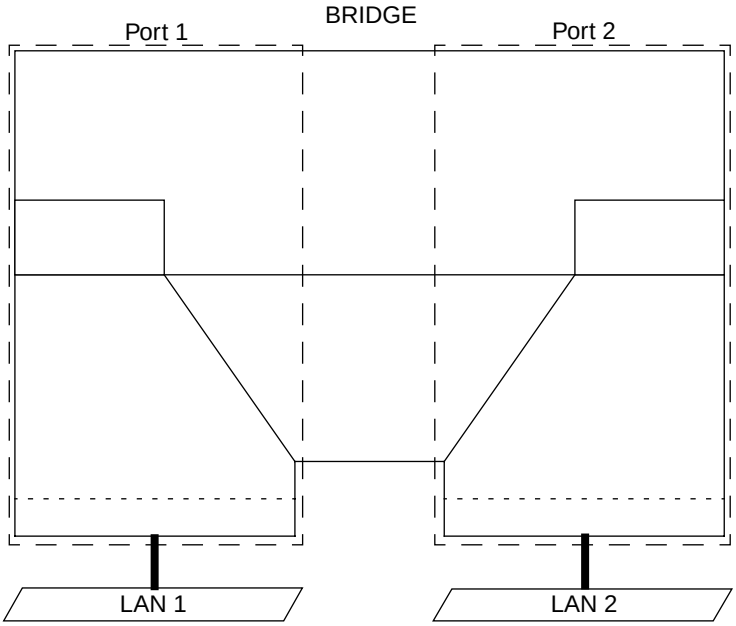


Figure 7-2—Bridge ports

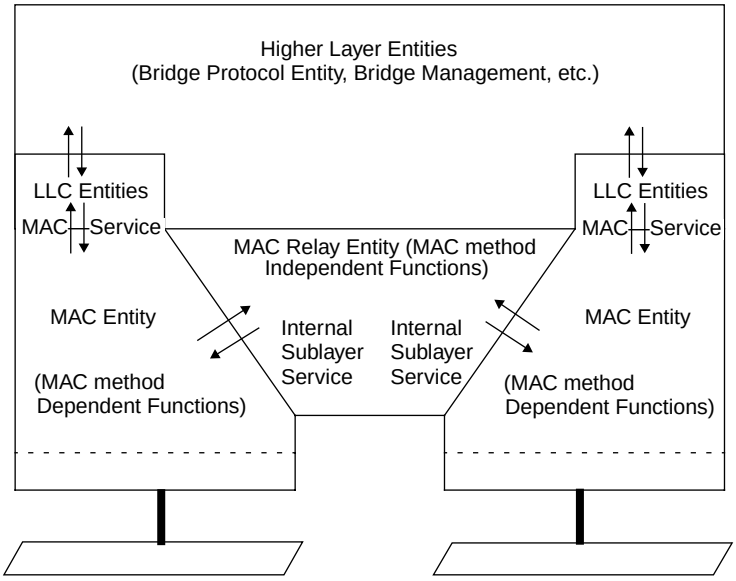


Figure 7-3—Bridge architecture

7.3 Model of operation

The model of operation is simply a basis for describing the functionality of the MAC Bridge. It is in no way intended to constrain real implementations of a MAC Bridge; these may adopt any internal model of operation compatible with the externally visible behavior that this standard specifies. Conformance of equipment to this standard is purely in respect of observable protocol.

Subclauses 7.5 and 7.6 specify the MAC Relay Entity's use of the Internal Sublayer Service. State information associated with each Port governs the Port's participation in the Bridged LAN. (Port States are specified in detail in 8.4.)

Frames are accepted for transmission and delivered on reception to and from Processes and Entities that model the operation of the MAC Relay Entity in a Bridge. These are

a) The Forwarding Process (7.7), which forwards received frames that are to be relayed to other Bridge Ports, filtering frames on the basis of information contained in the Filtering Database (7.9) and on the state of the Bridge Ports (7.4);
b) The Learning Process (7.8), which by observing the source addresses of frames received on each Port, updates the Filtering Database (7.9), conditionally on the Port state (7.4);
c) The Filtering Database (7.9), which holds filtering information and supports queries by the Forwarding Process as to whether frames with given values of the destination MAC Address field should be forwarded to a given Port.

Each Bridge Port also functions as an end station providing the MAC Service to LLC, which, in turn, supports operation of the Bridge Protocol Entity (7.10) and of other possible users of LLC, such as protocols providing Bridge Management (7.11).

Each Bridge Port shall support the operation of LLC Type 1 procedures in order to support the operation of the Bridge Protocol Entity. Bridge Ports may support other types of LLC procedures, which may be used by other protocols.

Figure 7-4 illustrates a single instance of frame relay between the Ports of a Bridge with two Ports.

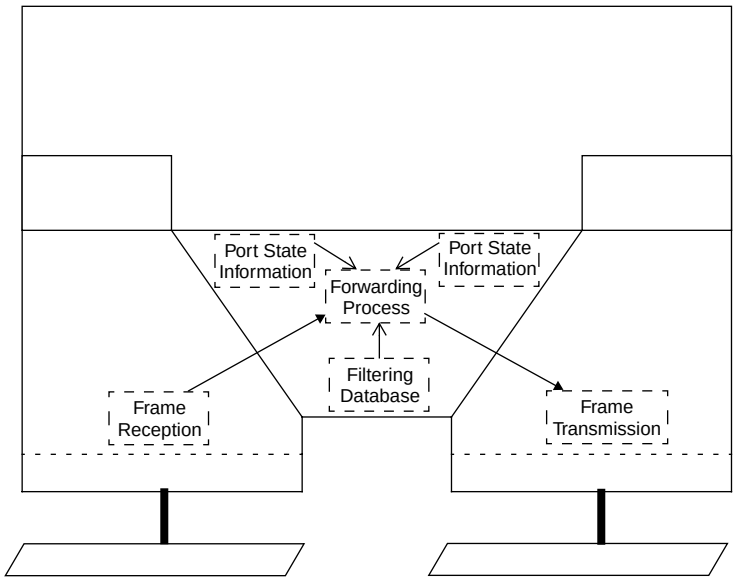


Figure 7-4—Relaying MAC frames

Figure 7-5 illustrates the inclusion of information carried by a single frame, received on one of the Ports of a Bridge with two Ports, in the Filtering Database.

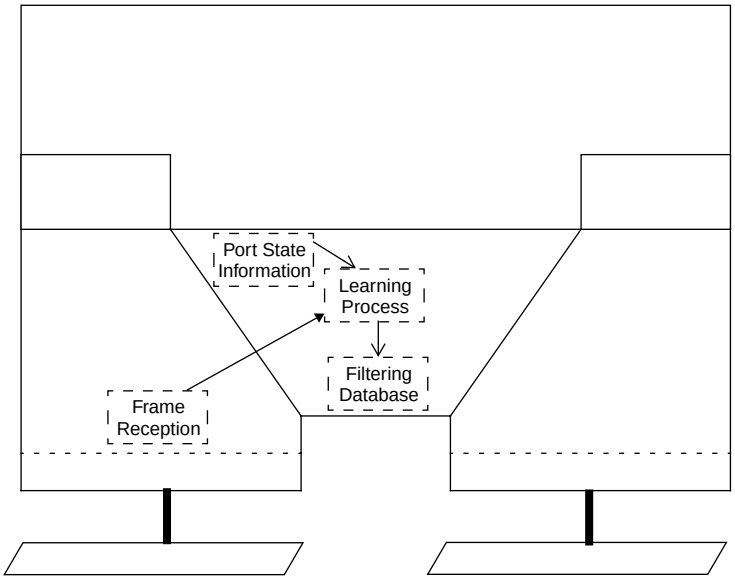


Figure 7-5—Observation of network traffic

Figure 7-6 illustrates the reception and transmission of Bridge Protocol Data Units by the Bridge Protocol Entity.

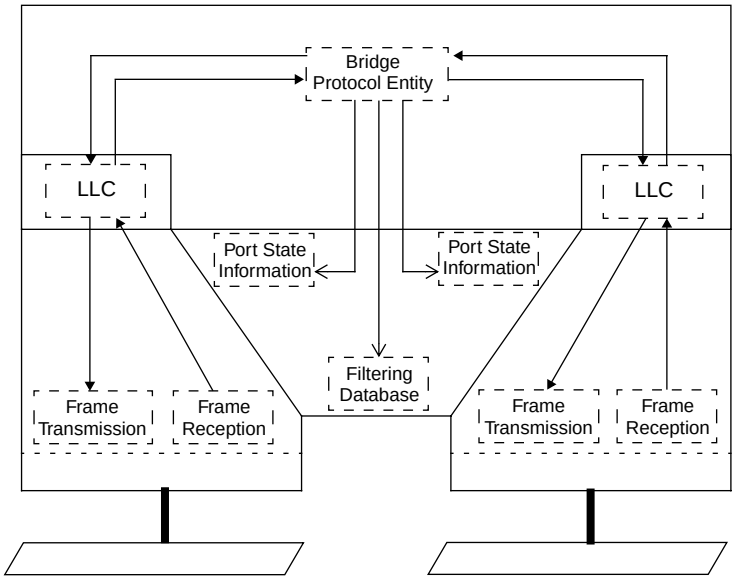


Figure 7-6—Operation of inter-bridge protocol

Figure 7-7 illustrates the reception and transmission of GARP Protocol Data Units by a GARP Protocol Entity (7.10).

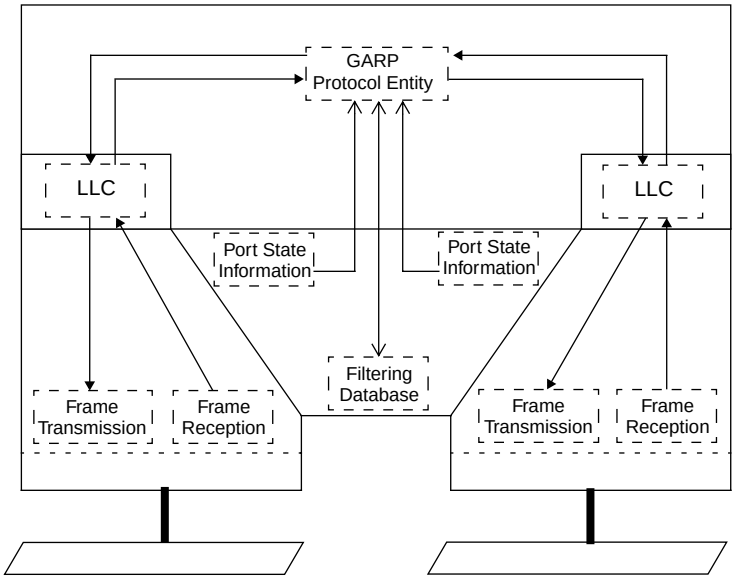


Figure 7-7—Operation of the GARP protocol

7.4 Port States, Active Ports, and the active topology

State information associated with each Bridge Port governs whether or not it participates in relaying MAC frames. A Port can be disabled by management, in which case it plays no part in the operation of the Bridged LAN; a Port that is not disabled can be dynamically excluded from participation in frame relaying by operation of the Spanning Tree algorithm. If neither of these applies to a Port, it is described as *forwarding*.

The *active topology* of a Bridged LAN at any time is the set of communication paths formed by interconnecting the LANs and Bridges by the forwarding Ports. The function of the distributed Spanning Tree algorithm (Clause 8) is to construct an active topology that is simply connected relative to communication between any given pair of MAC Addresses used to address end stations on the LANs.

Figure 7-6 illustrates the operation of the Bridge Protocol Entity, which operates the Spanning Tree Algorithm and its related protocols, and its modification of Port state information as part of determining the active topology of the Bridged LAN. The Port states associated with the determination of the active topology are specified in detail in 8.4.

Figure 7-4 illustrates the Forwarding Process's use of Port state information: first, for a Port receiving a frame, in order to determine whether the received frame is to be relayed through any other Ports; and second, for another Port in order to determine whether the relayed frame is to be forwarded through that particular Port.

The incorporation of end station location information in the Filtering Database by the Learning Process also depends on the active topology. If information associated with frames received on a Port is to be incorporated in the Filtering Database by the Learning Process, then the Port is described as being in a learning state; otherwise, it is in a nonlearning state. Figure 7-5 illustrates the use of the Port state information for a Port receiving a frame, by the Learning Process, in order to determine whether the station location information is to be incorporated in the Filtering Database.

7.5 Frame reception

The individual MAC Entity associated with each Bridge Port examines all frames transmitted on the LAN to which it is attached.

All error-free received frames give rise to M_UNITDATA indication primitives, which shall be handled as follows.

NOTE—A frame that is in error, as defined by the relevant MAC specification, is discarded by the MAC Entity without giving rise to any M_UNITDATA indication; see 6.4.

Frames with M_UNITDATA.indication primitive frame_type and mac_action parameter values of user_data_frame and request_with_no_response, respectively (6.4), shall be submitted to the Learning and Forwarding Processes.

Frames with other values of frame_type and mac_action parameters (e.g., request_with_response and response frames), shall not be submitted to the Forwarding Process. They may be submitted to the Learning Process.

Frames with a frame_type of user_data_frame and addressed to the Bridge Port as an end station shall be submitted to LLC. Such frames carry either the individual MAC Address of the Port or a group address associated with the Port (7.12) in the destination address field. Frames submitted to LLC can also be submitted to the Learning and Forwarding Processes, as specified above.

Frames addressed to a Bridge Port as an end station, and relayed to that Bridge Port from other Bridge Ports in the same Bridge by the Forwarding Process, shall also be submitted to LLC.

No other frames shall be submitted to LLC.

7.5.1 Regenerating user priority

The user_priority of received frames is regenerated using priority information contained in the frame and the User Priority Regeneration Table for the reception Port. For each reception Port, the User Priority Regeneration Table has eight entries, corresponding to the eight possible values of user_priority (0 through 7). Each entry specifies, for the given value of received user_priority, the corresponding Regenerated user_priority value.

NOTE 1—IEEE 802 LAN technologies signal a maximum of 8 user_priority values. Annex H.2 contains further explanation of the use of user_priority values and how they map to traffic classes.

Table 7-1 defines the default values of Regenerated user_priority for the eight possible values of the user_priority parameter received in a data indication; these values shall be used as the initial values of the corresponding entries of the User Priority Regeneration Table for each Port.

Optionally, the ability to modify the values in the User Priority Regeneration Table by management means may be supported, as described in Clause 14. If this capability is provided, the value of the table entries may be independently settable for each reception Port and for each value of received user_priority, and the Bridge may have the capability to use the full range of values in the parameter ranges specified in Table 7-1.

NOTE 2—It is important to ensure that the regeneration and mapping of user priority within the Bridge is consistent with the end-to-end significance attached to that user priority in the Bridged LAN. Within a given Bridge, the values chosen for the User Priority Regeneration Table for a given Port should be consistent with the priority to be associated with traffic received through that Port across the rest of the Bridged LAN, and should generate appropriate access priority values for each MAC method. The user priority value regenerated via the User Priority Regeneration Table on reception is used:

— Via the traffic class table (7.7.3) to determine the traffic class for a given outbound Port, and
— Via fixed, MAC method specific mappings (7.7.5) to determine the access priority that will be used for a given outbound MAC method.

Table 7-1 shows the default values for the regeneration of user priority. Table 7-2 shows the default values for the traffic class table, for all possible numbers of supported traffic classes. Table 7-3 shows the fixed mappings from user priority to access priority that are required for different outbound MAC methods.

Table 7-1—User Priority Regeneration

User Priority	Default Regenerated User Priority	Range
0	0	0–7
1	1	0–7
2	2	0–7
3	3	0–7
4	4	0–7
5	5	0–7
6	6	0–7
7	7	0–7

7.6 Frame transmission

The individual MAC Entity associated with each Bridge Port transmits frames submitted to it by the MAC Relay Entity.

Relayed frames are submitted for transmission by the Forwarding Process. The M_UNITDATA.request primitive associated with such frames conveys the values of the source and destination address fields received in the corresponding M_UNITDATA.indication primitive.

LLC Protocol Data Units are submitted by LLC as a user of the MAC Service provided by the Bridge Port. Frames transmitted to convey such Protocol Data Units carry the individual MAC Address of the Port in the source address field.

Each frame is transmitted subject to the MAC procedures to be observed for that specific IEEE 802 LAN technology. The values of the frame_type and mac_action parameters of the corresponding M_UNITDATA.request primitive shall be user_data_frame and request_with_no_response, respectively (6.5).

Frames transmitted following a request by the LLC user of the MAC Service provided by the Bridge Port shall also be submitted to the MAC Relay Entity.

7.7 The Forwarding Process

Frames submitted to the Forwarding Process after being received at any given Bridge Port (7.5) shall be forwarded through the other Bridge Ports subject to the constituent functions of the Forwarding Process. These functions enforce topology restrictions (7.7.1), use filtering database information to filter frames (7.7.2), queue frames (7.7.3), select queued frames for transmission (7.7.4), map priorities (7.7.5), and recalculate FCS if required (7.7.6).

The Forwarding Process functions are described in 7.7.1–7.7.6 in terms of the action taken for a given frame received on a given Port (termed “the reception Port”). The frame can be forwarded for transmission on some Ports (termed “transmission Ports”), and is discarded without being transmitted at the other Ports.

NOTE—The model of operation of the Forwarding Process described in this standard is limited to the operation of the relay function of the MAC Bridge, and does not take into consideration what may occur in real implementations once frames are passed to the MAC for transmission. In some MAC implementations, and under some traffic conditions, a degree of indeterminacy may be introduced between the modeled description of the process of passing selected frames to the MAC for transmission and the actual sequence of frames as visible on the LAN medium itself. Examples can be found in the handling of access_priority in Token-Passing Bus MACs, or in the effect of different values for Token Holding Time in FDDI LANs. Such indeterminacy could result in apparent violation of the queuing/de-queueing and prioritization rules described for the Forwarding Process, when observing traffic on the medium. As a consequence, in some implementations of this standard, it may prove to be impossible to test conformance to the standard simply by relating observed LAN traffic to the described model of the Forwarding Process; conformance tests would have to allow for the (permissible) behavior of the MAC implementations as well.

Figure 7-4 illustrates the operation of the Forwarding Process in a single instance of frame relay between the Ports of a Bridge with two Ports. Figure 7-8 illustrates the detailed operation of the Forwarding Process.

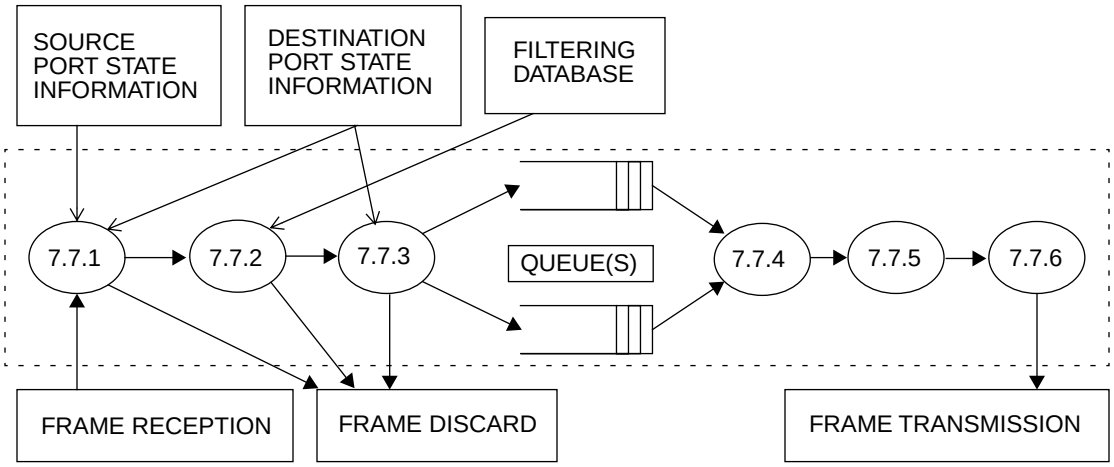


Figure 7-8—Illustration of the detailed operation of the Forwarding Process

7.7.1 Enforcing topology restriction

Each Port is selected as a potential transmission Port if, and only if

a) The Port on which the frame was received was in a forwarding state (8.4), and
b) The Port considered for transmission is in a forwarding state, and
c) The Port considered for transmission is not the same as the Port on which the frame was received, and
d) The size of the mac_service_data_unit conveyed by the frame does not exceed the maximum size of mac_service_data_unit supported by the LAN to which the Port considered for transmission is attached.

For each Port not selected as a potential transmission Port, the frame shall be discarded.

7.7.2 Filtering frames

Filtering decisions are taken by the Forwarding Process on the basis of

a) The destination MAC Address carried in a received frame;
b) The information contained in the Filtering Database for that MAC Address and reception Port;
c) The default Group filtering behavior for the potential transmission Port (7.9.4).

For each potential transmission Port selected as in 7.7.1, the frame shall be forwarded, or discarded (i.e., filtered), on the basis of this information, in accordance with the definition of the Filtering Database entry types (7.9.1, 7.9.2, and 7.9.3). The required forwarding and filtering behavior is summarized in 7.9.4, 7.9.5, Table 7-5, Table 7-6, and Table 7-7.

7.7.3 Queuing frames

The Forwarding Process provides storage for queued frames, awaiting an opportunity to submit these for transmission to the individual MAC Entities associated with each Bridge Port. The order of frames received on the same Bridge Port shall be preserved for

a) Unicast frames with a given user_priority for a given combination of destination_address and source_address;
b) Multicast frames with a given user_priority for a given destination_address.

The Forwarding Process may provide more than one transmission queue for a given Bridge Port. Frames are assigned to storage queue(s) on the basis of their user_priority using a traffic class table that is part of the state information associated with each Port. The table indicates, for each possible value of user_priority, the corresponding value of traffic class that shall be assigned. Values of user_priority range from 0 through 7. Queues correspond one-to-one with traffic classes.

NOTE 1—Annex H.2 contains further explanation of the use of user_priority values and how they map to traffic classes.

For management purposes, up to eight traffic classes are supported by the traffic class tables in order to allow for separate queues for each level of user_priority. Traffic classes are numbered 0 through N-1, where N is the number of traffic classes associated with a given outbound Port. Management of traffic class information is optional. Traffic class 0 corresponds to nonexpedited traffic; nonzero traffic classes are expedited classes of traffic.

NOTE 2—In a given Bridge, it is permissible to implement different numbers of traffic classes for each Port. Ports associated with MAC methods that support a single transmission priority, such as CSMA/CD, can support more than one traffic class.

Where the Forwarding Process does not support expedited classes of traffic for a given Port, in other words, where there is a single traffic class associated with the Port, all values of user_priority map to traffic class 0. In bridges that support expedited traffic, the recommended mapping of user_priority to traffic class, for the number of traffic classes implemented, is as shown in Table 7-2. Each entry in the body of the table is the traffic class assigned to traffic with a given user_priority, for a given number of available traffic classes.

A frame queued by the Forwarding Process for transmission on a Port shall be removed from that queue on submission to the individual MAC Entity for that Port. No further attempt shall be made to transmit the frame on that Port even if the transmission is known to have failed.

A frame queued by the Forwarding Process for transmission on a Port can be removed from that queue, and not subsequently transmitted, if the time for which buffering is guaranteed has been exceeded for that frame.

A frame queued for transmission on a Port shall be removed from that queue if that is necessary to ensure that the maximum bridge transit delay (6.3.6) will not be exceeded at the time at which the frame would subsequently be transmitted.

Table 7-2—Recommended user priority to traffic class mappings

		Number of Available Traffic Classes							
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
User Priority	**0 (Default)**	0	0	0	1	1	1	1	2
	1	0	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0	1
	3	0	0	0	1	1	2	2	3
	4	0	1	1	2	2	3	3	4
	5	0	1	1	2	3	4	4	5
	6	0	1	2	3	4	5	5	6
	7	0	1	2	3	4	5	6	7

NOTE—The rationale behind the choice of values shown in this table is discussed in H.2. A consequence of the mapping shown is that frames carrying the default user priority are given preferential treatment relative to user priority 1 and 2 in Bridges that implement four or more Traffic Classes.

A frame queued for transmission on a Port shall be removed from that queue if the associated Port leaves the forwarding state.

Removal of a frame from a queue for any particular Port does not of itself imply that it shall be removed from a queue for transmission on any other Port.

7.7.4 Selecting frames for transmission

The following algorithm shall be supported by all Bridges as the default algorithm for selecting frames for transmission:

a) For each Port, frames are selected for transmission on the basis of the traffic classes that the Port supports. For a given supported value of traffic class, frames are selected from the corresponding queue for transmission only if all queues corresponding to numerically higher values of traffic class supported by the Port are empty at the time of selection;

b) For a given queue, the order in which frames are selected for transmission shall maintain the ordering requirement specified in 7.7.3.

Additional algorithms, selectable by management means, may be supported as an implementation option, so long as the requirements of 7.7.3 are met.

7.7.5 Mapping priority

The user_priority parameter in an M_UNITDATA.request primitive (6.4) shall be equal to the user_priority parameter in the corresponding data indication.

The mapping of user_priority to outbound access_priority is achieved via fixed, MAC method-specific mappings. The access_priority parameter in an M_UNITDATA.request primitive (6.4) shall be determined from the user_priority in accordance with the values shown in Table 7-3 for the MAC method that will receive the data request. The values shown in Table 7-3 are not modifiable by management or other means.

Table 7-3—Outbound access priorities

user_priority	Outbound Access Priority per MAC method								
	802.3	**8802-4**	**8802-5 (default)**	**8802-5 (alternate)**	**8802-6**	**802.9a[a]**	**8802.11**	**8802-12**	**FDDI**
0	0	0	0	4	0	0	0	0	0
1	0	1	1	4	1	0	0	0	1
2	0	2	2	4	2	0	0	0	2
3	0	3	3	4	3	0	0	0	3
4	0	4	4	4	4	0	0	4	4
5	0	5	5	5	5	0	0	4	5
6	0	6	6	6	6	0	0	4	6
7	0	7	6	6	7	0	0	4	6

[a]In the absence of a definition in 6.5 of support by IEEE Std 802.9a-1995, it is assumed that for this MAC method, access priority 0 will map to “low.”

The table shows two columns for the 8802-5 MAC method. The mapping in the column marked “8802-5 (alternate)” is included in order to permit backwards compatibility with equipment manufactured in accordance with ISO/IEC 10038: 1993; however, the use of this mapping reduces the number of available access priority values to three. For this reason, it is recommended that the column marked “8802-5 (default)” be supported as the default mapping where backward compatibility is not an issue.

7.7.6 Recalculating FCS

Where a frame is being forwarded between two individual MAC Entities of the same IEEE 802 LAN type, and relaying the frame involves no changes to the data that is within the FCS coverage, the FCS received in the M_UNITDATA.indication primitive may be supplied in the corresponding M_UNITDATA.request primitive and not recalculated (6.3.7).

Where a frame is being forwarded between two individual MAC Entities of different types, recalculation of the FCS is necessary if the differences between the MAC methods is such that an FCS calculated according to the MAC procedures for the destination MAC method would differ from the FCS carried by the received frame, or if relaying the frame involves changes to the data that are within the FCS coverage. Where necessary, the FCS is recalculated according to the specific MAC procedures of the transmitting MAC Entity.

NOTE—There are two possibilities for recreating a valid FCS. The first is to generate a new FCS by algorithmically modifying the received FCS, based on knowledge of the FCS algorithm and the transformations that the frame has undergone between reception and transmission. The second is to rely on the normal MAC procedures to recalculate the

FCS for the outgoing frame. The former approach may be preferable in terms of its ability to protect against increased levels of undetected frame errors. Annex G discusses these possibilities in more detail. The frame_check_sequence parameter of the Internal Sublayer Service (6.4) is able to signal the validity, or otherwise, of the FCS; an unspecified value in this parameter in a data request indicates to the transmitting MAC that the received FCS is no longer valid, and the FCS must therefore be recalculated.

FCS recalculation is necessary if any of the following conditions are true:

a) The algorithm used to determine the FCS differs between the MAC methods used by the two MAC Entities;
b) The FCS coverage differs between the MAC methods used by the two MAC Entities;
c) Relaying the frame between the two MAC entities involves changes to the data that are within the coverage of the FCS.

7.8 The Learning Process

The Learning Process observes the source addresses of frames received on each Port and updates the Filtering Database conditionally on the state of the receiving Port.

Frames are submitted to the Learning Process by the individual MAC Entities associated with each Bridge Port as specified in 7.5.

The Learning Process may deduce the path through the Bridged LAN to particular end stations by inspection of the source address field of received frames. It shall create or update a Dynamic Filtering Entry (7.9, 7.9.2) in the Filtering Database, associating the Port on which the frame was received with the MAC Address in the source address field of the frame, if and only if

a) The Port on which the frame was received is in a state that allows learning (8.4), and
b) The source address field of the frame denotes a specific end station, i.e., is not a group address, and
c) No Static Filtering Entry (7.9, 7.9.1) for the associated MAC Address exists in which the Port Map specifies Forwarding or Filtering for that Port, and
d) The resulting number of entries would not exceed the capacity of the Filtering Database.

If the Filtering Database is already filled up to its capacity, but a new entry would otherwise be made, then an existing entry may be removed to make room for the new entry.

Figure 7-5 illustrates the operation of the Learning Process in the inclusion of station location information carried by a single frame, received on one of the Ports of a Bridge, in the Filtering Database.

7.9 The Filtering Database

The Filtering Database supports queries by the Forwarding Process as to whether frames received by the Forwarding Process from a given reception Port, and with given values of destination MAC Address parameter, are to be forwarded through a given potential transmission Port (7.7.1, 7.7.2). It contains filtering information in the form of filtering entries that are either

a) Static, and explicitly configured by management action; or
b) Dynamic, and automatically entered into the Filtering Database by the normal operation of the bridge and the protocols it supports.

A single entry type, the Static Filtering Entry, represents all static information in the Filtering Database, for individual and for group MAC Addresses. It allows administrative control of

c) Forwarding of frames with particular destination addresses; and
d) The inclusion in the Filtering Database of dynamic filtering information associated with Extended Filtering Services, and use of this information.

The Filtering Database shall contain entries of the Static Filtering Entry type.

Static filtering information is added to, modified, and removed from the Filtering Database only under explicit management control. It shall not be automatically removed by any ageing mechanism. Management of static filtering information may be carried out by use of the remote management capability provided by Bridge Management (7.11) using the operations specified in Clause 14.

Two entry types are used to represent dynamic filtering information. Dynamic Filtering Entries are used to specify the ports on which individual addresses have been learned. They are created and updated by the Learning Process (7.8), and are subject to ageing and removal by the Filtering Database. Group Registration Entries support the registration of group MAC Addresses. They are created, updated, and removed by the GMRP protocol in support of Extended Filtering Services (6.6.5, 7.9.3, and Clause 10). Dynamic filtering information may be read by use of the remote management capability provided by Bridge Management (7.11) using the operations specified in Clause 14.

Both static and dynamic entries comprise

e) A MAC Address specification;
f) A Port Map, with a control element for each outbound Port to specify filtering for the MAC Address specification.

The Filtering Services supported by a Bridge (Basic and Extended Filtering Services) determine the default behavior of the Bridge with respect to the forwarding of frames destined for group MAC Addresses. In Bridges that support Extended Filtering Services, the default forwarding behavior of each Port for group MAC Addresses can be configured both statically and dynamically by means of Static Filtering Entries and/or Group Registration Entries that can carry the following MAC Address specifications:

g) All Group Addresses, for which no more specific Static Filtering Entry exists;
h) All Unregistered Group Addresses (i.e., all group MAC Addresses for which no Group Registration Entry exists), for which no more specific Static Filtering Entry exists.

NOTE—The All Group Addresses specification (item g above), when used in a Static Filtering Entry with an appropriate control specification, provides the ability to configure a Bridge that supports Extended Filtering Services to behave as a Bridge that supports only Basic Filtering Services on some or all of its Ports. This might be done for the following reasons:

— The Ports concerned serve “legacy” devices that wish to receive multicast traffic, but are unable to register Group membership;
— The Ports concerned serve devices that need to receive all multicast traffic, such as routers or diagnostic devices.

The Filtering Database shall support the creation, updating, and removal of Dynamic Filtering Entries by the Learning Process (7.8). In Bridges that support Extended Filtering Services, the Filtering Database shall support the creation, updating, and removal of Group Registration Entries by GMRP (Clause 10).

Figure 7-4 illustrates the use of the Filtering Database by the Forwarding Process in a single instance of frame relay between the Ports of a Bridge with two Ports.

Figure 7-5 illustrates the creation or update of a dynamic entry in the Filtering Database by the Learning Process.

Figure 7-6 illustrates the operation of the Bridge Protocol Entity (7.10), which operates the Spanning Tree Algorithm and Protocol, and its notification of the Filtering Database of changes in active topology signaled by that protocol.

7.9.1 Static Filtering Entries

A Static Filtering Entry specifies

a) A MAC Address specification, comprising
   1) An Individual MAC Address; or
   2) A Group MAC Address; or
   3) All Group Addresses, for which no more specific Static Filtering Entry exists; or
   4) All Unregistered Group Addresses, for which no more specific Static Filtering Entry exists.
b) A Port Map, containing a control element for each outbound Port, specifying that a frame with a destination MAC Address that meets this specification is to be
   1) Forwarded, independently of any dynamic filtering information held by the Filtering Database; or
   2) Filtered, independently of any dynamic filtering information; or
   3) Forwarded or filtered on the basis of dynamic filtering information, or on the basis of the default Group filtering behavior for the outbound Port (7.9.4) if no dynamic filtering information is present specifically for the MAC Address.

All Bridges shall have the capability to support the first two values for the MAC Address specification, and the first two values for each control element for all Static Filtering Entries (i.e., shall have the capability to support a1, a2, b1, and b2 above).

A Bridge that supports Extended Filtering Services shall have the capability to support all four values for the MAC Address specification and all three control element values for Static Filtering Entries that specify group MAC Addresses, and may have the capability to support all three control element values for Static Filtering Entries that specify individual MAC Addresses (i.e., shall have the capability to support a1 through a4, and may have the capability to support b3, in addition to support of b1 and b2).

For a given MAC Address specification, a separate Static Filtering Entry with a distinct Port Map may be created for each inbound Port from which frames are received by the Forwarding Process.

In addition to controlling the forwarding of frames, Static Filtering Entries for group MAC Addresses provide the Registrar Administrative Control values for the GMRP protocol (10, 12, 12.9.1). Static configuration of forwarding of specific group addressed frames to an outbound port indicates Registration Fixed on that port: a desire to receive frames addressed to that Group even in the absence of dynamic information. Static configuration of filtering of frames which might otherwise be sent to an outbound port indicates Registration Forbidden. The absence of a Static Filtering Entry for the group address, or the configuration of forwarding or filtering on the basis of dynamic filtering information, indicates Normal Registration.

NOTE—The possibility of configuring a number of Static Filtering Entries, each for a different inbound port or ports, can appear to complicate registration controls. Group registration is propagated across the Bridge to the inbound Port, and that Port acts as a GMRP Applicant if any part of the propagated information indicates a desire to receive frames for the Group. Such frames will be filtered from Ports that do not wish to receive them, and correct operation is maintained.

7.9.2 Dynamic Filtering Entries

A Dynamic Filtering Entry specifies

a) An individual MAC Address;

b) A Port Map consisting of a control element that specifies forwarding of frames destined for that MAC Address to a single Port.

NOTE 1—This is equivalent to specifying a single port number; hence, this specification is directly equivalent to the specification of dynamic entries in ISO/IEC 10038: 1993.

Dynamic Filtering Entries are created and updated by the Learning Process (7.8). They shall be automatically removed after a specified time, the Ageing Time, has elapsed since the entry was created or last updated. No more than one Dynamic Filtering Entry shall be created in the Filtering Database for a given MAC Address.

A Dynamic Filtering Entry shall not be created or updated by the Learning Process if any Static Filtering Entry already exists for this MAC Address with a control element specification, for the outbound Port specified by the Learning Process, that specifies forwarding or filtering irrespective of dynamic filtering information.

NOTE 2—For Bridges that do not permit the (optional) ability of Static Filtering Entries to specify forwarding or filtering on the basis of dynamic filtering information (see 7.9.1), including Bridges that conform to ISO/IEC 10038: 1993, this effectively prevents the creation of Dynamic Filtering Entries where a Static Filtering Entry exists for the same MAC Address. This in turn ensures that these Bridges continue to conform to their own specification, which prohibits creation of a Dynamic Filtering Entry if a Static Filtering Entry already exists.

For Bridges that do permit the ability of Static Filtering Entries to specify forwarding or filtering on the basis of dynamic filtering information, it is possible for Dynamic and Static Filtering Entries to exist for the same MAC Address, as long as the address is not learned on a Port for which there is a Static Filtering Entry that specifies “forwarding or filtering independently of any dynamic filtering information.”

The facility provided by this updated specification allows source address learning to be confined to a subset of Ports.

Dynamic Filtering Entries cannot be created or updated by management.

If a Dynamic Filtering Entry exists for a given MAC Address, creation or updating of a Static Filtering Entry for the same address causes removal of any conflicting information that may be contained in the Dynamic Filtering Entry. If removal of such conflicting information would result in a Port Map that does not specify Forwarding on any Port, then that Dynamic Filtering Entry is removed from the Filtering Database.

The ageing out of Dynamic Filtering Entries ensures that end stations that have been moved to a different part of the Bridged LAN will not be permanently prevented from receiving frames. It also takes account of changes in the active topology of the Bridged LAN that can cause end stations to appear to move from the point of view of the bridge; i.e., the path to those end stations subsequently lies through a different Bridge Port.

The Ageing Time may be set by management (Clause 14). A range of applicable values and a recommended default is specified in Table 7-4; this is suggested to remove the need for explicit configuration in most cases. If the value of Ageing Time can be set by management, the Bridge shall have the capability to use values in the range specified, with a granularity of 1 s.

Table 7-4—Ageing time parameter value

Parameter	Recommended default value	Range
Ageing time	300.0 s	10.0–1 000 000.0 s

NOTE 3—The granularity is specified here in order to establish a common basis for the granularity expressed in the management operations defined in Clause 14, not to constrain the granularity of the actual timer supported by a conformant implementation. If the implementation supports a granularity other than 1 second, then it is possible that the value read back by management following a Set operation will not match the actual value expressed in the Set.

The Spanning Tree Algorithm and Protocol specified in Clause 8 includes a procedure for notifying all Bridges in the Bridged LAN of topology change. It specifies a short value for the Ageing Timer, to be enforced for a period after any topology change (see 8.3.5). While the topology is not changing, this procedure allows normal ageing to accommodate extended periods during which addressed end stations do not generate frames themselves, perhaps through being powered down, without sacrificing the ability of the Bridged LAN to continue to provide service after automatic configuration.

7.9.3 Group Registration Entries

A Group Registration Entry specifies

a) A MAC Address specification, comprising
   1) A Group MAC Address; or
   2) All Group Addresses, for which no more specific Static Filtering Entry exists; or
   3) All Unregistered Group Addresses, for which no more specific Static Filtering Entry exists.
b) A Port Map consisting of a control element for each outbound Port that specifies forwarding or filtering of frames destined to the MAC Address.

Group Registration Entries are created, modified, and deleted by the operation of GMRP (Clause 10). No more than one Group Registration Entry shall be created in the Filtering Database for a given MAC Address specification.

NOTE—It is possible to have a Static Filtering Entry which has values of Forward or Filter on some or all Ports that mask the dynamic values held in a corresponding Group Registration Entry. The values in the Group Registration Entry will continue to be updated by GMRP; hence, subsequent modification of that entry to allow the use of dynamic filtering information on one or more Ports immediately activates the true GMRP registration state that was hitherto masked by the static information.

7.9.4 Default Group filtering behavior

Forwarding and filtering of group addressed frames may be managed by specifying defaults for each outbound Port. The behavior of each of these defaults, as modified by the control elements of more explicit Filtering Database entries applicable to a given frame's MAC Address, reception Port, and outbound Port, is as follows.

NOTE 1—As stated in 7.9.1, a Bridge may optionally provide the capability to create separate Static Filtering Entries with a distinct Port Map for each reception Port. If this capability is not provided, then a given Static Filtering Entry applies to all reception Ports.

a) *Forward All Groups*. The frame is forwarded, unless an explicit Static Filtering Entry specifies filtering independent of any dynamic filtering information.
b) *Forward Unregistered Groups*. The frame is forwarded, unless
   1) An explicit Static Filtering Entry specifies filtering independent of any dynamic filtering information; or
   2) An explicit Static Filtering Entry specifies forwarding or filtering on the basis of dynamic filtering information, and an applicable explicit Group Registration Entry exists specifying filtering; or
   3) An applicable explicit Static Filtering Entry does not exist, but an applicable Group Registration entry specifies filtering.
c) *Filter Unregistered Groups*. The frame is filtered unless

1) An explicit Static Filtering Entry specifies forwarding independent of any dynamic filtering information; or
2) An explicit Static Filtering Entry specifies forwarding or filtering on the basis of dynamic filtering information, and an applicable explicit Group Registration Entry exists specifying forwarding; or
3) An applicable explicit Static Filtering Entry does not exist, but an applicable Group Registration entry specifies forwarding.

In Bridges that support only Basic Filtering Services, the default Group filtering behavior is Forward All Groups for all Ports of the Bridge.

NOTE 2—Forward All Groups corresponds directly to the behavior specified in ISO/IEC 10038: 1993 when forwarding group MAC Addressed frames for which no static filtering information exists in the Filtering Database. Forward All Groups makes use of information contained in Static Filtering Entries for specific group MAC Addresses, but overrides any information contained in Group Registration Entries. Forward Unregistered Groups is analogous to the forwarding behavior of a Bridge with respect to individual MAC Addresses; if there is no static or dynamic information for a specific group MAC Address, then the frame is forwarded, otherwise, the frame is forwarded in accordance with the statically configured or dynamically learned information.

In Bridges that support Extended Filtering Services, the default Group filtering behavior for each outbound Port is determined by the following information contained in the Filtering Database:

d) Any Static Filtering Entries applicable to the reception Port with a MAC Address specification of All Group Addresses or All Unregistered Group Addresses;
e) Any Group Registration Entries applicable to the reception Port with a MAC Address specification of All Group Addresses or All Unregistered Group Addresses.

The means whereby this information determines the default Group filtering behavior is specified in 7.9.5, Table 7-6 and Table 7-7.

NOTE 3—The result is that the default Group filtering behavior can be configured for each Port of the Bridge via Static Filtering Entries, determined dynamically via Group Registration Entries created/updated by GMRP (Clause 10), or both. For example, in the absence of any static or dynamic information in the Filtering Database for All Group Addresses or All Unregistered Group Addresses, the default Group filtering behavior will be Filter Unregistered Groups on all Ports. Subsequently, the creation of a Dynamic Group Registration Entry for All Unregistered Group Addresses indicating “Registered” on a given Port would cause that Port to exhibit Forward Unregistered Groups behavior. Similarly, creating a Static Filtering Entry for All Group Addresses indicating “Registration Fixed” on a given Port would cause that Port to exhibit Forward All Groups behavior.

Hence, by using appropriate combinations of “Registration Fixed,” “Registration Forbidden,” and “Normal Registration” in the Port Maps of Static Filtering Entries for the All Group Addresses and All Unregistered Group Addresses address specifications, it is possible, for a given Port, to

— Fix the default Group filtering behavior to be just one of the three behaviors described above; or
— Restrict the choice of behaviors to a subset of the three, and allow GMRP registrations (or their absence) to determine the final choice; or
— Allow any one of the three behaviors to be adopted, in accordance with any registrations received via GMRP.

7.9.5 Querying the Filtering Database

Each entry in the Filtering Database comprises

a) A MAC Address specification;
b) A Port Map, with a control element for each outbound Port.

A given individual MAC Address specification can be included in the Filtering Database in a Static Filtering Entry, a Dynamic Filtering Entry, both, or neither. Table 7-5 combines Static Filtering Entry and Dynamic

Filtering Entry information for an individual MAC Address to specify forwarding, or filtering, of a frame with that destination address through an outbound Port.

Table 7-5—Combining Static and Dynamic Filtering Entries for an individual MAC Address

Filtering Information	Static Filtering Entry Control Element for this individual MAC Address and Port specifies:				
	Forward	Filter	Use Dynamic Filtering Information, or no Static Filtering Entry present. Dynamic Filtering Entry Control Element for this individual MAC Address and Port specifies:		
			Forward	Filter	No Dynamic Filtering Entry present
Result	Forward	Filter	Forward	Filter	Forward

Table 7-6 specifies the result, Registered or Not Registered, of combining a Static Filtering Entry and a Group Registration Entry for the “All Group Addresses” address specification, and for the “All Unregistered Group Addresses” address specification.

Table 7-6—Combining Static Filtering Entry and Group Registration Entry for “All Group Addresses” and “All Unregistered Group Addresses”

Filtering Information	Static Filtering Entry Control Element for this group MAC Address and Port specifies:				
	Registration Fixed (Forward)	Registration Forbidden (Filter)	Use Group Registration Information, or no Static Filtering Entry present. Group Registration Entry Control Element for this group MAC Address and Port specifies:		
			Registered (Forward)	Not Registered (Filter)	No Group Registration Entry present
Result	Registered	Not Registered	Registered	Not Registered	Not Registered

Table 7-7 combines Static Filtering Entry and Group Registration Entry information for a specific group MAC Address with the Table 7-6 results for All Group Addresses and All Unregistered Group Addresses to specify forwarding, or filtering, of a frame with that destination group MAC Address through an outbound Port.

7.9.6 Permanent Database

The Permanent Database provides fixed storage for a number of Static Filtering Entries. The Filtering Database shall be initialized with the Filtering Database Entries contained in this fixed data store.

Table 7-7—Forwarding or Filtering for specific group MAC Addresses

All Group Addresses control elements for this Port specify (Table 7-6):	All Unregistered Group Addresses control elements for this Port specify (Table 7-6):	Static Filtering Entry Control Element for this group MAC Address and Port specifies:				
		Registration Fixed (Forward)	Registration Forbidden (Filter)	Use Group Registration Information, or no Static Filtering Entry present. Group Registration Entry Control Element for this group MAC Address and Port specifies:		
				Registered (Forward)	Not Registered (Filter)	No Group Registration Entry present
Not Registered	Not Registered	Forward	Filter	Forward	Filter	Filter (Filter Unregistered Groups)
	Registered	Forward	Filter	Forward	Filter	Forward (Forward Unregistered Groups)
Registered		Forward	Filter	Forward (Forward All Groups)	Forward (Forward All Groups)	Forward (Forward All Groups)

Entries can be added to and removed from the Permanent Database under explicit management control, using the management functionality defined in Clause 14. Changes to the contents of Static Filtering Entries in the Permanent Database do not affect forwarding and filtering decisions taken by the Forwarding Process until such a time as the Filtering Database is reinitialized.

NOTE—This aspect of the Permanent Database can be viewed as providing a “boot image” for the Filtering Database, defining the contents of all initial entries, before any dynamic filtering information is added.

7.10 Bridge Protocol Entity and GARP Protocol Entities

The Bridge Protocol Entity operates the Spanning Tree Algorithm and Protocol.

The Bridge Protocol Entities of Bridges attached to a given individual LAN in a Bridged LAN communicate by exchanging Bridge Protocol Data Units (BPDUs).

Figure 7-6 illustrates the operation of the Bridge Protocol Entity including the reception and transmission of frames containing BPDUs, the modification of the state information associated with individual Bridge Ports, and notification of the Filtering Database of changes in active topology.

The GARP Protocol Entities operate the Algorithms and Protocols associated with the GARP Applications supported by the Bridge, and consist of the set of GARP Participants for those GARP Applications (12.3, Clause 10).

The GARP Protocol Entities of Bridges attached to a given individual LAN in a Bridged LAN communicate by exchanging GARP Protocol Data Units (GARP PDUs).

Figure 7-7 illustrates the operation of a GARP Protocol Entity including the reception and transmission of frames containing GARP PDUs, the use of control information contained in the Filtering Database, and notification of the Filtering Database of changes in filtering information.

7.11 Bridge Management

Remote management facilities may be provided by the Bridge. Bridge Management is modeled as being performed by means of the Bridge Management Entity. The facilities provided by Bridge Management, and the operations that support these facilities, are specified in Clause 14.

Clause 15 specifies the protocol operations, identifiers, and values to be used in realizing these management operations through the use of ISO/IEC 15802-2.

Bridge Management protocols use the Service provided by the operation of LLC Procedures, which use the MAC Service provided by the Bridged LAN.

7.12 Addressing

All MAC Entities communicating across a Bridged LAN shall use 48-bit addresses. These may be Universally Administered Addresses, Locally Administered Addresses, or a combination of both.

7.12.1 End stations

Frames transmitted between end stations using the MAC Service provided by a Bridged LAN carry the MAC Address of the source and destination peer end stations in the source and destination address fields of the frames, respectively. The address, or other means of identification, of a Bridge is not carried in frames transmitted between peer users for the purpose of frame relay in the Bridged LAN.

The broadcast address and other group MAC Addresses apply to the use of the MAC Service provided by a Bridged LAN as a whole. In the absence of explicit filters configured via management as Static Filtering Entries, or via GMRP as Group Registration Entries (Clause 14, Clause 10, 7.9), frames with such destination addresses are relayed throughout the Bridged LAN.

7.12.2 Bridge Ports

The individual MAC Entity associated with each Bridge Port shall have a separate individual MAC Address. This address is used for any MAC procedures required by the particular MAC method employed.

Frames that are received from the LAN to which a Port is attached and that carry a MAC Address for the Port in the destination address field are submitted to the MAC Service User (LLC) exactly as for an end station.

7.12.3 Bridge Protocol Entities and GARP Protocol Entities

Bridge Protocol Entities only receive and transmit BPDUs. These are only received and transmitted from other Bridge Protocol Entities (or where two Bridge Ports are connected to the same LAN, to and from themselves).

GARP Protocol Entities only receive and transmit GARP PDUs (12.11) that are formatted according to the requirements of the GARP Applications they support. These are only received and transmitted from other GARP Protocol Entities.

A Bridge Protocol Entity or a GARP Protocol Entity uses the DL_UNITDATA.request primitive (see ISO/IEC 8802-2) provided by the individual LLC Entities associated with each active Bridge Port to transmit BPDUs or GARP PDUs. Each PDU is transmitted on one selected Bridge Port. PDUs are received through corresponding DL_UNITDATA.indication primitives. The source_address and destination_address parameters of the DL_UNITDATA.request primitive shall both denote the standard LLC address assigned to the Bridge Spanning Tree Protocol. This identifies the Bridge Protocol Entity and the GARP Protocol Entity among other users of LLC.

Each DL_UNITDATA.request primitive gives rise to the transmission of an LLC UI command PDU, which conveys the BPDU or GARP PDU in its information field. The source and destination LLC address fields are set to the values supplied in the request primitive.

The value assigned to the Bridge Spanning Tree Protocol LLC address is given in Table 7-8.[9]

Table 7-8—Standard LLC address assignment

Assignment	Value
Bridge spanning tree protocol	01000010

Code Representation: The least significant bit of the value shown is the right-most. The bits increase in significance from right to left. It should be noted that the code representation used here has been chosen in order to maintain consistency with the representation used elsewhere in this standard; however, it differs from the representation used in ISO/IEC TR 11802-1: 1997.

This standard defines a Protocol Identifier field, present in all BPDUs (Clause 9) and GARP PDUs (12.11), which serves to identify different protocols supported by Bridge Protocol Entities and GARP Protocol Entities, within the scope of the LLC address assignment. This standard specifies a single value of the Protocol Identifier in Clause 9 for use in BPDUs. This value serves to identify BPDUs exchanged between Bridge Protocol Entities operating the Spanning Tree Algorithm and Protocol specified in Clause 8. A second value of this protocol identifier for use in GARP PDUs is defined in 12.11. This value serves to identify GARP PDUs exchanged between GARP Participants operating the GARP protocol specified in Clause 12. Further values of this field are reserved for future standardization.

A Bridge Protocol Entity or GARP Protocol Entity that receives a BPDU or a GARP PDU with an unknown Protocol Identifier shall discard that PDU.

A Bridge Protocol Entity that operates the Spanning Tree Algorithm and Protocol specified in Clause 8 always transmits BPDUs addressed to all other Bridge Protocol Entities attached to the LAN on which the frame containing the BPDU is transmitted. A group address shall be used in the destination address field to address this group of Entities. This group address shall be configured in the Permanent Database (7.12.6) in order to confine BPDUs to the individual LAN on which they are transmitted.

[9]ISO/IEC TR 11802-1: 1997, Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Technical reports and guidelines—Part 1: The structure and coding of Logical Link Control addresses in Local Area Networks, contains the full list of standard LLC address assignments, and documents the criteria for assignment.

A 48-bit Universal Address, known as the Bridge Group Address, has been assigned for this purpose. Its value is specified in Table 7-9. Bridges that use 48-bit Universally Administered Addresses shall use this address in the destination address field of all MAC frames conveying BPDUs.

Table 7-9—Reserved addresses

Assignment	Value
Bridge Group Address	01-80-C2-00-00-00
IEEE Std. 802.3x Full Duplex PAUSE operation	01-80-C2-00-00-01
Reserved for future standardization	01-80-C2-00-00-02
Reserved for future standardization	01-80-C2-00-00-03
Reserved for future standardization	01-80-C2-00-00-04
Reserved for future standardization	01-80-C2-00-00-05
Reserved for future standardization	01-80-C2-00-00-06
Reserved for future standardization	01-80-C2-00-00-07
Reserved for future standardization	01-80-C2-00-00-08
Reserved for future standardization	01-80-C2-00-00-09
Reserved for future standardization	01-80-C2-00-00-0A
Reserved for future standardization	01-80-C2-00-00-0B
Reserved for future standardization	01-80-C2-00-00-0C
Reserved for future standardization	01-80-C2-00-00-0D
Reserved for future standardization	01-80-C2-00-00-0E
Reserved for future standardization	01-80-C2-00-00-0F

A GARP Protocol Entity that

a) Operates the GARP protocol specified in Clause 12; and
b) Supports a given GARP Application,

always transmits GARP PDUs addressed to all other GARP Protocol Entities that

c) Implement the same GARP Application; and
d) Are attached to the LAN on which the frame containing the GARP PDU is transmitted.

A group MAC Address, specific to the GARP Application concerned, shall be used as the destination MAC Address field to address this group of GARP Protocol Entities. A set of 48-bit Universal Addresses, known as GARP Application addresses, has been assigned for that purpose. The values of the GARP Application addresses are defined in Table 12-1. These group MAC Addresses are reserved for assignment to standard protocols, according to the criteria for such assignments (Clause 5.5 of ISO/IEC TR 11802-2).

In Bridges that provide only Basic Filtering Services, the set of GARP Application addresses shall not be configured in the Filtering Database (7.9) or the Permanent Database (7.9.6). In Bridges that provide

Extended Filtering Services, the set of GARP Application addresses shall be configured as Static Filtering Entries in the Filtering Database (7.9.1) and Permanent Database (7.9.6) as follows:

e) GARP Application addresses assigned to GARP Applications that are supported by the Bridge shall be configured in order to confine GARP PDUs for that GARP Application to the individual LAN on which they are transmitted;

f) GARP Application addresses assigned to GARP Applications that are not supported by the Bridge shall not be configured in the Filtering Database or Permanent Database.

Management shall not provide the capability to create, delete, or modify entries in the Permanent or Filtering Databases for any GARP application address.

The source address field of MAC frames conveying BPDUs or GARP PDUs contains the individual MAC Address for the Bridge Port through which the PDU is transmitted (7.12.2).

7.12.4 Bridge Management Entities

Bridge Management Entities transmit and receive protocol data units using the Service provided by the individual LLC Entities associated with each Bridge Port. Each of these in turn uses the MAC Service, which is provided by the individual MAC Entities associated with that Port and supported by the Bridged LAN as a whole.

As a user of the MAC Service provided by a Bridged LAN, the Bridge Management Entity may be attached to any point in the Bridged LAN. Frames addressed to the Bridge Management Entity will be relayed by Bridges if necessary to reach the LAN to which it is attached.

In order to ensure that received frames are not duplicated, the basic requirement in a single LAN or a Bridged LAN that a unique address be associated with each point of attachment shall be met.

A Bridge Management Entity for a specific Bridge is addressed by one or more individual MAC Addresses in conjunction with the higher layer protocol identifier and addressing information. It may share one or more points of attachment to the Bridged LAN with the Ports of the Bridge with which it is associated. It is recommended that it make use of the MAC Service provided by all the MAC Entities associated with each Bridge Port, i.e., that it be reachable through each Bridge Port using frames carrying the individual MAC Address of that Port in the destination address field.

This standard specifies a standard group address for public use that serves to convey management requests to the Bridge Management Entities associated with all Bridge Ports attached to a Bridged LAN. A management request that is conveyed in a MAC frame carrying this address value in the destination address field will generally elicit multiple responses from a single Bridge. This address is known as the All LANs Bridge Management Group Address and takes the value specified in Table 7-10.

Table 7-10—Addressing Bridge Management

Assignment	Value
All LANs Bridge Management Group Address	01-80-C2-00-00-10

7.12.5 Unique identification of a Bridge

A unique 48-bit Universally Administered MAC Address, termed the Bridge Address, shall be assigned to each Bridge. The Bridge Address may be the individual MAC Address of a Bridge Port, in which case use of the address of the lowest numbered Bridge Port (Port 1) is recommended.

NOTE—The Spanning Tree Protocol (Clause 8) requires that a single unique identifier be associated with each Bridge. That identifier is derived from the Bridge Address as specified in 8.5.1.3, 8.5.3.7, and 9.2.5.

7.12.6 Reserved addresses

Frames containing any of the group MAC Addresses specified in Table 7-9 in their destination address field shall not be relayed by the Bridge. They shall be configured in the Permanent Database. Management shall not provide the capability to modify or remove these entries from the Permanent or the Filtering Databases. These group MAC Addresses are reserved for assignment to standard protocols, according to the criteria for such assignments (Clause 5.5 of ISO/IEC TR 11802-2).

7.12.7 Points of attachment and connectivity for Higher Layer Entities

Higher Layer Entities, such as the Bridge Protocol Entity and GARP Protocol Entity (7.10), and Bridge Management (7.11), are modeled as being connected directly to the Bridged LAN via one or more points of attachment. From the point of view of their attachment to the Bridged LAN, Higher Layer Entities associated with a Bridge can be regarded as if they are distinct end stations, directly connected to one or more of the LAN segments served by the Bridge Ports, in the same way as any other end station is connected to the Bridged LAN. In practice, the Higher Layer Entities will, in many cases, share the same physical points of attachment used by the relay function of the Bridge, as stated in 7.12; however, from the point of view of the transmission and reception of frames by these functions, the behavior is the same as if they were contained in logically separate end stations with points of attachment “outside” the Port(s) with which they are associated. Figure 7-9 is functionally equivalent to Figure 7-3, but illustrates this logical separation between the points of attachment used by the Higher Layer Entities and points of attachment used by the MAC Relay Entity.

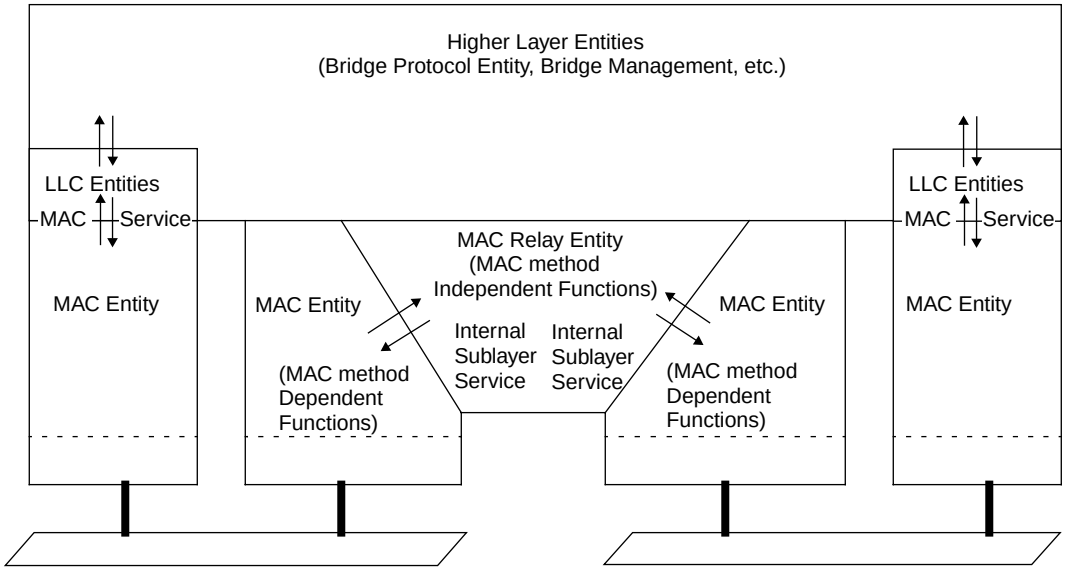


Figure 7-9—Logical separation of points of attachment used by Higher Layer Entities and the MAC Relay Entity

Higher Layer Entities fall into two distinct categories:

a) Those entities, such as the Bridge Management Entity, that require only a single point of attachment to the Bridged LAN;
b) Those entities, such as Bridge Protocol Entities and GARP Participants, that require a point of attachment per Port of the Bridge.

The fundamental distinction between these two categories is that for the latter, it is essential for the operation of the entity concerned that it is able to associate received frames with the LAN segment on which those frames were originally seen by the Bridge, and that it is able to transmit frames to peer entities that are connected directly to that LAN segment. It is therefore essential that

c) It does not receive frames via a point of attachment associated with one Port that have been relayed by the Bridge from other Ports; and
d) Frames that it transmits via one point of attachment are not relayed by the Bridge to any other Ports.

For this reason, the MAC Addresses used to reach entities of this type are permanently configured in the Filtering Database in order to prevent the Bridge from relaying such frames received via any Port to any other Port of the Bridge, as defined in 7.12.3 and 7.12.6.

NOTE—The MAC Addresses used to address such entities are generally group MAC Addresses.

The MAC Relay Entity Bridge forwards a frame received on one Port through the other Port(s) of the Bridge, subject to the following control information permitting such forwarding to take place:

— The Port state information (7.4) associated with the Port on which the frame was received;
— The information held in the Filtering Database (7.9);
— The Port state information (7.4) associated with the Port(s) on which the frame is potentially to be transmitted.

This is illustrated in Figure 7-10, where the control information represented by the Port state and Filtering Database information is represented as a series of switches (shown in the open, disconnected state) inserted in the forwarding path provided by the MAC Relay Entity. For the Bridge to forward a given frame between two Ports, all three switches must be in the closed state. This figure also illustrates that the controls placed in the forwarding path have no effect upon the ability of a Higher Layer Entity to transmit and receive frames directly onto a given LAN segment via the point of attachment to that segment (e.g., from entity A to segment A); they only affect the path taken by any indirect transmission/reception (e.g., from entity A to segment B).

Figure 7-11 illustrates the state of the forwarding path with respect to frames destined for Higher Layer Entities that require per-Port points of attachment. The fact that the Filtering Databases in all Bridges are permanently configured to prevent relay of frames addressed to these entities means that they can receive frames only via their direct points of attachment (i.e., from segment A to entity A, and from segment B to entity B), regardless of the Port states.

Figure 7-12 illustrates the state of the forwarding path with respect to frames destined for a Higher Layer Entity that requires only a single point of attachment, for the case where the Port states and Filtering Database states permit relay of frames. Frames destined for the Higher Layer Entity that originate on LAN segment B are relayed by the Bridge, and are both received by the entity and transmitted on LAN segment A.

Figure 7-13 illustrates the state of the forwarding path with respect to frames destined for a Higher Layer Entity that requires only a single point of attachment, for the case where one of the Port states does not permit relay. Frames destined for the Higher Layer Entity that originate on LAN segment A are received by the entity; however, frames that originate on LAN segment B are not relayed by the Bridge, and can therefore

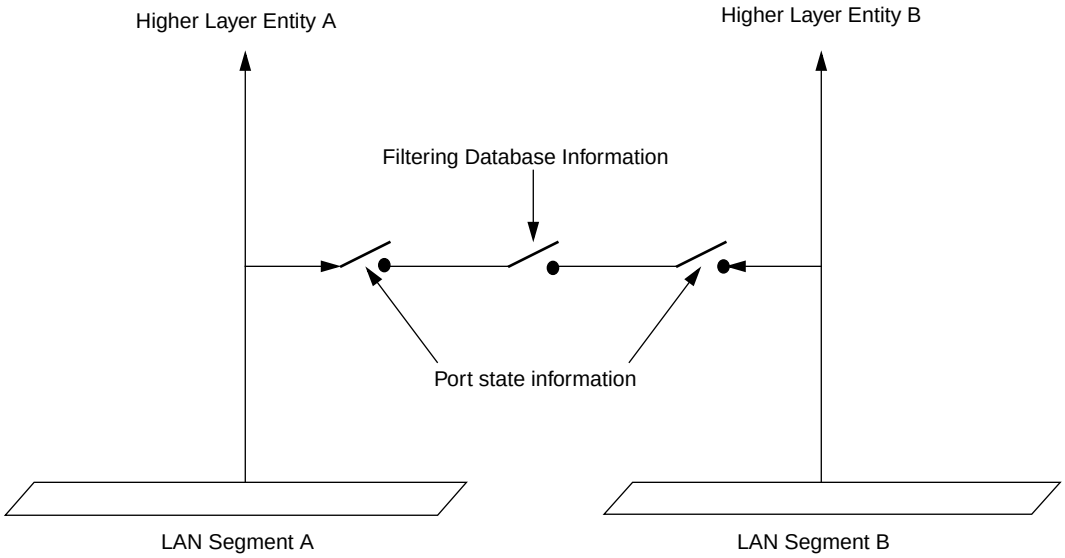


Figure 7-10—Effect of control information on the forwarding path

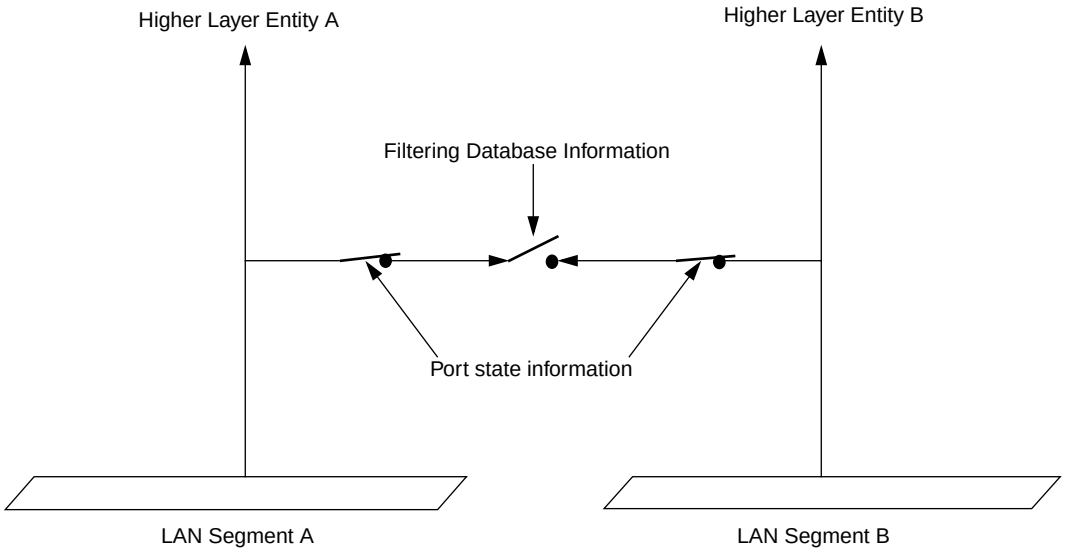


Figure 7-11—Per-Port points of attachment

only be received by the entity if there is some other forwarding path provided by other components of the Bridged LAN between segments A and B.

NOTE—If the Port state shown in Figure 7-13 occurs as a result of the normal operation of the Spanning Tree (as opposed to being a result of equipment failure, or administrative control of Port state information), then such a path will exist, either via another Port of this Bridge (not shown in the diagram) connected to segment A, or via one or more Bridges providing a path between segments A and B. If there is no active Spanning Tree path from segment B to segment A, then the Bridged LAN has partitioned into two separate Bridged LANs, one on either side of this Port, and the Higher Layer Entity shown is reachable only via segment A.

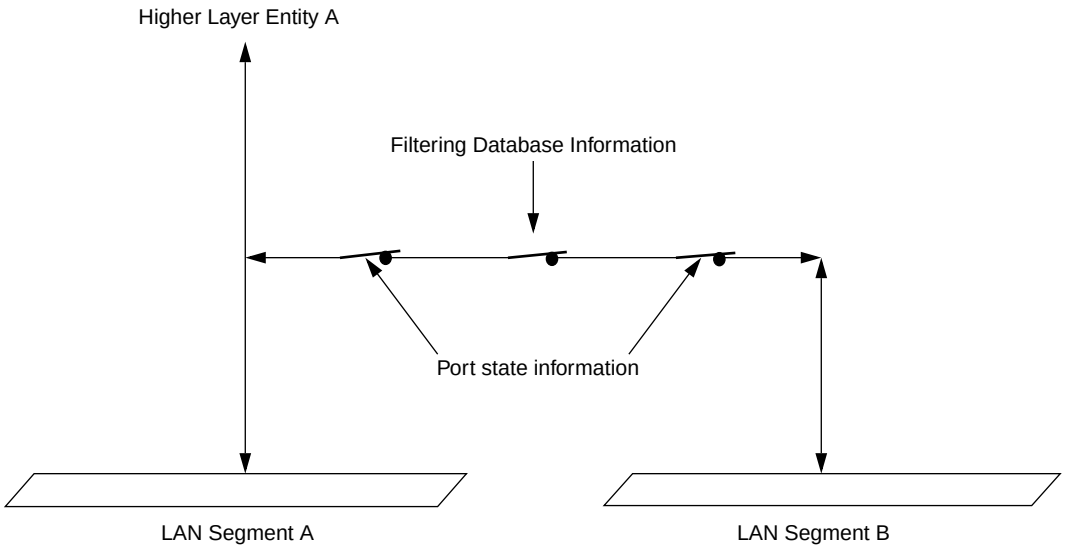


Figure 7-12—Single point of attachment—relay permitted

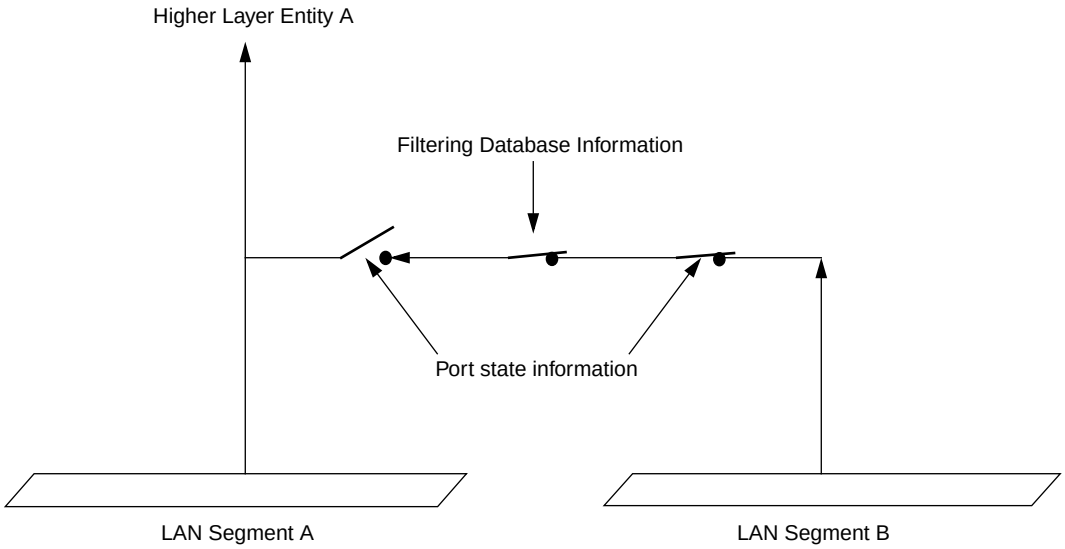


Figure 7-13—Single point of attachment—relay not permitted

8. The Spanning Tree Algorithm and Protocol

The configuration algorithm and protocol described in this clause reduce the Bridged LAN topology to a single Spanning Tree.

8.1 Requirements to be met by the algorithm

The Spanning Tree Algorithm and its associated Bridge Protocol operate to support, preserve, and maintain the quality of the MAC Service in all its aspects as discussed in Clause 6. In order to perform this function, the algorithm meets the following requirements, each of which is related to the discussion in that clause:

a) It will configure the active topology of a Bridged LAN of arbitrary topology into a single spanning tree, such that there is at most one data route between any two end stations, eliminating data loops (6.3.3; 6.3.4).
b) It will provide for fault tolerance by automatic reconfiguration of the spanning tree topology as a result of Bridge failure or a breakdown in a data path, within the confines of the available Bridged LAN components, and for the automatic accommodation of any Bridge or Bridge Port added to the Bridged LAN without the formation of transient data loops (6.1).
c) The entire active topology will stabilize in any sized Bridged LAN. It will, with a high probability, stabilize within a short, known bounded interval in order to minimize the time for which the service is unavailable for communication between any pair of end stations (6.1).
d) The active topology will be predictable and reproducible, and may be selected by management of the parameters of the algorithm, thus allowing the application of Configuration Management, following traffic analysis, to meet the goals of Performance Management (6.1; 6.3.10).
e) It will operate transparently to the end stations, such that they are unaware of their attachment to a single LAN or a Bridged LAN when using the MAC Service (6.2).
f) The communications bandwidth consumed by the Bridges in establishing and maintaining the spanning tree on any particular LAN will be a small percentage of the total available bandwidth and independent of the total traffic supported by the Bridged LAN regardless of the total number of Bridges or LANs (6.3.10).

Additionally, the algorithm and protocol meet the following goals, which limit the complexity of Bridges and their configuration:

g) The memory requirements associated with each Bridge Port are independent of the number of Bridges and LANs in the Bridged LAN.
h) Bridges do not have to be individually configured before being added to the Bridged LAN, other than having their MAC Addresses assigned through normal procedures.

8.2 Requirements of the MAC Bridges

In order for the Bridge Protocol to operate, the following are required:

a) A unique MAC group address, recognized by all the Bridges within the Bridged LAN, that identifies the Bridge Protocol Entities of all Bridges attached to an individual LAN.
b) An identifier for each Bridge, unique within the Bridged LAN.
c) A distinct Port Identifier for each Bridge Port, that can be assigned independently of the values used in other Bridges.

Values for each of these parameters, or a mechanism for assigning values to them, shall be provided by each Bridge. In the case of MAC Bridges that use 48-bit Universally Administered Addresses, the unique MAC Address that identifies the Bridge Protocol Entities is the Bridge Group Address (7.12.3).

In addition, to allow the configuration of the Spanning Tree active topology to be managed, the following are required:

1) A means of assigning the relative priority of each Bridge within the set of Bridges in the Bridged LAN.
2) A means of assigning the relative priority of each Port within the set of Ports of an individual Bridge.
3) A means of assigning a path cost component to each Port.

These parameters may be set by management when Bridge Management is supported.

The unique identifier for each Bridge is derived, in part, from the Bridge Address (7.12.5) and, in part, from a manageable priority component (9.2.5). The relative priority of Bridges is determined by the numerical comparison of the unique identifiers, with the lower numerical value indicating the higher priority identifier.

Part of the identifier for each Port is fixed and is different for each Port on a Bridge, and part is a manageable priority component (9.2.7). The relative priority of Ports is determined by the numerical comparison of the unique identifiers, with the lower numerical value indicating the higher priority identifier.

The path cost associated with each Port may be manageable. Additionally, 8.10.2 recommends default values for Ports attached to LANs of specific MAC types and speeds.

8.3 Overview

8.3.1 The active topology and its computation

The Spanning Tree Algorithm and Protocol configure a simply connected active topology from the arbitrarily connected components of a Bridged LAN. Frames are forwarded through some of the Bridge Ports in the Bridged LAN and not through others, which are held in a Blocking State. At any time, Bridges effectively connect just the LANs to which Ports in a Forwarding State are attached. Frames are forwarded in both directions through Bridge Ports that are in a Forwarding State. Ports that are in a Blocking State do not forward frames in either direction but may be included in the active topology, i.e., be put into a Forwarding State if components fail, are removed, or are added.

Figure 7-1 shows an example of a Bridged LAN. Figure 8-1 shows the active topology, i.e., the logical connectivity, of the same Bridged LAN following configuration.

One of the Bridges is known as the Root or the Root Bridge in the Bridged LAN. Each individual LAN has a Bridge Port connected to it that forwards frames from that LAN towards the Root, and forwards frames from the direction of the Root onto that LAN. This Port is known as the Designated Port for that LAN, and the Bridge of which it is part is the Designated Bridge for the LAN. The Root is the Designated Bridge for all the LANs to which it is connected. The Ports on each Bridge that are in a Forwarding State are the Root Port (that closest to the Root—see below) and the Designated Ports (if there are any). Ports that are not disabled and are neither Root Ports nor Designated Ports do not forward frames onto the LANs to which they connect; such Ports are known as Alternate Ports.

In Figure 8-1, Bridge 1 has been selected as the Root (though one cannot tell simply by looking at the topology which Bridge is the Root) and is the Designated Bridge for LAN A and LAN B. Bridge 2 is the Designated Bridge for LAN C and LAN D, and Bridge 4 is the Designated Bridge for LAN E. Figure 8-2 shows the logical tree topology of this configuration of the Bridged LAN.

The stable active topology of a Bridged LAN is determined by

a) The unique Bridge Identifiers associated with each Bridge.

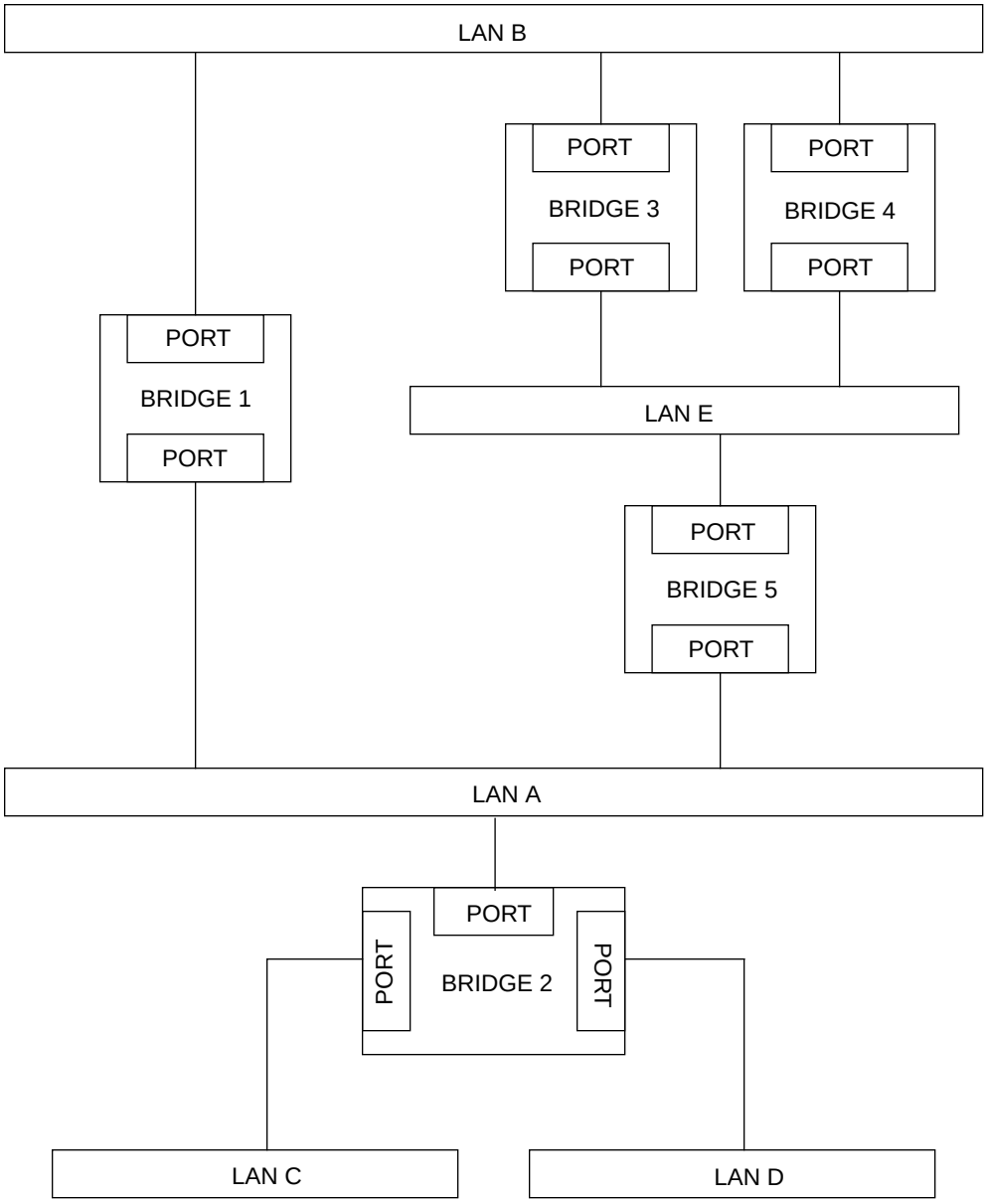


Figure 8-1—Active topology

b) The Path Cost associated with each Bridge Port.

c) The Port Identifier associated with each Bridge Port.

The Bridge with the highest priority Bridge Identifier is the Root (for convenience of calculation, this is the identifier with the lowest numerical value). Every Bridge Port in the Bridged LAN has a Root Path Cost associated with it. This is the sum of the Path Costs for each Bridge Port receiving frames forwarded from the Root on the least cost path to the Bridge. The Designated Port for each LAN is the Bridge Port for which the value of the Root Path Cost is the lowest: if two or more Ports have the same value of Root Path Cost, then first the Bridge Identifier of their Bridges and then their Port Identifiers are used as tie- breakers. Thus, a single Bridge Port is selected as the Designated Port for each LAN, the same computation selects the Root Port of a Bridge from among the Bridge's own Ports, and the active topology of the Bridged LAN is completely determined.

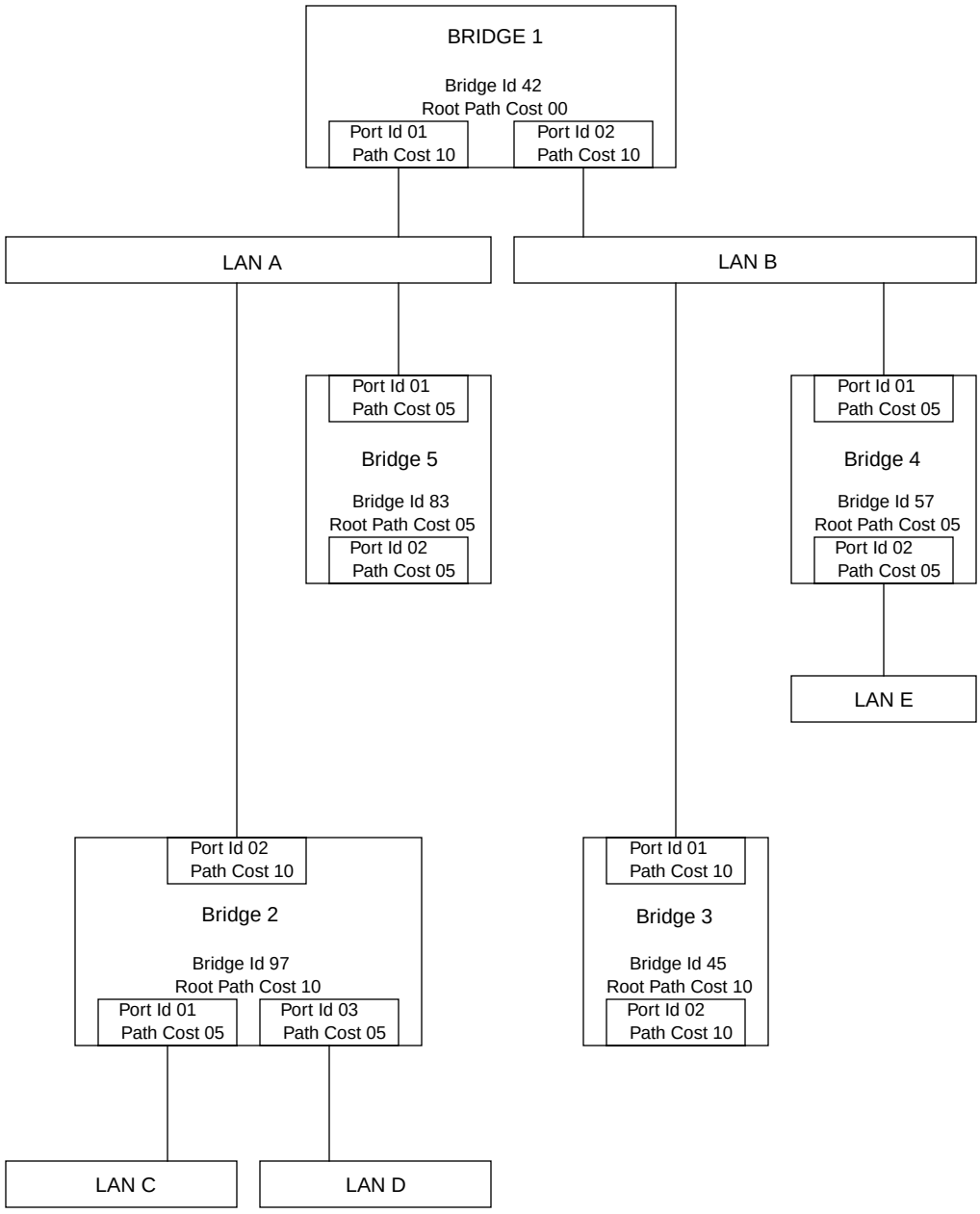


Figure 8-2—Spanning tree

A component of the Bridge Identifier of each Bridge, and the Path Cost and Port Identifier of each Bridge Port, can be managed, thus allowing a manager to select the active topology of the Bridged LAN.

8.3.2 Propagating the topology information

Bridges send a type of Bridge Protocol Data Unit known as a Configuration BPDU to each other in order to communicate and compute the above information. A MAC frame conveying a BPDU carries the Bridge Group Address in the destination address field and is received by all the Bridges connected to the LAN on which the frame is transmitted.

Bridge Protocol Data Units are not directly forwarded by Bridges, but the information in them may be used by a Bridge in calculating its own BPDU to transmit, and may stimulate that transmission. The Configuration BPDU, which is conveyed between the Bridge Ports attached to a single LAN, is distinguished from the notion of a Configuration Message, which expresses the propagation of the information carried throughout the Bridged LAN.

Each Configuration BPDU contains, among other parameters, the unique identifier of the Bridge that the transmitting Bridge believes to be the Root, the cost of the path to the Root from the transmitting Port, the identifier of the transmitting Bridge, and the identifier of the transmitting Port. This information is sufficient to allow a receiving Bridge to determine whether the transmitting Port has a better claim to be the Designated Port on the LAN on which the Configuration BPDU was received than the Port currently believed to be the Designated Port, and to determine whether the receiving Port should become the Root Port for the Bridge if it is not already.

Timely propagation throughout the Bridged LAN of the necessary information to allow all Bridge Ports to determine their state (Blocking or Forwarding) is achieved through three basic mechanisms:

a) A Bridge that believes itself to be the Root (all Bridges start by believing themselves to be the Root until they discover otherwise) originates Configuration Messages (by transmitting Configuration BPDUs) on all the LANs to which it is attached, at regular intervals.
b) A Bridge that receives a Configuration BPDU on what it decides is its Root Port conveying better information (i.e., highest priority Root Identifier, lowest Root Path Cost, highest priority transmitting Bridge and Port), passes that information on to all the LANs for which it believes itself to be the Designated Bridge.
c) A Bridge that receives inferior information, on a Port it considers to be the Designated Port on the LAN to which it is attached, transmits its own information in reply, for all other Bridges attached to that LAN to hear.

Hence, Spanning Tree paths to the Bridge with highest priority Root Identifier are quickly learned throughout the Bridged LAN, with inferior information about other potential roots and paths being contradicted.

8.3.3 Reconfiguration

To allow for reconfiguration of the Bridged LAN when components are removed or when management changes are made to parameters determining the topology, the topology information propagated throughout the Bridged LAN has a limited lifetime. This is effected by transmitting the age of the information conveyed (the time elapsed since the Configuration Message originated from the Root) in each Configuration BPDU. Every Bridge stores the information from the Designated Port on each of the LANs to which its Ports are connected, and monitors the age of that information.

In normal stable operation, the regular transmission of Configuration Messages by the Root ensures that topology information is not timed out.

If the Bridge times out the information held for a Port, it will attempt to become the Designated Bridge for the LAN to which that Port is attached, and will transmit protocol information received from the Root on its Root Port on to that LAN.

If the Root Port of the Bridge is timed out, then another Port may be selected as the Root Port. The information transmitted on LANs for which the Bridge is the Designated Bridge will then be calculated on the basis of information received on the new Root Port.

If no record of information from the current Root remains, then the Bridge will reconfigure by claiming to be the Root itself. If the Root has indeed failed, other Bridges will also be timing out protocol information; information as to the best successor and the new topology will rapidly propagate throughout the Bridged

LAN. It is also possible that the path to the current Root has changed, perhaps by increasing in cost, and that the reconfiguring Bridge has timed out because it considered more recent information from the Root inferior since it had a higher Root Path Cost. In this latter case, neighboring Bridges will immediately reply to BPDUs transmitted by the aspiring Root.

To ensure that all Bridges in the Bridged LAN share a common understanding of when old information should be timed out, the timeout value is transmitted in all Configuration Messages from the Root. This value takes account of the propagation delays in transmitting and receiving BPDUs on each of the LANs in the Bridged LAN, and thus of propagation of protocol information down the Spanning Tree. To minimize the probability of triggering reconfiguration through the loss of Configuration Messages, it includes an additional multiple of the time interval at which these are transmitted by the Root.

8.3.4 Changing Port State

Since there are propagation delays in passing protocol information throughout a Bridged LAN, there cannot be a sharp transition from one active topology to another. Topology changes may take place at different times in different parts of the Bridged LAN and to move a Bridge Port directly from nonparticipation in the active topology to the Forwarding State would be to risk having temporary data loops and the duplication and misordering of frames. It is also desirable to allow other Bridges time to reply to inferior protocol information before starting to forward frames.

Bridge Ports must therefore wait for new topology information to propagate throughout the Bridged LAN, and for the frame lifetime of any frames forwarded using the old active topology to expire, before forwarding frames.

During this time it is also desirable to time out station location information in the Filtering Database that may no longer be true and, during the latter part of this interval, to learn new station location information in order to minimize the effect of initial flooding of frames when the Port enters a Forwarding State. When the algorithm decides that a Port should be put into the Forwarding State, it is, therefore, first put into a Listening State where it waits for protocol information that suggests it should return to the Blocking State, and for the expiry of a protocol timer that would move it into a Learning State. In the Learning State, it still blocks the forwarding of frames, but learned station location information is included by the Learning Process in the Filtering Database. Finally the expiry of a protocol timer moves it into the Forwarding State where both forwarding of relayed frames and learning of station location information are enabled.

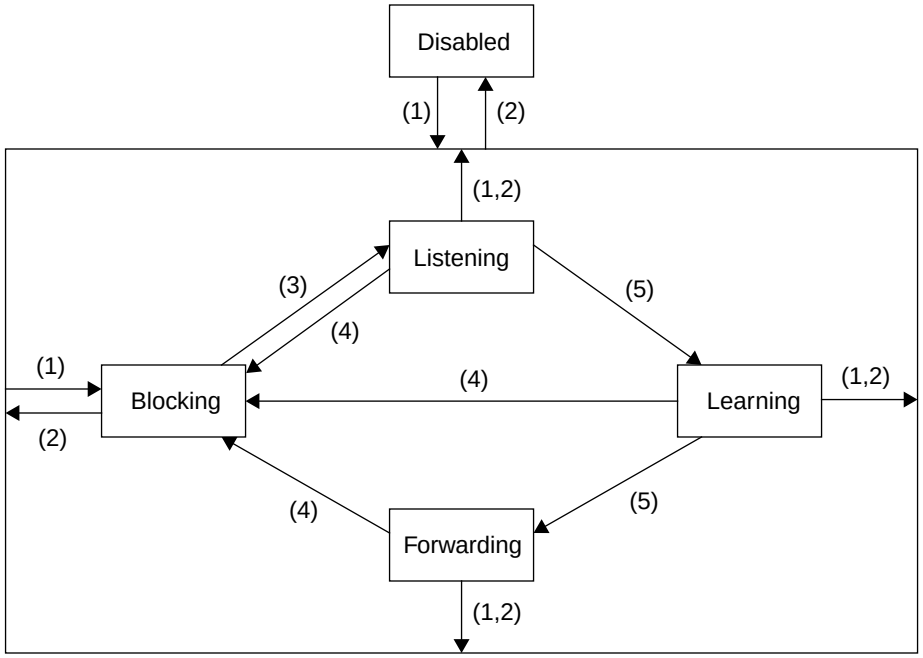
Figure 8-3 shows the transitions between the Port States.

8.3.5 Notifying topology changes

In normal stable operation, station location information in the Filtering Database need only change as a consequence of the physical relocation of stations. It may, therefore, be desirable to employ a long ageing time for entries in the Filtering Database, especially as many end stations transmit frames following power-up after relocation, which would cause station location information to be relearned.

However, when the active topology of a Bridged LAN reconfigures, end stations may appear to move from the point of view of a Bridge in the network. This is true even if the states of the Ports on that Bridge have not changed. It is necessary for station location to be relearned following a change in the active topology, even if only part of the Bridged LAN has reconfigured.

The Spanning Tree Algorithm and Protocol provide procedures for a Bridge that detects a change in active topology to notify the Root of the change reliably, and for the Root subsequently to communicate the change to all the Bridges. The Bridges then use a short value to age out dynamic entries in the Fitering Database for a period.



1) Port enabled, by management or initialization
2) Port disabled, by management or failure
3) Algorithm selects as Designated or Root Port
4) Algorithm selects as Alternate Port
5) Protocol timer expiry (Forwarding Timer)

Figure 8-3—Port States

When a Bridge that is not the Root changes the active topology of the Bridged LAN, it transmits a Topology Change Notification BPDU on the LAN to which its Root Port is attached. This transmission is repeated until the Bridge receives an acknowledgment from the Designated Bridge for that LAN. The acknowledgment is carried in a Configuration BPDU, thus the notification will eventually be acknowledged or further reconfiguration will take place. The Designated Bridge passes the notification to, or towards, the Root using the same procedure.

If the Root receives such a notification, or changes the topology itself, it will set a Topology Change flag in all Configuration Messages transmitted for some time. This time is such that all Bridges will receive one or more of the Configuration Messages, or further reconfiguration will take place. While this flag is set, Bridges use the value of Forwarding Delay (the time interval spent in each of the Listening and Learning States) to age out dynamic entries. When the flag is reset again, Bridges revert to using the Filtering Database Ageing Time.

The ability to detect topology changes can be enabled or disabled on a per-Port basis by means of the Change Detection Enabled parameter (8.5.5.10). The intent of this facility is to allow topology change detection to be disabled on Ports where it is known that a single end station is connected, and where powering that end station on and off would cause the Topology Change Notification mechanism to be triggered. Support of the ability to set this parameter to the Disabled State is optional.

8.4 Port States

The operation of an individual Bridge Port is described in terms of the state of the Port and the Processes (7.3) that provide and support the functions necessary for the operation of the Bridge (7.1).

The state of each Port governs the processing of frames received from the individual MAC Entity associated with the Port (7.5), the submission of frames to the MAC Entity for transmission (7.6), and the possible inclusion of the Port in the active topology of the Bridged LAN.

The operation of the Spanning Tree Algorithm and Protocol serves to maintain and change the state of each Port in order to meet the requirements placed on the algorithm (8.1). The possible Port States and the associated rules relating to the processing of frames are particular to this algorithm and Bridge Protocol.

The following are specified below for each of the five states—Blocking, Listening, Learning, Forwarding, or Disabled—that a Port may be in:

a) The purpose of the state.
b) Whether the Forwarding Process (7.7) discards received frames.
c) Whether the Forwarding Process (7.7) submits forwarded frames for transmission.
d) How the Learning Process (7.8) processes received frames.
e) Whether the Bridge Protocol Entity (7.10) includes the Port in its computation of the active topology.
f) Under which conditions a Port enters and leaves the state.

8.4.1 Blocking

A Port in this state does not participate in frame relay, thus preventing frame duplication arising through multiple paths existing in the active topology of the Bridged LAN.

The Forwarding Process shall discard received frames. It shall not submit forwarded frames for transmission. The Learning Process shall not add station location information to the Filtering Database.

The Bridge Protocol Entity shall include the Port in its computation of the active topology. BPDUs received shall be processed as required by the Spanning Tree Algorithm and Protocol.

This state is entered following initialization of the Bridge or from the Disabled State when the Port is enabled through the operation of management. This state can be entered from the Listening, Learning, or Forwarding States through the operation of the Spanning Tree Algorithm and Protocol. A Port enters the Blocking State because it has received information that another Bridge is the Designated Bridge for the LAN to which the Port is attached.

This state can be left upon expiry of a protocol timer or receipt of a Configuration Message on this or another Port, and the Listening State entered, through the operation of the Spanning Tree Algorithm and Protocol. This state can be left, and the Disabled State entered, through management action.

8.4.2 Listening

A Port in this state is preparing to participate in frame relay. Frame relay is temporarily disabled in order to prevent temporary loops, which may occur in a Bridged LAN during the lifetime of this state as the active topology of the Bridged LAN changes. Learning is disabled since changes in active topology can lead to the information acquired being incorrect when the active topology becomes stable.

The Forwarding Process shall discard received frames. It shall not submit forwarded frames for transmission. The Learning Process shall not add station location information to the Filtering Database.

The Bridge Protocol Entity shall include the Port in its computation of the active topology. BPDUs received shall be processed as required by the Spanning Tree Algorithm and Protocol. BPDUs can be submitted for transmission.

This state is entered from the Blocking State when the operation of the Spanning Tree Algorithm and Protocol determines that the Port should participate in frame relay.

This state can be left upon the expiry of a protocol timer, and the Learning State entered, through the operation of the Spanning Tree Algorithm and Protocol. This state can be left upon receipt of a Bridge Protocol Data Unit on this or another Port, and the Blocking State entered, through the operation of the Spanning Tree Algorithm and Protocol. This state can be left, and the Disabled or the Blocking State entered, through management action.

8.4.3 Learning

A Port in this state is preparing to participate in frame relay. Frame relay is temporarily disabled in order to prevent temporary loops, which may occur in a Bridged LAN during the lifetime of this state as the active topology of the Bridged LAN changes. Learning is enabled to allow information to be acquired prior to frame relay in order to reduce the number of frames unnecessarily relayed.

The Forwarding Process shall discard received frames. It shall not submit forwarded frames for transmission. The Learning Process shall incorporate station location information into the Filtering Database.

The Bridge Protocol Entity shall include the Port in its computation of the active topology. BPDUs received shall be processed as required by the Spanning Tree Algorithm and Protocol. BPDUs can be submitted for transmission.

This state is entered from the Listening State through the operation of the Spanning Tree Algorithm and Protocol, on the expiry of a protocol timer.

This state can be left upon the expiry of a protocol timer, and the Forwarding State entered, through the operation of the Spanning Tree Algorithm and Protocol. This state can be left upon receipt of a Bridge Protocol Data Unit on this or another Port, and the Blocking State entered, through the operation of the Spanning Tree Algorithm and Protocol. This state can be left, and the Disabled or the Blocking State entered, through management action.

8.4.4 Forwarding

A Port in this state is participating in frame relay.

The Forwarding Process can forward received frames. It can submit forwarded frames for transmission. The Learning Process shall incorporate station location information into the Filtering Database.

The Bridge Protocol Entity shall include the Port in its computation of the active topology. BPDUs received shall be processed as required by the Spanning Tree Algorithm and Protocol. BPDUs can be submitted for transmission.

This state is entered from the Learning State through the operation of the Spanning Tree Algorithm and Protocol, on the expiry of a protocol timer.

This state can be left upon receipt of a Bridge Protocol Data Unit on this or another Port, and the Blocking State entered, through the operation of the Spanning Tree Algorithm and Protocol. This state can be left, and the Disabled or the Blocking State entered, through management action.

8.4.5 Disabled

A Port in this state does not participate in frame relay or the operation of the Spanning Tree Algorithm and Protocol.

The Forwarding Process shall discard received frames. It shall not submit forwarded frames for transmission. The Learning Process shall not incorporate station location information into the Filtering Database.

The Bridge Protocol Entity shall not include the Port in its computation of the active topology. BPDUs received shall not be processed by the Spanning Tree Algorithm and Protocol. BPDUs shall not be submitted for transmission.

This state is entered from any other state by the operation of management.

This state is left when the Port is enabled by management action, and the Blocking State is entered.

8.5 Protocol parameters and timers

Information is transferred between the Protocol Entities of individual Bridges by the exchange of Bridge Protocol Data Units. This subclause specifies the parameters conveyed in the two types of BPDU specified: Configuration BPDUs and Topology Change Notification BPDUs. The encoding of these parameters and additional information elements are specified in Clause 9.

Each Bridge Protocol Entity maintains a number of parameters and timers independently of the individual Ports, and a number of timers and parameters for each Port. This subclause specifies those parameters, their use, and under what conditions they are updated.

8.5.1 Configuration BPDU parameters

8.5.1.1 Root Identifier

The unique Bridge Identifier of the Bridge assumed to be the Root by the Bridge transmitting the Configuration BPDU.

This parameter is conveyed to enable all Bridges to agree on the Root.

8.5.1.2 Root Path Cost

The cost of the path to the Root Bridge denoted by the Root Identifier from the transmitting Bridge.

This parameter is conveyed to enable a Bridge to decide which of the Bridges attached to the LAN on which the Configuration BPDU has been received offers the lowest Cost path to the Root for that LAN.

8.5.1.3 Bridge Identifier

The unique Bridge Identifier of the Bridge transmitting the Configuration BPDU.

This parameter is conveyed to enable a Bridge to

a) Decide, in the case of a LAN to which two or more Bridges are attached, that offer equal Cost paths to the Root, which of the Bridges should be selected as the Designated Bridge for that LAN.

b) Detect the case where two or more Ports on the same Bridge are attached to the same LAN, i.e., are in direct communication through a path of Bridged LAN components, none of which operate the Spanning Tree Algorithm and Protocol.

8.5.1.4 Port Identifier

The Port Identifier of the Port on the transmitting Bridge through which the Configuration BPDU was transmitted. This identifier uniquely identifies a Port on that Bridge.

This parameter is conveyed to enable a Bridge to decide, in the case of a LAN to which two or more Ports on the same Bridge are attached, which Ports are so attached.

8.5.1.5 Message Age

The age of the Configuration Message, being the time since the generation of the Configuration BPDU by the Root that instigated the generation of this Configuration BPDU.

This parameter is conveyed to enable a Bridge to discard information whose age exceeds Max Age (see below).

8.5.1.6 Max Age

A timeout value to be used by all Bridges in the Bridged LAN. The value of Max Age is set by the Root.

This parameter is conveyed to ensure that each Bridge in a Bridged LAN has a consistent value against which to test the age of stored configuration information.

8.5.1.7 Hello Time

The time interval between the generation of Configuration BPDUs by the Root.

This parameter is not directly used by the Spanning Tree Algorithm but is conveyed in Configuration BPDUs to facilitate the monitoring of protocol performance by management functions.

8.5.1.8 Forward Delay

A timeout value to be used by all Bridges in the Bridged LAN. The value of Forward Delay is set by the Root.

This parameter is conveyed to ensure that each Bridge in a Bridged LAN uses a consistent value for the Forward Delay Timer when transferring the state of a Port to the Forwarding State. This parameter is also used as the timeout value for ageing Filtering Database dynamic entries following changes in active topology.

8.5.1.9 Topology Change Acknowledgment

A flag set in a Configuration Message transmitted in response to Topology Change Notification received on a Designated Port. This parameter is conveyed to allow a reliable acknowledged protocol to operate for notifying the Root of changes in active topology.

8.5.1.10 Topology Change

A flag set by the Root in all Configuration BPDUs transmitted for a period of time following the notification or detection of a topology change.

This parameter is conveyed to notify Bridges throughout the Bridged LAN that there has been a change in active topology in part of the Bridged LAN and that the Filtering Database should age out entries more quickly in order to limit the effects of temporary isolation of end systems attached to the Bridged LAN brought about by the use of incorrect information in the Filtering Database.

The value of the ageing time applied to dynamic entries in the Filtering Database becomes equal to that of the value of the Forward Delay time parameter held for the Bridge; i.e., after Forward Delay time has elapsed while the Topology Change flag is set in all Configuration Messages received from the Root, then the only dynamic entries remaining in the Filtering Database are those that have been created or updated during that period.

8.5.2 Topology Change Notification BPDU parameters

No parameters are conveyed in a Topology Change Notification BPDU.

8.5.3 Bridge parameters

8.5.3.1 Designated Root

The unique Bridge Identifier of the Bridge assumed to be the Root.

This parameter is used as the value of the Root Identifier parameter in all Configuration BPDUs transmitted by the Bridge.

8.5.3.2 Root Path Cost

The cost of the path to the Root from this Bridge. When the Bridge is the Root this parameter has the value zero. Otherwise, it is equal to the sum of the values of the Designated Cost and Path Cost parameters held for the Root Port. This parameter is used to test the value of the Root Path Cost parameter conveyed in received Configuration Message information, and as the value of the Root Path Cost parameter in transmitted Configuration Message information.

8.5.3.3 Root Port

The Port Identifier of the Port that offers the lowest cost path to the Root, i.e., that Port for which the sum of the values of the Designated Cost and Path Cost parameters held for the Port is the lowest.

If two or more Ports offer equal least cost paths to the Root, the Root Port is selected to be that with the highest priority Bridge Identifier held as the Designated Bridge Parameter for that Port.

If two or more Ports offer equal least cost paths to the Root and hold the same Designated Bridge parameter values, then the Root Port is selected to be that with the highest-priority Designated Port held for that Port.

Finally, if two or more ports offer equal least-cost paths to the Root and hold the same Designated Bridge and Designated Port parameter values, then the Root Port is selected to be that with the highest-priority Port Identifier. The Port Identifiers for different Ports on the same Bridge are guaranteed to be different and thus enforce a tie-breaker.

This parameter is used to identify the Port through which the path to the Root is established. It is not significant when the Bridge is the Root, and is set to zero.

8.5.3.4 Max Age

The maximum age of received protocol information before it is discarded.

8.5.3.5 Hello Time

The time interval between the transmission of Configuration BPDUs by a Bridge that is attempting to become the Root or is the Root.

8.5.3.6 Forward Delay

The time spent by a Port in the Listening State and the Learning State before moving to the Learning or Forwarding State, respectively. It is also the value used for the ageing time of dynamic entries in the Filtering Database, while received Configuration Messages indicate a topology change.

8.5.3.7 Bridge Identifier

The unique Bridge Identifier of the Bridge. This parameter is used as the value of

a) The Bridge Identifier parameter in all Configuration BPDUs transmitted by the Bridge.
b) The Bridge's Designated Root when the Bridge is the Root, or is attempting to become the Root, following expiry of all information concerning the current Root, or following management action.

This parameter comprises two parts, one of which is derived from the unique Bridge Address (7.12.5) and assures the uniqueness of the Bridge Identifier in the Bridged LAN, the other of which allows the adjustment of the priority of the Bridge Identifier and is taken as the more significant part in priority comparisons. The priority part of this parameter may be updated by management action when Bridge Management is supported.

8.5.3.8 Bridge Max Age

The value of the Max Age parameter when the Bridge is the Root or is attempting to become the Root.

This parameter may be updated by management action when Bridge Management is supported.

8.5.3.9 Bridge Hello Time

The value of the Hello Time parameter when the Bridge is the Root or is attempting to become the Root.

This parameter is the time interval between transmissions of Topology Change Notification BPDUs towards the Root when the Bridge is attempting to notify the Designated Bridge on the LAN to which its Root Port is attached of a topology change.

This parameter may be updated by management action when Bridge Management is supported.

8.5.3.10 Bridge Forward Delay

The value of the Forward Delay parameter when the Bridge is the Root or is attempting to become the Root.

This parameter may be updated by management action when Bridge Management is supported.

8.5.3.11 Topology Change Detected

A Boolean parameter set True to record that a topology change has been detected by or notified to the Bridge. When set to True, this parameter is used to stimulate transmission of Topology Change Notifications towards the Root when the Bridge is not itself the Root; and to set the value of the Topology Change parameter for the Bridge to True if the Bridge is, or becomes, the Root. Transmission is subject to a reliable

acknowledgment mechanism, as at 8.3.5 and 8.5.1.9; the Topology Change Notification BPDUs are transmitted at regular intervals of Bridge Hello Time, until acknowledged.

8.5.3.12 Topology Change

A Boolean parameter set to record

a) For a Bridge that is not the Root, whether or not the most recently accepted Configuration Message indicates a change in the active topology; or
b) For the Root, whether or not a change in topology has been detected within the preceding Topology Change Time period.

This parameter is used to propagate the indication of Topology Change in transmitted Configuration Messages, and to determine whether the short (Forward Delay) or long (Ageing Time) timeout value is to be used for dynamic entries in the Filtering Database.

8.5.3.13 Topology Change Time

The time period for which the Bridge originates Configuration Messages indicating topology change following detection of a topology change, when the Bridge is the Root. The value of this parameter is equal to the sum of the values of the Bridge's Bridge Max Age and Bridge Forward Delay parameters.

8.5.3.14 Hold Time

The minimum time period to elapse between the transmission of Configuration BPDUs through a given LAN Port: at most one Configuration BPDU shall be transmitted in any Hold Time period. This parameter is a fixed parameter, with values as specified in Table 8-3.

8.5.4 Bridge timers

8.5.4.1 Hello Timer

This timer serves to ensure periodic transmission of Configuration BPDUs by the Bridge when it is, or is attempting to become, the Root.

The timeout value of the timer is that of the Bridge's Bridge Hello Time parameter.

8.5.4.2 Topology Change Notification Timer

This timer serves to ensure that the Designated Bridge on the LAN to which the Bridge's Root Port is attached is notified of any detected topology change.

The timeout value of the timer is that of the Bridge's Bridge Hello Time parameter.

8.5.4.3 Topology Change Timer

This timer serves to determine the time period for which Configuration BPDUs are transmitted with the Topology Change flag set by the Bridge when it is the Root following the detection of a topology change.

The timeout value of the timer is that of the Bridge's Topology Change Time parameter.

8.5.5 Port parameters

8.5.5.1 Port Identifier

The identifier of the Port, unique among the Ports of this Bridge. This parameter is used as the value of the Port Identifier parameter of all Configuration Messages transmitted through the Port.

The parameter consists of two parts. One part bears a fixed relationship to the physical or logical support of the Port by real-world equipment; this part assures the uniqueness of the Port Identifier among the Ports of a single Bridge, and is a small integer assigned in the range from one upwards. The other part of the parameter allows adjustment of the priority of the Port and is taken as the more significant part in priority comparisons. When Bridge Management is supported, the priority part of this parameter can be updated by management.

8.5.5.2 State

The current state of the Port (i.e., Disabled, Listening, Learning, Forwarding, or Blocking).

This parameter is used to control the acceptance of frames from the MAC Entity associated with the Port by the Forwarding and Learning Processes, the forwarding of frames by the Forwarding Process to that MAC Entity, and the transmission and reception of BPDUs (8.4).

This parameter is updated by the action of the protocol.

When Bridge Management is supported, this parameter may also be updated by management action.

8.5.5.3 Path Cost

The contribution of the path through this Port, when the Port is the Root Port, to the total cost of the path to the Root for this Bridge.

This parameter is used, added to the value of the Designated Cost parameter for the Root Port, as the value of the Root Path Cost parameter offered in all Configuration BPDUs transmitted by the Bridge, when it is not the Root.

When Bridge Management is supported, this parameter may be updated by management action.

8.5.5.4 Designated Root

The unique Bridge Identifier of the Bridge recorded as the Root in the Root Identifier parameter of Configuration BPDUs transmitted by the Designated Bridge for the LAN to which the Port is attached.

This parameter is used to test the value of the Root Identifier parameter conveyed in received Configuration BPDUs.

8.5.5.5 Designated Cost

a) For a Designated Port, the path cost (equal to the Root Path Cost of the Bridge) offered to the LAN to which the Port is attached; otherwise,
b) The cost of the path to the Root offered by the Designated Port on the LAN to which this Port is attached.

This parameter is used to test the value of the Root Path Cost parameter conveyed in received Configuration BPDUs.

8.5.5.6 Designated Bridge

The unique Bridge Identifier (8.5.3.7) of

a) The Bridge to which the Port belongs, in the case of a Designated Port; or otherwise,
b) The Bridge believed to be the Designated Bridge for the LAN to which this Port is attached.

This parameter is used

c) Together with the Designated Port and Port Identifier parameters for the Port to ascertain whether this Port should be the Designated Port for the LAN to which it is attached.
d) To test the value of the Bridge Identifier parameter conveyed in received Configuration BPDUs.

8.5.5.7 Designated Port

The Port Identifier (8.5.5.1) of the Bridge Port, on the Designated Bridge (8.5.5.6), through which the Designated Bridge transmits the Configuration Message information stored by this Port.

This parameter is used

a) Together with the Designated Bridge and Port Identifier parameters for the Port to ascertain whether this Port should be the Designated Port for the LAN to which it is attached.
b) By management to determine the topology of the Bridged LAN.

8.5.5.8 Topology Change Acknowledge

The value of the Topology Change Acknowledgment flag in the next Configuration BPDU to be transmitted on the associated Port.

This parameter is used to record the need to set the Topology Change Acknowledgment flag in reply to a received Topology Change Notification BPDU.

8.5.5.9 Configuration Pending

A Boolean parameter set to record that a Configuration BPDU should be transmitted on expiry of the Hold Timer for the associated Port.

This parameter is used, in conjunction with the Hold Timer for the Port, to ensure that Configuration BPDUs are not transmitted too frequently, but that up-to-date information is transmitted.

8.5.5.10 Change Detection Enabled

A Boolean parameter set to record whether or not detection of topology changes is enabled for the associated Port. If change detection is disabled, changes in topology detected by or notified to the Bridge via this Port are ignored from the point of view of communicating the topology change to the Root.

The default state of this parameter shall be Enabled for all Ports of the Bridge. Support of the ability to set this parameter to the Disabled State is optional.

8.5.6 Port timers

8.5.6.1 Message Age Timer

This timer serves to measure the age of the received protocol information recorded for a Port, and to ensure that this information is discarded when its age exceeds the value of the Max Age parameter recorded by the Bridge.

The timeout value of the timer is that of the Bridge's Max Age parameter.

8.5.6.2 Forward Delay Timer

This timer determines the time spent by a Port in the Listening and Learning States. The timeout value of the timer is that of the Bridge's Forward Delay parameter.

8.5.6.3 Hold Timer

This timer serves to ensure that Configuration BPDUs are not transmitted too frequently through any Bridge Port.

The timeout value of the timer is that of the Hold Time for the Bridge.

8.6 Elements of procedure

8.6.1 Transmit Configuration BPDU

8.6.1.1 Purpose

To convey knowledge of the Designated Root, Root Path Cost, Designated Bridge, Designated Port, and the values of protocol timers to other Bridge Ports attached to the same LAN as the Port on which the Configuration BPDU is transmitted.

8.6.1.2 Use

a) Case 1. As part of the Configuration BPDU Generation procedure (8.6.4).
b) Case 2. As part of the Reply to Configuration BPDU procedure (8.6.5).
c) Case 3. Following expiry of the Hold Timer for the Port (8.7.8) when the Configuration Pending flag parameter for the Port is set, as a consequence of a previous invocation of the procedure.
d) Case 4. As part of the Acknowledge Topology Change procedure (8.6.16).

8.6.1.3 Procedure

8.6.1.3.1 Step 1

If the Hold Timer for the Port is active then the Configuration Pending flag parameter for the Port shall be set. This completes the procedure. Otherwise, if the Hold Timer is not active, the following steps are invoked.

8.6.1.3.2 Step 2

If the Hold Timer for the Port is not active, a Configuration BPDU is prepared for transmission. The Configuration BPDU shall have parameters set as follows:

a) The Root Identifier shall be set to the value of the Designated Root parameter held by the Bridge.
b) The Root Path Cost shall be set to the value of the Root Path Cost parameter held by the Bridge.
c) The Bridge Identifier shall be set to the value of the Bridge Identifier parameter held by the Bridge.
d) The Port Identifier shall be set to the value of the Port Identifier parameter held for the Bridge Port through which the Configuration BPDU is transmitted.
e) If the Bridge has been selected as the Root, i.e., if the values of the Designated Root and Bridge Identifier parameters held by the Bridge are the same, the Message Age shall be set to zero.
f) Otherwise, the value of the Message Age shall be set such that the transmitted Configuration BPDU does not convey an underestimate of the age of the Protocol Message received on the Root Port; i.e., the value transmitted shall be no less than that recorded by the Message Age Timer for that Port, shall be greater than the value received, and will incorporate any transmission delay. The value of the parameter shall not exceed its true value by more than the maximum Message Age increment overestimate as specified in 8.10.2.
g) The Max Age, Hello Time, and Forward Delay shall be set to the values of the Max Age, Hello Time, and Forward Delay parameters held for the Bridge.
h) The Topology Change Acknowledgment flag shall be set to the value of the Topology Change Acknowledge flag parameter for the Port.
i) The Topology Change flag shall be set to the value of the Topology Change flag parameter for the Bridge.

8.6.1.3.3 Step 3

If the value of the Message Age parameter in the Configuration BPDU is less than that of the Max Age parameter, then

a) The Topology Change Acknowledge flag parameter for the Port is reset.
b) The Configuration Pending flag parameter for the Port is reset.
c) The BPDU shall be transmitted through the Port within a time maximum BPDU transmission delay (as specified in 8.10.2).
d) The Hold Timer for the Port is started.

8.6.2 Record Configuration Information

8.6.2.1 Purpose

To record, for a Port, protocol parameters conveyed by a Configuration BPDU received on that Port.

8.6.2.2 Use

Following the receipt of a Configuration BPDU conveying protocol information that supersedes that already held, i.e., if

a) Case 1. The Root Identifier denotes a Bridge of higher priority than that recorded as the Designated Root, or
b) Case 2. The Root Identifier is the same as the Designated Root, and the Root Path Cost is lower than that recorded as the Designated Cost for the Port, or
c) Case 3. The Root Identifier and Root Path Cost are as recorded for the Port, and the Bridge Identifier denotes a Bridge of higher priority than that recorded as the Designated Bridge for the Port, or
d) Case 4. The Root Identifier and Root Path Cost are as recorded for the Port, and the Bridge Identifier is the same as that recorded as the Designated Bridge for the Port, and either
    1) The Bridge receiving the BPDU is not the Designated Bridge for the Port, or
    2) The Port Identifier denotes a Port of priority not less than that recorded as the Designated Port.

8.6.2.3 Procedure

8.6.2.3.1 Step 1

The Designated Root, Designated Cost, Designated Bridge, and Designated Port parameters held for the Port are set to the values of the Root Identifier, Root Path Cost, Bridge Identifier, and Port Identifier parameters conveyed in the received Configuration BPDU.

8.6.2.3.2 Step 2

The Message Age Timer for the Port is started, to run from the value of the Message Age parameter conveyed in the received Configuration BPDU.

8.6.3 Record Configuration Timeout Values

8.6.3.1 Purpose

To update the Max Age, Hello Time, Forward Delay, and Topology Change flag parameters to the latest values received from the Root.

8.6.3.2 Use

Following receipt of a Configuration BPDU on the Root Port which invokes the Record Configuration Information procedure (8.6.2.2).

8.6.3.3 Procedure

The Max Age, Hello Time, Forward Delay, and Topology Change parameters held by the Bridge are set to the values conveyed in the received Configuration BPDU.

8.6.4 Configuration BPDU Generation

8.6.4.1 Purpose

To convey to Bridges attached to each LAN for which the Bridge is Designated Bridge knowledge of the Designated Root, Root Path Cost, Designated Bridge, Designated Port, and the values of protocol timers.

8.6.4.2 Use

a) Case 1. Following receipt of a Configuration BPDU on the Root Port that invokes the Record Configuration Information procedure (8.6.2.2).
b) Case 2. Following expiry of the Hello Timer.
c) Case 3. Following selection of the Bridge as the Designated Root by the Configuration Update procedure on expiry of a Message Age Timer for a Bridge Port.
d) Case 4. Following selection of the Bridge as the Designated Root by management action.

8.6.4.3 Procedure

For each Port that is the Designated Port for the LAN to which it is attached (i.e., the value of the Designated Bridge and Designated Port parameters held for the Port are the same as that of the Bridge Identifier and the Port Identifier for that Port, respectively, which is not in the Disabled State), the Transmit Configuration BPDU procedure (8.6.1) is used.

8.6.5 Reply to Configuration BPDU

8.6.5.1 Purpose

To establish the Designated Bridge and Designated Port for a LAN in the case where another Bridge Port has transmitted a Configuration BPDU on that LAN. This arises if Configuration Messages from the current Root have not been received by the transmitting Bridge, either due to that Root having been newly established or to BPDU loss and subsequent expiry of a Message Age Timer.

8.6.5.2 Use

Following receipt of a Configuration BPDU on a Port that is the Designated Port for the LAN to which it is attached, which does not update the information held for that Port, i.e., does not satisfy the conditions (8.6.2.2) for the use of the Record Configuration Information procedure.

8.6.5.3 Procedure

The Transmit Configuration BPDU procedure (8.6.1) is used for the Port on which the Configuration BPDU was received.

8.6.6 Transmit Topology Change Notification BPDU

8.6.6.1 Purpose

To notify the Bridge on the path towards the Root that an extension of the topology has been detected by the transmitting Bridge. Eventually this will result in the Root being notified of the topology change.

8.6.6.2 Use

a) Case 1. Following the detection or receipt of notification of a topology change by a Bridge that is not the Root.
b) Case 2. Following expiry of the Topology Change Notification Timer.

8.6.6.3 Procedure

A Topology Change Notification BPDU shall be transmitted through the Root Port within a time of maximum BPDU transmission delay (8.10.2).

8.6.7 Configuration Update

8.6.7.1 Purpose

To update the configuration information held by the Bridge and the Bridge Ports.

8.6.7.2 Use

a) Case 1. Following receipt of a Configuration BPDU which invokes the Record Configuration Information procedure (8.6.2.2).
b) Case 2. Following a Port becoming the Designated Port for the LAN to which it is attached on expiry of the Message Age Timer for that Port.
c) Case 3. Following a change in Port State through management action.

8.6.7.3 Procedure

8.6.7.3.1 Step 1

The procedure for Root Selection (8.6.8) shall be used to select the Designated Root and the Root Port, and to calculate the Root Path Cost for this Bridge.

8.6.7.3.2 Step 2

The procedure for Designated Port Selection (8.6.9) shall be used to determine for each Port whether the Port should become the Designated Port for the LAN to which it is attached.

8.6.8 Root selection

8.6.8.1 Purpose

To select the Designated Root and the Root Port, and to calculate the Root Path Cost for this Bridge.

8.6.8.2 Use

This procedure is used by the Configuration Update procedure (8.6.7).

8.6.8.3 Procedure

8.6.8.3.1 Step 1

The Root Port is set to identify the Port that, among those that are not the Designated Port for the LAN to which they are attached, are not Disabled, and have a Designated Root parameter of higher priority than the Bridge's Bridge Identifier;

a) Has the highest priority Root associated with it, i.e., recorded as the Designated Root for the Port.
b) Of two or more Ports with the highest priority Designated Root parameter, has the lowest Root Path Cost associated with it, i.e., the lowest sum of the Designated Cost and Path Cost parameters for any Port; or
c) Of two or more Ports with the highest priority Designated Root parameter and lowest value of associated Root Path Cost, has the highest priority Bridge Identifier recorded as the Designated Bridge for the LAN to which the Port is attached; or
d) Of two or more Ports with the highest priority Designated Root parameter, lowest value of associated Root Path Cost, and highest priority Designated Bridge, has the highest priority Port Identifier recorded as the Designated Port for the LAN to which the Port is attached; or
e) Of two or more Ports with the highest priority Designated Root parameter, lowest value of associated Root Path Cost, and highest priority Designated Bridge and Designated Port, has the highest priority Port Identifier.

8.6.8.3.2 Step 2

If there is no such Port, the value of the Root Port parameter is set to zero, and

a) The Designated Root parameter held by the Bridge is set to the Bridge Identifier parameter held for the Bridge, and
b) The value of the Root Path Cost parameter held by the Bridge is set to zero.

8.6.8.3.3 Step 3

Otherwise, i.e., if one of the Bridge Ports has been identified as the Root Port, then

a) The Designated Root parameter held by the Bridge is set to the Designated Root parameter held for the Root Port, and
b) The value of the Root Path Cost parameter held by the Bridge is set to the value of the Root Path Cost parameter associated with the Root Port, i.e., the sum of the values of the Path Cost and the Designated Cost parameters recorded for the Root Port.

8.6.9 Designated Port selection

8.6.9.1 Purpose

To determine, for each Port, whether the Port should be the Designated Port for the LAN to which it is attached.

8.6.9.2 Use

As part of the Configuration Update procedure (8.6.7).

8.6.9.3 Procedure

The procedure to Become Designated Port (8.6.10) shall be invoked for each Port that

a) Has already been selected as the Designated Port for the LAN to which it is attached, i.e., the value of the Designated Bridge and Designated Port parameters held for the Port are the same as that of the Bridge Identifier and the Port Identifier for that Port respectively, or for which
b) The Designated Root parameter recorded for the Bridge differs from that recorded for the Port (note that this procedure follows root selection), or
c) The Bridge offers a Path of lower cost to the Root for the LAN to which the Port is attached, i.e., the Root Path Cost recorded by the Bridge is less than the Designated Cost recorded for the Port, or
d) The Bridge offers a Path of equal cost to the Root, and the Bridge's Bridge Identifier denotes a Bridge of higher priority than that recorded as the Designated Bridge for that Port, or
e) The Bridge offers a Path of equal cost to the Root, and the Bridge is the Designated Bridge for the LAN to which the Port is attached, and the Port Identifier of the Port is of higher priority than that recorded as the Designated Port.

8.6.10 Become Designated Port

8.6.10.1 Purpose

Given that a Port is to be the Designated Port on the LAN to which it is attached, to assign appropriate values to those Port parameters that determine the active topology of the Bridged LAN.

8.6.10.2 Use

a) Case 1. Following expiry of the Message Age Timer for the Port.
b) Case 2. Following selection of the Port as the Designated Port for the LAN to which it is attached by the Designated Port Selection procedure (8.6.9) as part of the Configuration Update procedure (8.6.7).
c) Case 3. Following a change of Port State through management action (8.8.1, 8.8.2, 8.8.3, and 8.8.5).

8.6.10.3 Procedure

a) The Designated Root parameter held for the Port is set to the value of the Designated Root parameter held by the Bridge;
b) The Designated Cost parameter held for the Port is set to the value of the Root Path Cost held by the Bridge;
c) The Designated Bridge parameter held for the Port is set to the Bridge Identifier of the Bridge;
d) The Designated Port parameter held for the Port is set to the Port Identifier of the Port.

8.6.11 Port State selection

8.6.11.1 Purpose

To select the state of the Bridge's Ports based upon updated configuration information that indicates, for each Port, its part in the active topology of the Bridged LAN, i.e., whether it should

a) Be the Root Port for the Bridge.
b) Be a Designated Port.
c) Be an Alternate Port, i.e., act as a backup Port in a redundantly connected Bridged LAN.

8.6.11.2 Use

Following use of the Configuration Update procedure after

a) Case 1. Receipt of a Configuration BPDU conveying information that supersedes that recorded for a Port.
b) Case 2. The expiry of the Message Age timer for a Port, which causes that Port to become the Designated Port for the LAN to which it is attached.
c) Case 3. A change in the state of a Port arising through management action.

8.6.11.3 Procedure

For each of the Bridge's Ports:

a) If the Port is the Root Port for the Bridge, then
    1) The Configuration Pending flag parameter and Topology Change Acknowledge flag parameter for the Port are reset.
    2) The Make Forwarding procedure (8.6.12) is used for the Port.
b) Otherwise, if the Port is the Designated Port for the LAN to which it is attached, i.e., the Designated Bridge parameter for the Port is the same as the Bridge Identifier parameter held by the Bridge, and the Designated Port and Port Identifier parameters held for the Port are the same and the Port is not in the Disabled State, then
    1) The Message Age Timer for the Port is stopped, if running.
    2) The Make Forwarding procedure (8.6.12) is used for the Port.
c) Otherwise, if the Port is to be an alternate Port, i.e., is neither the Root Port nor a Designated Port, then
    1) The Configuration Pending flag parameter and Topology Change Acknowledge flag parameter for the Port are reset.
    2) The procedure to Make Blocking (8.6.13) is used.

8.6.12 Make forwarding

8.6.12.1 Purpose

To permit a Port to participate in frame relay, following a suitable interval to ensure that temporary loops in the Bridged LAN do not cause duplication of frames.

8.6.12.2 Use

As part of the Port State Selection procedure (8.6.11).

8.6.12.3 Procedure

If the Port State is Blocking, then

a) The Port State is set to Listening, and
b) The Forward Delay Timer for the Port is started.

8.6.13 Make blocking

8.6.13.1 Purpose

To terminate the participation of a Port in frame relay.

8.6.13.2 Use

As part of the Port State Selection procedure (8.6.11).

8.6.13.3 Procedure

If the Port is not in the Disabled or the Blocking State, then

a) If the Port is in the Forwarding or Learning State and the Change Detection Enabled parameter for the Port is set, the Topology Change Detection procedure (8.6.14) is invoked;
b) The Port State for the Port is set to Blocking;
c) The Forward Delay Timer for the Port is stopped.

8.6.14 Topology change detection

8.6.14.1 Purpose

To record a topology change that has been detected by or notified to the Bridge, and to initiate action to communicate the fact that a topology change has been detected to the Root.

8.6.14.2 Use

8.6.14.2.1 Case 1

On receipt of a Topology Change Notification BPDU on a Port that is the Designated Port for the LAN to which it is attached.

8.6.14.2.2 Case 2

When a Bridge Port is put into the Forwarding State following the expiry of the Forward Delay Timer for the Port, provided that the Change Detection Enabled parameter for that Port is set and the Bridge is the Designated Bridge for at least one of the LANs to which its Ports are attached.

8.6.14.2.3 Case 3

When a Bridge Port in either the Forwarding or the Learning State is put into the Blocking State, provided that the Change Detection Enabled parameter for that Port is set.

8.6.14.2.4 Case 4

When the Bridge becomes the Root.

8.6.14.3 Procedure

a) If the Bridge has been selected as the Root, i.e., the Designated Root and Bridge Identifier parameters held for the Bridge are the same, then
    1) The Topology Change flag parameter held for the Bridge is set.
    2) The Topology Change Timer for the Bridge is started.
b) If the Bridge has not been selected as the Root and the Topology Change Detected flag parameter held for the Bridge is not already set, then
    1) The Transmit Topology Change Notification BPDU procedure (8.6.6) is invoked.
    2) The Topology Change Notification Timer is started.
c) The Topology Change Detected flag parameter for the Bridge is set.

8.6.15 Topology change acknowledged

8.6.15.1 Purpose

To terminate the transmission of Topology Change Notification BPDUs.

8.6.15.2 Use

Following receipt of a Configuration BPDU with the Topology Change Acknowledgment flag parameter set from the Designated Bridge for the LAN to which the Root Port is attached.

8.6.15.3 Procedure

a) The Topology Change Detected flag parameter held for the Bridge is reset;
b) The Topology Change Notification Timer is stopped.

8.6.16 Acknowledge topology change

8.6.16.1 Purpose

To acknowledge the notification of a detected topology change by another Bridge.

8.6.16.2 Use

Following receipt of a Topology Change Notification BPDU on a Port that is the Designated Port for the LAN to which it is attached.

8.6.16.3 Procedure

a) The Topology Change Acknowledge flag parameter for the Port is set;
b) The Transmit Configuration BPDU procedure (8.6.1) is used for the Port.

8.7 Operation of the protocol

A Bridge Protocol Entity shall

a) Communicate with its Peer Entities in other Bridges by the transmission of Bridge Protocol Data Units;
b) Update stored protocol variables and timers;
c) Change the state of the Bridge Ports;

following

— The reception of Bridge Protocol Data Units;
— The expiry of Bridge and Port Timers;

as required by the specification below (8.7.1, 8.7.2, 8.7.3, 8.7.4, 8.7.5, 8.7.6, 8.7.7, and 8.7.8). In any case of ambiguity, reference shall be made to the Procedural Model (8.9), which constitutes the definitive description of the operation of the protocol.

This specification uses the Elements of Procedure of the Protocol described in 8.6, which, taken together with this subclause and the Protocol Parameters and Timers described in 8.5, provide an abstract description of the Spanning Tree Algorithm and Protocol. Conformance to this specification is achieved through maintenance of the Protocol Parameters and Timers and the transmission of BPDUs as described. Implementations are not otherwise constrained; in particular, there is no conformance to individual elements of procedure.

8.7.1 Received Configuration BPDU

8.7.1.1 Case 1

If the Configuration BPDU received conveys protocol information that supersedes that already held for a Port as specified in 8.6.2.2, then the following sequence of Procedures is used:

a) Step 1. The Record Configuration Information procedure (8.6.2).
b) Step 2. The Configuration Update procedure (8.6.7).
c) Step 3. The Port State Selection procedure (8.6.11).
d) Step 4. If the Bridge was selected as the Root prior to Configuration Update, but is no longer, then the Hello Timer (8.5.4.1) is stopped.
e) Step 5. If the Bridge was selected as the Root prior to Configuration Update, but is no longer, and the Topology Change Detected flag parameter is set, then the Topology Change Timer is stopped, the Transmit Topology Change Notification BPDU procedure (8.6.6) is used, and the Topology Change Notification Timer is started.
f) Step 6. If the Configuration BPDU was received on the Root Port (i.e., the Port selected as the Root Port by the Configuration Update procedure), the Record Configuration Timeout Values (8.6.3) and the Configuration BPDU Generation (8.6.4) procedures are used.
g) Step 7. If the Configuration BPDU was received on the Root Port and the Topology Change Acknowledgment flag parameter was set, the Topology Change Acknowledged procedure (8.6.15) is used.

8.7.1.2 Case 2

If the Configuration BPDU received does not convey information superseding that already held for the Port and that Port is the Designated Port for the LAN to which it is attached, i.e., the value of the Designated Bridge and Designated Port parameters held for the Port are the same as that of the Bridge Identifier for the Bridge and the Port Identifier for that Port respectively, then the Reply to Configuration BPDU procedure (8.6.5) is used.

8.7.2 Received Topology Change Notification BPDU

If the Port on which the Topology Change Notification BPDU was received is the Designated Port for the LAN to which it is attached, then

a) Step 1. The Topology Change Detection procedure (8.6.14) is used.
b) Step 2. The Acknowledge Topology Change procedure (8.6.16) is used.

8.7.3 Hello Timer expiry

The Configuration BPDU Generation procedure (8.6.4) is used and the Hello Timer (8.5.4.1) is started.

8.7.4 Message Age Timer expiry

a) Step 1. The procedure to Become Designated Port (8.6.10) is used for the Port for which Message Age Timer has expired.
b) Step 2. The Configuration Update procedure (8.6.7) is used.
c) Step 3. The Port State Selection procedure (8.6.11) is used.
d) Step 4. If the Bridge is selected as the Root following Configuration Update, then
    1) Step 1. The Max Age, Hello Time, and Forward Delay parameters held by the Bridge are set to the values of the Bridge Max Age, Bridge Hello Time, and Bridge Forward Delay parameters.
    2) Step 2. The Topology Change Detection procedure (8.6.14) is used.
    3) Step 3. The Topology Change Notification Timer (8.5.4.2) is stopped.
    4) Step 4. The Configuration BPDU Generation procedure (8.6.4) is used and the Hello Timer is started.

8.7.5 Forward Delay Timer expiry

a) Step 1. If the state of the Port for which the Forward Delay Timer (8.5.6.2) has expired was Listening, then
    1) Step 1. The Port State is set to Learning, and
    2) Step 2. The Forward Delay Timer is restarted.
b) Step 2. Otherwise, if the state of the Port for which the Forward Delay Timer (8.5.6.2) has expired was Learning, then
    1) Step 1. The Port State is set to Forwarding, and
    2) Step 2. If the Bridge is the Designated Bridge for at least one of the LANs to which its Ports are attached and the Change Detection Enabled parameter for the Port is set, the Topology Change Detection procedure (8.6.14) is invoked.

8.7.6 Topology Change Notification Timer Expiry

a) Step 1. The Transmit Topology Change Notification BPDU procedure (8.6.6) is used.
b) Step 2. The Topology Change Notification Timer (8.5.4.2) is restarted.

8.7.7 Topology Change timer

a) Step 1. The Topology Change Detected flag parameter held by the Bridge is reset.
b) Step 2. The Topology Change flag parameter held by the Bridge is reset.

8.7.8 Hold Timer expiry

If the Configuration Pending flag parameter for the Port for which the Hold Timer (8.5.6.3) has expired is set, the Transmit Configuration BPDU procedure (8.6.1) is invoked for that Port.

8.8 Management of the Bridge Protocol Entity

Management control of the Bridge Protocol Entity, which operates the Spanning Tree Algorithm and Protocol, may be exerted in order to

— Meet any requirements for local information and configuration services.
— Support management operations.

This subclause specifies the interaction of the following management operations with the parameters and procedures of the Spanning Tree Algorithm and Protocol:

a) Initialization;
b) Enabling an individual Port;
c) Disabling an individual Port;
d) Changing the priority part of a Bridge Identifier;
e) Changing the priority part of a Port Identifier;
f) Change the Path Cost associated with an individual Port;
g) Enable or disable topology change detection associated with an individual Port.

These operations shall modify the Protocol Parameters and Timers and transmit BPDUs as described below (8.8.1, 8.8.2, 8.8.3, 8.8.4, 8.8.5, 8.8.6, 8.8.7, and 8.8.8). Implementations are not otherwise constrained; in particular, there is no conformance to individual elements of procedure. In any case of ambiguity, reference shall be made to the Procedural Model (8.9), which constitutes the definitive description of these operations.

This subclause does not specify which operations are made available to a remote management station, nor how these are combined and conveyed. Operations and facilities that can be provided by remote management are detailed in Clause 14. Similarly, this subclause does not specify the availability of local information and configuration procedures.

8.8.1 Initialization

a) Step 1. The Designated Root parameter held for the Bridge is set equal to the value of the Bridge Identifier, and the values of the Root Path Cost and Root Port parameter held for the Bridge are set to zero.
b) Step 2. The Max Age, Hello Time, and Forward Delay parameters held by the Bridge are set to the values of the Bridge Max Age, Bridge Hello Time, and Bridge Forward Delay parameters.
c) Step 3. The Topology Change Detected and Topology Change flag parameters for the Bridge are reset, and the Topology Change Notification Timer (8.5.4.2) and Topology Change Timer (8.5.4.3) are stopped, if running.
d) Step 4. For each of the Bridge's Ports
    1) Step 1. The Become Designated Port procedure (8.6.10) is used to assign values to the Designated Root, Designated Cost, Designated Bridge, and Designated Port parameters for the Port.

2) Step 2. The Port State is set to Blocking if the Port is to be enabled following initialization; alternatively, the Port State is set to Disabled.
3) Step 3. The Topology Change Acknowledge flag parameter is reset.
4) Step 4 .The Configuration Pending flag parameter is reset.
5) Step 5. The Message Age Timer (8.5.6.1) is stopped, if running.
6) Step 6. The Forward Delay Timer (8.5.6.2) is stopped, if running.
7) Step 7. The Hold Timer (8.5.6.3) is stopped, if running.
8) Step 8. The Change Detection Enabled flag (8.5.5.10) is set.

e) Step 5. The Port State Selection procedure (8.6.11) is used to select the state of each of the Bridge's Ports.
f) Step 6. The Configuration BPDU Generation procedure (8.6.4) is invoked and the Hello Timer (8.5.4.1) started.

8.8.2 Enable Port

a) Step 1. The Become Designated Port procedure (8.6.10) is used to assign values to the Designated Root, Designated Cost, Designated Bridge, and Designated Port parameters for the Port.
b) Step 2. The Port State is set to Blocking.
c) Step 3. The Topology Change Acknowledge flag parameter is reset.
d) Step 4. The Configuration Pending flag parameter is reset.
e) Step 5. The Message Age Timer (8.5.6.1) is stopped, if running.
f) Step 6. The Forward Delay Timer (8.5.6.2) is stopped, if running.
g) Step 7. The Hold Timer (8.5.6.3) is stopped, if running.
h) Step 8. The Port State Selection procedure (8.6.11) is used.

8.8.3 Disable Port

a) Step 1. The Become Designated Port procedure (8.6.10) is used to assign values to the Designated Root, Designated Cost, Designated Bridge, and Designated Port parameters for the Port.
b) Step 2. The Port State is set to Disabled.
c) Step 3. The Topology Change Acknowledge flag parameter is reset.
d) Step 4. The Configuration Pending flag parameter is reset.
e) Step 5. The Message Age Timer (8.5.6.1) is stopped, if running.
f) Step 6. The Forward Delay Timer (8.5.6.2) is stopped, if running.
g) Step 7. The Configuration Update procedure (8.6.7) is used.
h) Step 8. The Port State Selection procedure (8.6.11) is used.
i) Step 9. If the Bridge has been selected as the Root following Configuration Update, then
    1) Step 1. The Max Age, Hello Time, and Forward Delay parameters held by the Bridge are set to the values of the Bridge Max Age, Bridge Hello Time, and Bridge Forward Delay parameters.
    2) Step 2. The Topology Change Detection procedure (8.6.14) is used.
    3) Step 3. The Topology Change Notification Timer (8.5.4.2) is stopped.
    4) Step 4. The Configuration BPDU Generation procedure (8.6.4) is used and the Hello Timer is started.

8.8.4 Set Bridge Priority

a) Step 1. The new value of the Bridge Identifier is calculated.
b) Step 2. The value of the Designated Bridge parameter held for each Port that has been selected as the Designated Port for the LAN to which it is attached (i.e., for which the value of the Designated Bridge and Designated Port parameters were the same as that of the Bridge Identifier and the Port Identifier for that Port, respectively; is set to the new value of the Bridge Identifier.
c) Step 3. The Bridge Identifier parameter held by the Bridge is set to the new value.
d) Step 4. The Configuration Update procedure (8.6.7) is used.
e) Step 5. The Port State Selection procedure (8.6.11) is used.

f) Step 6. If the Bridge has been selected as the Root following Configuration Update, then
   1) Step 1. The Max Age, Hello Time, and Forward Delay parameters held by the Bridge are set to the values of the Bridge Max Age, Bridge Hello Time, and Bridge Forward Delay parameters.
   2) Step 2. The Topology Change Detection procedure (8.6.14) is used.
   3) Step 3. The Topology Change Notification Timer (8.5.4.2) is stopped.
   4) Step 4. The Configuration BPDU Generation procedure (8.6.4) is used and the Hello Timer is started.

8.8.5 Set Port Priority

a) Step 1. The new value of the Port Identifier is calculated.
b) Step 2. If the Port has been selected as the Designated Port for the LAN to which it is attached (i.e., the value of the Designated Bridge and Designated Port parameters were the same as that of the Bridge Identifier and the Port Identifier, respectively), the Designated Port parameter held for the Port is set to the new value of the Port Identifier.
c) Step 3. The Port Identifier parameter held for the Port is set to the new value.
d) Step 4. If the value of the Designated Bridge parameter held for the Port is equal to that of the Bridge's Bridge Identifier, and the new value of the Port Identifier is of higher priority than that recorded as the Designated Port, then
   1) Step 1. The Become Designated Port procedure (8.6.10) is used to assign values to the Designated Root, Designated Cost, Designated Bridge, and Designated Port parameters for the Port.
   2) Step 2. The Port State Selection procedure (8.6.11) is used.

8.8.6 Set Path Cost

a) Step 1. The Path Cost parameter for the Port is set to the new value.
b) Step 2. The Configuration Update procedure (8.6.7) is used.
c) Step 3. The Port State Selection procedure is used.

8.8.7 Enable Change Detection

The Change Detection Enabled flag for the Port (8.5.5.10) is set.

8.8.8 Disable Change Detection

The Change Detection Enabled flag for the Port (8.5.5.10) is reset.

8.9 Procedural model

This subclause constitutes the definitive description of the operation of the Spanning Tree Algorithm and Protocol. The natural language text in 8.6, 8.7, and 8.8 of this standard is intended to informally present the semantics of operation specified here. Should differences of interpretation exist between that text and this procedural model, the latter shall take precedence.

8.9.1 Overview

The parameters, timers, elements of procedure, and operation of the protocol are presented below as a compilable program in the computer language C (ANSI X3.159).

The objective of presenting this program is to precisely and unambiguously specify the operation of the algorithm and protocol. The description of the operation of the protocol in a computer language is in no way intended to constrain the implementation of the protocol; a real implementation may employ any appropriate technology.

Conformance of equipment to this standard is purely in respect of observable protocol. The program contained in this subclause contains modeling details that are of local concern to an implementation; there is no conformance in respect of these details.

The natural language text in 8.6, 8.7, and 8.8 follows the computer language text contained in this subclause. In order to preserve the compactness of the program text, all comments are made by reference to the natural language description and are of the form 4.n.n.n. Where a program statement invokes an element of procedure, a further reference is made to the particular condition, of those listed for the procedure, that has caused the procedure to be invoked.

In the computer language description of the procedural model, it should be noted that the BPDUs are shown in their unpacked form, i.e., unpacked into individual parameters. The actual format of the BPDUs, as passed across the LLC service boundary, is shown in Clause 9.

```
/**************************************************************************
*
*  SPANNING TREE ALGORITHM AND PROTOCOL
*
**************************************************************************/

/**************************************************************************
*  DEFINED CONSTANTS
**************************************************************************/

#define Zero        0
#define One         1

#define False       0
#define True        1

/** port states. **/

#define Disabled       0                                  /* (8.4.5)        */
#define Listening      1                                  /* (8.4.2)        */
#define Learning       2                                  /* (8.4.3)        */
#define Forwarding     3                                  /* (8.4.4)        */
#define Blocking       4                                  /* (8.4.1)        */

/** BPDU type constants **/

#define Config_bpdu_type     0
#define Tcn_bpdu_type      128

/** pseudo-implementation constants. **/

#define No_of_ports 2
        /* arbitrary choice, to allow the code below to compile */

#define All_ports No_of_ports+1
        /* ports start at 1, arrays in C start at 0 */

#define Default_path_cost 10
        /* arbitrary */

#define Message_age_increment 1
        /* minimum increment possible to avoid underestimating age, allows
           for BPDU transmission time */

#define No_port 0
        /* reserved value for Bridge's root port parameter indicating no
           root port, used when Bridge is the root */
```



```
/*****************************************************************************
*  TYPEDEFS, STRUCTURES, AND UNION DECLARATIONS
*****************************************************************************/

/** basic types. **/

typedef int Int;        /* to align with convention used here for use of
                           case. Types and defined constants have their
                           initial letters capitalized. */

typedef Int Boolean;    /* : (True, False) */

typedef Int State;      /* : (Disabled, Listening, Learning,
                              Forwarding, Blocking) */


/** BPDU encoding types defined in Clause 9, “Encoding of Bridge Protocol
Data Units” are:

Protocol_version     (9.2.2)

Bpdu_type            (9.2.3)

Flag                 (9.2.4)

Identifier           (9.2.5)

Cost                 (9.2.6)

Port_id              (9.2.7)

Time                 (9.2.8)

**/

#include “types.c”      /* defines BPDU encoding types */

/** Configuration BPDU Parameters (8.5.1) **/

typedef struct
{
   Bpdu_type  type;

   Identifier root_id;                                  /* (8.5.1.1)      */

   Cost       root_path_cost;                           /* (8.5.1.2)      */

   Identifier bridge_id;                                /* (8.5.1.3)      */

   Port_id    port_id;                                  /* (8.5.1.4)      */

   Time       message_age;                              /* (8.5.1.5)      */

   Time       max_age;                                  /* (8.5.1.6)      */

   Time       hello_time;                               /* (8.5.1.7)      */

   Time       forward_delay;                            /* (8.5.1.8)      */

   Flag       topology_change_acknowledgment;           /* (8.5.1.9)      */

   Flag       topology_change;                          /* (8.5.1.10)     */

} Config_bpdu;
```

```
/** Topology Change Notification BPDU Parameters (8.5.2) **/

typedef struct
{
   Bpdu_type  type;

} Tcn_bpdu;

/** Bridge Parameters (8.5.3) **/

typedef struct
{
   Identifier designated_root;                          /* (8.5.3.1)      */

   Cost       root_path_cost;                           /* (8.5.3.2)      */

   Int        root_port;                                /* (8.5.3.3)      */

   Time       max_age;                                  /* (8.5.3.4)      */

   Time       hello_time;                               /* (8.5.3.5)      */

   Time       forward_delay;                            /* (8.5.3.6)      */

   Identifier bridge_id;                                /* (8.5.3.7)      */

   Time       bridge_max_age;                           /* (8.5.3.8)      */

   Time       bridge_hello_time;                        /* (8.5.3.9)      */

   Time       bridge_forward_delay;                     /* (8.5.3.10)     */

   Boolean    topology_change_detected;                 /* (8.5.3.11)     */

   Boolean    topology_change;                          /* (8.5.3.12)     */

   Time       topology_change_time;                     /* (8.5.3.13)     */

   Time       hold_time;                                /* (8.5.3.14)     */

} Bridge_data;

/** Port Parameters (8.5.5) **/

typedef struct
{
   Port_id    port_id;                                  /* (8.5.5.1)      */

   State      state;                                    /* (8.5.5.2)      */

   Int        path_cost;                                /* (8.5.5.3)      */

   Identifier designated_root;                          /* (8.5.5.4)      */

   Int        designated_cost;                          /* (8.5.5.5)      */

   Identifier designated_bridge;                        /* (8.5.5.6)      */

   Port_id    designated_port;                          /* (8.5.5.7)      */

   Boolean    topology_change_acknowledge;              /* (8.5.5.8)      */

   Boolean    config_pending;                           /* (8.5.5.9)      */
```

```
   Boolean    change_detection_enabled;                    /* (8.5.5.10)      */

} Port_data;

/** types to support timers for this pseudo-implementation. **/

typedef struct
{
   Boolean   active;         /* timer in use. */

   Time      value;          /* current value of timer, counting up. */

} Timer;



/*****************************************************************************
*  STATIC STORAGE ALLOCATION
*****************************************************************************/


Bridge_data   bridge_info;                                 /* (8.5.3)        */

Port_data     port_info[All_ports];                        /* (8.5.5)        */

Config_bpdu   config_bpdu[All_ports];

Tcn_bpdu      tcn_bpdu[All_ports];

Timer         hello_timer;                                 /* (8.5.4.1)      */

Timer         tcn_timer;                                   /* (8.5.4.2)      */

Timer         topology_change_timer;                       /* (8.5.4.3)      */

Timer         message_age_timer[All_ports];                /* (8.5.6.1)      */

Timer         forward_delay_timer[All_ports];              /* (8.5.6.2)      */

Timer         hold_timer[All_ports];                       /* (8.5.6.3)      */

/*****************************************************************************
*  CODE
*****************************************************************************/

/** Elements of Procedure (8.6) **/

transmit_config(port_no)                                   /* (8.6.1)        */
Int     port_no;
{
   if (hold_timer[port_no].active)                         /* (8.6.1.3.1)    */
   {
      port_info[port_no].config_pending = True;            /* (8.6.1.3.1)    */
   }
   else                                                    /* (8.6.1.3.2)    */
   {
      config_bpdu[port_no].type = Config_bpdu_type;

      config_bpdu[port_no].root_id = bridge_info.designated_root;
                                                           /* (8.6.1.3.2(a))*/
      config_bpdu[port_no].root_path_cost = bridge_info.root_path_cost;
                                                           /* (8.6.1.3.2(b))*/
      config_bpdu[port_no].bridge_id = bridge_info.bridge_id;
                                                           /* (8.6.1.3.2(c))*/
      config_bpdu[port_no].port_id = port_info[port_no].port_id;
```

```
                                                        /* (8.6.1.3.2(d))*/
      if (root_bridge())
      {
         config_bpdu[port_no].message_age = Zero;          /* (8.6.1.3.2(e))*/
      }
      else
      {
         config_bpdu[port_no].message_age
           =  message_age_timer[bridge_info.root_port].value
              + Message_age_increment;                     /* (8.6.1.3.2(f))*/
      }
      config_bpdu[port_no].max_age = bridge_info.max_age;  /* (8.6.1.3.2(g))*/
      config_bpdu[port_no].hello_time = bridge_info.hello_time;
      config_bpdu[port_no].forward_delay = bridge_info.forward_delay;
      config_bpdu[port_no].topology_change_acknowledgment
        =  port_info[port_no].topology_change_acknowledge;
                                                        /* (8.6.1.3.2(h)) */
      config_bpdu[port_no].topology_change
        = bridge_info.topology_change;                  /* (8.6.1.3.2(i)) */

      if (config_bpdu[port_no].message_age < bridge_info.max_age)
      {
         port_info[port_no].topology_change_acknowledge = False;
                                                           /* (8.6.1.3.3) */
         port_info[port_no].config_pending = False;        /* (8.6.1.3.3)*/
         send_config_bpdu(port_no, &config_bpdu[port_no]);
         start_hold_timer(port_no);                        /* (8.6.3.3(b))*/
       }
   }

   }
        /* where

        send_config_bpdu(port_no,bpdu)
        Int         port_no;
        Config_bpdu *bpdu;

        is a pseudo-implementation specific routine that transmits
        the bpdu on the specified port within the specified time.

        */

        /* and */

        Boolean root_bridge()
        {
           return(bridge_info.designated_root == bridge_info.bridge_id);
        }
Boolean supersedes_port_info(port_no, config)           /* (8.6.2.2)      */
Int port_no;
Config_bpdu *config;
{
   return (
   (  config->root_id
      <  port_info[port_no].designated_root             /* (8.6.2.2 a)    */
   )
   ||
   (  (  config->root_id
         == port_info[port_no].designated_root
      )
      &&
      (  (  config->root_path_cost
            <  port_info[port_no].designated_cost       /* (8.6.2.2 b)    */
         )
         ||
```

```
         (  (  config->root_path_cost
               == port_info[port_no].designated_cost
            )
            &&
            (  (  config->bridge_id
                  <  port_info[port_no].designated_bridge  /* (8.6.2.2 c)    */
               )
               ||
               (  (  config->bridge_id
                     == port_info[port_no].designated_bridge
                  )                                       /* (8.6.2.2 d)    */
                  &&
                  (  (  config->bridge_id != bridge_info.bridge_id
                     )                                    /* (8.6.2.2 d1) */
                     ||
                     (  config->port_id
                        <= port_info[port_no].designated_port
                     )                                    /* (8.6.2.2 d2) */
   )  )  )  )  )  )
          );
}

record_config_information(port_no, config)                 /* (8.6.2)       */
Int port_no;
Config_bpdu *config;
{
   port_info[port_no].designated_root = config->root_id;    /* (8.6.2.3.1)   */
   port_info[port_no].designated_cost = config->root_path_cost;
   port_info[port_no].designated_bridge = config->bridge_id;
   port_info[port_no].designated_port = config->port_id;

   start_message_age_timer(port_no, config->message_age);   /* (8.6.2.3.2)   */
}

record_config_timeout_values(config)                       /* (8.6.3)       */
Config_bpdu *config;
{
   bridge_info.max_age = config->max_age;                  /* (8.6.3.3)     */
   bridge_info.hello_time = config->hello_time;
   bridge_info.forward_delay = config->forward_delay;
   bridge_info.topology_change = config->topology_change;
}

config_bpdu_generation()                                   /* (8.6.4)       */
{
   Int port_no;

   for (port_no = One; port_no <= No_of_ports; port_no++)  /* (8.6.4.3)     */
   {
      if (  designated_port(port_no)                       /* (8.6.4.3)     */
            &&
            (port_info[port_no].state != Disabled)
         )
      {
         transmit_config(port_no);                         /* (8.6.4.3)     */
      }                                                    /* (8.6.1.2)     */
   }
}

        /* where */

        Boolean designated_port(port_no)
        Int port_no;
        {
           return (  (  port_info[port_no].designated_bridge
```

```
                        == bridge_info.bridge_id
                     )
                     &&
                     (  port_info[port_no].designated_port
                         == port_info[port_no].port_id
                     )
                  );
        }
reply(port_no)                                          /* (8.6.5)        */
Int port_no;
{
   transmit_config(port_no);                            /* (8.6.5.3)      */
}


transmit_tcn()                                          /* (8.6.6)        */
{
   Int port_no;

   port_no = bridge_info.root_port;
   tcn_bpdu[port_no].type = Tcn_bpdu_type;

   send_tcn_bpdu(port_no, &tcn_bpdu[bridge_info.root_port]);/* (8.6.6.3)     */

}

        /* where

        send_tcn_bpdu(port_no,bpdu)
        Int         port_no;
        Tcn_bpdu    *bpdu;

        is a pseudo-implementation-specific routine that transmits
        the bpdu on the specified port within the specified time.

        */

configuration_update()                                  /* (8.6.7)        */
{
   root_selection();                                    /* (8.6.7.3.1)    */
                                                        /* (8.6.8.2)      */

   designated_port_selection();                         /* (8.6.7.3.2)    */
                                                        /* (8.6.9.2)      */
}

root_selection()                                         /* (8.6.8)       */
{
   Int root_port;
   Int port_no;

   root_port = No_port;

   for (port_no = One; port_no <= No_of_ports; port_no++)     /* (8.6.8.3.1) */
   {
      if (  (  (!designated_port(port_no))
               &&
               (port_info[port_no].state != Disabled)
               &&
               (port_info[port_no].designated_root < bridge_info.bridge_id)
            )
            &&
            (  (root_port == No_port)
               ||
               (  port_info[port_no].designated_root
```

```
                  <  port_info[root_port].designated_root  /* (8.6.8.3.1(a)) */
               )
               ||
               (  (  port_info[port_no].designated_root
                     ==  port_info[root_port].designated_root
                  )
                  &&
                  (  (  (  port_info[port_no].designated_cost
                           + port_info[port_no].path_cost
                        )
                        <
                        (  port_info[root_port].designated_cost
                           + port_info[root_port].path_cost
                        )                                  /* (8.6.8.3.1(b)) */
                     )
                     ||
                     (  (  (  port_info[port_no].designated_cost
                              + port_info[port_no].path_cost
                           )
                           ==
                           (  port_info[root_port].designated_cost
                              + port_info[root_port].path_cost
                           )
                        )
                        &&
                        (  (  port_info[port_no].designated_bridge
                              <  port_info[root_port].designated_bridge
                           )                               /* (8.6.8.3.1(c)) */
                           ||
                           (  (  port_info[port_no].designated_bridge
                                 ==  port_info[root_port].designated_bridge
                              )
                              &&
                              (  (  port_info[port_no].designated_port
                                    <  port_info[root_port].designated_port
                                 )                         /* (8.6.8.3.1(d)) */
                                 ||
                                 (  (  port_info[port_no].designated_port
                                       == port_info[root_port].designated_port
                                    )
                                    &&
                                    (  port_info[port_no].port_id
                                       <  port_info[root_port].port_id
                                    )                      /* (8.6.8.3.1(e)) */
      )  )  )  )  )  )  )  )  )
   {
      root_port = port_no;
   }
}

bridge_info.root_port = root_port;                         /* (8.6.8.3.1)    */

if (root_port == No_port)                                  /* (8.6.8.3.2)    */
{

   bridge_info.designated_root = bridge_info.bridge_id;
                                                           /* (8.6.8.3.2(a)) */
   bridge_info.root_path_cost = Zero;                      /* (8.6.8.3.2(b)) */
}
else                                                       /* (8.6.8.3.3)    */
{
   bridge_info.designated_root = port_info[root_port].designated_root;
                                                           /* (8.6.8.3.3(a)) */
   bridge_info.root_path_cost = (  port_info[root_port].designated_cost
                                 + port_info[root_port].path_cost
```

```
                                    );                      /* (8.6.8.3.3(b)) */
   }
}
designated_port_selection()                                 /* (8.6.9)        */
{
   Int port_no;

   for (port_no = One; port_no <= No_of_ports; port_no++)  /* (8.6.9.3)      */
   {
      if (  designated_port(port_no)                        /* (8.6.9.3 a)    */
            ||
            (
               port_info[port_no].designated_root
               != bridge_info.designated_root               /* (8.6.9.3 b)    */
            )
            ||
            (  bridge_info.root_path_cost
               <  port_info[port_no].designated_cost
            )                                               /* (8.6.9.3 c)    */
            ||
            (  (  bridge_info.root_path_cost
                  == port_info[port_no].designated_cost
               )
               &&
               (  (  bridge_info.bridge_id
                     <  port_info[port_no].designated_bridge
                  )                                         /* (8.6.9.3 d)    */
                  ||
                  (  (  bridge_info.bridge_id
                           == port_info[port_no].designated_bridge
                     )
                     &&
                     (  port_info[port_no].port_id
                        <=  port_info[port_no].designated_port
                     )                                      /* (8.6.9.3 e)    */
         )  )  )  )
      {
         become_designated_port(port_no);                   /* (8.6.10.2 a) */
      }
   }
}

become_designated_port(port_no)                             /* (8.6.10)       */
Int port_no;
{
    port_info[port_no].designated_root = bridge_info.designated_root;
                                                            /* (8.6.10.3 a)   */

    port_info[port_no].designated_cost = bridge_info.root_path_cost;
                                                            /* (8.6.10.3 b)   */

    port_info[port_no].designated_bridge = bridge_info.bridge_id;
                                                            /* (8.6.10.3 c)   */

    port_info[port_no].designated_port = port_info[port_no].port_id;
                                                            /* (8.6.10.3 d)   */
}

port_state_selection()                                      /* (8.6.11)       */
{
   Int port_no;

   for (port_no = One; port_no <= No_of_ports; port_no++)
   {
      if (port_no == bridge_info.root_port)                 /* (8.6.11.3 a)   */
```

```
      {
         port_info[port_no].config_pending = False;        /* (8.6.11.3 a1)*/
         port_info[port_no].topology_change_acknowledge = False;

         make_forwarding(port_no);                         /* (8.6.11.3 a2)*/
      }
      else if (designated_port(port_no))                   /* (8.6.11.3 b)   */
      {
         stop_message_age_timer(port_no);                  /* (8.6.11.3 b1)*/

         make_forwarding(port_no);                         /* (8.6.11.3 b2)*/
      }
      else                                                 /* (8.6.11.3 c)   */
      {
         port_info[port_no].config_pending = False;        /* (8.6.11.3 c1)*/
         port_info[port_no].topology_change_acknowledge = False;

         make_blocking(port_no);                           /* (8.6.11.3 c2)*/
      }
   }
}
make_forwarding(port_no)                                   /* (8.6.12)       */
Int port_no;
{
   if (port_info[port_no].state == Blocking)               /* (8.6.12.3)     */
   {
      set_port_state(port_no, Listening);                  /* (8.6.12.3 a)   */

      start_forward_delay_timer(port_no);                  /* (8.6.12.3 b)   */
   }
}



make_blocking(port_no)                                     /* (8.6.13)       */
Int port_no;
{
   if (  (port_info[port_no].state != Disabled)
         &&
         (port_info[port_no].state != Blocking)
      )                                                    /* (8.6.13.3)     */
   {
      if (  (port_info[port_no].state == Forwarding)
            ||
            (port_info[port_no].state == Learning)
         )
      {
         if (port_info[port_no].change_detection_enabled == True)
                                                           /* (8.5.5.10)     */
         {
             topology_change_detection();                  /* (8.6.13.3 a)   */
         }                                                 /* (8.6.14.2.3)   */
      }

      set_port_state(port_no, Blocking);                   /* (8.6.13.3 b)   */

      stop_forward_delay_timer(port_no);                   /* (8.6.13.3 c)   */
   }
}

        /* where */

        set_port_state(port_no, state)
        Int port_no;
        State state;
```

```
        {
              port_info[port_no].state = state;
        }

topology_change_detection()                                  /* (8.6.14)       */
{
   if (root_bridge())                                        /* (8.6.14.3 a)   */
   {
      bridge_info.topology_change = True;                    /* (8.6.14.3 a1)*/

      start_topology_change_timer();                         /* (8.6.14.3 a2)*/
   }

   else if (bridge_info.topology_change_detected == False) /* (8.6.14.3 b)   */
   {
      transmit_tcn();                                        /* (8.6.14.3 b1)*/

      start_tcn_timer();                                     /* (8.6.14.3 b2)*/
   }

   bridge_info.topology_change_detected = True;              /* (8.6.14.3 c)   */
}

topology_change_acknowledged()                               /* (8.6.15)      */
{
   bridge_info.topology_change_detected = False;             /* (8.6.15.3 a)  */

   stop_tcn_timer();                                         /* (8.6.15.3 b)  */
}

acknowledge_topology_change(port_no)                         /* (8.6.16)      */
Int port_no;
{
   port_info[port_no].topology_change_acknowledge = True;    /* (8.6.16.3 a)  */

   transmit_config(port_no);                                 /* (8.6.16.3 b)  */
}

/** Operation of the Protocol (8.7) **/

received_config_bpdu(port_no, config)                        /* (8.7.1)        */
Int port_no;
Config_bpdu *config;
{
   Boolean root;

   root = root_bridge();

   if (port_info[port_no].state != Disabled)
   {
      if (supersedes_port_info(port_no, config))             /* (8.7.1.1)      */
      {                                                      /* (8.6.2.2)      */
         record_config_information(port_no, config);         /* (8.7.1.1 a)    */
                                                             /* (8.6.2.2)      */
         configuration_update();                             /* (8.7.1.1 b)    */
                                                             /* (8.6.7.2 a)    */
         port_state_selection();                             /* (8.7.1.1 c)    */
                                                             /* (8.6.11.2 a)   */

         if ((!root_bridge()) && root)                       /* (8.7.1.1 d)    */
         {
            stop_hello_timer();

            if (bridge_info.topology_change_detected)        /* (8.7.1.1 e)    */
            {
```

```
               stop_topology_change_timer();

               transmit_tcn();                                  /* (8.6.6.1)      */

               start_tcn_timer();
            }
         }

         if (port_no == bridge_info.root_port)
         {
            record_config_timeout_values(config);            /* (8.7.1.1 e)    */
                                                             /* (8.6.3.2)      */
            config_bpdu_generation();                        /* (8.6.4.2 a)    */

            if (config->topology_change_acknowledgment)      /* (8.7.1.1 g)    */
            {
               topology_change_acknowledged();               /* (8.6.15.2)     */
            }
         }
      }

      else if (designated_port(port_no))                     /* (8.7.1.2)      */
      {
         reply(port_no);                                     /* (8.7.1.2)    */
                                                             /* (8.6.5.2)      */
      }
   }
}

received_tcn_bpdu(port_no, tcn)                              /* (8.7.2)        */
Int port_no;
Tcn_bpdu *tcn;
{
   if (port_info[port_no].state != Disabled)
   {
      if (designated_port(port_no))
      {
         topology_change_detection();                        /* (8.7.2 a)      */
                                                             /* (8.6.14.2.1)   */
         acknowledge_topology_change(port_no);               /* (8.7.2 b)      */
      }                                                      /* (8.6.16.2)     */
   }
}

hello_timer_expiry()                                         /* (8.7.3)        */
{
   config_bpdu_generation();                                 /* (8.6.4.2 b)    */

   start_hello_timer();
}

message_age_timer_expiry(port_no)                            /* (8.7.4)        */
Int port_no;
{
   Boolean root;

   root = root_bridge();

   become_designated_port(port_no);                          /* (8.7.4 a)      */
                                                             /* (8.6.10.2 b)   */
   configuration_update();                                   /* (8.7.4 b)      */
                                                             /* (8.6.7.2 b)    */
   port_state_selection();                                   /* (8.7.4 c)      */
                                                             /* (8.6.11.2 b)   */
```

```
   if ((root_bridge()) && (!root))                          /* (8.7.4 d)      */
   {
      bridge_info.max_age = bridge_info.bridge_max_age;     /* (8.7.4 d1)     */
      bridge_info.hello_time = bridge_info.bridge_hello_time;
      bridge_info.forward_delay = bridge_info.bridge_forward_delay;

      topology_change_detection();                          /* (8.7.4 d2)     */
                                                            /* (8.6.14.2.4)   */
      stop_tcn_timer();                                     /* (8.7.4 d3)     */

      config_bpdu_generation();                             /* (8.7.4 d4)     */

      start_hello_timer();
   }
}

forward_delay_timer_expiry(port_no)                         /* (8.7.5)        */
Int port_no;
{
   if (port_info[port_no].state == Listening)               /* (8.7.5 a)      */
   {
      set_port_state(port_no, Learning);                    /* (8.7.5 a1)     */

      start_forward_delay_timer(port_no);                   /* (8.7.5 a2)     */
   }

   else if (port_info[port_no].state == Learning)           /* (8.7.5 b)      */
   {
      set_port_state(port_no, Forwarding);                  /* (8.7.5 b1)     */

      if (designated_for_some_port())                       /* (8.7.5 b2)     */
      {
         if (port_info[port_no].change_detection_enabled == True)
                                                            /* (8.5.5.10)     */
         {
             topology_change_detection();                   /* (8.6.14.2.2)   */
         }
      }
   }
}

        /* where */

        Boolean designated_for_some_port()
        {
           Int port_no;

           for (port_no = One; port_no <= No_of_ports; port_no++)
           {
              if ( port_info[port_no].designated_bridge
                   == bridge_info.bridge_id
                 )
              {
                 return(True);
              }
           }

           return(False);
        }

tcn_timer_expiry()                                          /* (8.7.6)        */
{
      transmit_tcn();                                       /* (8.7.6 a)      */

      start_tcn_timer();                                    /* (8.7.6 b)      */
```



```
}

topology_change_timer_expiry()                              /* (8.7.7)        */
{
      bridge_info.topology_change_detected = False;         /* (8.7.7 a)      */

      bridge_info.topology_change = False;                  /* (8.7.7 b)      */
}

hold_timer_expiry(port_no)                                  /* (8.7.8)        */
Int port_no;
{
   if (port_info[port_no].config_pending)
   {
      transmit_config(port_no);                             /* (8.6.1.2)      */
   }
}
/** Management of the Bridge Protocol Entity (8.8) **/
initialisation()                                            /* (8.8.1)        */
{
   Int port_no;

   bridge_info.designated_root = bridge_info.bridge_id;     /* (8.8.1 a)      */
   bridge_info.root_path_cost = Zero;
   bridge_info.root_port = No_port;

   bridge_info.max_age = bridge_info.bridge_max_age;        /* (8.8.1 b)      */
   bridge_info.hello_time = bridge_info.bridge_hello_time;
   bridge_info.forward_delay = bridge_info.bridge_forward_delay;

   bridge_info.topology_change_detected = False;            /* (8.8.1 c)      */
   bridge_info.topology_change = False;
   stop_tcn_timer();
   stop_topology_change_timer();

   for (port_no = One; port_no <= No_of_ports; port_no++)   /* (8.8.1 d)      */
   {

      initialize_port(port_no);

   }

   port_state_selection();                                  /* (8.8.1 e)      */

   config_bpdu_generation();                                /* (8.8.1 f)      */
   start_hello_timer();
}
                                                                               /*

                                                                               */
   initialize_port(port_no)
   Int port_no;
   {

      become_designated_port(port_no);                      /* (8.8.1 d1)    */

      set_port_state(port_no, Blocking);                    /* (8.8.1 d2)    */

      port_info[port_no].topology_change_acknowledge = False;
                                                            /* (8.8.1 d3)    */

      port_info[port_no].config_pending = False;            /* (8.8.1 d4)    */

      port_info[port_no].change_detection_enabled = True;   /* (8.8.1 d8)    */
```

```
      stop_message_age_timer(port_no);                    /* (8.8.1 d5)    */

      stop_forward_delay_timer(port_no);                  /* (8.8.1 d6)    */

      stop_hold_timer(port_no);                           /* (8.8.1 d7)    */
   }

enable_port(port_no)                                      /* (8.8.2)        */
Int port_no;
{

   initialize_port(port_no);

   port_state_selection();                                /* (8.8.2 g)      */
}
                                                          /*

                                                          */

disable_port(port_no)                                     /* (8.8.3)       */
Int port_no;
{
   Boolean root;

   root = root_bridge();

   become_designated_port(port_no);                       /* (8.8.3 a)      */

   set_port_state(port_no, Disabled);                     /* (8.8.3 b)      */

   port_info[port_no].topology_change_acknowledge = False; /* (8.8.3 c)      */

   port_info[port_no].config_pending = False;             /* (8.8.3 d)      */

   stop_message_age_timer(port_no);                       /* (8.8.3 e)      */

   stop_forward_delay_timer(port_no);                     /* (8.8.3 f)      */

   configuration_update();                                /* (8.8.3 g)      */

   port_state_selection();                                /* (8.8.3 h)      */

   if ((root_bridge()) && (!root))                        /* (8.8.3 i)      */
   {
      bridge_info.max_age = bridge_info.bridge_max_age;   /* (8.8.3 i1)    */
      bridge_info.hello_time = bridge_info.bridge_hello_time;
      bridge_info.forward_delay = bridge_info.bridge_forward_delay;

      topology_change_detection();                        /* (8.8.3 i2)    */

      stop_tcn_timer();                                   /* (8.8.3 i3)    */

      config_bpdu_generation();                           /* (8.8.3 i4)    */

      start_hello_timer();
   }
}

set_bridge_priority(new_bridge_id)                        /* (8.8.4)        */
Identifier new_bridge_id;                                 /* (8.8.4 a)      */
{
   Boolean root;
   Int port_no;

   root = root_bridge();
```



```
   for (port_no = One; port_no <= No_of_ports; port_no++)  /* (8.8.4 b)      */
   {
      if (designated_port(port_no))
      {
         port_info[port_no].designated_bridge = new_bridge_id;
      }
   }

   bridge_info.bridge_id = new_bridge_id;                  /* (8.8.4 c)      */

   configuration_update();                                 /* (8.8.4 d)      */

   port_state_selection();                                 /* (8.8.4 e)      */

   if ((root_bridge()) && (!root))                         /* (8.8.4 f)      */
   {
      bridge_info.max_age = bridge_info.bridge_max_age;    /* (8.8.4 f1)    */
      bridge_info.hello_time = bridge_info.bridge_hello_time;
      bridge_info.forward_delay = bridge_info.bridge_forward_delay;

      topology_change_detection();                         /* (8.8.4 f2)    */

      stop_tcn_timer();                                    /* (8.8.4 f3)    */

      config_bpdu_generation();                            /* (8.8.4 f4)    */

      start_hello_timer();
   }
}

set_port_priority(port_no, new_port_id)                    /* (8.8.5)        */
Int port_no;
Port_id new_port_id;                                       /* (8.8.5 a)      */
{
   if (designated_port(port_no))                           /* (8.8.5 b)      */
   {
      port_info[port_no].designated_port = new_port_id;
   }

   port_info[port_no].port_id = new_port_id;               /* (8.8.5 c)      */

   if ( (  bridge_info.bridge_id                           /* (8.8.5 d)      */
            == port_info[port_no].designated_bridge
        )
        &&
        (  port_info[port_no].port_id
           <  port_info[port_no].designated_port
        )
      )
   {
      become_designated_port(port_no);                     /* (8.8.5 d1)   */

      port_state_selection();                              /* (8.8.5 d2)   */
   }
}

                                                                          /*

                                                                          */
set_path_cost(port_no, path_cost)                          /* (8.8.6)        */
Int port_no;
Cost path_cost;
{
   port_info[port_no].path_cost = path_cost;               /* (8.8.6 a)      */
```

```
   configuration_update();                              /* (8.8.6 b)      */

   port_state_selection();                              /* (8.8.6 c)      */
}

                                                                        /*

                                                                        */
enable_change_detection(port_no)                        /* (8.8.7)        */
Int port_no;
{
   port_info[port_no].change_detection_enabled = True;
}
                                                                        /*

                                                                        */
disable_change_detection(port_no)                       /* (8.8.8)        */
Int port_no;
{
   port_info[port_no].change_detection_enabled = False;
}
/** pseudo-implementation-specific timer running support **/

tick()
{
   Int port_no;

   if (hello_timer_expired())
   {  hello_timer_expiry();
   }

   if (tcn_timer_expired())
   {  tcn_timer_expiry();
   }

   if (topology_change_timer_expired())
   {  topology_change_timer_expiry();
   }

   for (port_no = One; port_no <= No_of_ports; port_no++)
   {
      if (message_age_timer_expired(port_no))
      {
         message_age_timer_expiry(port_no);
      }
   }

   for (port_no = One; port_no <= No_of_ports; port_no++)
   {
      if (forward_delay_timer_expired(port_no))
      {
         forward_delay_timer_expiry(port_no);
      }
      if (hold_timer_expired(port_no))
      {
         hold_timer_expiry(port_no);
      }
   }
}

/* where */

start_hello_timer()
{  hello_timer.value = (Time) Zero;
```

```
   hello_timer.active = True;
}

stop_hello_timer()
{  hello_timer.active = False;
}
Boolean hello_timer_expired()
{  if (hello_timer.active && (++hello_timer.value >= bridge_info.hello_time))
   {   hello_timer.active = False;
       return(True);
   }
   return(False);
}

start_tcn_timer()
{  tcn_timer.value = (Time) Zero;
   tcn_timer.active = True;
}

stop_tcn_timer()
{  tcn_timer.active = False;
}

Boolean tcn_timer_expired()
{  if (tcn_timer.active && (++tcn_timer.value >= bridge_info.bridge_hello_time))
   {   tcn_timer.active = False;
       return(True);
   }
   return(False);
}

start_topology_change_timer()
{  topology_change_timer.value = (Time) Zero;
   topology_change_timer.active = True;
}

stop_topology_change_timer()
{  topology_change_timer.active = False;
}

Boolean topology_change_timer_expired()
{  if ( topology_change_timer.active
        && ( ++topology_change_timer.value
             >= bridge_info.topology_change_time
           )
       )
   {   topology_change_timer.active = False;
       return(True);
   }
   return(False);
}

start_message_age_timer(port_no, message_age)
Int  port_no;
Time message_age;
{  message_age_timer[port_no].value = message_age;
   message_age_timer[port_no].active = True;
}
stop_message_age_timer(port_no)
Int  port_no;
{  message_age_timer[port_no].active = False;
}

Boolean message_age_timer_expired(port_no)
Int port_no;
```

```
{  if (message_age_timer[port_no].active &&
         (++message_age_timer[port_no].value >= bridge_info.max_age))
   {  message_age_timer[port_no].active = False;
      return(True);
   }
      return(False);
}
start_forward_delay_timer(port_no)
Int port_no;
{  forward_delay_timer[port_no].value = Zero;
   forward_delay_timer[port_no].active = True;
}

stop_forward_delay_timer(port_no)
Int port_no;
{  forward_delay_timer[port_no].active = False;
}

Boolean forward_delay_timer_expired(port_no)
Int port_no;
{  if (forward_delay_timer[port_no].active &&
       (++forward_delay_timer[port_no].value >= bridge_info.forward_delay))
   {   forward_delay_timer[port_no].active = False;
       return(True);
   }
   return(False);
}

start_hold_timer(port_no)
Int port_no;
{  hold_timer[port_no].value = Zero;
   hold_timer[port_no].active = True;
}

stop_hold_timer(port_no)
Int port_no;
{  hold_timer[port_no].active = False;
}

Boolean hold_timer_expired(port_no)
Int port_no;
{  if (hold_timer[port_no].active &&
       (++hold_timer[port_no].value >= bridge_info.hold_time))
   {   hold_timer[port_no].active = False;
       return(True);
   }
   return(False);
}


/** pseudo-implementation specific transmit routines **/

#include “transmit.c”
```

8.10 Performance

This subclause places requirements on the performance of the Bridges in a Bridged LAN and on the setting of the parameters of the Spanning Tree Algorithm and Protocol. These are necessary to ensure that the algorithm and protocol operate correctly.

It recommends default operational values for performance parameters. These have been specified in order to avoid the need to set values prior to operation, and have been chosen with a view to maximizing the ease with which Bridged LAN components interoperate.

It specifies absolute maximum values for performance parameters. The ranges of applicable values are specified to assist in the choice of operational values and to provide guidance to implementors.

8.10.1 Requirements

For correct operation, the parameters and configuration of Bridges in the Bridged LAN ensure that

— Bridges do not initiate reconfiguration if none is needed. This means that a Bridge Protocol Message is not timed out before its successor arrives, unless a failure has occurred.
— Following reconfiguration, frames are not forwarded on the new active topology, while frames that were initially forwarded on the previous active topology are still in the Bridged LAN. This ensures that frames are not duplicated.

These requirements are met through placing restrictions on

— The **maximum bridge diameter** of the Bridge LAN: The maximum number of Bridges between any two points of attachment of end stations.
— The **maximum bridge transit delay**: The maximum time elapsing between reception and transmission by a Bridge of a forwarded frame, frames that would otherwise exceed this limit being discarded.
— The **maximum BPDU transmission delay**: The maximum delay prior to the transmission of a Bridge Protocol Data Unit following the need to transmit such a BPDU arising, as specified in 8.7.
— The **maximum Message Age increment overestimate** that may be made to the value of the Message Age parameter in transmitted BPDUs or to the age of stored Bridge Protocol Message information.
— The values of the **Bridge Hello Time**, **Bridge Max Age**, **Bridge Forward Delay**, and **Hold Time** parameters.

Additionally, a Bridge shall not

— Underestimate the increment to the Message Age parameter in transmitted BPDUs.
— Underestimate Forward Delay.
— Overestimate the Hello Time interval when acting as the Root.

8.10.2 Parameter values

Recommended default, absolute maximum, and ranges of parameters are specified in Tables 8-1 through 8-5.

A Bridge shall not exceed the absolute maximum values specified in Table 8-2 for **maximum bridge transit delay**, **maximum BPDU transmission delay**, and **maximum Message Age increment overestimate**.

If the values of **Bridge Hello Time**, **Bridge Max Age**, and **Bridge Forward Delay** can be set by management, the Bridge shall have the capability to use the full range of values in the parameter ranges specified in Table 8-3, with a granularity of 1 s.

Table 8-1—Maximum Bridge Diameter

Parameter	Recommended value
Maximum bridge diameter	7

Table 8-2—Transit and transmission delays

Parameter	Recommended value	Absolute maximum
Maximum bridge transit delay	1.0	4.0
maximum BPDU transmission delay	1.0	4.0
maximum Message Age increment overestimate	1.0	4.0

All times are in seconds.

A Bridge shall use the value of **Hold Time** shown in Table 8-3.

A Bridge shall enforce the following relationships:

$2 \times (Bridge\_Forward\_Delay - 1.0\ seconds) >= Bridge\_Max\_Age$

$Bridge\_Max\_Age >= 2 \times (Bridge\_Hello\_Time + 1.0\ seconds)$

It is recommended that default values of the **Path Cost** parameter for each Bridge Port be based on the values shown in Table 8-5, the values being chosen according to the speed of the LAN segment to which each Port is attached.

NOTE—The values shown in Table 8-5 apply to both full duplex and half duplex operation. The intent of the recommended values and ranges shown is to minimize the number of Bridges in which path costs need to be managed in order to exert control over the topology of the Bridged LAN.

Table 8-3—Spanning Tree Algorithm timer values

Parameter	Recommended or default value	Fixed value	Range
Bridge Hello Time	2.0	—	1.0–10.0
Bridge Max Age	20.0	—	6.0–40.0
Bridge Forward Delay	15.0	—	4.0–30.0
Hold Time	—	1.0	—

All times are in seconds.
— Not applicable.
Subclause 8.10.2 constrains the relationship between Bridge Max Age and Bridge Forward Delay.

If the values of the **Bridge Priority** and the **Port Priority** for each of the Ports can be set by management, the Bridge shall have the capability to use the full range of values in the parameter ranges specified in Table 8-4, with a granularity of 1.

Table 8-4—Bridge and Port Priority parameter values

Parameter	Recommended or default value	Range
Bridge Priority	32 768	0–65 535
Port Priority	128	0–255

A Bridge shall not use a lower value for the Path Cost parameter associated with any Port than the absolute minimum value specified in 8-5.

If the value of **Path Cost** can be set by management, the Bridge shall have the capability to use the full range of values in the parameter ranges specified in Table 8-5, with a granularity of 1.

Table 8-5—Path Cost Parameter Values

Parameter	Link Speed	Recommended value	Recommended range	Range
Path Cost	4 Mb/s	250	100–1000	1–65 535
Path Cost	10 Mb/s	100	50–600	1–65 535
Path Cost	16 Mb/s	62	40–400	1–65 535
Path Cost	100 Mb/s	19	10–60	1–65 535
Path Cost	1 Gb/s	4	3–10	1–65 535
Path Cost	10 Gb/s	2	1–5	1–65 535

9. Encoding of Bridge Protocol Data Units (BPDUs)

This clause specifies the structure and encoding of the BPDUs for the Spanning Tree Protocol, exchanged between Bridge Protocol Entities.

9.1 Structure

9.1.1 Transmission and representation of octets

All BPDUs shall contain an integral number of octets. The octets in a BPDU are numbered starting from 1 and increasing in the order they are put into a Data Link Service Data Unit (DLSDU). The bits in an octet are numbered from 1 to 8, where 1 is the low-order bit.

When consecutive octets are used to represent a binary number, the lower octet number has the most significant value.

All Bridge Protocol Entities respect these bit and octet ordering conventions, thus allowing communications to take place.

9.1.2 Components

A Protocol Identifier is encoded in the initial octets of all BPDUs. This standard reserves a single Protocol Identifier value. This standard places no further restriction on the structure, encoding, or use of BPDUs with different values of the Protocol Identifier field, should these exist, by other standard protocols.

BPDUs used by Bridge Protocol Entities operating the Spanning Tree Algorithm and Protocol specified in Clause 8 use the reserved Protocol Identifier value and have the following structure.

Each BPDU comprises a fixed number of parameters essential to the operation of the protocol. Each parameter is encoded in one or more octets and is of fixed length. The order of parameters in a BPDU is fixed.

The parameters of each BPDU are determined by the BPDU type; all BPDUs of the same type shall comprise the same parameters, in the same order, encoded in the same way.

9.2 Encoding of parameter types

9.2.1 Encoding of Protocol Identifiers

A Protocol Identifier shall be encoded in two octets.

9.2.2 Encoding of Protocol Version Identifiers

A Protocol Version Identifier shall be encoded in one octet. If two Protocol Version Identifiers are interpreted as unsigned binary numbers, then the greater number will be associated with the more recently defined Protocol Version.

9.2.3 Encoding of BPDU types

The type of the BPDU shall be encoded as a single octet. The bit pattern contained in the octet merely serves to distinguish the type; no ordering relationship between BPDUs of different types is implied.

9.2.4 Encoding of flags

A flag shall be encoded as a bit in a single octet. A number of flags may be thus encoded in a single octet. A flag is set if the corresponding bit in the octet takes the value 1. Bit positions in the octet that do not correspond to flags defined for a given type of BPDU are reset, i.e., shall take the value 0. No additional flags will be defined for a BPDU of given protocol version and type.

9.2.5 Encoding of Bridge Identifiers

A Bridge Identifier shall be encoded as eight octets, taken to represent an unsigned binary number. Two Bridge Identifiers may be numerically compared and the lesser number shall denote the Bridge of the higher priority.

The two most significant octets of a Bridge Identifier comprise a settable priority component that permits the relative priority of Bridges to be managed (8.5.3.7 and Clause 14). The six least significant octets ensure the uniqueness of the Bridge Identifier; they shall be derived from the globally unique Bridge Address (7.12.5) according to the following procedure.

The third most significant octet is derived from the initial octet of the MAC Address; the least significant bit of the octet (Bit 1) is assigned the value of the first bit of the Bridge Address, the next most significant bit is assigned the value of the second bit of the Bridge Address, and so on. In a Bridged LAN utilizing 48-bit MAC Addresses, the fourth through eighth octets are similarly assigned the values of the second to the sixth octets of the Bridge Address.

9.2.6 Encoding of Root Path Cost

Root Path Cost shall be encoded as four octets, taken to represent an unsigned binary number, a multiple of arbitrary cost units. Subclause 8.10.2 contains recommendations as to the increment to the Root Path Cost, in order that some common value can be placed on this parameter without requiring a management installation practice for Bridges in a Bridged LAN.

9.2.7 Encoding of Port Identifiers

A Port Identifier shall be encoded as two octets, taken to represent an unsigned binary number. If two Port Identifiers are numerically compared, the lesser number shall denote the Port of higher priority. The more significant octet of a Port Identifier is a settable priority component that permits the relative priority of Ports on the same Bridge to be managed (8.5.5 and Clause 14). The less significant octet is the Port Number expressed as an unsigned binary number. The value 0 is not used as a Port Number.

9.2.8 Encoding of Timer Values

Timer Values shall be encoded in two octets, taken to represent an unsigned binary number multiplied by a unit of time of 1/256 of a second. This permits times in the range 0 to, but not including, 256 s to be represented.

9.3 BPDU formats and parameters

9.3.1 Configuration BPDUs

The format of the Configuration BPDUs is shown in Figure 9-1. Each transmitted Configuration BPDU shall contain the following parameters (8.5.1) and no others:

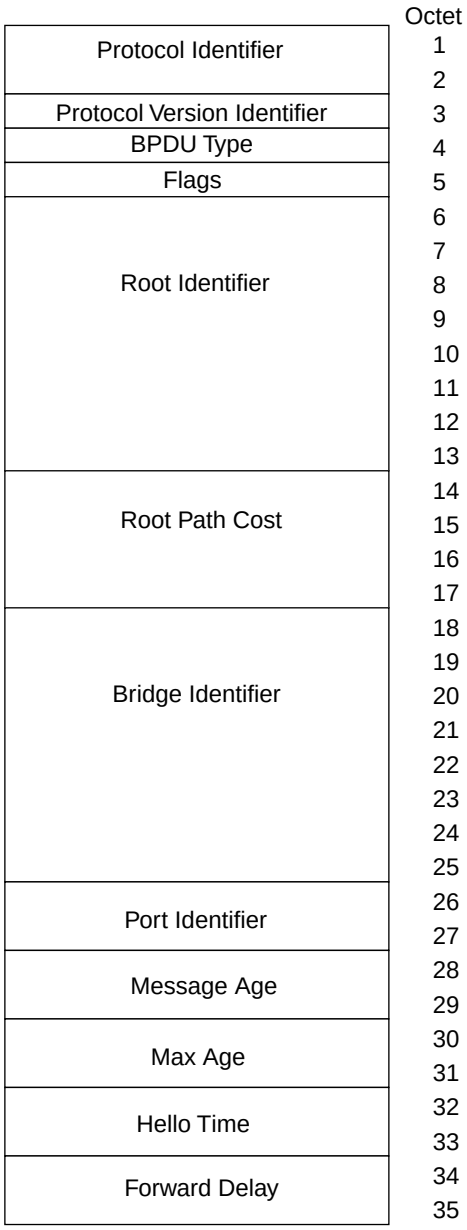


Figure 9-1—Configuration BPDU parameters and format

a) The Protocol Identifier is encoded in Octets 1 and 2 of the BPDU. It takes the value 0000 0000 0000 0000, which identifies the Spanning Tree Algorithm and Protocol as specified in Clause 8.
b) The Protocol Version Identifier is encoded in Octet 3 of the BPDU. It takes the value 0000 0000.
c) The BPDU Type is encoded in Octet 4 of the BPDU. This field shall take the value 0000 0000. This denotes a Configuration BPDU.
d) The Topology Change Acknowledgment flag is encoded in Bit 8 of Octet 5 of the BPDU.
e) The Topology Change flag is encoded in Bit 1 of Octet 5 of the BPDU.
f) The Root Identifier is encoded in Octets 6 through 13 of the BPDU.
g) The Root Path Cost is encoded in Octets 14 through 17 of the BPDU.
h) The Bridge Identifier is encoded in Octets 18 through 25 of the BPDU.

i) The Port Identifier is encoded in Octets 26 and 27 of the BPDU.

j) The Message Age timer value is encoded in Octets 28 and 29 of the BPDU.

k) The Max Age timer value is encoded in Octets 30 and 31 of the BPDU.

l) The Hello Time timer value is encoded in Octets 32 and 33 of the BPDU.

m) The Forward Delay timer value is encoded in Octets 34 and 35 of the BPDU.

The Message Age (Octets 28 and 29) shall be less than Max Age (Octets 30 and 31).

9.3.2 Topology Change Notification BPDUs

The format of the Topology Change Notification BPDUs is shown in Figure 9-2. Each transmitted Topology Change Notification BPDU shall contain the following parameters (8.5.2) and no others:

a) The Protocol Identifier is encoded in Octets 1 and 2 of the BPDU. It takes the value 0000 0000 0000 0000, which identifies the Spanning Tree Algorithm and Protocol as specified in Clause 8 of this standard.

b) The Protocol Version Identifier is encoded in Octet 3 of the BPDU. It takes the value 0000 0000.

c) The BPDU Type is encoded in Octet 4 of the BPDU. This field shall take the value 1000 0000 (where bit 8 is shown at the left of the sequence). This denotes a Topology Change Notification BPDU.

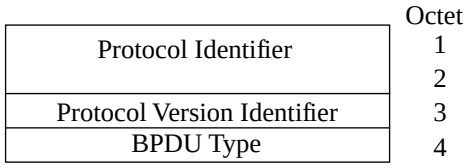


Figure 9-2—Topology change notification BPDU parameters and format

9.3.3 Validation of received BPDUs

A Bridge Protocol Entity shall process a received BPDU as specified in 8.7 if and only if the BPDU contains at least four octets and the Protocol Identifier has the value specified for BPDUs (9.3.2), and

a) The BPDU Type denotes a Configuration BPDU and the BPDU contains at least 35 octets, and the value of the BPDUs Message Age parameter is less than that of its Max Age parameter; or

b) The BPDU Type denotes a Topology Change Notification BPDU.

In case a), any octets that are present beyond Octet 35 are ignored, as far as processing according to this standard is concerned. Similarly, in case b), any octets beyond Octet 4 are ignored.

NOTE—The Protocol Version Identifier is not checked on receipt, in order to allow the possibility of future specification of extensions to the Spanning Tree Protocol, identified as new versions by different values of the Protocol Version Identifier.

10. GARP Multicast Registration Protocol (GMRP)

10.1 Purpose

The GARP (Generic Attribute Registration Protocol) Multicast Registration Protocol (GMRP) provides a mechanism that allows Bridges and end stations to dynamically register (and subsequently, de-register) Group membership information with the MAC Bridges attached to the same LAN segment, and for that information to be disseminated across all Bridges in the Bridged LAN that support Extended Filtering Services. The operation of GMRP relies upon the services provided by GARP, defined in Clause 12.

The information registered, de-registered, and disseminated via GMRP is in the following forms:

a) *Group membership information*. This indicates that one or more GMRP participants that are members of a particular Group (or Groups) exist, and carries the group MAC Address(es) associated with the Group(s). The exchange of specific Group membership information can result in the creation or updating of Group Registration Entries in the Filtering Database to indicate the Port(s) on which members of the Group(s) have been registered. The structure of these entries is described in 7.9.3;
b) *Group service requirement information*. This indicates that one or more GMRP participants require Forward All Groups or Forward Unregistered Groups to be the default Group filtering behavior (see 6.6.7 and 7.9.4).

Registration of Group membership information allows the Bridges in a Bridged LAN to be made aware that frames destined for the group MAC Address concerned should only be forwarded in the direction of the registered members of the Group. Forwarding of frames destined for the address associated with that Group therefore occurs only on Ports on which such membership registration has been received.

Registration of Group service requirement information allows the Bridges in a Bridged LAN to be made aware that any of their Ports that can forward frames in the direction from which the Group service requirement information has been received should modify their default Group forwarding behavior in accordance with the Group service requirement expressed.

NOTE—This ability to dynamically manipulate the default Group forwarding behavior allows Bridge Ports to take account of the needs of GMRP-unaware devices in the Bridged LAN with respect to the forwarding of frames destined for unregistered group MAC Addresses.

The operation of GMRP can result in

c) The propagation of Group membership information and Group service requirement information across a Bridged LAN, and consequent creation, updating, or deletion of Group Registration Entries in the Filtering Databases of all Bridges in the Bridged LAN that support Extended Filtering Services;
d) Consequent changes to the Group filtering behavior of such Bridges.

10.2 Model of operation

GMRP defines a *GARP Application* (12.3, 12.3.1, and 10.3) that provides the extended filtering services defined in 6.6.5 and 6.6.7. To this end, GMRP makes use of

a) The declaration and propagation services offered by *GARP Information Distribution* (GID; 12.3 and 12.3.2) and *GARP Information Propagation* (GIP; 12.3 and 12.3.3) to declare and propagate Group membership information and Group service requirement information within the Bridged LAN;

b) The registration services offered by GID (12.3 and 12.3.2) to allow Group membership information and Group service requirement information declared in the Bridged LAN to control the frame filtering behavior of participating devices with respect to frames destined for group MAC Addresses.

Any new information registered with (or de-registered from) a particular Bridge is propagated by GARP across the Bridged LAN to the other Bridges that may be present, in the manner described in 12.2. This information is used to update Group Registration Entries (7.9.3) in the Filtering Database(s) of participating end stations and Bridges.

10.2.1 Propagation of Group Membership information

The Forwarding Process uses the Group Registration Entries in the Filtering Databases to ensure that frames are transmitted only through those Bridge Ports that are necessary in order to reach LANs to which Group members are attached. Figure 10-1 illustrates the Group Registration Entries created by GMRP for a single Group.

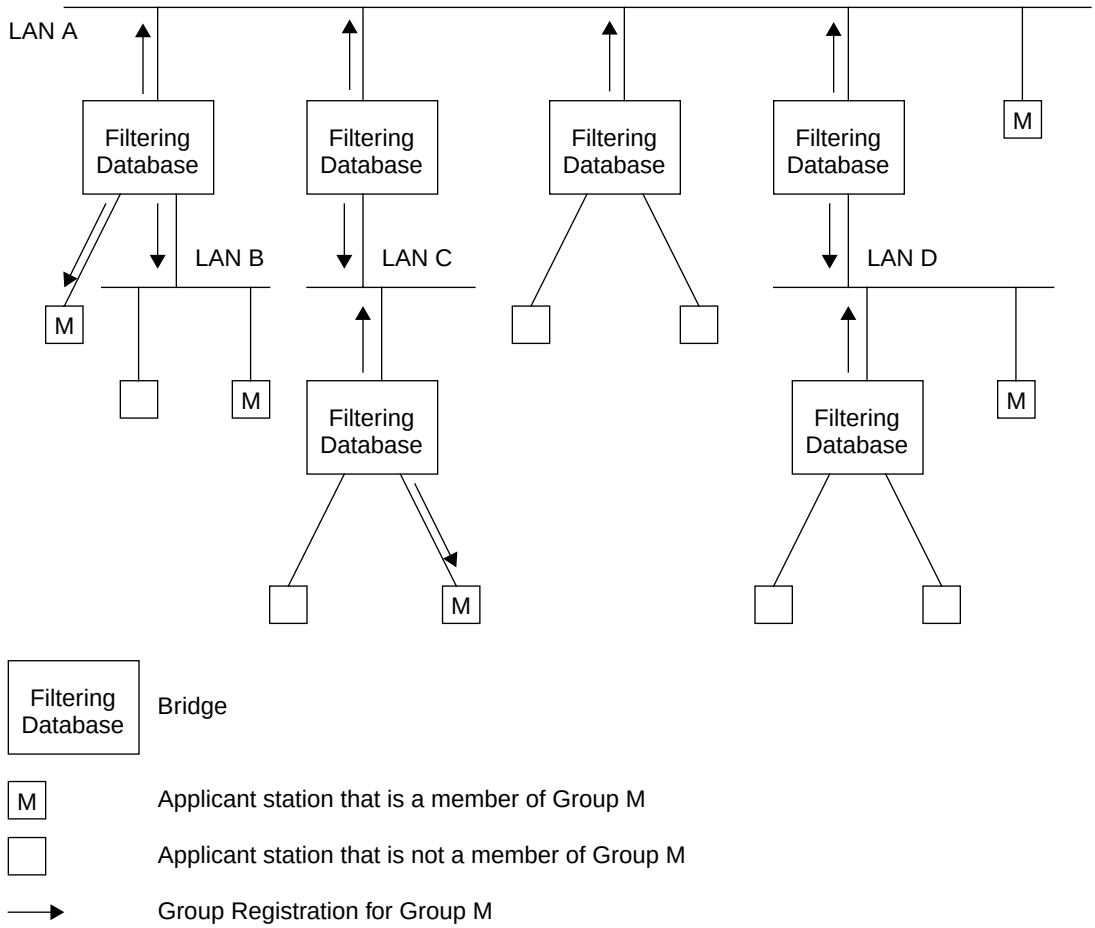


Figure 10-1—Example Directed Graph

By receiving frames from all Ports and forwarding only through Ports for which GMRP has created Group Registration Entries, Bridges facilitate the implementation of Group distribution mechanisms based on the concept of an Open Host Group. Any GMRP Participants (12.3) that wish to receive frames transmitted to a particular Group or Groups register their intention to do so by requesting membership of the Group(s) concerned. Any MAC Service user that wishes to send frames to a particular Group can do so from any point of attachment to the Bridged LAN. These frames can be received on all LAN segments to which registered

GMRP Participants are attached, but the filtering applied by Bridges for ports that have not had Group Registration Entries created by GMRP ensures that frames are not transmitted on LAN segments that neither have registered GMRP participants attached nor are in the path through the active topology between the sources of the frames and the registered Group members. GMRP and the Group Registration Entries thus act to restrict the frames to pruned subsets of the overall loop free active topology defined by the Spanning Tree Protocol, where each pruned subset only includes LAN segments necessary to reach Group members from a given source.

NOTE—The term “Open Host Group” used here comes from the terminology introduced in the definition of the Internet Group Membership Protocol (IGMP) defined by the IETF.

MAC Service users that are sources of MAC frames destined for the Group do not have to register as members of the Group themselves unless they also wish to receive frames transmitted to the Group address by other sources.

10.2.2 Propagation of Group service requirement information

GMRP propagates Group service requirement information in the same manner as for Group Registration information. If any Port in a given Bridge has a registered Group service requirement of All Groups or All Unregistered Groups (expressed in terms of the control information in Static Filtering Entries and/or Dynamic Filtering Entries with a MAC Address specification of All Groups or All Unregistered Groups), this fact is propagated on all other Ports of the Bridge, resulting in the registration of that information on Ports of adjacent Bridges. As a consequence of that registration, the default Group filtering behavior of those Ports may change in order to maintain compatibility with the service requirements expressed by the registered information, as defined in 7.9.4. This ensures that connectivity can be maintained in LANs where the service requirements of different regions of the Bridged LAN differ.

NOTE—In a Bridged LAN where the default Group filtering behavior is not the same for all “edge” Ports, service requirement propagation will tend to result in all “backbone” Ports switching to the highest precedence Group filtering behavior in use in the Bridged LAN. The precedence rules are defined in 7.9.4.

10.2.3 Source pruning

As described in 10.2.1, the operation of GMRP defines a subtree of the Spanning Tree as a result of the creation of Group Registration Entries in the Filtering Databases of the Bridges in the Bridged LAN. End stations in the Bridged LAN are also able to make use of the Group Membership information registered via GMRP to allow them to keep track of the set of Groups for which active members currently exist, and the service requirements of upstream devices. This allows end stations that are sources of frames destined for a Group to suppress the transmission of such frames, if their registered Group membership and Group service requirement information indicates that there are no valid recipients of those frames reachable via the LAN segment to which they are attached.

NOTE—In effect, for the purposes of frame transmission, the end station can be viewed as if it operates as a single Port Bridge, with its own default Group filtering behavior and “Filtering Database” entries updated via GMRP that tell it whether or not multicast frames that it has generated should be forwarded onto the attached LAN. In order to achieve this, it is necessary for the end station to implement both the Registrar and the Applicant functionality of GARP, as described in 12.7.3, 12.8.1, and 12.8.2. The Applicant Only and Simple Applicant Participants described in 12.7.7 and 12.7.8 do not contain the Registrar functionality that would be required for Source Pruning.

This end system behavior is known as *source pruning. S*ource pruning allows MAC Service users that are sources of MAC frames destined for a number of Groups, such as server stations or routers, to avoid unnecessary flooding of traffic on their local LAN segments in circumstances where there are no current Group members in the Bridged LAN that wish to receive such traffic.

10.2.4 Use of Group service requirement registration by end stations

The ability to propagate Group service requirement information is described in this standard primarily as a means of propagating the requirements of the Bridges themselves. However, this mechanism may also be used by end stations that have requirements involving some aspect of promiscuous reception, such as Routers or network monitors. A GMRP-aware end station requiring to be able to receive all multicast traffic can achieve this end by declaring membership of All Groups on the LAN segment to which it is attached; similarly, an end station that wishes to receive unregistered multicast traffic can do so by declaring membership of All Unregistered Groups. The circumstances under which these facilities might be used are further discussed in F.3.

10.3 Definition of the GMRP Application

10.3.1 Definition of GARP protocol elements

10.3.1.1 Use of GIP Contexts by GMRP

GMRP, as defined in this standard, operates within the Base Spanning Tree Context, as defined in 12.3.4. A GIP Context identifier value of 0 shall be used to identify this context. The use of GMRP in any other GIP Context is outside the scope of this standard.

10.3.1.2 GMRP Application address

The group MAC Address used as the destination address for GARP PDUs destined for GMRP Participants shall be the GMRP Address identified in Table 12-1.

10.3.1.3 Encoding of GMRP Attribute Types

The operation of GMRP defines two Attribute Types (12.11.2.2) that are carried in GARP protocol exchanges, as follows:

a) The Group Attribute Type;
b) The Service Requirement Attribute Type.

The Group Attribute Type is used to identify values of group MAC Addresses. The value of the Group Attribute Type carried in GARP PDUs (12.11.2.2) shall be 1.

The Service Requirement Attribute Type is used to identify values of Group service requirements. The value of the Service Requirement Attribute Type carried in GARP PDUs (12.11.2.2) shall be 2.

10.3.1.4 Encoding of GMRP Attribute Values

Values of instances of the Group Attribute Type shall be encoded as Attribute Values in GARP PDUs (12.11.2.6) as six octets, each taken to represent an unsigned binary number. The octets are derived from the Hexadecimal Representation of a 48-bit MAC Address (defined in IEEE Std 802) as follows:

a) Each two-digit hexadecimal numeral in the Hexadecimal Representation is taken to represent an unsigned hexadecimal value, in the normal way, i.e., the rightmost digit of each numeral represents the least significant digit of the value, the leftmost digit is the most significant.
b) The first octet of the attribute value encoding is derived from the left-most hexadecimal value in the Hexadecimal Representation of the MAC Address. The least significant bit of the octet (bit 1) is assigned the least significant bit of the hexadecimal value, the next most significant bit is assigned the value of the second significant bit of the hexadecimal value, and so on.

c) The second through sixth octets of the encoding are similarly assigned the value of the second through sixth hexadecimal values in the Hexadecimal Representation of the MAC Address.

Values of this Attribute Type shall not include individual MAC Addresses.

Values of instances of the Service Requirement Attribute Type shall be encoded as Attribute Values in GARP PDUs (12.11.2.6) as a single octet, taken to represent an unsigned binary number. Only two values of this Type are defined:

a) All Groups shall be encoded as the value 0;
b) All Unregistered Groups shall be encoded as the value 1.

The remaining possible values (2 through 255) are undefined.

10.3.2 Provision and support of Extended Filtering Services

10.3.2.1 End system registration and de-registration

The GMRP Application element of a GARP Participant provides the dynamic registration and de-registration services defined in 6.6.7.1, as follows:

On receipt of a REGISTER_GROUP_MEMBER service primitive, the GMRP Participant issues a GID_Join.request service primitive. The attribute_type parameter of the request carries the value of the Group Attribute Type (10.3.1.3) and the attribute_value parameter carries the value of the MAC_ADDRESS parameter carried in the REGISTER_GROUP_MEMBER primitive.

On receipt of a DEREGISTER_GROUP_MEMBER service primitive, the GMRP Participant issues a GID_Leave.request service primitive. The attribute_type parameter of the request carries the value of the Group Attribute Type (10.3.1.3) and the attribute_value parameter carries the value of the MAC_ADDRESS parameter carried in the DEREGISTER_GROUP_MEMBER primitive.

On receipt of a REGISTER_SERVICE_REQUIREMENT service primitive, the GMRP Participant issues a GID_Join.request service primitive. The attribute_type parameter of the request carries the value of the Service Requirement Attribute Type (10.3.1.3) and the attribute_value parameter carries the value of the REQUIREMENT_SPECIFICATION parameter carried in the REGISTER_SERVICE_REQUIREMENT primitive.

On receipt of a DEREGISTER_SERVICE_REQUIREMENT service primitive, the GMRP Participant issues a GID_Leave.request service primitive. The attribute_type parameter of the request carries the value of the Service Requirement Attribute Type (10.3.1.3) and the attribute_value parameter carries the value of the REQUIREMENT_SPECIFICATION parameter carried in the DEREGISTER_SERVICE_REQUIREMENT primitive.

10.3.2.2 Registration and de-registration events

The GMRP Application element of a GARP Participant responds to registration and de-registration events signalled by GID as follows:

On receipt of a GID_Join.indication whose attribute_type is equal to the value of the Group Attribute Type or the Service Requirement Attribute Type (10.3.1.3), the GMRP Application element updates any Group Registration Entry (7.9.3) in the Filtering Database for the MAC Address specification carried in the attribute_value parameter, to indicate that the MAC Address specification concerned is registered on the Port associated with the GMRP Participant. This is achieved by specifying the Port concerned as Forwarding in

the Port Map field of the entry. If such a Group Registration Entry does not exist in the Filtering Database, a new Group Registration Entry is created.

NOTE—Both Group Membership and Group service requirement information is recorded in the Filtering Database by means of Group Registration Entries (see 7.9.3). In the case of Group Membership Information, the MAC Address specification in the Group Registration Entry is a Group MAC Address. In the case of Group service requirement information, the MAC Address specification is either "All Group Addresses" or "All Unregistered Group Addresses."

On receipt of a GID_Leave.indication whose attribute_type is equal to the value of the Group Attribute Type or the Service Requirement Attribute Type (10.3.1.3), the GMRP Application element updates any Group Registration Entry (7.9.3) in the Filtering Database for the MAC Address specification carried in the attribute_value parameter to indicate that the MAC Address specification concerned is no longer registered on the Port associated with the GMRP Participant. This is achieved by specifying the Port concerned as Filtering in the Port Map field of the entry. If such a Filtering Database entry does not exist in the Filtering Database, then the indication is ignored. If setting that Port to Filtering results in there being no Ports in the Port Map specified as Forwarding (i.e., all Group members are de-registered), then that Group Registration Entry is removed from the Filtering Database.

10.3.2.3 Administrative controls

The provision of static control over the registration state of the state machines associated with the GMRP Application is achieved by means of the Registrar Administrative Control parameters associated with the operation of GARP (12.9.1). These parameters are represented in the Filtering Database by the information held in Static Filtering Entries (7.9.1). If no Static Filtering Entry exists for a given MAC Address specification, the value of the Registrar Administrative Control parameter for the corresponding attribute value is Normal Registration for all Ports of the Bridge.

The initial state of the Permanent Database (i.e., the state of the Permanent Database in a Bridge that has not been otherwise configured by management action) includes a Static Filtering Entry with a MAC Address specification of All Groups, in which the Port Map indicates Registration Fixed. This Static Filtering Entry will have the effect of determining the default Group filtering behavior of all Ports of the Bridge to be Forward All Groups. This entry in the Permanent Database may be deleted or updated by management action.

NOTE—This specification of an initial Static Filtering Entry means that operation using Forward Unregistered Groups or Filter Unregistered Groups requires a conscious action on the part of the network manager or administrator.

Where management capability is implemented, the Registrar Administrative Control parameters can be applied and modified by means of the management functionality defined in 14.7.

The provision of static control over the ability of Applicant state machines to participate in protocol exchanges is achieved by means of the Applicant Administrative Control parameters associated with the operation of GARP (12.9.2). Where management capability is implemented, the Applicant Administrative Control parameters can be applied and modified by means of the management functionality defined in 14.9.

10.3.3 Procedural model

Clauses 11 and 13 contain example "C" code descriptions of GMRP and GARP.

10.4 Conformance to GMRP

This subclause defines the conformance requirements for implementations claiming conformance to GMRP. Two cases are covered: implementation of GMRP in MAC Bridges and implementation of GMRP in end stations. Although this standard is principally concerned with defining the requirements for MAC Bridges, the conformance requirements for end station implementations of GMRP are included in order to give useful

guidance to such implementations. The PICS Proforma defined in Annex A is concerned only with conformance claims with respect to MAC Bridges.

10.4.1 Conformance to GMRP in MAC Bridges

A MAC Bridge for which conformance to GMRP is claimed shall

a) Conform to the operation of the GARP Applicant and Registrar state machines, and the LeaveAll generation mechanism, as defined in 12.8.1, 12.8.2, and 12.8.3;
b) Exchange GARP PDUs as required by those state machines, formatted in accordance with the generic PDU format described in 12.11, and able to carry application-specific information as defined in 10.3.1, using the GMRP Application address as defined in Table 12-1;
c) Propagate registration information in accordance with the operation of GIP for the Base Spanning Tree Context, as defined in 12.3.3 and 12.3.4;
d) Implement the GMRP Application component as defined in 10.3;
e) Forward, filter or discard MAC frames carrying any GARP Application address as the destination MAC Address in accordance with the requirements of 7.12.3.

10.4.2 Conformance to GMRP in end stations

An end station for which conformance to GMRP is claimed shall

a) Conform to the operation of one of
   1) the Applicant state machine, as defined in 12.8.1; or
   2) the Applicant Only state machine, as defined in 12.8.5; or
   3) the Simple Applicant state machine, as defined in 12.8.6;
b) Exchange GARP PDUs as required by the GARP state machine(s) implemented, formatted in accordance with the generic PDU format described in 12.11, and able to carry application-specific information as defined in 10.3.1, using the GMRP Application address as defined in Table 12-1;
c) Support the provision of end system registration and de-registration as defined in 10.3.2.1;
d) Discard MAC frames carrying any GARP Application address as the destination MAC Address in accordance with the requirements of 7.12.3.

An end station for which conformance to the operation of the Applicant state machine (12.8.1) is claimed shall also

e) Conform to the operation of the GARP Registrar state machine and the LeaveAll generation mechanism, as defined in 12.8.2 and 12.8.3; and
f) Support the provision of Group and Group service requirement registration and de-registration as defined in 10.3.2.2; and
g) Filter outgoing frames destined for group MAC Addresses in accordance with registered Group and Group service requirement information, in a manner consistent with the operation of the filtering function of the forwarding process described in 7.7.2.

It is recommended that only those end stations that require the ability to perform Source Pruning (10.2.3) conform to the operation of the Applicant state machine (12.8.1).

For the reasons stated in 12.7.9, it is recommended that end stations that do not require the ability to perform Source Pruning implement the Applicant Only state machine (12.8.5), in preference to the Simple Applicant state machine (12.8.6).

NOTE—End stations that implement only a) 2) and b) through d) are equivalent to the description of the Applicant Only Participant (12.7.7); those that implement only a) 3) and b) through d) are equivalent to the description of the Simple Applicant Participant (12.7.8). Such end stations require only the ability to register membership of one or more Groups,

and revoke that membership at some later point in time; for this reason, there is no requirement to support the operation of the Registrar or Leave All state machines.

End stations that implement a) 1) and b) through g) are able to perform “source pruning” as described in 10.2.3, i.e., to suppress the transmission of frames destined for Groups that currently have no membership. Consequently, such end stations need to support the full Applicant state machine, in combination with the Registrar and Leave All state machines.

11. Example “C” code implementation of GMRP

11.1 Purpose

This section contains an example implementation of the GMRP application (Clause 10). The example implementation makes use of the example implementation shown in Clause 13 of the GARP protocol and state machinery required in order to support GMRP and other applications that use GARP (Clause 12). This “C” code description is included in order to demonstrate the structure of GMRP, and to show that a reasonably low overhead implementation can be constructed. The implementation has been designed with the intent of maximizing clarity and generality, not for compactness.

The example implementation is shown in the following sections:

a) Header files for the GMRP application (11.2);
b) The GMRP application code (11.3).

The separation shown between the application dependent (this clause) and application independent (Clause 13) aspects of the implementation gives a clear illustration of what is involved in implementing additional applications using the same basic GARP state machines. The code is intended to be largely self-documenting, by means of in-line comments.

NOTE—For clarity, the example implementation assumes that only one Static Filtering Entry can be present in the Filtering Database for a given MAC Address (see 7.9.1).

11.2 GMRP application-specific header files

11.2.1 gmr.h

```
/* gmr.h */
#ifndef gmr_h__
#define gmr_h__

#include “garp.h”
#include “gid.h”
#include “gip.h”

/*****************************************************************************
 * GMR : GARP MULTICAST REGISTRATION APPLICATION : GARP ATTRIBUTES
 *****************************************************************************
 */

typedef enum {All_attributes, Legacy_attribute, Multicast_attribute}
             Attribute_type;

typedef enum {Forward_all, Forward_unregistered} Legacy_control;

typedef enum {Number_of_legacy_controls = 1};

/*****************************************************************************
 * GMR : GARP MULTICAST REGISTRATION APPLICATION : CREATION, DESTRUCTION
 *****************************************************************************
 */

extern Boolean gmr_create_gmr(int process_id, unsigned vlan_id,
                              void **gmr);
   /*
    * Creates a new instance of GMR, allocating and initializing a control
    * block, returning True and a pointer to this instance if creation succeeds.
    * Also creates instances of MCD (the MultiCast registration Database) and
    * of GIP (which controls information propagation).
    *
    * Ports are created by the system and added to GMR separately (see
    * gmr_added_port() and gmr_removed_port() below).
    *
    * The operating system supplied process_id is for use in subsequent calls
    * to operating system services. The system itself ensures the temporal
    * scope of process_id, guarding against timers yet to expire for destroyed
    * processes, etc. Although process_id will be implicitly supplied by many
    * if not most systems, it is made explicit in this implementation for
    * clarity.
    *
    * The vlan_id provides the context for this instance of GMR.
    * vlan_id 0 is taken to refer to the base LAN, i.e., the LAN as seen by
    * 802.1D prior to the invention of VLANs. This assumption may be subject
    * to further 802.1 discussion.
    */

extern void gmr_destroy_gmr(void *gmr);
   /*
    * Destroys an instance of GMR, destroying and deallocating the associated
    * instances of MCD and GIP, and any instances of GID remaining.
    */

extern void gmr_added_port(void *my_gmr, int port_no);
   /*
    * The system has created a new port for this application and added it to
    * the ring of GID ports. This function should provide any management
    * initilization required for the port for legacy control or multicast
```

```
    * filtering attributes, such as might be stored in a permanent database
    * either specifically for the port or as part of a template.
    *
    * Newly created ports are 'connected' for the purpose of GARP information
    * propagation using the separate function gip_connect_port(). Prior to
    * doing this, the system should initialize the newly created ports with
    * any permanent database controls for specific multicast values.
    *
    * It is assumed that new ports will be 'connected' correctly before the
    * application continues. The rules for connection are not encoded within
    * GMR. They depend on the relaying connectivity of the system as a whole,
    * and can be summarized as follows:
    *
    *   if (my_gmr->vlan_id == Lan)
    *   {
    *       if stp_forwarding(port_no) gmr_connect_port(port_no);
    *   }
    *   else if vlan_forwarding(vlan_id, port_no)
    *   {
    *       gmr_connect_port(port_no);
    *   }
    *
    * As the system continues to run it should invoke gmr_disconnect_port()
    * and gmr_connect_port() as required to maintain the required connectivity.
    */

extern void gmr_removed_port(void *my_gmr, int port_no);
   /*
    * The system has removed and destroyed the GID port. This function should
    * provide any application-specific cleanup required.
    */

/*****************************************************************************
 * GMR : GARP MULTICAST REGISTRATION APPLICATION : JOIN, LEAVE INDICATIONS
 *****************************************************************************
 */

extern void gmr_join_indication(void *my_gmr, void *my_port,
                                unsigned joining_gid_index);
   /*
    * Deals with joins for both Legacy Attributes and Multicast Attributes.
    * The former are represented by the first few GID indexes, and give rise
    * to three cases:
    *
    * 1. Neither Forward All or Forward Unregistered are currently set (i.e.,
    *    registered for this port), so the Filtering Database is in
    *    "filter_by_default" mode, and the only multicasts that are being
    *    forwarded through (out of) this port are those "registered here."
    *    In addition there may be other entries in the Filtering Database
    *    whose effect on this port is currently duplicated by
    *    "filter_by_default": if an entry for a multicast is present for any
    *    port, the model of the Filtering Database requires that its filter or
    *    forward behavior be represented explicitly for all other ports
    *    - there is no per port setting which means "behave as default" mode.
    *
    * 2. Forward Unregistered is currently set, but Forward All is not.
    *    The Filtering Database is in "forward_by_default" mode. If a Filtering
    *    Database entry has been made for a multicast for any port, it specifies
    *    filtering for this port if the multicast is not registered here but
    *    is registered for a port to which this port is connected (in the GIP
    *    sense), and forwarding otherwise (i.e., multicast registered here or
    *    not registered for any port to which this port is connected).
    *
    * 3. Forward All is currently set and takes precedence - Forward
    *    Unregistered may or may not be set. The Filtering Database is in
```



```
    *    “forward_by_default” mode. If a Filtering Database entry has been
    *    made for any port, it specifies forwarding for this port. Not all
    *    multicasts registered for this port have been entered into the
    *    Filtering Database.
    *
    * If a call to “fdb_filter” or “fdb_forward,” made for a given port and
    * multicast address, causes a Filtering Database entry to be created, the
    * other ports are set to filter or forward for that address according to
    * the setting of “forward_by_default” or “filter_by_default” current when
    * the call is made. This behavior can be used to avoid temporary filtering
    * glitches.
    *
    * This function, gmr_join_indication(), changes filtering database entries
    * for the port that gives rise to the indication alone. If another port
    * is in Legacy mode B (Forward_unregistered set (registered) for that port,
    * but not Forward_all), then registration of a multicast address on this
    * port can cause it to be filtered on that other port. This is handled by
    * gmr_join_propagated() for the other ports that may be affected. It will
    * be called as a consequence of the GIP propagation of the newly registered
    * attribute (multicast address).
    */

extern void gmr_join_propagated(void *my_gmr, void *my_port,
                                unsigned joining_gid_index);
   /*
    *
    */

extern void gmr_leave_indication(void *my_gmr, void *my_port,
                                 unsigned leaving_gid_index);
   /*
    *
    */

extern void gmr_leave_propagated(void *my_gmr, void *my_port,
                                 unsigned leaving_gid_index);
   /*
    *
    */

/*****************************************************************************
 * GMR : GARP MULTICAST REGISTRATION APPLICATION : PROTOCOL & MGT EVENTS
 *****************************************************************************
 */

extern void gmr_rcv(void *my_gmr, void *my_port, void *pdu);
   /*
    * Process an entire received pdu for this instance of GMR.
    */

extern void gmr_tx(void *my_gmr, void *my_port);
   /*
    * Transmit a pdu for this instance of GMR.
    */

#endif /* gmr_h__ */
```

11.2.2 gmd.h

```
/* gmd.h */
#ifndef gmd_h__
#define gmd_h__
```

```
#include “sys.h”

/*****************************************************************************
 * GMD : GARP MULTICAST REGISTRATION APPLICATION DATABASE
 *****************************************************************************
 */

extern Boolean gmd_create_gmd(unsigned max_multicasts, void **gmd);
   /*
    * Creates a new instance of GMD, allocating space for up to max_multicasts
    * MAC addresses.
    *
    * Returns True if the creation succeeded together with a pointer to the
    * GMD information.
    */

extern void gmd_destroy_gmd(void *gmd);
   /*
    * Destroys the instance of gmd, releasing previously allocated database and
    * control space.
    */


extern Boolean gmd_find_entry(  void *my_gmd, Mac_address key,
                              unsigned *found_at_index);

extern Boolean gmd_create_entry(void *my_gmd, Mac_address key,
                              unsigned *created_at_index);

extern Boolean gmd_delete_entry(void *my_gmd,
                              unsigned  delete_at_index);

extern Boolean gmd_get_key(     void *my_gmd, unsigned index, Mac_address *key);

#endif /* gmd_h__ */
```

11.2.3 gmf.h

```
/* gmf.h */
#ifndef gmf_h__
#define gmf_h__

#include “sys.h”
#include “prw.h”
#include “GMR.h”

/*****************************************************************************
 * GMF : GARP MULTICAST REGISTRATION APPLICATION PDU FORMATTING
 *****************************************************************************
 */

typedef struct
{  /*
    * This data structure saves the temporary state required to parse GMR
    * PDUs in particular. Gpdu provides a common basis for GARP application
    * formatters; additional state can be added here as required by GMF.
    */

    Gpdu gpdu;

} Gmf;
```



```
typedef struct /* Gmf_msg_data */
{
    Attribute_type attribute;

    Gid_event   event;

    Mac_address    key1;

    Mac_address    key2;

    Legacy_control legacy_control;

} Gmf_msg;

extern void    gmf_rdmsg_init(Pdu *pdu, Gmf **gmf);

extern void    gmf_wrmsg_init(Gmf *gmf, Pdu  *pdu, int vlan_id);

extern Boolean gmf_rdmsg(     Gmf *gmf, Gmf_msg *msg);

extern Boolean gmf_wrmsg(     Gmf *gmf, Gmf_msg *msg);

#endif /* gmf_h__ */
```

11.2.4 fdb.h

```
/* fdb.h */
#ifndef fdb_h__
#define fdb_h__

#include “sys.h”

/*****************************************************************************
 * FDB : FILTERING DATABASE INTERFACE
 *****************************************************************************
 */

extern void fdb_filter(unsigned vlan_id, int port_no, Mac_address address);

extern void fdb_forward(unsigned vlan_id, int port_no, Mac_address address);

extern void fdb_filter_by_default(unsigned vlan_id, int port_no);

extern void fdb_forward_by_default(unsigned vlan_id, int port_no);

#endif /* fdb_h__ */
```

11.3 GMRP application code

11.3.1 gmr.c

```
/* gmr.c */

#include “gmr.h”
#include “gid.h”
#include “gip.h”
#include “garp.h”
#include “gmd.h”
#include “gmf.h”
#include “fdb.h”


/*****************************************************************************
 * GMR : GARP MULTICAST REGISTRATION APPLICATION : IMPLEMENTATION SIZING      *
 *****************************************************************************
 */

enum {Max_multicasts = 100};

enum {Number_of_gid_machines = Number_of_legacy_controls + Max_multicasts};

enum {Unused_index = Number_of_gid_machines};

/*****************************************************************************
 * GMR : GARP MULTICAST REGISTRATION APPLICATION : CREATION, DESTRUCTION      *
 *****************************************************************************
 */

typedef struct /* Gmr */
{
    Garp      g;

    unsigned  vlan_id;

    void     *gmd;

    unsigned  number_of_gmd_entries;

    unsigned  last_gmd_used_plus1;

} Gmr;


Boolean gmr_create_gmr(int process_id, unsigned vlan_id, void **gmr)
{  /*
    */
    Gmr *my_gmr;

    if (!sysmalloc(sizeof(Gmr), &my_gmr))
        goto gmr_creation_failure;

    my_gmr->g.process_id = process_id;
    my_gmr->g.gid        = NULL;

    if (!gip_create_gip(Number_of_gid_machines, &my_gmr->g.gip))
        goto gip_creation_failure;

    my_gmr->g.max_gid_index = Number_of_gid_machines - 1;
    my_gmr->g.last_gid_used = Number_of_legacy_controls - 1;

    my_gmr->g.join_indication_fn  = gmr_join_indication;
```



```
    my_gmr->g.leave_indication_fn = gmr_leave_indication;
    my_gmr->g.join_propagated_fn  = gmr_join_propagated;
    my_gmr->g.leave_propagated_fn = gmr_leave_propagated;
    my_gmr->g.transmit_fn         = gmr_tx;
    my_gmr->g.added_port_fn       = gmr_added_port;
    my_gmr->g.removed_port_fn     = gmr_removed_port;

    my_gmr->vlan_id = vlan_id;

    if (!gmd_create_gmd(Max_multicasts, &my_gmr->gmd))
        goto gmd_creation_failure;

    my_gmr->number_of_gmd_entries = Max_multicasts;
    my_gmr->last_gmd_used_plus1   = 0;

    *gmr = my_gmr;    return(True);
gmd_creation_failure: gip_destroy_gip(my_gmr->g.gip);
gip_creation_failure: sysfree(my_gmr);
gmr_creation_failure: return(False);
}


void gmr_destroy_gmr(Gmr *my_gmr)
{
    Gid *my_port;

    gmd_destroy_gmd(my_gmr->gmd);
    gip_destroy_gip(my_gmr->g.gip);

    while ((my_port = my_gmr->g.gid) != NULL)
        gid_destroy_port(&my_gmr->g, my_port->port_no);
}


void gmr_added_port(Gmr *my_gmr, int port_no)
{  /*
    * Provide any management initialization of legacy control or multicast
    * attributes from templates here for the new port.
    */
}


void gmr_removed_port(Gmr *my_gmr, int port_no)
{  /*
    * Provide any GMR specific cleanup or management alert functions for the
    * removed port.
    */
}


/*****************************************************************************
 * GMR : GARP MULTICAST REGISTRATION APPLICATION : JOIN, LEAVE INDICATIONS
 *****************************************************************************
 */

void gmr_join_indication(Gmr *my_gmr, Gid *my_port, unsigned joining_gid_index)
{  /*
    * This implementation of gmr_join_indication() respects the three cases
    * described in the header file for the state of the Filtering Database and
    * the registered Legacy controls (Forward All, Forward Unregistered) and
    * Multicasts. It makes some, but not a perfect, attempt to optimize
    * calls to the Filtering Database when one Legacy mode transitions to
    * another.
    *
    */
```

```
    unsigned     gmd_index;
    unsigned     gid_index;
    Mac_address  key;

    if (!gid_registered_here(my_port, Forward_all))
    {
        if (   (joining_gid_index == Forward_all)
            || (joining_gid_index == Forward_unregistered))
        {
            gmd_index = 0; gid_index = gmd_index + Number_of_legacy_controls;
            while (gmd_index < my_gmr->last_gmd_used_plus1)
            {
                if (!gid_registered_here(my_port, gid_index))
                {
                    if (joining_gid_index == Forward_all)
                    {
                        gmd_get_key(my_gmr->gmd, gmd_index, &key);
                        fdb_forward(my_gmr->vlan_id, my_port->port_no, key);
                    }
                    else if (!gip_propagates_to(my_port, gid_index))
                    {     /*(joining_gid_index == Forward_unregistered) */
                        gmd_get_key(my_gmr->gmd, gmd_index, &key);
                        fdb_forward(my_gmr->vlan_id, my_port->port_no, key);
                    }
                }
                gmd_index ++; gid_index ++;
            }

            fdb_forward_by_default(my_gmr->vlan_id, my_port->port_no);
        }
        else /* Multicast Attribute */
        {
            gmd_index = joining_gid_index - Number_of_legacy_controls;
            gmd_get_key(my_gmr->gmd, gmd_index, &key);

            fdb_forward(my_gmr->vlan_id, my_port->port_no, key);
}   }   }


void gmr_join_propagated(Gmr *my_gmr, Gid *my_port, unsigned joining_gid_index)
{  /*
    *
    */
    unsigned     gmd_index;
    Mac_address  key;

    if (joining_gid_index >= Number_of_legacy_controls)
    {/* Multicast attribute */

        if (   (!gid_registered_here(my_port, Forward_all))
            &&  (gid_registered_here(my_port, Forward_unregistered))
            && (!gid_registered_here(my_port, joining_gid_index)))
        {
            gmd_index = joining_gid_index - Number_of_legacy_controls;
            gmd_get_key(my_gmr->gmd, gmd_index, &key);

            fdb_filter(my_gmr->vlan_id, my_port->port_no, key);
}   }   }


void gmr_leave_indication(Gmr *my_gmr, Gid *my_port, unsigned leaving_gid_index)
{  /*
    *
    */
    unsigned     gmd_index;
```



```
    unsigned     gid_index;
    Boolean  mode_a;
    Boolean      mode_c;
    Mac_address  key;

    mode_a =  gid_registered_here(my_port, Forward_all);
    mode_c = !gid_registered_here(my_port, Forward_unregistered);

    if (       (leaving_gid_index == Forward_all)
        || (   (!mode_a)
            && (leaving_gid_index == Forward_unregistered)))
    {
        gmd_index = 0; gid_index = gmd_index + Number_of_legacy_controls;
        while (gmd_index < my_gmr->last_gmd_used_plus1)
        {
            if (!gid_registered_here(my_port, gid_index))
            {
                if (mode_c || gip_propagates_to(my_port, gid_index))
                {
                    gmd_get_key(my_gmr->gmd, gmd_index, &key);
                    fdb_filter(my_gmr->vlan_id, my_port->port_no, key);
            }   }

            gmd_index ++; gid_index ++;
        }

        if (mode_c)
            fdb_filter_by_default(my_gmr->vlan_id, my_port->port_no);
    }
    else if (!mode_a)
    {
        if (mode_c || gip_propagates_to(my_port, gid_index))
        { /* Multicast Attribute */
            gmd_index = leaving_gid_index - Number_of_legacy_controls;

            gmd_get_key(my_gmr->gmd, gmd_index, &key);
            fdb_filter(my_gmr->vlan_id, my_port->port_no, key);
}   }   }


void gmr_leave_propagated(Gmr *my_gmr, Gid *my_port, unsigned leaving_gid_index)
{  /*
    *
    */
    unsigned     gmd_index;
    Mac_address  key;

    if (leaving_gid_index >= Number_of_legacy_controls)
    {/* Multicast attribute */

        if (   (!gid_registered_here(my_port, Forward_all))
            &&  (gid_registered_here(my_port, Forward_unregistered))
            && (!gid_registered_here(my_port, leaving_gid_index)))
        {
            gmd_index = leaving_gid_index - Number_of_legacy_controls;
            gmd_get_key(my_gmr->gmd, gmd_index, &key);

            fdb_forward(my_gmr->vlan_id, my_port->port_no, key);
}   }   }



/*****************************************************************************
 * GMR : GARP MULTICAST REGISTRATION APPLICATION : RECEIVE MESSAGE PROCESSING
```

```
 ****************************************************************************
 */

static void gmr_db_full(Gmr *my_gmr, Gid *my_port)
{  /*
    * If it is desirable to be able to operate correctly with an undersized
    * database, add code here. The best approach seems to be to use GID
    * management controls to configure the attribute for the Legacy mode
    * control Forward_all to be Registration fixed on all ports on which join
    * messages have been discarded because their keys are not in the database.
    * Then start a retry timer, which attempts to scavenge space from the
    * database at a later time, and, if it succeeds, waits for a few LeaveAll
    * times before switching Forward_all back to Normal_registration.
    */
}

static void gmr_rcv_msg(Gmr *my_gmr, Gid *my_port, Gmf_msg *msg)
{  /*
    * Process one received message.
    *
    * Dispatch messages by message event, and by attribute type (legacy mode
    * control, or multicast address) except in the case of the LeaveAll
    * message event, which applies equally to all attributes.
    *
    * A LeaveAll message never causes an indication (join or leave directly),
    * even for the point to point link protocol enhancements (where an
    * ordinary Leave does). No further work is needed here.
    *
    * A LeaveAllRange message is currently treated exactly as a LeaveAll
    * (i.e., the range is ignored).
    *
    * All the remaining messages refer to a single attribute (i.e., a single
    * registered group address). Try to find a matching entry in the MCD
    * database. If one is found, dispatch the message to a routine that will
    * handle both the local GID effects and the GIP propagation to other ports.
    *
    * If no entry is found, Leave and Empty messages can be discarded, but
    * JoinIn and JoinEmpty messages demand further treatment. First, an attempt
    * is made to create a new entry using free space (in the database, which
    * corresponds to a free GID machine set). If this fails, an attempt may be
    * made to recover space from a machine set that is in an unused or less
    * significant state. Finally, the database is considered full and the received
    * message is discarded.
    *
    * Once (if) an entry is found, Leave, Empty, JoinIn, and JoinEmpty are
    * all submitted to GID (gid_rcv_msg()).
    *
    * JoinIn and JoinEmpty may cause Join indications, which are then propagated
    * by GIP.
    *
    * On a shared medium, Leave and Empty will not give rise to indications
    * immediately. However, this routine does test for and propagate
    * Leave indications so that it can be used unchanged with a point-to-point
    * protocol enhancement.
    *
    */

    unsigned gmd_index = Unused_index;
    unsigned gid_index = Unused_index;

    if (   (msg->event == Gid_rcv_leaveall)
        || (msg->event == Gid_rcv_leaveall_range))
    {
        gid_rcv_leaveall(my_port);
    }
```



```
    else
    {
        if (msg->attribute == Legacy_attribute)
        {
            gid_index = msg->legacy_control;
        }
        else if (!gmd_find_entry(my_gmr->gmd, msg->key1, &gmd_index))
        {/*  && (msg->attribute == Multicast_attribute) */

            if (   (msg->event == Gid_rcv_joinin)
                || (msg->event == Gid_rcv_joinempty))
            {
                if (!gmd_create_entry(my_gmr->gmd, msg->key1, &gmd_index))
                {
                    if (gid_find_unused(&my_gmr->g,
                                        Number_of_legacy_controls, &gid_index))
                    {
                        gmd_index = gid_index - Number_of_legacy_controls;
                        gmd_delete_entry(my_gmr->gmd, gmd_index);
                        (void) gmd_create_entry(my_gmr->gmd, msg->key1,
                                                &gmd_index);
                    }
                    else
                        gmr_db_full(my_gmr, my_port);
        }   }   }

        if (gmd_index != Unused_index)
            gid_index = gmd_index + Number_of_legacy_controls;

        if (gid_index != Unused_index)
            gid_rcv_msg(my_port, gid_index, msg->event);
}   }


void gmr_rcv(Gmr *my_gmr, Gid *my_port, Pdu *pdu)
{  /*
    * Process an entire received pdu for this instance of GMR: initialize
    * the Gmf pdu parsing routine, and, while messages last, read and process
    * them one at a time.
    */
    Gmf      gmf;
    Gmf_msg  msg;

    gmf_rdmsg_init(&gmf, pdu);

    while (gmf_rdmsg(&gmf, &msg))
        gmr_rcv_msg(my_gmr, my_port, &msg);
}


/*****************************************************************************
 * GMR : GARP MULTICAST REGISTRATION APPLICATION : TRANSMIT PROCESSING
 *****************************************************************************
 */

static void gmr_tx_msg(Gmr *my_gmr, unsigned gid_index, Gmf_msg *msg)
{
    unsigned gmd_index;

    if (msg->event == Gid_tx_leaveall)
    {
        msg->attribute      = All_attributes;
    }
    else if (gid_index == Forward_all)
    {
```

```
        msg->attribute      = Legacy_attribute;
        msg->legacy_control = Forward_all;
    }
    else /* index for Multicast_attribute */
    {
        msg->attribute = Multicast_attribute;

        gmd_index = gid_index - Number_of_legacy_controls;
        gmd_get_key(my_gmr->gmd, gmd_index, &msg->key1);
}   }


void gmr_tx(Gmr *my_gmr, Gid *my_port)
{  /*
    * Get and prepare a pdu for the transmission, if one is not available,
    * simply return; if there is more to transmit, GID will reschedule a call
    * to this function.
    *
    * Get messages to transmit from GID and pack them into the pdu using Gmf
    * (MultiCast pdu Formatter).
    */
    Pdu         *pdu;
    Gmf          gmf;
    Gmf_msg      msg;
    Gid_event    tx_event;
    unsigned     gid_index;


    if ((tx_event = gid_next_tx(my_port, &gid_index)) != Gid_null)
    {
        if (syspdu_alloc(&pdu))
        {
            gmf_wrmsg_init(&gmf, pdu, my_gmr->vlan_id);

            do
            {
                msg.event = tx_event;

                gmr_tx_msg(my_gmr, gid_index, &msg);

                if (!gmf_wrmsg(&gmf, &msg))
                {
                    gid_untx(my_port);
                    break;
                }
            } while ((tx_event  = gid_next_tx(my_port, &gid_index))
                          != Gid_null);

        syspdu_tx(pdu, my_port->port_no);
}   }   }
```

12. Generic Attribute Registration Protocol (GARP)

12.1 Purpose

The GARP provides a generic attribute dissemination capability that is used by participants in GARP Applications (GARP Participants) to register and de-register attribute values with other GARP Participants within a Bridged LAN. The definition of the attribute types, the values that they can carry, and the semantics that are associated with those values when registered, are specific to the operation of the GARP Application concerned.

NOTE—Clause 10 defines one such application, GMRP, which makes use of GARP to allow the registration of two attribute types: group MAC Addresses and Group service requirements. Values of these attribute types are used to dynamically control the filtering behavior of GMRP participants within the Bridged LAN.

12.2 Overview of GARP operation

The operation of GARP allows a participant in a given GARP Application to make *declarations*, or withdraw declarations, relative to *attribute* values, and for those declarations (or withdrawals) to result in the *registration* (or *de-registration*) of those parameter values with the other GARP Participants for that Application within the Bridged LAN.

The fact that a GARP Participant has declared (or withdrawn a declaration for) a given attribute value is recorded by means of the state variables associated with an Applicant state machine for that attribute value, for the Port from which the declaration was made.

Registration occurs only with respect to those Ports that receive the GARP PDU containing a declaration or withdrawal. The current registration state of an attribute value on a Port is recorded by means of the state variables associated with a Registrar state machine for that attribute value.

De-registration of a given attribute value on a given Port occurs only if all GARP Participants connected to the same LAN segment as the Port (other than the Participant associated with the Port itself) withdraw the declaration.

Parameter values that are registered on Bridge Ports that are part of the *active topology* (7.4) for the *GIP Context* (12.3.4) are propagated as declarations through all other Bridge Ports that are part of the active topology. Hence, a given declaration will be propagated to all devices in a Bridged LAN in which a GARP Participant exists for the application concerned, and in each Bridge, the attribute value will be registered on those Ports that are "nearest" to the source of the declaration in the active topology.

NOTE 1—Registration can occur on any Port, regardless of the Spanning Tree state of the Port; however, propagation of registered information follows the active topology.
NOTE 2—Unless otherwise stated, the description of GARP that follows assumes operation within GIP Context 0, the Spanning Tree established and maintained by the operation of the protocol and procedures defined in Clause 8.

Figure 12-1 illustrates the registration and propagation of an Attribute value across the components of a Bridged LAN, resulting from a single end station declaring that Attribute value. The diagram shows which Bridge Ports also declare the Attribute value in order to propagate the value. All Ports of the Bridges shown are assumed to be in the Forwarding state. As can be seen, the propagation arcs result in the Attribute value being propagated to all LAN segments; however, the directional nature of the propagation results in registration of the Attribute value only on Bridge Ports that receive (as opposed to transmit) the propagated information.

Figure 12-2 illustrates the registration and propagation of an attribute value across the components of a Bridged LAN, resulting from two end stations declaring the same attribute value on different LAN seg-

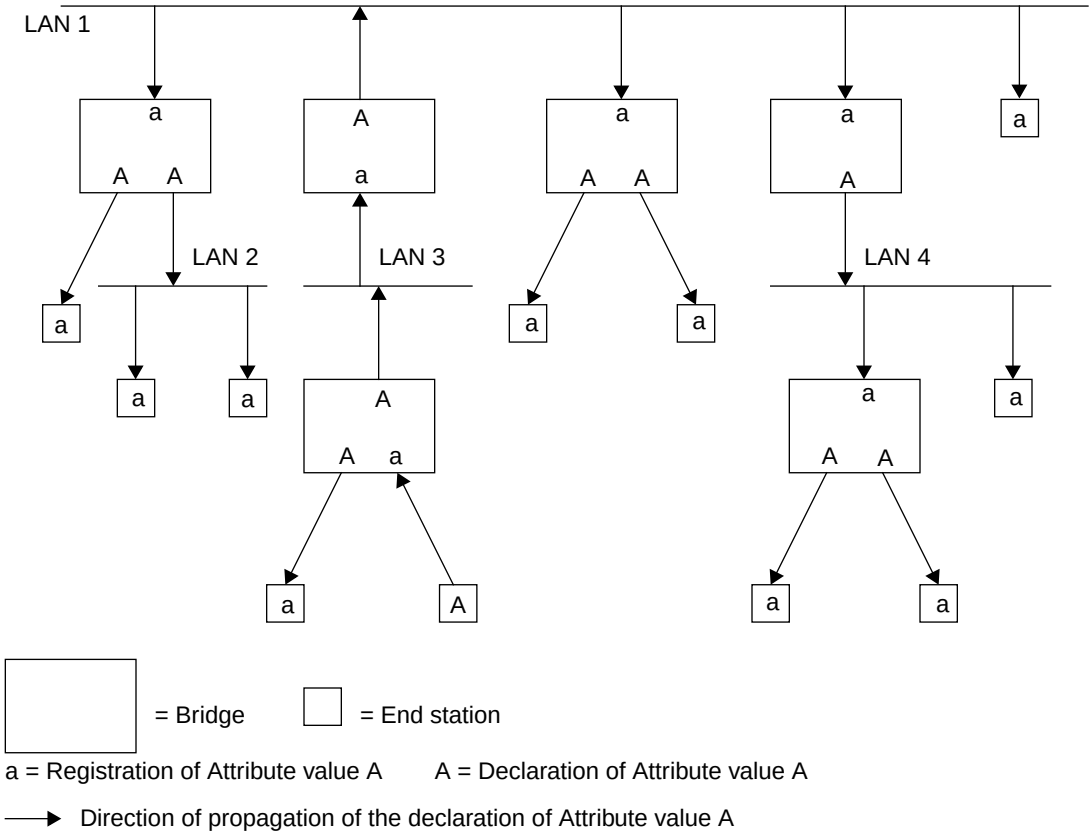


Figure 12-1—Example Attribute value propagation from one station

ments. It can be seen from this diagram that the addition of a second source of declaration results in all end stations registering the attribute value, and some Bridges registering the value on more than one Port.

The registration of these Attribute values on Bridge Ports is asymmetric in nature. From within any Bridge, the set of Ports that

a) Have registered a given attribute value, and
b) Are part of the active topology,

completely defines the subset of the active Spanning Tree topology that contains all GARP Participants that have declared that Attribute value. Similarly, for any end stations, the presence of a registered Attribute value indicates that one or more GARP Participants that are part of the Spanning Tree topology to which the end station is attached have declared that value.

Hence, a registered attribute value can be regarded as a pointer from a GARP Participant to a subtree of the active topology that contains one or more GARP Participants that have declared that Attribute value. The set of such registrations in a given Participant can therefore represent a subtree of the active topology that contains all GARP Participants that have declared that Attribute value.

The set of registrations of a given Attribute value within the Bridged LAN can therefore be considered to form a set of subtrees, each indicating, from a given GARP Participant, the subset of the active topology in which all GARP Participants that have declared that Attribute value can be found. In Figure 12-3, Ports that form this set of subtrees are shown as the origin of the arrows, based on the registrations that are shown in Figure 12-2.

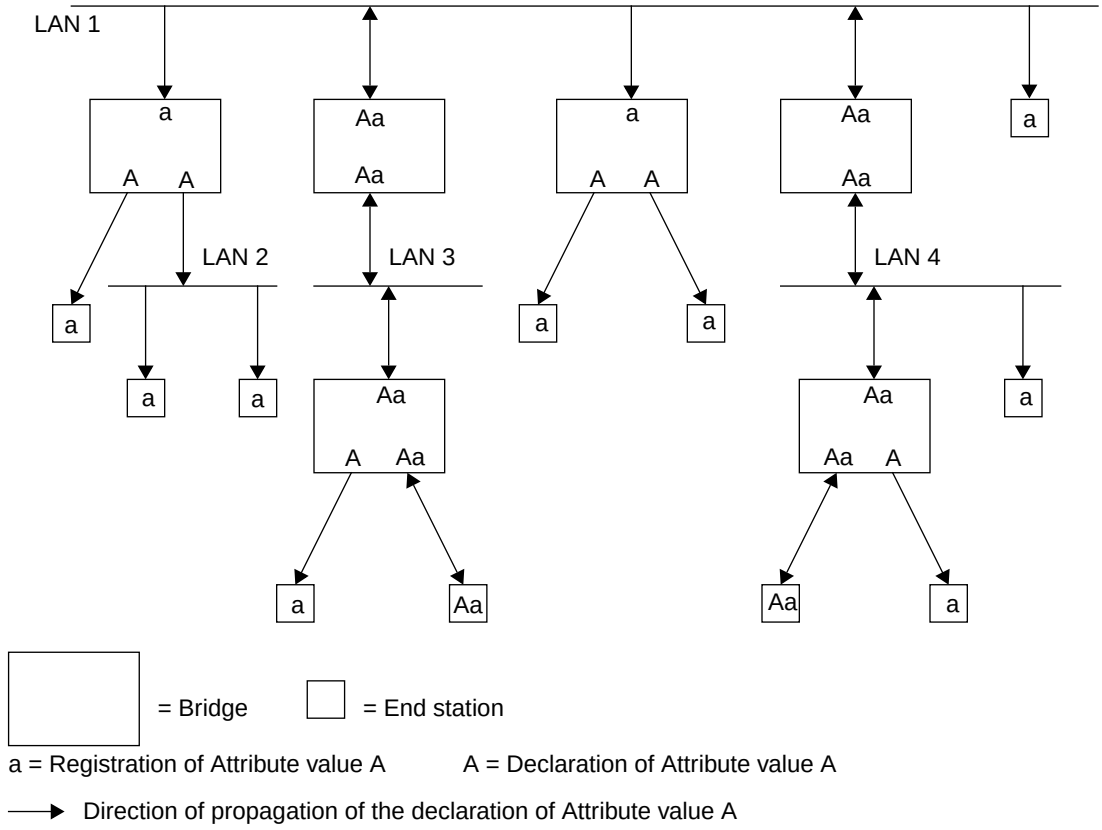


Figure 12-2—Example Attribute value propagation from two stations

This property of GARP gives an indication of the kinds of application to which GARP is best put. Applications where it is desirable to form “reachability” trees that constitute the subset of the active topology that encompasses all registered Participants are generally good candidates for the use of GARP. For example, if the attribute value A in Figure 12-3 is a MAC Address that carries the semantics “I wish to receive details of the final score in the Superbowl,” and it is deemed desirable for those results to be sent only to the subset of the active topology that contains end stations that have declared Attribute A, then an end station that has these results available could use the presence or absence of a registration for A on its Port as an indication of whether or not to send the results on the LAN segment to which it is attached, and any Bridge receiving the results could determine on which Ports the results should be forwarded.

In MAC Bridges, GARP operates only when the Bridge Filtering Mode is set to Extended Filtering Mode. Bridges that are unable to operate in Extended Filtering Mode, or have been set to operate in Basic Filtering Mode, are transparent with respect to GARP protocol exchanges, and forward GARP PDUs on all Ports that are in Forwarding. Similarly, Bridges that do not implement a given GARP Application are transparent to GARP protocol exchanges destined for that Application.

12.3 GARP architecture

Figure 12-4 illustrates the components of GARP Participants in a two-Port Bridge and an end station.

A *GARP Participant* in a Bridge or an end station consists of a *GARP Application* component, and a *GARP Information Declaration (GID)* component associated with each Port of the Bridge. One such GARP Participant exists per Port, per GARP Application. The propagation of information between GARP Participants for

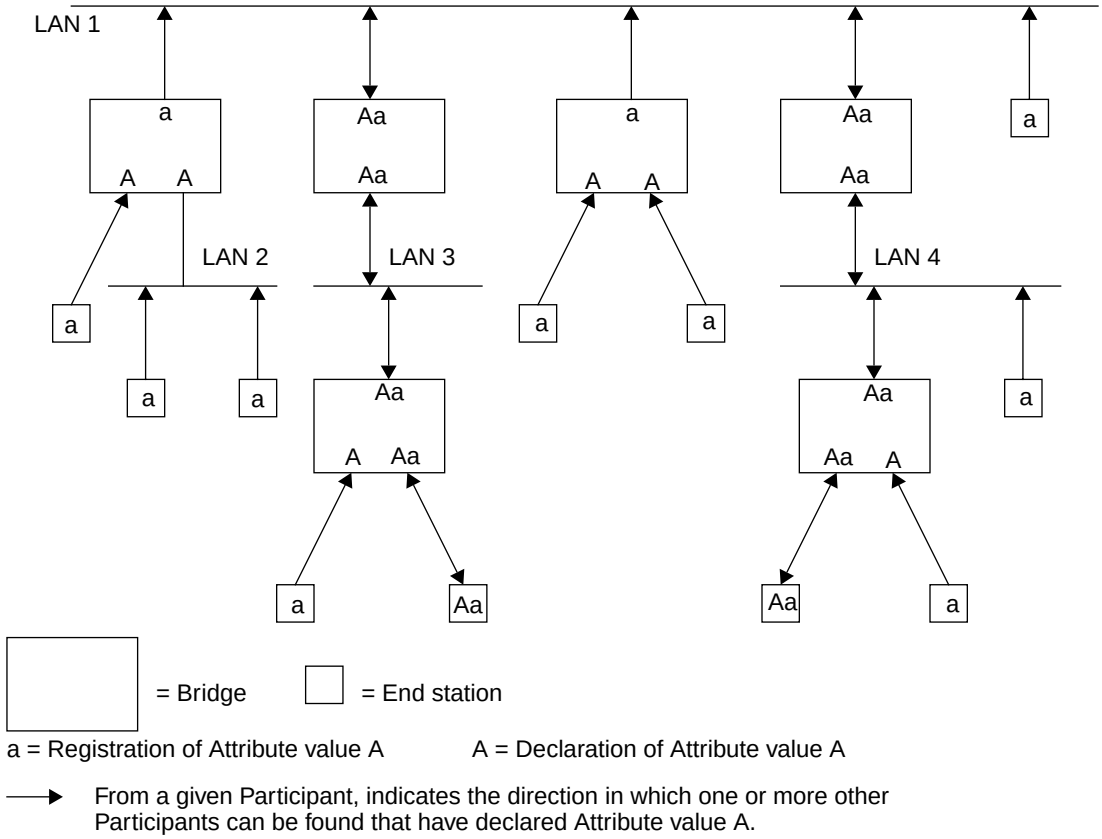


Figure 12-3—Example of the formation of subtrees of the active topology

the same Application in a Bridge is carried out by the *GARP Information Propagation (GIP)* component. Protocol exchanges take place between GARP Participants by means of LLC Type 1 services, using the group MAC Address and PDU format defined for the GARP Application concerned.

NOTE—This clause defines a generic PDU structure for use in GARP protocol exchanges (12.11); however, each GARP Application defines a further specification of that generic structure in order to carry the application identification and attribute values that are required for the operation of that application.

12.3.1 The GARP Application component

For each GARP Application, the following are defined:

a) A set of one or more Attribute types that are used by the Application;
b) The set of values that each Attribute type can carry;
c) The semantics associated with each Attribute type and value;
d) The group MAC Address used to exchange GARP PDUs between GARP Participants for that Application;
e) The structure and encoding of the Attribute types and values in GARP PDUs;
f) The requirements placed on the support of the GARP state machines in end stations and Bridges that support the application.

The GARP Application component of the GARP Participant is responsible for defining the semantics associated with parameter values and operators received in GARP PDUs, and for generating GARP PDUs for transmission. The Application component makes use of the GID component, and the state machines associ-

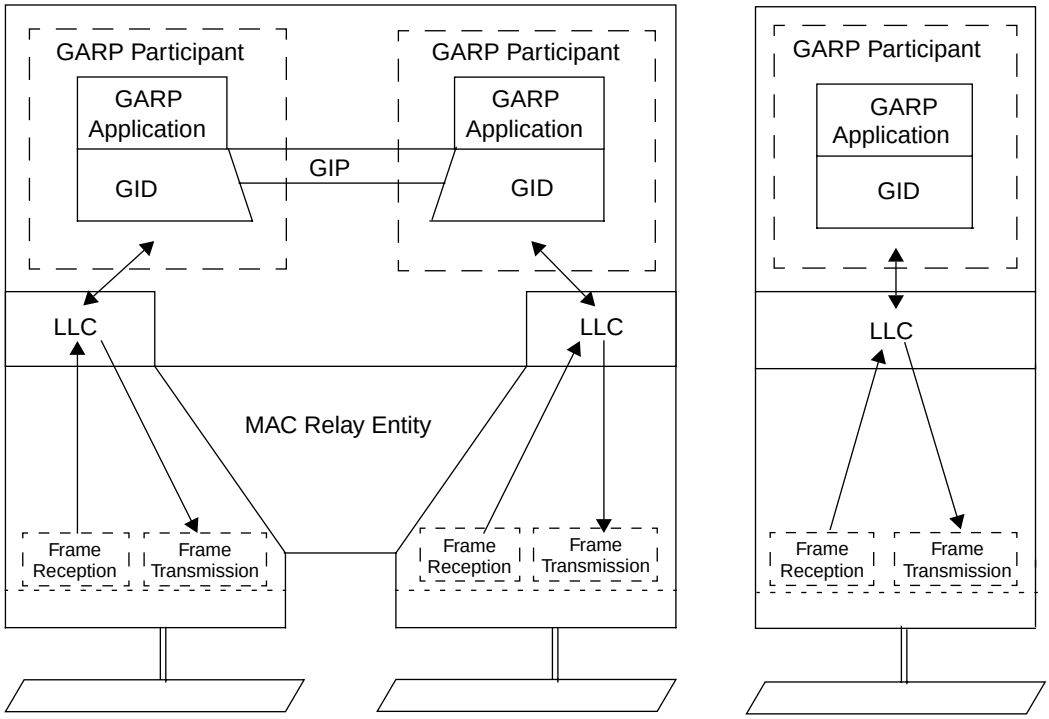


Figure 12-4—GARP architecture

ated with GID's operation, in order to control its protocol interactions. The service offered to the Application component by the GID component is defined by the set of primitives described in 12.3.2.

12.3.2 GID

An instance of GID consists of the set of state machines that define the current registration and declaration state of all attribute values associated with the GARP Participant of which the GID instance is a component. Figure 12-5 illustrates the set of state machines associated with a GID instance.

The operation of GID is defined by

a) The Applicant State Transition Table (Table 12-3);
b) The Registrar State Transition Table (Table 12-4);
c) The state machines that represent the current declaration state (Applicant state machines) and registration state (Registrar state machines) for each attribute value associated with the GARP Participant;
d) The service primitives available to users of GID.

12.3.2.1 Declarations

Two primitives are defined in order to allow users of the GID service to request GID to make (Join) or revoke (Leave) Attribute declarations through a given Port, as follows:

GID_Join.request (attribute_type, attribute_value)

GID_Leave.request (attribute_type, attribute_value)

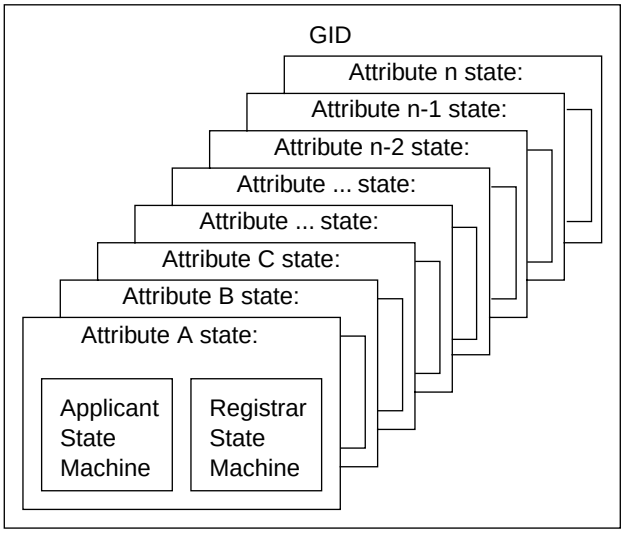


Figure 12-5—GID architecture

GID determines the action to be taken on receipt of these primitives in accordance with the current Applicant state for the Attribute concerned, and the action defined for that state and event in the Applicant State Transition Table (Table 12-3). The new state so determined is recorded in the Applicant state variable for that Attribute.

GID Requests can be generated by both the GARP Application component and the GARP Information Propagation function.

12.3.2.2 Registration

Two primitives are defined in order to allow the GID service to indicate to its users that a given Attribute value has been registered (Join) or de-registered (Leave) on a given Port, as a result of protocol activity on the LAN segment to which the Port is attached, as follows:

GID_Join.indication (attribute_type, attribute_value)

GID_Leave.indication (attribute_type, attribute_value)

GID determines the action to be taken on receipt of registration and de-registration information contained in GARP PDUs in accordance with the current Registrar state for the Attribute concerned, and the action defined for that state and event in the Registrar State Transition Table (Table 12-4). The new state so determined is recorded in the Registrar state variable for that Attribute.

GID Indications are received by both the GARP Application and the GARP Information Propagation function.

12.3.3 GIP

The GARP Information Propagation (GIP) function operates in the same manner for all GARP applications, and enables the Attribute information registered on Bridge Ports to be propagated across the Bridged LAN to other GARP Participants.

The operation of GIP is as follows, for a given GARP Application and GIP Context (12.3.4), and for the set of Ports that are in a Forwarding state as defined by that GIP Context:

a) Any GID_Join.indication received by GIP from a given Port in the set is propagated as a GID_Join.request to the instance(s) of GID associated with any other Port in the set;
b) Any GID_Leave.indication received by GIP from a given Port in the set is propagated as a GID_Leave.request to the instance(s) of GID associated with any other Port in the set (Port P, say) if and only if no registration now exists for that Attribute on any other Port in the set excluding Port P.

The effect of these propagation rules is that, for the set of Ports defined by the GIP Context, registrations are propagated on a given Port if any other Port has seen a registration for the Attribute concerned, and de-registrations are propagated on a given Port if all other Ports are now de-registered for the Attribute concerned.

As the set of Ports that are in a Forwarding state for a given GIP Context can change dynamically, for example as a result of Spanning Tree reconfiguration, GIP operates as follows on detection of a change in the set of forwarding Ports:

c) If a Port is added to the set of Ports that are in a Forwarding state, and that Port has previously registered an attribute (a GID_Join.indication has occurred more recently than any GID_Leave.indication for that attribute), then GID_Join.requests for that attribute are propagated to the instance(s) of GID associated with any other Port in the set;
d) If a Port is removed from the set of Ports that are in a Forwarding state, and that Port has previously registered an attribute (a GID_Join.indication has occurred more recently than any GID_Leave.indication for that attribute), then GID_Leave.requests for that attribute are propagated to the instance(s) of GID associated with any other Port in the set.

12.3.4 GARP Information Propagation Context

For a given Port of a GARP-aware Bridge and GARP Application supported by that Bridge, an instance of a GARP Participant can exist for each *GARP Information Propagation Context* (GIP Context) that is understood by that Bridge. A GIP Context defines a subset of the Ports of the Bridge that form the active topology (7.4) that applies in that Context. A GIP Context is defined relative to some other active topology, and defines a subset of that active topology.

An example of a GIP Context is the context defined by the active topology formed by the operation of the Spanning Tree algorithm and protocol (Clause 8). For this GIP Context, the active topology within a given Bridge that is defined by this Context is exactly equal to the active topology formed by the operation of Spanning Tree. This GIP Context is known as the *Base Spanning Tree Context*.

NOTE—This standard only makes use of the Base Spanning Tree Context to define the operation of GMRP; however, GARP has been defined in a manner that permits the use of other GIP Contexts should this be necessary for other GARP applications, or for extension to the existing functionality of GMRP. In particular, this flexibility has been included in order to allow for the use of GARP Applications in networking architectures based on the use of multiple Spanning Trees and/or Virtual LAN (VLAN) technologies, where the active topology would be defined by the instance of Spanning Tree, or the VLAN, within which the context was defined. VLANs are the subject of further development under project IEEE P802.1Q.

GARP protocol exchanges can occur on all of the Ports of a Bridge; however, the propagation of GARP registration information across a Bridged LAN, controlled by the operation of GIP (12.3.3), for a given GARP Application, occurs only among the set of Ports that comprise the active topology defined by the GIP Context. GIP Contexts are defined such that the active topology selected for propagation of registration information allows GARP registrations to be propagated towards all regions of the Bridged LAN that (potentially) contain GARP Participants for that GARP Application and GIP Context. Each GARP Application defines the set of GIP Contexts within which it can operate, defines the rules whereby a set of forwarding Ports is

selected for each Context, and assigns GIP Context identifiers to each Context for use in conjunction with the operation of GARP and its administrative controls (12.9).

A value of 0 shall be used as the GIP Context identifier for the GIP Context defined by the operation of the Spanning Tree algorithm and protocol defined in Clause 8. Where GARP is used in any GIP Context other than 0, the definition of the GARP Application shall specify how PDUs exchanged between GARP Participants are identified as belonging to the GIP Context concerned.

12.4 Requirements to be met by GARP

GARP and its associated algorithms operate in order to establish, maintain, and dissolve Attribute registrations within a Bridged LAN and to disseminate registration information among the GARP Participants attached to the LAN. In order to perform this function, the protocol and its associated algorithms meet the following requirements, which are addressed in the subclauses indicated. The requirements addressed include the requirements of Applicant and Registrar behavior, error recovery in failure conditions, performance, scalability, backwards compatibility with non-GARP aware devices, and the load imposed on Bridges, end stations and the network:

a) It will allow GARP Participants connected to a Bridged LAN to issue declarations for Attribute values associated with GARP applications (12.3.2, 12.8.1, and 12.10);
b) It will allow GARP Participants connected to a Bridged LAN to revoke declarations for Attribute values associated with GARP applications (12.3.2, 12.8.1, and 12.10);
c) It will propagate declarations received by a Bridge to GARP Participants reachable via LAN segments to which that Bridge is attached (12.3.3);
d) It will allow GARP Participants to maintain state information that indicates the current state of declaration and registration of Attributes on each Port of the participant device (12.8.1, and 12.8.2);
e) It will allow GARP Participants to remove state information that relates to Attributes that are no longer active within part or all of the Bridged LAN, for example, as a result of the failure of a Participant (12.8.2, and 12.8.3);
f) The latency involved in establishing or revoking Attribute registration information, and in the propagation of that information across the Bridged LAN, will be small (i.e., comparable to the frame propagation delay across the Bridged LAN) and will increase linearly as a function of the diameter of the Bridged LAN (12.8.1, 12.10, 12.8.2, and 12.8.3);
g) It will operate in a manner that is resilient in the face of the failure of GARP Participants (H.1.4);
h) It will operate in a manner that is resilient in the face of single packet loss;
i) It will operate correctly
    1) In homogeneous Bridged LANs, i.e., Bridged LANs where all Bridges support both Basic Filtering Services and Extended Filtering Services (12.8.1, 12.10, 12.8.2, and 12.8.3);
    2) In heterogeneous Bridged LANs, i.e., Bridged LANs where there are some Bridges that support only Basic Filtering Services and some Bridges that support both Basic Filtering Services and Extended Filtering Services (12.5, 12.8.1, 12.10, 12.8.2, 12.8.3, and F.2).
j) The communications bandwidth consumed on any particular LAN segment by Applicants and Registrars in the PDU exchanges associated with the protocol will be a small percentage of the total available bandwidth, and independent of the total traffic supported by the Bridged LAN. The bandwidth consumed will be a function of the number of Groups for which Applicants on that LAN wish to maintain membership.

12.5 Requirements for interoperability between GARP Participants

In order to ensure the interoperability of GARP, the following are required:

a) For each defined GARP Application, a unique group MAC Address, known as the GARP Application address, shall be used as the destination MAC Address for GARP protocol exchanges between

GARP Participants for that Application. Table 12-1 defines the set of group MAC Addresses that have been allocated to existing GARP Applications; the unallocated values in that table have been reserved for use by future standardized GARP Applications. Bridges that implement the GARP Application corresponding to a given entry in that table are required not to forward frames destined for that MAC Address; Bridges that do not implement the GARP Application corresponding to a given entry in that table are required to forward frames destined for that MAC Address received on any Port that is part of the active topology on all other Ports that are part of the active topology;

b) The transmission and reception of GARP PDUs between GARP Participants, formatted as defined for the GARP Application concerned, using the generic PDU format defined in 12.11, shall be achieved by means of LLC Type 1 procedures. The standard LLC Address assignment for the Spanning Tree Protocol as defined in Table 7-8 shall be used as the source and destination LLC address. This use of LLC Type 1 procedures is shown in Figure 12-4;

c) PDUs received by a GARP Participant [i.e., PDUs with destination MAC and LLC addresses as specified in a) and b)], but which are not well formed (i.e., not structured and encoded as defined in 12.11 and with attribute types and values encoded as defined by the Application concerned), shall be discarded;

d) The protocol behavior of GARP Participants shall be in accordance with the state machine descriptions and procedures defined in 12.8 and 12.10.

Table 12-1—GARP Application addresses

Assignment	Value
GMRP address (See Clause 10)	01-80-C2-00-00-20
Reserved	01-80-C2-00-00-21
Reserved	01-80-C2-00-00-22
Reserved	01-80-C2-00-00-23
Reserved	01-80-C2-00-00-24
Reserved	01-80-C2-00-00-25
Reserved	01-80-C2-00-00-26
Reserved	01-80-C2-00-00-27
Reserved	01-80-C2-00-00-28
Reserved	01-80-C2-00-00-29
Reserved	01-80-C2-00-00-2A
Reserved	01-80-C2-00-00-2B
Reserved	01-80-C2-00-00-2C
Reserved	01-80-C2-00-00-2D
Reserved	01-80-C2-00-00-2E
Reserved	01-80-C2-00-00-2F

NOTE 1—The second entry in Table 12-1 is allocated in IEEE P802.1Q for use by the GARP VLAN Registration Protocol (GVRP).
NOTE 2—GARP makes use of the same LLC Address as the Spanning Tree protocol; however, the use of distinct MAC Addresses and protocol identifiers for GARP Applications and Spanning Tree ensures that the appropriate PDUs can be delivered to the appropriate protocol entities. PDUs with a destination MAC Address equal to the Bridge Group Address identified in Table 7-9, an LLC address equal to the value assigned in Table 7-8, and that carry the Spanning Tree protocol identifier as defined in Clause 9, are received by the Bridge Protocol Entity (see 7.12.3). PDUs with a destination MAC Address equal to any of the GARP Application addresses as identified in Table 12-1, an LLC address equal to the value assigned in Table 7-8, and that carry the GARP protocol identifier as defined in 12.11, are handled as described in a) through d) above.

12.6 Conformance to GARP Applications

The specification of GARP as described in this clause defines aspects of behavior that are generic to GARP Applications. Conformance to GARP is defined with reference to a specific GARP Application or Applications. Each GARP Application defines the aspects of the GARP functionality that are required in order to claim conformance to that application, as stated in 12.3.1. A conformant implementation of GARP is therefore an implementation that supports at least the GARP functionality that is required in order to claim conformance to the set of GARP Applications supported by that implementation.

12.7 Overview of GARP protocol operation

This overview is intended to serve only as an informal introduction to the operation of the GARP protocol. The definitive description is contained in the State Machine Descriptions (12.8), Procedures (12.10), and Encoding of GARP Protocol Data Units (12.11).

In the following description, propagation of information between Ports of a system follows the active topology for the GIP context.

12.7.1 Basic notions

The basic notions behind the operation of GARP are that

a) A simple, fully distributed many-to-many protocol is possible. There is no need for an additional election protocol to change the problem to allow a many-to-one design;
b) The protocol should be resilient against the loss of a single message, in a set of related messages, but does not need to be stronger;
c) A GARP Participant that wishes to make a declaration (an Applicant) sends Join messages;
d) If an Applicant sees other Participants sending two Join messages, it does not need to send a Join message itself in order to participate in the declaration concerned;
e) An Applicant that wishes to withdraw a declaration need only send a single Leave message; it can then forget all about the registration concerned. There is no need for it to confirm that de-registration has taken place, as other Participants may wish to maintain the registration themselves.
f) Missing or spuriously continued registrations that arise from multiple lost messages are cleared up by a periodic mechanism that sends LeaveAll messages. A LeaveAll message declares that all registrations will shortly be terminated unless one or more Participants declares a continuing interest in specific registrations by issuing further Joins.

To guard against the possibility that a group member misses a Leave message, thus causing another Participant's Registrar (the component that records membership of the Group) to think that there are no members, one additional mechanism is necessary. If a Participant receives a Leave message, and no subsequent joins, it sends a further message to prompt rejoining before terminating the registration.

12.7.2 GARP messages

The description so far introduces three basic message types used in GARP: Join, Leave, and LeaveAll. However, making do with only these three message types would add to the eventual complexity of the protocol:

— Consider the case of two GARP Participants at either end of a point-to-point link. The fact that one of them (Andy, say) sends the other (Bill) two Join messages says nothing about Andy's knowledge of Bill's wish to make that declaration. The recitation of the basic GARP notions above obscures the fact that would-be Applicants may also need to know if there are other Applicants making the same declaration, a requirement for bridge ports, for example. Attempts to work around the problem by setting join timers such that no single Participant can send two messages in an interval within which two or more Participants can be expected to send messages results in timer correctness dependencies that are fraught with danger in terms of determining the correctness of the protocol operation.
— Consider the sending of a second Leave by a Registrar to prompt a rejoin. If a Join is just being sent, the protocol now depends on the second Join not being lost. The protocol depends on the relative values of the Registrar's leave timer and other Participants' join timers.

The protocol has therefore been based on the general design principle that protocol Participants should communicate their current state, rather than send directions. Four Attribute-specific message types are used:

a) Empty: I am not trying to declare this Attribute value. I have not registered this Attribute value, but I care if there are any Participants that wish to declare it.
b) JoinEmpty: I wish to declare this Attribute value. I have not registered this Attribute value, but I care if there are any Participants that wish to declare it.
c) JoinIn: I wish to declare this Attribute value. I have either registered this Attribute value, or I do not care if there are any other Participants that wish to declare it (I will behave as if there are).
d) Leave: I had registered this Attribute value, but am now in the process of de-registering it.

along with the garbage collection message, as before

e) LeaveAll: All registrations will shortly be de-registered; if any Participants have a continuing interest in any of the registrations they need to rejoin in order to maintain the registration.

In theory there could be LeaveIn and LeaveEmpty variants of the Leave message; these are coded in the GARP PDUs to provide maximum visibility into what implementations are doing, and to avoid missing or illegal codes. However, it will be seen that the state machines treat these two message variants identically.

There is no reason to send a simple In message, i.e., one that means "I do not wish to make this declaration but have registered the Attribute value on behalf of other Participants (or will behave as if there are other Participants that have made the declaration)."

The protocol makes good use of the distinction between JoinEmpty and JoinIn messages, and between Leave and Empty.

The JoinIn message meets the requirements for Join message suppression. If an Applicant sees a JoinIn message it can avoid sending a Join itself for that declaration, as it knows that both the recipient(s) and the transmitter of the JoinIn believe there are Participants that have made the declaration. The JoinIn is not treated as an acknowledgment, because on a multiaccess segment, there are potentially many Participants who need to register the Attribute value. Moreover, Participants who don't care whether there are other Participants interested in that registration or not can always send JoinIns instead of JoinEmptys. However, on the assumption that only one JoinIn message is lost, two suffice to ensure that all Registrars have registered the group, to a high probability.

The Leave message will cause its recipients to de-register membership, while the JoinEmpty and Empty messages will just prompt them to rejoin, so JoinEmpty and Empty messages can be used at any time to prompt for rejoin without throwing recently joined members out again.

12.7.3 Applicant and Registrar

Each GARP participant maintains a single Leave All protocol component (12.7.6). It also maintains two protocol components per Attribute that it is interested in: an Applicant and a Registrar.

The job of the Registrar is to record Attribute registration declared by the other participants on the segment. It does not send any protocol messages.

The job of the Applicant is twofold:

a) To ensure that this Participant's declarations are registered by other Participants' registrars—if it wants to maintain those registrations.
b) To ensure that other Participants have a chance to re-declare (rejoin), after anyone withdraws a declaration (leaves)—if there are any Participants that want to maintain the registration.

NOTE—Item b) above applies only to the behavior of the full Applicant state machine (12.7.5); the Applicant Only and Simple Applicant state machines (12.7.7 and 12.7.8) are concerned only with item a).

The Applicant is therefore looking after the interests of all would-be Participants. This allows the Registrar to be very simple.

12.7.4 Registrar behavior

The Registrar has a single timer, the leave timer, and three states:

a) IN: I have registered the fact that this Attribute value has been declared on this segment.
b) MT: (Empty) All declarations for this Attribute value on this segment have been withdrawn.
c) LV: I had registered this Attribute value, but am now timing out the registration (using the leave timer). If I do not see a declaration for this Attribute before leave timer expires, I will become MT.

The Registrar reacts to received messages as follows:

d) A Join message, either JoinIn or JoinEmpty, causes the Registrar to become IN (I have registered the Attribute).
e) If the Registrar was IN, then a Leave or LeaveAll causes it to become LV (I am timing out the registration) and starts the leave timer. Otherwise (LV or MT) there is no effect.
f) An Empty message (someone else has no registration for this Attribute) has no effect [see 12.7.2 a)].

While the Registrar does not send messages, it affects the type of Join message sent by the Applicant. If the Registrar is IN, a JoinIn is sent; otherwise, a JoinEmpty is sent.

12.7.5 Applicant behavior

Against the background of this simple Registrar, the next consideration is the behavior of the Applicant that wishes to make a declaration, starting from a point where it has neither seen nor sent any messages.

If no messages were ever lost, the Applicant could either send a Join or receive a JoinIn, and then be content that all Registrars would have registered its declaration. On the single message loss assumption it needs to send two Joins, or receive two JoinIns, or send one Join and receive one JoinIn (in either order). This part of its state could be recorded in a simple counter:

my_membership_msgs = 0, 1, or 2

which is incremented for every Join sent or JoinIn received. If the counter value is 0 or 1 when there is an opportunity to transmit a PDU, a Join message will be sent and the counter incremented.

NOTE 1—A counter value of greater than 2 is unnecessary for the purposes of successful registration.
NOTE 2—A randomized Join timer is set running to ensure such an opportunity is scheduled. There only needs to be one Join timer running for the entire Participant, not one per attribute—assuming that messages related to the maximum number of attributes can be packed into a single PDU.

If a JoinEmpty, Empty, Leave, or LeaveAll message is received, the counter is reset to 0.

When the Applicant leaves the Group, it sends a single Leave message.

12.7.5.1 Anxious Applicants

Expressing protocol behavior in terms of counter and flag variables is not always the best approach if enabling thorough analysis and maximizing implementation flexibility are primary goals. From this point on, the values assigned to the join message count are given state name prefixes:

a) V or Very anxious equates to my_join_msgs = 0. No Join messages have been sent, and no JoinIns received since the Applicant started, or leave or empty messages received. The Applicant has no reason to be comfortable that other Registrars have registered the Attribute value concerned.
b) A or Anxious equates to my_join_msgs = 1. If no messages have been lost, other Registrars will have registered this Attribute value.
c) Q or Quiet equates to my_join_msgs = 2. The Applicant feels no need to send further messages.

12.7.5.2 Members and Observers

The Applicant described so far needs have no existence unless it is trying to make a declaration. Bridge Ports and end stations that make active use of registered Attribute values (e.g., in the case of the GMRP Application defined in Clause 10, Bridges and end stations that implement source pruning for transmission), need to maintain their GARP machines even if they do not want to make (or have just withdrawn) a declaration. (The term GARP machine refers to the total state maintained for a given Attribute value, both Applicant and Registrar, in a Participant.)

In the context of the Applicant state machine, a Member is a Participant that is attempting to make or maintain a declaration for a given Attribute value, or that has not yet sent the Leave message to allow it to become simply an Observer. An Observer tracks the Attribute state but does not wish to make a declaration.

12.7.5.3 Active and Passive Members

The concept of Active and Passive Members is introduced to permit the minimum number of messages to be sent when a number of Participants are actively joining and leaving with respect to the same registration.

Since a Member may become Quiet without ever sending a Join, it follows that it should be allowed to become an Observer once more without sending a Leave. All Observers are passive, so there are three potential (sub)states, distinguished by the following state name suffixes:

a) A, or Active member.
b) P, or Passive member.
c) O, or Observer.

If an Observer is required to become a Member, it first becomes a Passive Member. If it was a Quiet Observer (i.e., its count of Join messages is already at two and it is therefore content that other Registrars have registered the declaration), then it has no need to transmit a Join, and becomes Passive and Quiet. Otherwise, i.e., its count of Join messages is less than two, it requests the earliest possible message transmission opportunity in order to transmit a Join.

If a Passive Member sends a Join message, it becomes an Active Member.

If an Active Member receives a Leave or LeaveAll message, it becomes a Passive Member.

12.7.5.4 Receiving a Leave

When an Applicant that is, and wishes to continue being, a Member receives a Leave Message, it becomes Very Anxious. Unless it receives a Join message from another Member, it will send a JoinEmpty itself. This has the following effect on other Members. First, it will cause them to register that Attribute. Second, it will cause them to become Very Anxious themselves if they wish to continue to be Members, and to transmit JoinIns.

This latter effect protects any Participant that is a Member from accidentally de-registering other Members due to a single packet loss following a Leave.

An Applicant that is an Observer has to prompt other Members to re-join in case they have missed the Leave. A further (sub) state is added to the Very Anxious, Anxious, Quiet set with state name prefix:

— L or Leaving, which records the pending need to send a message at the next transmission opportunity. An Observer will send an Empty message, and then become Very Anxious.

12.7.5.5 Leaving

An Active Member has to send a Leave message in order to withdraw a declaration. The Leaving substate is used to record that fact.

12.7.5.6 Applicant State Summary

The following matrix summarizes the Applicant states and their short names: VA for Very anxious Active member, QO for Quiet Observer, etc.

Table 12-2—Applicant: Summary of states

	Very Anxious	**Anxious**	**Quiet**	**Leaving**
Active Member	VA	AA	QA	LA
Passive Member	VP	AP	QP	
Observer	VO	AO	QO	LO

Note that there is no LP (Leaving Passive Member) state, since a Passive Member can transition directly to an Observer state when it wishes to withdraw a declaration.

12.7.6 The Leave All protocol component

The Leave All protocol component is responsible for initiating garbage collection by the Participant. This is achieved by the regular generation of LeaveAll messages by the LeaveAll state machine, as defined in 12.8.3.

The operation of the Leave All state machine causes the Participant to generate a LeaveAll message when the leaveAllTimer expires. Reception of a Leave All message from another Participant causes the timer to be restarted without generating a message, thus ensuring that, where several Participants are connected to the same LAN segment, multiple LeaveAll messages are suppressed.

Receipt of a LeaveAll message causes all Applicants to become Very Anxious, and all Registrars to enter the Leaving (LV) state. The effect of this is to force any Applicants that are still active to rejoin; if no Join message is seen by a Registrar state machine before its leave timer expires, it will de-register the attribute. This effectively causes the Participant to remove all registrations for which active Applicants no longer exist.

12.7.7 Applicant Only Participants

It is possible to simplify the GARP Participant in circumstances where the Participant only wishes to make declarations; for example, as might be required in an end station implementing the ability to register to receive group addressed frames, via GMRP (Clause 10), but that does not operate as a source of such frames, and therefore does not need to implement source pruning. Such a Participant has no need to take note of declarations made by other Participants (i.e., it does not need to implement the Registrar state machine), does not send Leave All or Join Empty messages, and offers no additional administrative controls.

This leads to a potential simplification of the state machinery that such a participant would need to support, as follows:

a) No requirement to support a LeaveAll timer, or the generation of LeaveAll PDUs;
b) No requirement to support the operation of the Registrar state machine. The “IN” state for the Registrar is assumed for the purposes of the operation of the Applicant state machine, hence Join messages are always sent as JoinIn;
c) No requirement for the state machine to support the LO state or generate Empty messages;
d) No requirement to support the administrative controls defined in 12.9.

The operation of the Applicant Only state machine is described in 12.8.5 and Table 12-8.

12.7.8 Simple Applicant Participants

The operation of the Simple Applicant Participant is a further simplification of the Applicant Only state machine (Table 12-8), modified to reduce the Applicant to its simplest form, achieved by removing the Passive Member and Observer states. Consequently, the Simple Applicant Participant makes no attempt to suppress its initial Join and final Leave messages. The result is the simplest possible Applicant state machine that is compatible with the operation of the full GARP Participant.

The operation of the Simple Applicant state machine is described in 12.8.6 and Table 12-9.

12.7.9 Choice of Applicant Only Participant or Simple Applicant Participant

The fact that the Applicant Only Participant retains the Passive Member and Observer states of the full Applicant means that it retains the ability of the full Participant to suppress Join or Leave messages when these are unnecessary (see 12.7.5.3); in contrast, the Simple Applicant Participant is unable to perform such suppression. Where there is the possibility of several Simple Applicant Participants appearing on the same LAN segment, this could result in significant additional, and unnecessary, Join and Leave traffic. Consequently, it is recommended that the Applicant Only Participant is implemented in preference to the Simple Applicant Participant in devices that have no need to perform registration.

12.8 State machine descriptions

The following conventions are used in the abbreviations used in this subclause:

rXXX	receive PDU XXX
sXXX	send PDU XXX
ReqXXX	GID service request XXX
IndXXX	GID service indication XXX

The following abbreviations are used in the state machine descriptions. For a formal definition of their meaning, see 12.10:

rJoinIn	receive Join In message
rJoinEmpty	receive Join Empty message
rEmpty	receive Empty message
rLeaveIn	receive Leave In message
rLeaveEmpty	receive Leave Empty message
LeaveAll	receive or send a LeaveAll message
sJ[E,I]	send Join In message if Registrar state = IN; otherwise, send Join Empty message
sJ[I]	send Join In message
sE	send Empty message
sLE	send Leave Empty message
sLeaveAll	send Leave All message
ReqJoin	GID Service Request to declare an Attribute value
ReqLeave	GID Service Request to withdraw an Attribute value declaration
IndJoin	Issue GID Service Indication signaling an Attribute value has been registered
IndLeave	Issue GID Service Indication signaling an Attribute value has been de-registered
transmitPDU!	An opportunity to transmit a GARP PDU has occurred. Such events occur at intervals randomly chosen in the interval 0 - JoinTime. JoinTime is as defined in Table 12-10
leavetimer	Leave period timer
leavetimer!	leavetimer has expired
leavealltimer	Leave All period timer.
leavealltimer!	leavealltimer has expired.
-x-	Inapplicable event/state combination. No action or state transition occurs.

Timers are used in the state machine descriptions in order to cause actions to be taken after defined time periods have elapsed. The following terminology is used in the state machine descriptions to define timer states and the actions that can be performed upon them:

a) A timer is said to be *running* if the most recent action to be performed upon it was a *start* or a *restart.*

b) A running timer is said to have *expired* when the time period associated with the timer has elapsed since the most recent start or restart action took place.

c) A timer is said to be *stopped* if it has expired or if the most recent action to be performed upon it was a *stop* action.

d) A *start* action sets a stopped timer to the running state, and associates a time period with the timer. This time period supersedes any periods that might have been associated with the timer by previous start events.

e) A *restart* action stops a running timer and then performs a start action upon it.

f) A *stop* action sets a timer to the stopped state.

The following abbreviations are used for the state names in the state tables and state diagrams:

Registrar states

LV Leaving
IN In
MT Empty

Applicant states

VA Very anxious, active member
AA Anxious, active member
QA Quiet, active member
LA Leaving, active member
VP Very anxious, passive member
AP Anxious, passive member
QP Quiet, passive member
VO Very anxious, observer
AO Anxious, observer
QO Quiet, observer
LO Leaving, observer

Simple Applicant states

V Very anxious
A Anxious
Q Quiet

12.8.1 Applicant state machine

A full GARP Participant maintains a single instance of this state machine for each Attribute value for which the Participant needs to maintain state information.

NOTE—Conceptually, state information is maintained for all possible values of all Attribute types that are defined for a given Application; however, in real implementations of GARP, it is likely that the range of possible Attribute values in some Applications will preclude this, and the implementation will limit the state to those Attribute values in which the Participant has an immediate interest, either as a Member or as a likely future Member.

The detailed operation of this state machine is described in Table 12-3. The state transitions shown handle the possibilities of receiving either Leave In or Leave Empty messages; however, only Leave Empty messages are generated by the state machine. Sending a Leave Empty message also causes an event against the Registrar state machine, causing a transition from IN to LV.

12.8.2 Registrar state machine

A full GARP Participant maintains a single instance of this state machine for each Attribute value that is currently registered, or that the Registrar state machine is in the process of de-registering.

Table 12-3—Applicant state table

		STATE										
		VA	**AA**	**QA**	**LA**	**VP**	**AP**	**QP**	**VO**	**AO**	**QO**	**LO**
EVENT	**transmitPDU!**	sJ[E,I] AA	sJ[E,I] QA	-x-	sLE VO	sJ[E,I] AA	sJ[E,I] QA	-x-	-x-	-x-	-x-	sE VO
	rJoinIn	AA	QA	QA	LA	AP	QP	QP	AO	QO	QO	AO
	rJoinEmpty	VA	VA	VA	VO	VP	VP	VP	VO	VO	VO	VO
	rEmpty	VA	VA	VA	LA	VP	VP	VP	VO	VO	VO	VO
	rLeaveIn	VA	VA	VA	LA	VP	VP	VP	LO	LO	LO	VO
	rLeaveEmpty	VP	VP	VP	VO	VP	VP	VP	LO	LO	LO	VO
	LeaveAll	VP	VP	VP	VO	VP	VP	VP	LO	LO	LO	VO
	ReqJoin	-x-	-x-	-x-	VA	-x-	-x-	-x-	VP	AP	QP	VP
	ReqLeave	LA	LA	LA	-x-	VO	AO	QO	-x-	-x-	-x-	-x-

NOTE—As with the Applicant, state information is conceptually maintained for all possible values of all Attribute types that are defined for a given Application; however, in real implementations of GARP, it is likely that the range of possible Attribute values in some Applications will preclude this, and the implementation will limit the state to those Attribute values in which the Participant has an immediate interest. In the case of simple devices that have no interest in what other Participants have registered, it may be appropriate for that device to ignore Registrar operation altogether.

The detailed operation of this state machine is described in Table 12-4.

12.8.3 Leave All state machine

A single Leave All state machine exists for each full GARP Participant. Leave All messages generated by this state machine also generate LeaveAll events against all the Applicant and Registrar state machines associated with that Participant and Port; hence, LeaveAll generation is treated by those state machines in the same way as reception of a LeaveAll message from an external source.

The detailed operation of this state machine is described in Table 12-5.

12.8.4 Combined Applicant/Registrar state machine

Table 12-6 shows all the reachable states, with cells containing the joint state names, Applicant.Registrar, and unreachable states marked ---. The MT and LV Registrar states are grouped together since the only event that differentiates the two is the expiry of the leave timer, which does not affect any of the other states. There are 24 reachable states in all.

The combined state machine is shown in Table 12-7. For compactness, the actions—what message is transmitted, when the implementation should check or start timers, when to indicate joins and leaves to the higher layer user—have been omitted.

Table 12-4—Registrar state table

		STATE		
		IN	**LV**	**MT**
EVENT	**rJoinIn**	IN	Stop leavetimer IndJoin IN	IndJoin IN
	rJoinEmpty	IN	Stop leavetimer IndJoin IN	IndJoin IN
	rEmpty	IN	LV	MT
	rLeaveIn	Start leavetimer LV	LV	MT
	rLeaveEmpty	Start leavetimer LV	LV	MT
	LeaveAll	Start leavetimer LV	LV	MT
	leavetimer!	-x-	IndLeave MT	-x-

Table 12-5—Leave All state table

		STATE	
		Active	**Passive**
EVENT	**transmitPDU!**	sLeaveAll Passive	-x-
	LeaveAll	Start leavealltimer Passive	Start leavealltimer Passive
	leavealltimer!	Start leavealltimer Active	Start leavealltimer Active
	(All other events)	-x-	-x-

Table 12-6—Combined Applicant/Registrar states

	Very Anxious		**Anxious**		**Quiet**		**Leaving**	
Active Member	VA.MT VA.LV	VA.IN	AA.MT AA.LV	AA.IN	QA.MT QA.LV	QA.IN	LA.MT LA.LV	LA.IN
Passive Member	VP.MT VP.LV	VP.IN	--- ---	AP.IN	---	QP.IN		
Observer	VO.MT VO.LV	VO.IN	--- ---	AO.IN	---	QO.IN	LO.MT LO.LV	---

Table 12-7—Combined Applicant/Registrar state table

		EVENT								
		leavetimer!	**transmitPDU!**	**rJoinIn**	**rJoinEmpty**	**rEmpty**	**rLeaveIn**	**rLeaveEmpty, LeaveAll**	**ReqJoin**	**ReqLeave**
STATE	**VA.MT**	-x-	AA.MT	AA.IN	VA.IN	VA.MT	VP.MT	VP.MT	-x-	LA.MT
	VA.LV	VA.MT	AA.LV	AA.IN	VA.IN	VA.LV	VP.LV	VP.LV	-x-	LA.LV
	VA.IN	-x-	AA.IN	AA.IN	VA.IN	VA.IN	VP.LV	VP.LV	-x-	LA.IN
	AA.MT	-x-	QA.MT	QA.IN	VA.IN	VA.MT	VP.MT	VP.MT	-x-	LA.MT
	AA.LV	AA.MT	QA.LV	QA.IN	VA.IN	VA.LV	VP.LV	VP.LV	-x-	LA.LV
	AA.IN	-x-	QA.IN	QA.IN	VA.IN	VA.IN	VP.LV	VP.LV	-x-	LA.IN
	QA.MT	-x-	—	QA.IN	VA.IN	VA.MT	VP.MT	VP.MT	-x-	LA.MT
	QA.LV	QA.MT	—	QA.IN	VA.IN	VA.LV	VP.LV	VP.LV	-x-	LA.LV
	QA.IN	-x-	—	QA.IN	VA.IN	VA.IN	VP.LV	VP.LV	-x-	LA.IN
	LA.MT	-x-	VO.MT	LA.IN	VO.IN	LA.MT	LA.MT	VO.MT	VA.MT	-x-
	LA.LV	LA.MT	VO.LV	LA.IN	VO.IN	LA.LV	LA.LV	VO.LV	VA.LV	-x-
	LA.IN	-x-	VO.LV	LA.IN	VO.IN	LA.IN	LA.LV	VO.LV	VA.IN	-x-
	VP.MT	-x-	AA.MT	AP.IN	VP.IN	VP.MT	VP.MT	VP.MT	-x-	VO.MT
	VP.LV	VP.MT	AA.LV	AP.IN	VP.IN	VP.LV	VP.LV	VP.LV	-x-	VO.LV
	VP.IN	-x-	AA.IN	AP.IN	VP.IN	VP.IN	VP.LV	VP.LV	-x-	VO.IN
	AP.IN	-x-	QA.IN	QP.IN	VP.IN	VP.IN	VP.LV	VP.LV	-x-	AO.IN
	QP.IN	-x-	—	QP.IN	VP.IN	VP.IN	VP.LV	VP.LV	-x-	QO.IN
	VO.MT	-x-	—	AO.IN	VO.IN	VO.MT	LO.MT	LO.MT	VP.MT	-x-
	VO.LV	VO.MT	—	AO.IN	VO.IN	VO.LV	LO.LV	LO.LV	VP.LV	-x-
	VO.IN	-x-	—	AO.IN	VO.IN	VO.IN	LO.LV	LO.LV	VP.IN	-x-
	AO.IN	-x-	—	QO.IN	VO.IN	VO.IN	LO.LV	LO.LV	AP.IN	-x-
	QO.IN	-x-	—	QO.IN	VO.IN	VO.IN	LO.LV	LO.LV	QP.IN	-x-
	LO.MT	-x-	VO.MT	AO.IN	VO.IN	VO.MT	LO.MT	LO.MT	VP.MT	-x-
	LO.LV	LO.MT	VO.LV	AO.IN	VO.IN	VO.LV	LO.LV	LO.LV	VP.LV	-x-

12.8.5 Applicant Only GARP Participant

An Applicant Only GARP Participant maintains a single instance of the Applicant Only state machine for each Attribute value for which the Participant needs to maintain state information.

The detailed operation of this state machine is described in Table 12-8. The state transitions shown handle the possibilities of receiving either Leave In or Leave Empty messages; however, only Leave Empty messages are generated by the state machine.

Table 12-8—Applicant Only State Machine

		STATE									
		VA	**AA**	**QA**	**LA**	**VP**	**AP**	**QP**	**VO**	**AO**	**QO**
EVENT	**transmit-PDU!**	sJ[I] AA	sJ[I] QA	-x-	sLE VO	sJ[I] AA	sJ[I] QA	-x-	-x-	-x-	-x-
	rJoinIn	AA	QA	QA	LA	AP	QP	QP	AO	QO	QO
	rJoinEmpty	VA	VA	VA	VO	VP	VP	VP	VO	VO	VO
	rEmpty	VA	VA	VA	LA	VP	VP	VP	VO	VO	VO
	rLeaveIn	VA	VA	VA	LA	VP	VP	VP	VO	VO	VO
	rLeave-Empty	VP	VP	VP	VO	VP	VP	VP	VO	VO	VO
	LeaveAll	VP	VP	VP	VO	VP	VP	VP	VO	VO	VO
	ReqJoin	-x-	-x-	-x-	VA	-x-	-x-	-x-	VP	AP	QP
	ReqLeave	LA	LA	LA	-x-	VO	AO	QO	-x-	-x-	-x-

12.8.6 Simple Applicant Participant

A Simple Applicant Participant maintains a single instance of the Simple Applicant state machine for each Attribute value for which the Participant needs to maintain state information.

The detailed operation of this state machine is described in Table 12-9. The state transitions shown handle the possibilities of receiving either Leave In or Leave Empty messages; however, only Leave Empty messages are generated by the state machine.

12.9 Administrative controls

Associated with each instance of the Registrar state machines are *Registrar Administrative Control* parameters. These parameters allow administrative control to be exercised over the registration state of each Attribute value, and hence, via the propagation mechanism provided by GIP, allow control to be exercised over the propagation of declarations.

An overall control parameter for each Applicant state machine, the *Applicant Administrative Control*, determines whether or not the Applicant state machine participates in GARP protocol exchanges.

Table 12-9—Simple Applicant State Machine

		STATE		
		V	**A**	**Q**
EVENT	**transmitPDU!**	sJ[I] A	sJ[I] Q	-x-
	rJoinIn	A	Q	Q
	rJoinEmpty	V	V	V
	rEmpty	V	V	V
	rLeaveIn	V	V	V
	rLeaveEmpty	V	V	V
	LeaveAll	V	V	V
	ReqJoin	-x-	-x-	-x-
	ReqLeave	sLE O	sLE O	sLE O

These parameters can be set to the values defined in 12.9.1 and 12.9.2.

12.9.1 Registrar Administrative Control values

a) *Normal Registration.* The Registrar responds normally to incoming GARP messages.
b) *Registration Fixed.* The Registrar ignores all GARP messages, and remains in the IN (registered) state.
c) *Registration Forbidden.* The Registrar ignores all GARP messages, and remains in the EMPTY (unregistered) state.

The default value of this parameter is Normal Registration.

Optionally, an implementation may support the ability to record against each Registrar state machine the MAC Address of the originator of the GARP PDU that caused the most recent state change for that state machine.

NOTE—The Registrar Administrative Controls are realized by means of the contents of the Port Map parameters of static entries in the Filtering Database for all GARP applications. In the case of GMRP, the static entries concerned are Static Filtering Entries (7.9.1 and 10.3.2.3). The contents of the Port Map parameters in static entries can be modified by means of the management operations defined in 14.7. In the absence of such control information for a given attribute, the default value “Normal Registration” is assumed.

12.9.2 Applicant Administrative Control values

a) *Normal Participant.* The state machine participates normally in GARP protocol exchanges.
b) *Non-Participant.* The state machine does not send any GARP messages.

The default value of this parameter is Normal Participant.

NOTE—The Applicant Administrative Control parameters can be modified for any GARP application by means of the management operations defined in 14.9. In the absence of such information for a given attribute, the default value “Normal Participant” is assumed.

12.10 Procedures

The following subclauses define the protocol actions and procedures that are identified in the description of the State Machines contained in 12.8.

12.10.1 Discarding badly formed GARP PDUs

A GARP Participant that receives a GARP PDU shall discard that PDU if any of the following are true:

a) The PDU carries an unknown protocol identifier;
b) The PDU is not formatted according to the GARP PDU format defined in 12.11.

Items of information contained within a GARP PDU that are not understood by the GARP Application shall be discarded as described in 12.11.3.

12.10.2 Protocol parameters and timers

12.10.2.1 jointimer

The Join Period Timer, jointimer, controls the interval between transmitPDU! events that are applied to the Applicant State Machine. An instance of this timer is required on a per-Port, per-GARP Participant basis. The maximum time period between transmitPDU! events is defined by JoinTime, as defined in Table 12-10.

12.10.2.2 leavetimer

The Leave Period Timer, leavetimer, controls the period of time that the Registrar State Machine will wait in the LV state before transiting to the MT state. An instance of the timer is required for each state machine that is in the LV state. The Leave Period Timer is set to the value LeaveTime when it is started or restarted; LeaveTime is defined in Table 12-10.

12.10.2.3 leavealltimer

The Leave All Period Timer, leavealltimer, controls the frequency with which the Leave All state machine generates LeaveAll PDUs. The timer is required on a per-Port, per-GARP Participant basis. The Leave All Period Timer is set to a random value, T, in the range LeaveAllTime < T < 1.5*LeaveAllTime when it is started or restarted. LeaveAllTime is defined in Table 12-10.

12.10.3 Protocol event definitions

Unless stated otherwise in these event definitions, GARP PDU reception in a Bridge can occur through all Ports of a Bridge, and events generated as a result of such reception affect only those state machines that are associated with the Port through which the PDU was received.

12.10.3.1 ReqJoin

For an instance of the combined Applicant/Registrar state machine, the Applicant Only state machine, or the Simple Applicant state machine, the ReqJoin event is deemed to have occurred if the GID Service User issues a GID_Join.request service primitive for the Attribute instance associated with that state machine.

12.10.3.2 ReqLeave

For an instance of the combined Applicant/Registrar state machine, the Applicant Only state machine, or the Simple Applicant state machine, the ReqLeave event is deemed to have occurred if the GID Service User issues a GID_Leave.request service primitive for the Attribute instance associated with that state machine.

12.10.3.3 rJoinIn

For an instance of the combined Applicant/Registrar state machine, the Applicant Only state machine, or the Simple Applicant state machine, the rJoinIn event is deemed to have occurred if a GARP PDU (12.11) is received, and the following conditions are true:

a) The PDU was addressed to the GARP Application address (Table 12-1) of the GARP Application associated with the state machine;
b) The PDU contains a Message (12.11.1) in which the Attribute Type is the type associated with the state machine;
c) The Message contains an Attribute in which the Attribute Event (12.11.2.4) specifies the JoinIn event, and the Attribute Value (12.11.2.6) is equal to the value associated with the state machine.

12.10.3.4 rJoinEmpty

For an instance of the combined Applicant/Registrar state machine, the Applicant Only state machine, or the Simple Applicant state machine, the rJoinEmpty event is deemed to have occurred if a GARP PDU (12.11) is received, and the following conditions are true:

a) The PDU was addressed to the GARP Application address (Table 12-1) of the GARP Application associated with the state machine;
b) The PDU contains a Message (12.11.1) in which the Attribute Type is the type associated with the state machine;
c) The Message contains an Attribute in which the Attribute Event (12.11.2.4) specifies the JoinEmpty event, and the Attribute Value (12.11.2.6) is equal to the value associated with the state machine.

12.10.3.5 rEmpty

For an instance of the combined Applicant/Registrar state machine, the Applicant Only state machine, or the Simple Applicant state machine, the rEmpty event is deemed to have occurred if a GARP PDU (12.11) is received, and the following conditions are true:

a) The PDU was addressed to the GARP Application address (Table 12-1) of the GARP Application associated with the state machine;
b) The PDU contains a Message (12.11.1) in which the Attribute Type is the type associated with the state machine;
c) The Message contains an Attribute in which the Attribute Event (12.11.2.4) specifies the Empty event, and the Attribute Value (12.11.2.6) is equal to the value associated with the state machine.

12.10.3.6 rLeaveIn

For an instance of the combined Applicant/Registrar state machine, the Applicant Only state machine, or the Simple Applicant state machine, the rLeaveIn event is deemed to have occurred if a GARP PDU (12.11) is received, and the following conditions are true:

a) The PDU was addressed to the GARP Application address (Table 12-1) of the GARP Application associated with the state machine;

b) The PDU contains a Message (12.11.1) in which the Attribute Type is the type associated with the state machine;
c) The Message contains an Attribute in which the Attribute Event (12.11.2.4) specifies the LeaveIn event, and the Attribute Value (12.11.2.6) is equal to the value associated with the state machine.

12.10.3.7 rLeaveEmpty

For an instance of the combined Applicant/Registrar state machine, the Applicant Only state machine, or the Simple Applicant state machine, the rLeaveEmpty event is deemed to have occurred if a GARP PDU (12.11) is received, and the following conditions are true:

a) The PDU was addressed to the GARP Application address (Table 12-1) of the GARP Application associated with the state machine;
b) The PDU contains a Message (12.11.1) in which the Attribute Type is the type associated with the state machine;
c) The Message contains an Attribute in which the Attribute Event (12.11.2.4) specifies the LeaveEmpty event, and the Attribute Value (12.11.2.6) is equal to the value associated with the state machine.

12.10.3.8 LeaveAll

For an instance of the combined Applicant/Registrar state machine, the Applicant Only state machine, the Simple Applicant state machine or the Leave All state machine, the LeaveAll event is deemed to have occurred if

a) A GARP PDU (12.11) is received, and the following conditions are all true:
    1) The PDU was addressed to the GARP Application address (Table 12-1) of the GARP Application associated with the state machine;
    2) The PDU contains a Message (12.11.1) in which the Attribute Type is the type associated with the state machine;
    3) The Message contains a LeaveAll Attribute in which the LeaveAll Event (12.11.2.5) is present.

For an instance of the combined Applicant/Registrar state machine, the Applicant Only state machine or the Simple Applicant state machine, the LeaveAll event is deemed to have occurred if

b) The Leave All state machine associated with that state machine performs the sLeaveAll action (12.10.4.5).

NOTE—The LeaveAll state machine operates on a per-Application (not per-Attribute Type) basis, but the LeaveAll message operates on a per-Attribute Type basis. Hence, when the LeaveAll state machine issues a LeaveAll, it must generate a LeaveAll Attribute for each Attribute Type supported by the Application concerned.

12.10.3.9 leavetimer!

For an instance of the combined Applicant/Registrar state machine, the leavetimer! event is deemed to have occurred when the leavetimer associated with that state machine expires.

12.10.3.10 leavealltimer!

For an instance of the combined Applicant/Registrar state machine, the Applicant Only state machine, the Simple Applicant state machine or LeaveAll state machine, the leavealltimer! event is deemed to have occurred when the leavealltimer associated with that state machine expires.

12.10.3.11 transmitPDU!

For an instance of the combined Applicant/Registrar state machine, the Applicant Only state machine, or the Simple Applicant state machine, the transmitPDU! event is deemed to have occurred when the jointimer associated with that state machine expires.

For an instance of the LeaveAll state machine, the transmitPDU! event is deemed to have occurred when the state machine has an opportunity to transmit a LeaveAll message.

12.10.4 Action definitions

Unless stated otherwise in these action definitions, GARP PDU transmission as a result of the operation of a state machine in a Bridge occurs only through the Port associated with that state machine, and only if that Port is in the Forwarding state.

12.10.4.1 -x-

No action is taken.

12.10.4.2 sJ[E, I], sJ[I]

A GARP PDU, formatted as defined in 12.11.1, is transmitted. The PDU shall be formatted such that

a) The PDU contains a Message (12.11.1.2) which carries an Attribute Type (12.11.2.2) that specifies the type associated with the state machine;
b) The Message contains an Attribute (12.11.1.2) that specifies an Attribute Event (12.11.2.4) equal to JoinIn (if the Registrar is in the IN state, or if no Registrar functionality is implemented) or JoinEmpty (if the Registrar is in either the LV or the MT state), and an Attribute Value equal to the value associated with the state machine.

The PDU shall be transmitted using, as the destination MAC Address, the GARP Application address of the GARP Application associated with the state machine.

12.10.4.3 sE

A GARP PDU, formatted as defined in 12.11.1, is transmitted. The PDU shall be formatted such that

a) The PDU contains a Message (12.11.1.2) which carries an Attribute Type (12.11.2.2) that specifies the type associated with the state machine;
b) The Message contains an Attribute (12.11.1.2) that specifies an Attribute Event (12.11.2.4) equal to Empty, and an Attribute Value equal to the value associated with the state machine.

The PDU shall be transmitted using, as the destination MAC Address, the GARP Application address of the GARP Application associated with the state machine.

12.10.4.4 sLE

A GARP PDU, formatted as defined in 12.11.1, is transmitted. The PDU shall be formatted such that

a) The PDU contains a Message (12.11.1.2) which carries an Attribute Type (12.11.2.2) that specifies the type associated with the state machine;
b) The Message contains an Attribute (12.11.1.2) that specifies an Attribute Event (12.11.2.4) equal to LeaveEmpty, and an Attribute Value equal to the value associated with the state machine.

The PDU shall be transmitted using, as the destination MAC Address, the GARP Application address of the GARP Application associated with the state machine.

12.10.4.5 sLeaveAll

A GARP PDU, formatted as defined in 12.11.1, is transmitted. The PDU shall be formatted such that, for each Attribute Type associated with the GARP Application

a) The PDU contains a Message (12.11.1.2) which carries an Attribute Type (12.11.2.2) that specifies the Attribute Type concerned;
b) The Message contains a LeaveAll Attribute (12.11.1.2).

The PDU shall be transmitted using, as the destination MAC Address, the GARP Application address of the GARP Application associated with the state machine.

The sLeaveAll action also gives rise to a LeaveAll event against all instances of the combined Applicant/Registrar state machine, the Applicant Only state machine and the Simple Applicant state machine associated with the GARP Application.

12.10.4.6 Start leavetimer

Causes leavetimer to be started, in accordance with the definition of the timer in 12.10.2.2.

12.10.4.7 Stop leavetimer

Causes leavetimer to be stopped.

12.10.4.8 Start leavealltimer

Causes leavealltimer to be started, in accordance with the definition of the timer in 12.10.2.3.

12.10.4.9 IndJoin

When an instance of the Registrar state machine makes a transition from the LV or MT state to the IN state, the IndJoin action causes a GID_Join.indication primitive to be issued to the GID Service User, indicating the Attribute instance corresponding to the state machine concerned.

12.10.4.10 IndLeave

When an instance of the Registrar state machine makes a transition from the LV state to the MT state, the IndLeave action causes a GID_Leave.indication primitive to be issued to the GID Service User, indicating the Attribute instance corresponding to the state machine concerned.

12.10.4.11 Failure to register

Each GARP Participant maintains a count of the number of times that it has received a registration request, but has failed to register the attribute concerned due to lack of space in the Filtering Database to record the registration. The value of this count may be examined by management.

NOTE—Further action to be taken on such events is a matter for implementation choice.

12.11 Structure and encoding of GARP Protocol Data Units

This subclause describes the generic structure and encoding of the GARP Protocol Data Units (GARP PDUs) exchanged between all GARP Participants. The structure and encoding of elements that are specific to the operation of the GARP Applications are defined by the Applications themselves.

Each GARP PDU identifies the GARP Application by which it was generated, and to which it is being transmitted. Bridges that receive GARP PDUs identified as belonging to a GARP Application which they do not support shall forward such PDUs on all other Ports that are in a Forwarding state.

NOTE 1—If GARP is used to support an Application that can operate in any GIP Context other than 0 (the base Spanning Tree), then the means whereby that context is identified in protocol exchanges is defined by the GARP Application concerned.

Each GARP PDU carries one or more GARP messages, each of which identify a GARP event (e.g., Join, Leave, LeaveAll) and the attribute class(es) and value(s) to which that event applies. A given GARP Participant shall process GARP PDUs in the order in which they are received, and within a given GARP PDU, shall process GARP Messages in the order in which they were put into a Data Link Service Data Unit (DLSDU).

NOTE 2—Any messages generated as a consequence of state machine responses to an sLeaveAll action and its associated LeaveAll events will be put into the DLSDU after the LeaveAll message(s), or into a later DLSDU.

12.11.1 Structure

12.11.1.1 Transmission and representation of octets

All GARP PDUs consist of an integral number of words, numbered starting from 1 and increasing in the order that they are put into a Data Link Service Data Unit (DLSDU). The bits in each octet are numbered from 1 to 8, where 1 is the low-order bit.

When consecutive octets are used to represent a binary number, the lower octet number has the most significant value.

When the encoding of (an element of) a GARP PDU is represented using a diagram in this clause, the following representations are used:

a) Octet 1 is shown towards the top of the page, higher numbered octets being towards the bottom;
b) Where more than one octet appears on a given line, octets are shown with the lowest numbered octet to the left, higher numbered octets being to the right;
c) Within an octet, bits are shown with bit 8 to the left and bit 1 to the right.

12.11.1.2 Structure definition

A Protocol Identifier shall be encoded in the initial octets of all GARP PDUs. This standard reserves a single Protocol Identifier value to identify the Generic Attribute Registration Protocol. GARP PDUs operating the Protocol specified in this Clause carry this reserved Protocol Identifier value, and shall have the following structure:

a) The first two octets contain the *Protocol Identifier* value.
b) Following the Protocol Identifier are one or more *Messages*. The last element in the PDU is an *End Mark*.
c) Each Message consists of an *Attribute Type* and an *Attribute List*, in that order.

d) An Attribute List consists of one or more *Attribute*s. The last element in the Attribute List is an End Mark.

e) An Attribute consists of an *Attribute Length*, an *Attribute Event*, and an *Attribute Value*, in that order. In the case where the Attribute Event is "LeaveAll," the Attribute Value is omitted.

The following BNF productions give the formal description of the GARP PDU structure:

GARP PDU ::= Protocol ID, Message {, Message}, End Mark
Protocol ID SHORT ::= 1
Message ::= Attribute Type, Attribute List
Attribute Type BYTE ::= *Defined by the specific GARP Application*
Attribute List ::= Attribute {,Attribute}, End Mark
Attribute ::= Ordinary Attribute | LeaveAll Attribute
Ordinary Attribute ::= Attribute Length, Attribute Event, Attribute Value
LeaveAll Attribute ::= Attribute Length, LeaveAll Event
Attribute Length BYTE ::= 2-255
Attribute Event BYTE ::= JoinEmpty | JoinIn | LeaveEmpty | LeaveIn | Empty
LeaveAll Event BYTE ::= LeaveAll
Attribute Value ::= *Defined by the specific GARP Application*
End Mark ::= 0x00 | *End of PDU*
LeaveAll ::= 0
JoinEmpty ::= 1
JoinIn ::= 2
LeaveEmpty ::= 3
LeaveIn ::= 4
Empty ::= 5

The parameters carried in GARP PDUs, as identified in this structure definition, shall be encoded as specified in 12.11.2.

Figure 12-6 illustrates the structure of the GARP PDU and its components.

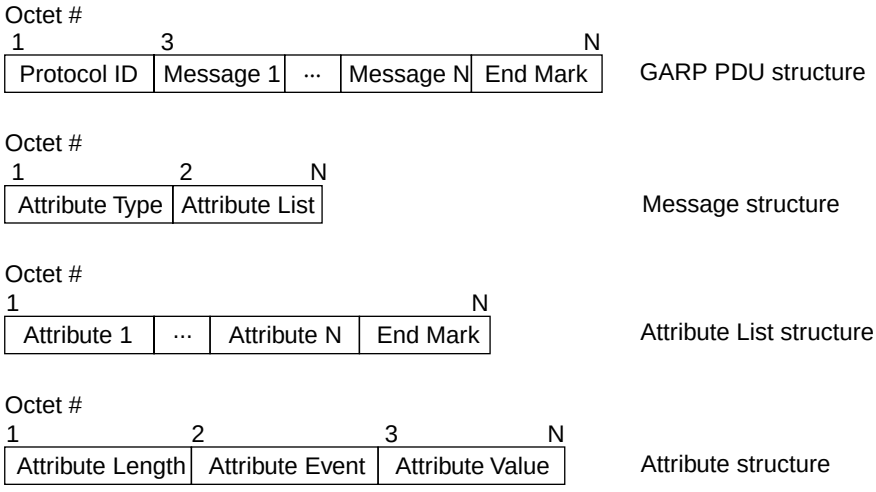


Figure 12-6—Format of the major components of a GARP PDU

12.11.2 Encoding of GARP PDU parameters

12.11.2.1 Encoding of Protocol Identifier

A Protocol Identifier shall be encoded in two octets, taken to represent an unsigned binary number. It takes the hexadecimal value 0x0001, which identifies the GARP protocol as defined in this clause.

12.11.2.2 Encoding of Attribute Type

An Attribute Type shall be encoded as a single octet, taken to represent an unsigned binary number. The Attribute Type identifies the type of Attribute to which the message applies. The range of values that can be taken by the Attribute Type, and the meanings of those values, are defined by the Application concerned. The value 0 is reserved, and shall not be used as an Attribute Type by any GARP Application. GARP Applications may otherwise allocate meanings to any set of values of Attribute Type in the range 1 through 255.

12.11.2.3 Encoding of Attribute Length

An Attribute Length shall be encoded as a single octet, taken to represent an unsigned binary number, equal to the number of octets occupied by an Attribute, inclusive of the Attribute Length field. Valid values of Attribute Length are in the range 2 through 255.

Further values of Attribute Length are reserved and shall not be used.

12.11.2.4 Encoding of Attribute Event

An Attribute Event shall be encoded as a single octet, taken to represent an unsigned binary number. The permitted values and meanings of the Attribute Event are as follows:

1: JoinEmpty operator
2: JoinIn operator
3: LeaveEmpty operator
4: LeaveIn operator
5: Empty operator

Further values of Attribute Event are reserved.

The Attribute Event is interpreted on receipt as a GID event to be applied to the state machine for the Attribute defined by the Attribute Type and Attribute Value fields.

12.11.2.5 Encoding of LeaveAll Event

A LeaveAll Event shall be encoded as a single octet, taken to represent an unsigned binary number. The permitted values and meanings of LeaveAll Event are as follows:

0: LeaveAll operator

Further values of LeaveAll Event are reserved.

The LeaveAll Event is interpreted on receipt as a GID Leave All event to be applied to the state machines for all Attributes of the type defined by the Attribute Type field.

12.11.2.6 Encoding of Attribute Value

An Attribute Value is encoded in N octets, in accordance with the specification for the Attribute Type, as defined by the GARP Application concerned.

12.11.2.7 Encoding of End Mark

An End Mark shall be encoded as a single octet, taken to represent the unsigned binary number. It takes the value 0.

Further values of End Mark are reserved and shall not be used.

NOTE—As defined by the GARP PDU structure definition in 12.11.1, if the end of the GARP PDU is encountered, this is taken to be an End Mark from the point of view of processing the PDU contents.

12.11.3 Packing and parsing GARP PDUs

The use of the End Mark (12.11.2.7) to signal the end of an Attribute List and the end of a GARP PDU, and the fact that the (physical) end of the PDU is interpreted as an End Mark, simplifies the requirements both for packing information into GARP PDUs and for correctly interpreting that information on receipt.

12.11.3.1 Packing

Successive Messages are packed into the GARP PDU, and within each Message, successive Attributes are packed into each Message, until the end of the PDU is encountered or there are no more attributes to pack at that time. The following cases can occur:

a) The PDU has sufficient room for all the Attributes that require to be transmitted at that time to be packed. In this case, the PDU is transmitted, and subsequent PDUs are transmitted when there are further Attributes to transmit;
b) The PDU has exactly enough room for the first N Attributes that require to be transmitted at that time to be packed. In this case, the PDU is transmitted, and the next N Attributes are encoded in a subsequent PDU;
c) The PDU has enough room for the first N Attributes that require to be transmitted at that time to be packed, but the remaining space in the PDU is too small for Attribute N+1, so the last few octets of the PDU carry a partial encoding of Attribute N+1. In this case, the PDU can be transmitted as it is, and Attribute N+1 and its successors are encoded in full in a subsequent PDU.

12.11.3.2 Parsing

Successive Messages, and within each Message, successive Attributes, are unpacked from the PDU. If this process terminates because the end of the PDU is reached, then the end of the PDU is taken to signal termination both of the current Attribute List and the overall PDU. Two cases can occur:

a) The last Attribute to be unpacked was complete. In this case, the Attribute is processed normally, and processing of the PDU terminates;
b) The last Attribute to be unpacked was incomplete. In this case, the partial Attribute is discarded, and processing of the PDU terminates.

12.11.3.3 Discarding unrecognized information

In order to allow backward compatibility with previous versions of a given GARP Application, the following procedure is adopted when unrecognized elements within a received GARP PDU are encountered:

a) If a Message is encountered in which the Attribute Type is not recognized, then that Message is discarded. This is achieved by discarding the successive Attributes in the Attribute List until either an End Mark or the end of the PDU is reached. If an End Mark is reached, processing continues with the next Message.

b) If an Attribute is encountered in which the Attribute Event is not recognized for the Attribute Type concerned, then the Attribute is discarded and processing continues with the next Attribute or Message if the end of the PDU has not been reached.

12.12 Timer values, granularity and relationships

12.12.1 Timer values

The default timer values used in the GARP protocol are defined in Table 12-10. The values used for the GARP timers may be modified on a per-Port basis by means of the management functionality defined in Clause 14.

Table 12-10—GARP timer parameter values

Parameter	Value (centiseconds)
JoinTime	20
LeaveTime	60
LeaveAllTime	1000

NOTE—The default values for the GARP timers are independent of media access method or data rate. This is a deliberate choice, made in the interests of maximizing the “plug and play” characteristics of the protocol.

12.12.2 Timer resolution

Implementation of the timers for GARP shall be based on a timer resolution of 5 centiseconds or less.

12.12.3 Timing relationships

GARP protocol *correctness* does not depend critically on timing relationships; however, the protocol operates more efficiently, and with less likelihood of unwanted de-registrations, if the following relationships are maintained between the protocol timers operating in state machines that exchange GARP PDUs on the same LAN segment:

a) JoinTime should be chosen such that at least two JoinTimes can occur within the value of LeaveTime being used on the LAN segment. This ensures that after a Leave or LeaveAll message has been issued, the Applicants can re-Join before a further Leave is issued;

b) LeaveAllTime should be larger than the value of LeaveTime being used on the LAN segment. In order to minimize the volume of re-joining traffic generated following a Leave All, the value chosen for LeaveAllTime should be large relative to LeaveTime.

These relationships are illustrated in Figure 12-7. The time intervals labeled A (Leave Time minus two Join Times) and B (Leave All Period minus Leave Time) should all be positive and non-zero in value for the efficient operation of GARP. The time parameter values specified in Table 12-10 have been chosen in order to ensure that these timer relationships are maintained.

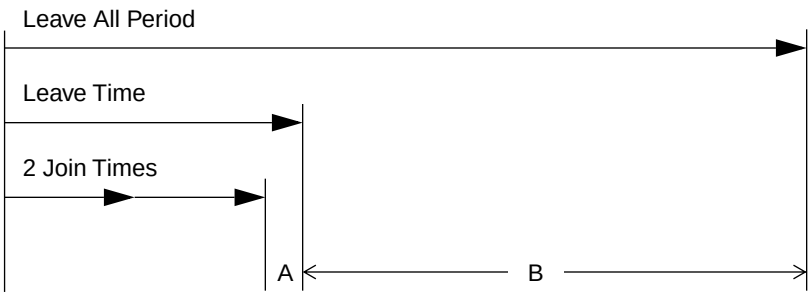


Figure 12-7—GARP timing relationships

12.13 Procedural model

Clause 13 contains an example “C” code description of GARP.

12.14 Interoperability considerations

Correct operation of the GARP protocol for a given GARP application requires that protocol exchanges among a given set of communicating GARP Participants maintain sequentiality; i.e., that Participant A cannot receive GARP PDU B (generated as a consequence of Participant B receiving GARP PDU A) before Participant A has received GARP PDU A. In circumstances where the Participants concerned are all attached to the same LAN segment, such sequentiality is ensured. However, if a set of GARP Participants communicates via an intervening Bridge that does not implement that GARP application (or does not implement GARP at all), the sequentiality constraints expressed in 7.7.3 are insufficient to guarantee the correct operation of the GARP protocol. In order for the correct sequencing of PDUs to be maintained through such a Bridge, the following constraint must be met:

> If GARP PDU A is received on Port X, and is due to be forwarded on Ports Y and Z, and subsequent to being forwarded on Y, GARP PDU B is received on Port Y for forwarding on Port Z, then forwarding of B cannot precede A on Port Z.

NOTE—This expresses a stronger sequencing constraint for multicast frames than is stated in 7.7.3, but a weaker constraint than was required for conformance to ISO/IEC 10038: 1993.

The consequence of failure to meet this constraint is that the users of a given GARP application may experience an increased incidence of loss of registration. Therefore, it is inadvisable to construct LAN configurations involving forwarding of GARP PDUs through intervening Bridges if those Bridges do not meet the constraint expressed above.

13. Example “C” code implementation of GARP

13.1 Purpose

This section contains an example implementation of the GARP protocol and state machinery required in order to support GMRP (Clause 10) and other applications that use GARP (Clause 12). This “C” code description is included in order to demonstrate the structure of GARP and its components, and to show that a reasonably low-cost implementation can be constructed. The implementation has been designed with the intent of maximizing clarity and generality, not for compactness.

The example implementation is shown in the following subclauses:

a) Header files for the GARP application independent code (13.2);
b) The GARP application independent code (13.3).

The separation shown between the application dependent (Clause 11) and application independent (this clause) aspects of the implementation gives a clear illustration of what is involved in implementing additional applications using the same basic GARP state machines. The code is intended to be largely self-documenting, by means of in-line comments.

NOTE—The code shown is based on the functionality and state machines defined for the full GARP Participant; code for the Applicant Only Participant and Simple Participant would involve simplification of the functionality shown, as discussed in 12.7.7 and 12.7.8.

13.2 GARP application independent header files

13.2.1 garp.h

```
/* garp.h */
#ifndef garp_h__
#define garp_h__

#include “sys.h”

/*****************************************************************************
 * GARP : GENERIC ATTRIBUTE REGISTRATION PROTOCOL : COMMON APPLICATION ELEMENTS
 *****************************************************************************
 */

typedef struct /* Garp */
{  /*
    * Each GARP, i.e., each instance of an application that uses the GARP
    * protocol, is represented as a struct or control block with common initial
    * fields.  These comprise pointers to application-specific functions that
    * are by the GID and GIP components to signal protocol events to the
    * application, and other controls common to all applications. The pointers
    * include a pointer to the instances of GID (one per port) for the application,
    * and to GIP (one per application). The signaling functions include the
    * addition and removal of ports, which the application should use to
    * initialize port attributes with any management state required.
    */

    int      process_id;

    void     *gid;

    unsigned *gip;

    unsigned  max_gid_index;

    unsigned  last_gid_used;

    void(*join_indication_fn)( void *, void *my_port, unsigned joining_gid_index);
    void(*leave_indication_fn)(void *, void *gid,
                                       unsigned leaving_gid_index);
    void(*join_propagated_fn)( void *, void *gid,
                                       unsigned joining_gid_index);
    void(*leave_propagated_fn)(void *, void *gid,
                                       unsigned leaving_gid_index);

    void(*transmit_fn)(        void *, void *gid);
    void    (*receive_fn)(         void *, void *gid, Pdu *pdu);

    void    (*added_port_fn)(      void *, int port_no);
    void    (*removed_port_fn)(    void *, int port_no);

} Garp;

#endif /* garp_h__ */
```

13.2.2 gid.h

```
/* gid.h */
#ifndef gid_h__
#define gid_h__
```

```
#include “sys.h”
#include “garp.h”


/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : OVERVIEW
 *****************************************************************************
 *
 * In this reference implementation there is a single instance of GID, the
 * GARP Information Distribution protocol per application instance per
 * physical or logical port in the system [the base spanning tree operates
 * over physical ports; VLAN registration operates over physical ports to
 * offer one (or zero) logical ports per physical port per VLAN; an instance
 * of the Multicast registration operates either over physical ports or over
 * the logical ports of a VLAN, depending on whether the registration is within
 * the scope of a VLAN or not].
 *
 * This single GID instance operates a number of GID machines, one for each
 * attribute of interest to the application instance (an attribute is the
 * smallest unit that can be registered by GARP, e.g., a single multicast
 * address — join and leave indications occur for individual attributes).
 * GID knows nothing of the semantics of individual attributes — it is only
 * interested in the state of the GID machine for each attribute and the
 * GID events which change that state and give rise to further protocol
 * events. Each attribute is identified by its GID index, which in this
 * implementation is a simple index into an array of GID machines. An
 * attribute’s GID index is the same for every port belonging to an application
 * instance.
 *
 * The point of operating a single GID instance instead of completely separate
 * machines is to allow there to be a single set of GID timers per port, and
 * to facilitate easy packing of messages for individual attributes into a
 * single PDU.
 */

/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : GID MACHINES
 *****************************************************************************
 */

typedef struct /* Gid_machine */
{
   /* The GID state of each attribute on each port is held in a GID ‘machine’
    * which comprises the Applicant and Registrar states for the port,
    * including control modifiers for the states.
    *
    * The GID machine and its internal representation of GID states is not
    * accessed directly: this struct is defined here to allow the GID
    * Control Block (which is accessed externally) to be defined below.
    */

    unsigned applicant : 5;  /* : Applicant_states */

    unsigned registrar : 5;  /* : Registrar_states */

} Gid_machine;

/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : MANAGEMENT STATES
 *****************************************************************************
 *
 * Implementation independent representations of the GID states of a single
 * attribute including management controls, following the standard state
 * machine specification as follows:
 *
```



```
 *  Applicant : Major state :           Very anxious, Anxious, Quiet, Leaving
 *
 *              Current participation : Active, Passive
 *
 *               Management controls :   Normal, No protocol
 *
 *  Registrar : Major state :           In, Leave, Empty
 *
 *               Management controls :   Normal registration,
 *                                     Registration fixed,
 *                                       Registration forbidden
 *
 * with a struct defined to return the current state (caller supplies by
 * reference).
 */

typedef enum {Very_anxious, Anxious, Quiet, Leaving} Gid_applicant_state;
typedef enum {Normal, No_protocol} Gid_applicant_mgt;
typedef enum {In, Leave, Empty}                      Gid_registrar_state ;
typedef enum {Normal_registration, Registration_fixed, Registration_forbidden
            }                                        Gid_registrar_mgt;

typedef struct /* Gid_states */
{
    unsigned applicant_state : 2;

    unsigned applicant_mgt   : 1;

    unsigned registrar_state : 2;

    unsigned registrar_mgt   : 2;

} Gid_states;

/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : PROTOCOL EVENTS
 *****************************************************************************
 *
 * A complete set of events is specified to allow events to be passed around
 * without rewriting. The other end of the spectrum would have been one
 * set of events for received messages, a set of receive indications, a
 * set of user requests, etc., and separate transition tables for each.
 *
 * Event definitions are ordered in an attempt not to waste space in
 * transition tables and to pack switch cases together.
 */
typedef enum /* Gid_event */
{

    Gid_null, Gid_rcv_leaveempty, Gid_rcv_leavein, Gid_rcv_empty,
    Gid_rcv_joinempty, Gid_rcv_joinin,
    Gid_join, Gid_leave,
    Gid_normal_operation, Gid_no_protocol,
    Gid_normal_registration, Gid_fix_registration, Gid_forbid_registration,
    Gid_rcv_leaveall, Gid_rcv_leaveall_range,
    Gid_tx_leaveempty, Gid_tx_leavein, Gid_tx_empty, Gid_tx_joinempty,
    Gid_tx_joinin, Gid_tx_leaveall, Gid_tx_leaveall_range
} Gid_event;

enum {Number_of_gid_rcv_events  = (Gid_rcv_joinin + 1),
      Number_of_gid_req_events  = 2,
      Number_of_gid_amgt_events = 2,
      Number_of_gid_rmgt_events = 3,
      Number_of_gid_tx_events   = 7};
```

```
/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : PROTOCOL TIMER VALUES
 *****************************************************************************
 *
 * Describe goals of timers, subsecond response times, etc.
 */
typedef int milliseconds;

enum {Gid_default_join_time     =   200}; /* milliseconds */
enum {Gid_default_leave_time    =   600}; /* milliseconds */
enum {Gid_default_hold_time     =   100}; /* milliseconds */
enum {Gid_leaveall_count        =     4};
enum {Gid_default_leaveall_time = 10000}; /* miliiseconds */

/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : PROTOCOL INSTANCES
 *****************************************************************************
 */

typedef struct /* Gid */
{
    /* Each instance of GID is represented by one of these control blocks.
     * There is a single instance of GID per port for each GARP application.
     * The control blocks are linked together through the next_in_port_ring
     * pointer to form a complete ring.
     *
     * Each control block contains a pointer to the GARP control block
     * representing the application instance that specifies the application's
     * join, leave, transmit, and receive functions, and its process identifier
     * (for identifying the application instance to the rest of the system
     * including timer functions).
     *
     * The port number associated with this instance of GID is specified.
     * The GID control blocks for all of the ports of an application instance
     * are linked together through the next_in_port_ring pointer to form a
     * complete ring. Ports which are 'connected,' e.g., are all in a spanning
     * tree forwarding state, are linked through the next_in_connected_ring
     * pointer. The is_connected flag is also set for these ports, and it
     * handles the case of a single connected port [is_connected is true (set)]
     * only if is_enabled is also true. The GID control block definition is
     * shared to allow GIP to traverse these fields.
     *
     * GID processing for the port as a whole may be enabled or disabled : the
     * current state is recorded here in case received PDUs or other events
     * are pending in the system.
     *
     * Standard GID operates as if the GID entity were connected to a shared
     * medium. GARP information may be transported more quickly if the link
     * is known to be point-to-point. Specifically received Leave messages
     * can give rise to immediate Leave indications, without the need to
     * solicit further Joins from other potential members attached to the shared
     * medium.
     *
     * The control block provides a 'scratchpad' for recording actions
     * arising from GID machine processing during this invocation of GID
     * ('invocation' meaning when GARP runs, e.g., when a received packet is
     * being processed). After each invocation gid_do_actions() is called
     * to schedule a transmission immediately, start the join timer (if
     * not already started), or start the leave timer (again, if not already
     * started) depending on the setting of cschedule_tx_now,
     * cstart_join_timer, and cstart_leave_timer, and whether these timers
     * have already been started (or immediate scheduling requested) as
     * recorded in tx_now_scheduled, join_timer_running, and
     * leave_timer_running. If hold_tx is true scheduling and starting timers
     * are held pending expiry of the hold timer.
```



```
 *
 * Timeout values for the join, leave, hold, and leaveall timers are
 * recorded here to allow them to be managed according to the media type
 * and speed, and whether the port attaches to point-to-point switch-to-
 * switch link, to shared media, or through a non-GARP aware switch to
 * switches of possibly varying link speeds.
 *
 * The leave time for an individual GID machine comprises three to four
 * expirations of the leave_timeout_4. This gives sufficient granularity
 * for the leave timer to protect against premature expiration (while
 * another GID participant may be preparing to send a join) without making
 * the worst-case leave time overlong. It is also sufficiently coarse to
 * allow the leave timer state for each entry to be easily stored within
 * the Registrar state.
 *
 * The leaveall timeout is implemented as leaveall_countdown expirations
 * of leaveall_timeout_n. This supports suppression of Leaveall generation
 * by this machine when a Leaveall has been received, without requiring
 * the operating system to support cancelling or restarting of timers.
 * When leaveall_countdown reaches zero, the join timer is started (if not
 * already running). Whenever the join timer expires, the application's
 * transmit function is invoked, which will lead to a call to gid_next_tx,
 * which in turn will return Gid_tx_leaveall. This ensures that Leavealls
 * are transmitted at the beginning of PDUs that contain the results of
 * the local Leaveall processing - such as immediate rejoins. This is a
 * good idea for protocol robustness in the face of receiver packet loss
 * - the rejoins are only lost if the Leaveall itself is lost - and it
 * minimizes the number of PDUs sent.
 *
 * This GID control block points to an array of GID machines allocated
 * when this instance of GID is created. It supports more GID attributes
 * than can be packed by an application formatter into a single PDU
 * (required for most simple encodings of 4096 VLANs). The tx_pending
 * flag indicates that some of the GID machines between last_transmitted
 * and last_to_transmit indices probably have messages to send. The
 * application state prior to message generation of the last_transmitted
 * machine is stored in the GID machine indexed by untransmit_machine -
 * space for this is reserved at the very end of the GID machine array,
 * which is one larger than would otherwise be required to store the
 * maximum number of attributes specified at GID creation. This allows
 * the implementation of a simple untransmit function, like the C
 * library function ungetc().
 */

Garp    *application;

int          port_no;

void        *next_in_port_ring;

void        *next_in_connected_ring;

unsigned     is_enabled          : 1;

unsigned     is_connected        : 1;

unsigned     is_point_to_point   : 1;

unsigned     cschedule_tx_now    : 1;

unsigned     cstart_join_timer   : 1;

unsigned     cstart_leave_timer  : 1;

unsigned     tx_now_scheduled    : 1;
```

```
    unsigned     join_timer_running  : 1;
    unsigned     leave_timer_running : 1;
    unsigned     hold_tx             : 1;
    unsigned     tx_pending          : 1;
    int          join_timeout;
    int          leave_timeout_4;
    int          hold_timeout;
    int          leaveall_countdown;
    int          leaveall_timeout_n;
    Gid_machine *machines;
    unsigned     last_transmitted;
    unsigned     last_to_transmit;
    unsigned     untransmit_machine;
} Gid;

/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : CREATION, DESTRUCTION, ETC.
 *****************************************************************************
 */

extern Boolean gid_create_port(Garp *application, int port_no);
   /*
    * Creates a new instance of GID, allocating space for GID machines as
    * required by the application, adding the port to the ring of ports
    * for the application, and signaling the application that the new port
    * has been created.
    *
    * On creation each GID machine is set to operate as Normal or with
    * No_protocol!!
    *
    * The port is enabled when created, but not connected (see GIP).
    */

extern void gid_destroy_port(Garp *application, int port_no);
   /*
    * Destroys the instance of GID, disconnecting the port if it is still
    * connected (causing leaves to propagate as required), then causing
    * leave indications for this port as required, finally releasing all
    * allocated space, signaling the application that the port has been
    * removed.
    */

/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : USEFUL FUNCTIONS
 *****************************************************************************
 */

extern Boolean gid_find_port(Gid *first_port, int port_no, void **gid);
   /*
    * Finds the GID instance for port number port_no.
    */
```



```
extern Gid *gid_next_port(Gid *this_port);
   /*
    * Finds the next port in the ring of ports for this application.
    */

extern Boolean gid_find_unused(Garp *application, unsigned from_index,
                                                  unsigned *found_index);
   /*
    * Finds an unused GID machine (i.e., one with an Empty registrar and a
    * Very Anxious Observer applicant) starting the search at GID index
    * from_index, and searching to gid_last_used.
    */


/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : MGT
 *****************************************************************************
 */

extern void gid_read_attribute_state(Gid *my_port, unsigned index,
                                     Gid_states *state);
   /*
    *
    */

extern void gid_manage_attribute(Gid *my_port, unsigned index,
                                 Gid_event directive);
   /*
    * Changes the attribute's management state on my_port. The directive
    * can be Gid_normal_operation, Gid_no_protocol, Gid_normal_registration,
    * Gid_fix_registration, or Gid_forbid_registration. If the change in
    * management state causes a leave indication, this is sent to the user
    * and propagated to other ports.
    */

/*
 * Further management functions, including disabling and enabling GID ports,
 * are to be provided. A significant part of the purpose of this reference
 * implementation is to facilitate the unambiguous specification of such
 * management operations.
 */

/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : EVENT PROCESSSING
 *****************************************************************************
 */

extern void gid_rcv_msg(Gid *my_port, unsigned gid_index, Gid_event msg);
   /*
    * Only for Gid_rcv_leave, Gid_rcv_empty, Gid_rcv_joinempty, Gid_rcv_joinin.
    * See gid_rcv_leaveall for Gid_rcv_leaveall, Gid_rcv_leaveall_range.
    *
    * Joinin and JoinEmpty may cause Join indications; this function calls
    * GIP to propagate these.
    *
    * On a shared medium, Leave and Empty will not give rise to indications
    * immediately. However, this routine does test for and propagate
    * Leave indications so that it can be used unchanged with a point-to-point
    * protocol enhancement.
    */

extern void gid_join_request(Gid *my_port, unsigned gid_index);
   /*
    * can be called multiple times with no ill effect.
```

```
    *
    */

extern void gid_leave_request(Gid *my_port, unsigned gid_index);
   /*
    * can be called multiple times with no ill effect.
    *
    */

extern void gid_rcv_leaveall(Gid *my_port);
   /*
    */

extern Boolean gid_registered_here(Gid *my_port, unsigned gid_index);
   /*
    * Returns True if the Registrar is not Empty, or if Registration is fixed.
    */

/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : TIMER PROCESSING
 *****************************************************************************
 */

void gid_do_actions(Gid *my_port);
   /*
    * Carries out 'scratchpad' actions accumulated in this invocation of GID,
    * and outstanding 'immediate' transmissions and join timer starts that
    * have been delayed by the operation of the hold timer.
    */

/*
 * Timer expiration routines for timers started by GID, mainly by
 * gid_do_actions(). join_timer_expired() is scheduled immediately by
 * gid_do_actions().
 */
extern void gid_leave_timer_expired(   void *application, int port_no);
extern void gid_join_timer_expired(    void *application, int port_no);
extern void gid_leaveall_timer_expired(void *application, int port_no);
extern void gid_hold_timer_expired(    void *application, int port_no);


/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : TRANSMIT PROCESSSING
 *****************************************************************************
 */

extern Gid_event gid_next_tx(Gid *my_port, unsigned *gid_index);
   /*
    * Scan the GID machines for messsages that require transmission.
    * If there is no message to transmit return Gid_null; otherwise,
    * return the message as a Gid_event.
    *
    * If message transmission is currently held [pending expiry of a hold
    * timer and a call to gid_release_tx()], this function will return Gid_null
    * so it may be safely called if that is convenient.
    *
    * Supports sets of GID machines that can generate more messages than
    * can fit in a single application PDU. This allows this implementation
    * to support, for example, GARP registration for all 4096 VLAN identifiers.
    *
    * To support the application's packing of messages into a single PDU
    * without the detailed knowledge of frame format and message encodings
    * required to tell whether a message will fit, and to avoid the application
    * having to make two calls to GID for every message - one to get the
    * message and another to say it has been taken - this routine supports the
```



```
    * gid_untx() function. This is conceptually similar to the C library
    * ungetc() function. It restores the applicant state of the last machine
    * from which a message was taken, and establishes that machine as the next
    * one from which a message should be taken - effectively pushing the
    * message back into the set of GID machines. It should only be used
    * within a single invocation of GID; otherwise, intervening events and their
    * consequent state changes may be lost with unspecified results.
    */

extern void gid_untx(Gid *my_port);
   /*
    * See description above.
    */

#endif /* gid_h__ */
```

13.2.3 gidtt.h

```
/* gidtt.h */
#ifndef gidtt_h__
#define gidtt_h__

#include “gid.h”


/*****************************************************************************
 * GIDTT : GARP INFORMATION DISTRIBUTION PROTOCOL : TRANSITION TABLES
 *****************************************************************************
 */

extern Gid_event gidtt_event(Gid         *my_port,
                             Gid_machine *machine,
                             Gid_event    event);

extern Gid_event gidtt_tx(Gid         *my_port,
                          Gid_machine *machine);

extern Gid_event gidtt_leave_timer_expiry(Gid         *my_port,
                                          Gid_machine *machine);

extern Boolean gidtt_in(Gid_machine *machine);
   /*
    * Returns True if the Registrar is in, or if registration is fixed.
    */

extern Boolean gidtt_machine_active(Gid_machine *machine);
   /*
    * Returns False iff the Registrar is Normal registration, Empty, and the
    * Application is Normal membership, Very Anxious Observer.
    */

extern void gidtt_states(Gid_machine *machine,
                         Gid_states  *state);
   /*
    * Reports the the GID machine state : Gid_applicant_state,
    * Gid_applicant_mgt, Gid_registrar_state, Gid_registrar_mgt.
    */

#endif /* gidtt_h__ */
```

13.2.4 gip.h

```
/* gip.h */
#ifndef gip_h__
#define gip_h__

#include “gid.h”

/****************************************************************************
 * GIP : GARP INFORMATION PROPAGATION : CREATION, DESTRUCTION
 ****************************************************************************
 */

extern Boolean gip_create_gip(unsigned max_attributes, unsigned **gip);
   /*
    * Creates a new instance of GIP, allocating space for propagation counts
    * for up to max_attributes.
    *
    * Returns True if the creation succeeded together with a pointer to the
    * GIP information. This pointer is passed to gid_create_port() for ports
    * using this instance of GIP and is saved along with GID information.
    */

extern void gip_destroy_gip(void *gip);
   /*
    * Destroys the instance of GIP, releasing previously allocated space.
    */

/****************************************************************************
 * GIP : GARP INFORMATION PROPAGATION : PROPAGATION FUNCTIONS
 ****************************************************************************
 */

extern void gip_connect_port(Garp *application, int port_no);
   /*
    * Finds the port, checks that it is not already connected, and connects
    * it into the GIP propagation ring that uses GIP field(s) in GID control
    * blocks to link the source port to the ports to which the information is
    * to be propagated.
    *
    * Propagates joins from and to the other already connected ports as
    * necessary.
    */

extern void gip_disconnect_port(Garp *application, int port_no);
   /*
    * Checks to ensure that the port is connected, and then disconnects it from
    * the GIP propagation ring. Propagates leaves to the other ports that
    * remain in the ring and causes leaves to my_port as necessary.
    */

extern void gip_propagate_join(Gid *my_port, unsigned index);
   /*
    * Propagates a join indication for a single attribute (identified
    * by a combination of its attribute class and index) from my_port to other
    * ports, causing join requests to those other ports if required.
    *
    * GIP maintains a joined membership count for the connected ports for each
    * attribute (in a given context) so that leaves are not caused when joins
    * from other ports would maintain membership.
    *
    * Because this count is maintained by ‘dead-reckoning’ it is important
    * that this function only be called when there is a change indication for
    * the source port and index.
```



```
    */

extern void gip_propagate_leave(Gid *my_port, unsigned index);
   /*
    * Propagates a leave indication for a single attribute, causing leave
    * requests to those other ports if required.
    *
    * See the comments for gip_propagate_join() before reading further.
    * This function decrements the 'dead-reckoning' membership count.
    */

extern Boolean gip_propagates_to(Gid *my_port, unsigned index);
   /*
    * True if any other port is propagating the attribute associated with index
    * to my_port.
    */

extern void gip_do_actions(Gid *my_port);
   /*
    * Calls GID to carry out GID 'scratchpad' actions accumulated during this
    * invocation of GARP for all the ports in the GIP propagation list,
    * including the source port.
    */

#endif /* gip_h__ */
```

13.2.5 prw.h

```
/* prw.h */
#ifndef prw_h__
#define prw_h__

#include "sys.h"

/*****************************************************************************
 * PRW : PDU READ WRITE ACCESS
 *****************************************************************************
 */

typedef struct
{
    Pdu *pdu;

    int     record_id;

    int     record_len;

} Gpdu;

extern void    prw_rdrec_init(Pdu *pdu, Gpdu *gpdu);

extern Boolean prw_rdrec(Gpdu gpdu);

extern Boolean prw_skiprec(Gpdu gpdu);


#define Garp_terminating_record_id 0x0000

#endif /* prw_h__ */
```

13.2.6 sys.h

```
/* sys.h */
#ifndef sys_h__
#define sys_h__

#include <stddef.h>

/*****************************************************************************
 * SYS : SYSTEM SUPPLIED ROUTINES AND PRIMITIVES
 *****************************************************************************
 *
 * This header file represents system supplied routines and primitives grouped
 * into five categories:
 *
 * SYS     : System conventions.
 *
 * SYSMEM  : General memory allocation.
 *
 * SYSPDU  : Protocol buffer allocation, access, transmit, and receive.
 *
 * SYSTIME : Scheduling routines : immediate, in fixed (approximate) time, in
 *           random time period.
 *
 * SYSERR  : System error routines for gross errors in program logic detected
 *           in the course of execution, for which there is no sensible course
 *           of action at the point of detection.
 */

/*****************************************************************************
 * SYS : SYSTEM CONVENTIONS
 *****************************************************************************
 */

typedef enum {False = 0, True = 1} Boolean;

enum {Zero = 0, One = 1, Two = 2};

typedef unsigned char   Octet;

typedef unsigned short  Int16;

typedef unsigned char  *Mac_address;

/*****************************************************************************
 * SYS : SYSTEM SUPPLIED MEMORY ALLOCATION ROUTINES
 *****************************************************************************
 */

extern Boolean sysmalloc(int size, void **allocated);

extern void    sysfree(void *allocated);


/*****************************************************************************
 * SYSPDU : SYSTEM SUPPLIED PDU ACCESS PRIMITIVES
 *****************************************************************************
 */

typedef void   Pdu;
```

```
extern Boolean syspdu_alloc(Pdu **pdu);

extern void    syspdu_free( Pdu  *pdu);

extern Boolean rdcheck(     Pdu  *pdu, int number_of_octets_remaining);

extern Boolean rdoctet(     Pdu  *pdu, Octet *val);

extern Boolean rdint16(     Pdu  *pdu, Int16 val);

extern Boolean rdskip(      Pdu  *pdu, int number_to_skip);

extern void    syspdu_tx(   Pdu  *pdu, int port_no);

/*****************************************************************************
 * SYSTIME : SYSTEM SUPPLIED SCHEDULING FUNCTIONS
 *****************************************************************************
 */

extern void systime_start_random_timer(int process_id,
                  void (*expiry_fn)(void *, int instance_id),
                  int    instance_id,
                  int    timeout);

extern void systime_start_timer(int    process_id,
                            void (*expiry_fn) (void *, int instance_id),
                            int    instance_id,
                            int    timeout);

extern void systime_schedule(int    process_id,
                          void (*expiry_fn) (void *, int instance_id),
                          int    instance_id);

/*****************************************************************************
 * SYSERR : FATAL ERROR HANDLING
 *****************************************************************************
 */

extern syserr_panic();

#endif /* sys_h__ */
```

13.3 GARP application independent code

13.3.1 gid.c

```
/* gid.c */

#include “sys.h”
#include “gid.h”
#include “gidtt.h”
#include “gip.h”
#include “garp.h”

/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : CREATION, DESTRUCTION
 *****************************************************************************
 */

static Boolean gid_create_gid(Garp  *application, int port_no, void **gid)
{  /*
    * Creates a new instance of GID.
    */

    Gid *my_port;

    if (!sysmalloc(sizeof(Gid), &my_port))
        goto gid_creation_failure;

    my_port->application              = application;
    my_port->port_no                  = port_no;
    my_port->next_in_port_ring        = my_port;
    my_port->next_in_connected_ring   = my_port;

    my_port->is_enabled               = False;
    my_port->is_connected             = False;
    my_port->is_point_to_point        = True;

    my_port->cschedule_tx_now         = False;
    my_port->cstart_join_timer        = False;
    my_port->cstart_leave_timer       = False;
    my_port->tx_now_scheduled         = False;
    my_port->join_timer_running       = False;
    my_port->leave_timer_running      = False;
    my_port->hold_tx                  = False;

    my_port->join_timeout             = Gid_default_join_time;
    my_port->leave_timeout_4          = Gid_default_leave_time/4;
    my_port->hold_timeout             = Gid_default_hold_time;

    if (!sysmalloc(sizeof(Gid_machine)*(application->max_gid_index + 2),
                   &my_port->machines))
        goto gid_mcreation_failure;

    my_port->leaveall_countdown       = Gid_leaveall_count;
    my_port->leaveall_timeout_n       = Gid_default_leaveall_time/
                                       Gid_leaveall_count;
    systime_start_timer(my_port->application->process_id,
                        gid_leaveall_timer_expired,
                        my_port->port_no,
                        my_port->leaveall_timeout_n);


    my_port->tx_pending               = False;
    my_port->last_transmitted         =  application->last_gid_used;
    my_port->last_to_transmit         =  application->last_gid_used;
```

```
    my_port->untransmit_machine        =  application->last_gid_used + 1;

    *gid = my_port;    return(True);
gid_mcreation_failure: sysfree(my_port);
gid_creation_failure:  return(False);
}


static void gid_destroy_gid(Gid *gid)
{  /*
    * Destroys the instance of GID, releasing previously allocated space.
    * Sends leave indications to the application for previously registered
    * attributes.
    */
    unsigned gid_index;

    for (gid_index = 0; gid_index <= gid->application->last_gid_used;
         gid_index++)
    {
        if (gid_registered_here(gid, gid_index))
            gid->application->leave_indication_fn(gid->application,
                                                  gid, gid_index);
    }

    sysfree(gid->machines);
    sysfree(gid);
}

static Gid *gid_add_port(Gid *existing_ports, Gid *new_port)
{  /*
    * Adds new_port to the port ring.
    */

    Gid *prior;
    Gid *next;
    int  new_port_no;

    if (existing_ports != NULL)
    {
        new_port_no = new_port->port_no;

        next = existing_ports;
        for(;;)
        {
            prior = next; next = prior->next_in_port_ring;

            if (prior->port_no <= new_port_no)
            {
                if (    (next->port_no <= prior->port_no)
                     || (next->port_no >  new_port_no)
                   ) break;
            }
            else /* if (prior_>port_no > new_port_no) */
            {
                if (    (next->port_no <= prior->port_no)
                     && (next->port_no >  new_port_no)
                   ) break;
        }   }

        if (prior->port_no == new_port_no) syserr_panic();

        prior->next_in_port_ring    = new_port;
        new_port->next_in_port_ring = next;
    }
```

```
    new_port->is_enabled = True;
    return(new_port);
}


static Gid *gid_remove_port(Gid *my_port)
{
    Gid *prior;
    Gid *next;

    prior = my_port;
    while ((next = prior->next_in_port_ring) != my_port)
        prior = next;

    prior->next_in_port_ring = my_port->next_in_port_ring;

    if (prior == my_port) return(NULL);
    else                  return(prior);
}


Boolean gid_create_port(Garp  *application, int port_no)
{
     Gid *my_port;

     if (!gid_find_port(application->gid, port_no, &my_port))
     {
        if (gid_create_gid(application, port_no, &my_port))
        {
            application->gid = gid_add_port(application->gid, my_port);
            application->added_port_fn(application, port_no);
            return(True);
     }  }

    return(False);
 }


void gid_destroy_port(Garp  *application, int port_no)
{
     Gid *my_port;

     if (gid_find_port(application->gid, port_no, &my_port))
     {
        gip_disconnect_port(application, port_no);

        application->gid = gid_remove_port(my_port);

        gid_destroy_gid(my_port);

        application->removed_port_fn(application, port_no);
 }  }


/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : USEFUL FUNCTIONS
 *****************************************************************************
 */

Boolean gid_find_port(Gid *first_port, int port_no, void **gid)
{
    Gid   *next_port           = first_port;
    while (next_port->port_no != port_no)
    {
        if ((next_port = next_port->next_in_port_ring) == first_port)
```



```
            return(False);
    }

    *gid = next_port; return(True);
}


Gid *gid_next_port(Gid *this_port)
{
    return(this_port->next_in_port_ring);
}


/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : MGT
 *****************************************************************************
 */

void gid_read_attribute_state(Gid *my_port, unsigned index, Gid_states *state)
{  /*
    */
    gidtt_states(&my_port->machines[index], state);
}


void gid_manage_attribute(Gid *my_port, unsigned index, Gid_event directive)
{  /*
    *
    */

    Gid_machine *machine;
    Gid_event    event;

    machine = &my_port->machines[index];
    event   =  gidtt_event(my_port, machine, directive);
    if (event == Gid_join)
    {
        my_port->application->join_indication_fn(my_port->application,
                                                 my_port, index);
        gip_propagate_join(my_port, index);
    }
    else if (event == Gid_leave)
    {
        my_port->application->leave_indication_fn(my_port->application,
                                                  my_port, index);
        gip_propagate_leave(my_port, index);
}   }


Boolean gid_find_unused(Garp *application, unsigned  from_index,
                                          unsigned *found_index)
{
    unsigned  gid_index;
    Gid      *check_port;

    gid_index = from_index; check_port = application->gid;
    for (;;)
    {
        if (gidtt_machine_active(&check_port->machines[gid_index]))
        {
            if (gid_index++ > application->last_gid_used)
                return(False);
            check_port = application->gid;
        }
        else if ((check_port = check_port->next_in_port_ring)
```

```
                        == application->gid)
        {
            *found_index = gid_index;
            return (True);
}   }   }


/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : EVENT PROCESSSING
 *****************************************************************************
 */

static void gid_leaveall(Gid *my_port)
{  /*
    * only for shared media at present
    */
    unsigned  i;
    Garp     *application;

    application = my_port->application;
    for (i = 0; i <= application->last_gid_used; i++)
        (void) gidtt_event(my_port, &my_port->machines[i], Gid_rcv_leaveempty);
}


void gid_rcv_leaveall(Gid *my_port)
{
    my_port->leaveall_countdown = Gid_leaveall_count;
    gid_leaveall(my_port);
}


void gid_rcv_msg(Gid *my_port, unsigned index, Gid_event msg)
{
    Gid_machine *machine;
    Gid_event    event;

    machine = &my_port->machines[index];
    event   =  gidtt_event(my_port, machine, msg);
    if (event == Gid_join)
    {
        my_port->application->join_indication_fn(my_port->application,
                                                 my_port, index);
        gip_propagate_join(my_port, index);
    }
    else if (event == Gid_leave)
    {
        my_port->application->leave_indication_fn(my_port->application,
                                                  my_port, index);
        gip_propagate_leave(my_port, index);
}   }


void gid_join_request(Gid *my_port, unsigned gid_index)
{
    (void)(gidtt_event(my_port, &my_port->machines[gid_index], Gid_join));
}


void gid_leave_request(Gid *my_port, unsigned gid_index)
{
    (void)(gidtt_event(my_port, &my_port->machines[gid_index], Gid_leave));
}
```



```
Boolean gid_registrar_in(Gid_machine *machine)
{
    return(gidtt_in(machine));
}

/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : RECEIVE PROCESSING
 *****************************************************************************
 */

void gid_rcv_pdu(Garp *application, int port_no, void *pdu)
{  /*
    * If a GID instance for this application and port number is found and is
    * enabled, pass the PDU to the application, which will parse it (using the
    * application's own PDU formatting conventions) and call gid_rcv_msg() for
    * each of the conceptual GID message components read from the PDU. Once
    * the application is finished with the PDU, call gip_do_actions() to start
    * timers as recorded in the GID scratchpad for this port and any ports to
    * which it may have propagated joins or leaves.
    *
    * Finally release the received pdu.
    */
    Gid     *my_port;

    if (gid_find_port(application->gid, port_no, &my_port))
    {
        if (my_port->is_enabled)
        {
            application->receive_fn(application, my_port, pdu);

            gip_do_actions(my_port);
    }   }
/* gid_rlse_rcv_pdu: Insert any system specific action required. */
}

/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : TRANSMIT PROCESSSING
 *****************************************************************************
 */

Gid_event gid_next_tx(Gid *my_port, unsigned *index)
{  /*
    * Check to see if a leaveall should be sent; if so, return Gid_tx_leaveall;
    * otherwise, scan the GID machines for messsages that require transmission.
    *
    * Machines will be checked for potential transmission starting with the
    * machine following last_transmitted and up to and including last_to_transmit.
    * If all machines have transmitted, last_transmitted equals last_to_transmit
    * and tx_pending is False (in this case tx_pending distinguished the
    * case of 'all machines are yet to be checked for transmission' from 'all have
    * been checked.')
    *
    * If tx_pending is True and all machines are yet to be checked, transmission
    * will start from the machine with GID index 0, rather than from immediately
    * following last_transmitted.
    */
    unsigned     check_index;
    unsigned     stop_after;
    Gid_event    msg;

    if (my_port->hold_tx) return(Gid_null);

    if (my_port->leaveall_countdown == 0)
    {
        my_port->leaveall_countdown = Gid_leaveall_count;
```

```
        systime_start_timer(my_port->application->process_id,
                            gid_leaveall_timer_expired,
                            my_port->port_no,
                            my_port->leaveall_timeout_n);
        return(Gid_tx_leaveall);
    }

    if (!my_port->tx_pending) return(Gid_null);

    check_index    = my_port->last_transmitted + 1;
    stop_after     = my_port->last_to_transmit;
    if (stop_after < check_index)
        stop_after = my_port->application->last_gid_used;

    for(;; check_index++)
    {
        if (check_index > stop_after)
        {
            if (stop_after == my_port->last_to_transmit)
            {
                my_port->tx_pending = False;
                return(Gid_null);
            }
            else if (stop_after == my_port->application->last_gid_used)
            {
                check_index = 0;
                stop_after = my_port->last_to_transmit;
        }   }

        if ((msg = gidtt_tx(my_port, &my_port->machines[check_index]))
                != Gid_null)
        {
            *index = my_port->last_transmitted = check_index;

            my_port->machines[my_port->untransmit_machine].applicant =
            my_port->machines[check_index].applicant;

            my_port->tx_pending = (check_index == stop_after);

            return(msg);
}   }   } /* end for(;;) */


void gid_untx(Gid *my_port)
{
    my_port->machines[my_port->last_transmitted].applicant =
    my_port->machines[my_port->untransmit_machine].applicant;

    if (my_port->last_transmitted == 0)
        my_port->last_transmitted  = my_port->application->last_gid_used;
    else
        my_port->last_transmitted--;

    my_port->tx_pending = True;
}

/*****************************************************************************
 * GID : GARP INFORMATION DISTRIBUTION PROTOCOL : TIMER PROCESSING
 *****************************************************************************
 */

void gid_do_actions(Gid *my_port)
{  /*
    * Carries out 'scratchpad' actions accumulated in this invocation of GID,
    * and outstanding 'immediate' transmissions and join timer starts that
```



```
    * have been delayed by the operation of the hold timer. Note the way in
    * which the hold timer works here. It could have been specified just to
    * impose a minimum spacing on transmissions - and run in parallel with the
    * join timer - with the effect that the longer of the hold timer and actual
    * join timer values would have determined the actual transmission time.
    * This approach was not taken because it could have led to bunching
    * transmissions at the hold time.
    *
    * The procedure restarts the join timer if there are still transmissions
    * pending (if leaveall_countdown is zero. a Leaveall is to be sent; if
    * tx_pending is true, individual machines may have messages to send.)
    */

    int my_port_no = my_port->port_no;

    if (my_port->cstart_join_timer)
    {
        my_port->last_to_transmit  = my_port->last_transmitted;
        my_port->tx_pending        = True;
        my_port->cstart_join_timer = False;
    }

    if (!my_port->hold_tx)
    {
        if (my_port->cschedule_tx_now)
        {
            if (!my_port->tx_now_scheduled)
                systime_schedule(my_port->application->process_id,
                                 gid_join_timer_expired,
                                 my_port->port_no);

            my_port->cschedule_tx_now  = False;
        }
        else if (     (my_port->tx_pending || (my_port->leaveall_countdown == 0))
                  && (!my_port->join_timer_running))
        {
            systime_start_random_timer(my_port->application->process_id,
                                       gid_join_timer_expired,
                                       my_port->port_no,
                                       my_port->join_timeout);

            my_port->join_timer_running = True;
    }   }

    if (my_port->cstart_leave_timer && (!my_port->leave_timer_running))
    {
        systime_start_timer(my_port->application->process_id,
                            gid_leave_timer_expired,
                            my_port->port_no,
                            my_port->leave_timeout_4);

    }
    my_port->cstart_leave_timer  = False;
}


void gid_leave_timer_expired(Garp *application, int port_no)
{
    Gid      *my_port;
    unsigned  gid_index;

    if (gid_find_port(application->gid, port_no, &my_port))
    {
        for (gid_index = 0; gid_index < my_port->application->last_gid_used;
            gid_index++)
```

```
        {
            if (   gidtt_leave_timer_expiry(my_port,&my_port->machines[gid_index])
                == Gid_leave)
            {
                my_port->application->leave_indication_fn(my_port->application,
                                                          my_port, gid_index);
                gip_propagate_leave(my_port, gid_index);
}   }   }   }


void gid_leaveall_timer_expired(Garp *application, int port_no)
{
    Gid *my_port;

    if (gid_find_port(application->gid, port_no, &my_port))
    {
        if (my_port->leaveall_countdown > 1)
            my_port->leaveall_countdown--;
        else
        {
            gid_leaveall(my_port);

            my_port->leaveall_countdown = 0;
            my_port->cstart_join_timer  = True;

            if ((!my_port->join_timer_running) && (!my_port->hold_tx))

            systime_start_random_timer(my_port->application->process_id,
                                       gid_join_timer_expired,
                                       my_port->port_no,
                                       my_port->join_timeout);

            my_port->cstart_join_timer = False;
            my_port->join_timer_running = True;
}   }   }


void gid_join_timer_expired(Garp *application, int port_no)
{
    Gid *my_port;

    if (gid_find_port(application->gid, port_no, &my_port))
    {
        if (my_port->is_enabled)
        {
            application->transmit_fn(application, my_port);

            systime_start_timer(my_port->application->process_id,
                                gid_hold_timer_expired,
                                my_port->port_no,
                                my_port->hold_timeout);
}   }   }


void gid_hold_timer_expired(Garp *application, int port_no)
{
    Gid *my_port;

    if (gid_find_port(application->gid, port_no, &my_port))
    {
        my_port->hold_tx = False;

        gid_do_actions(my_port);
}   }
```


13.3.2 gidtt.c

```
/* gidtt.c */

#include “gidtt.h”
#include “gid.h”


/*****************************************************************************
 * GIDTT : GID PROTOCOL TRANSITION TABLES : IMPLEMENTATION OVERVIEW
 *****************************************************************************
 */
/* This implementation of GID uses transition tables directly. This makes
 * the implementation appear very bulky, but the net code size impact may be
 * less than the page of code required for an algorithmic-based implementation,
 * depending on the processor. A processing-based implementation is planned as
 * both alternatives may be interesting.
 *
 * The Applicant and the Registrar use separate transition tables. Both use
 * a general transition table to handle most events, and separate smaller
 * tables to determine behavior when a transmit opportunity occurs (both
 * Applicant and Registrar), and when the leave timer expires (Registrar only).
 *
 * The stored states support management control directly - which leads to a
 * total of 14 applicant states and 18 registrar states (the registrar states
 * also incorporate leave timer support):
 *
 *    The Applicant may be in one of the following management states:
 *
 *     1. Normal
 *
 *     2. No protocol
 *
 *        The protocol machine is held quiet and never sends a message, even
 *        to prompt another GID participant on the same LAN to respond. In
 *        this state the Applicant does messages on the media so it can
 *        be toggled between Normal and No protocol with minimum
 *        disruption.
 *
 *    The Registrar may be in one of the following management states:
 *
 *     1. Normal Registration.
 *
 *     2. Registration Fixed.
 *
 *        The Registrar always reports “In” to the application and to GIP
 *        whatever has occurred on the LAN.
 *
 *     3. Registration Forbidden.
 *
 *        The Registrar always reports “Empty” to the application and to GIP.
 *
 * A set of small tables is used to report aspects of the management state of
 * both applicant and registrar.
 *
 * The main applicant transition table (applicant_tt) is indexed by current
 * applicant state and GID event, and returns
 *
 *     1. The new applicant state.
 *
 *     2. A start join timer instruction, when required.
 *
 *
 *
```

```
 * The main registrar transition table (registrar_tt) is indexed by current
 * registrar state and GID event, and returns
 *
 *     1. The new registrar state.
 *
 *     2. A join indication or a leave indication, when required.
 *
 *     3. A start leave timer instruction, when required.
 *
 * The only user interface to both these tables is through the public
 * function gidtt_event(), which accepts and returns Gid_events (to report
 * join or leave indications), and which writes timer start requests to the
 * GID scratchpad directly.
 *
 * The Applicant transmit transition table (applicant_txtt) returns the new
 * applicant state, the message to be transmitted, and whether the join timer
 * should be restarted to transmit a further message. A modifier that
 * determines whether a Join (selected for transmission by the Applicant table)
 * should be transmitted as a JoinIn or as a JoinEmpty is taken from a
 * Registrar state reporting table. The Registrar state is never modified by
 * transmission.
 */


/*****************************************************************************
 * GIDTT : GID PROTOCOL TRANSITION TABLE : TABLE ENTRY DEFINITIONS
 *****************************************************************************
 */
enum Applicant_states
{
    Va,  /* Very anxious, active   */
    Aa,  /* Anxious,      active   */
    Qa,  /* Quiet,        active   */
    La,  /* Leaving,      active   */
    Vp,  /* Very anxious, passive  */
    Ap,  /* Anxious,      passive  */
    Qp,  /* Quiet,        passive  */
    Vo,  /* Very anxious observer  */
    Ao,  /* Anxious observer       */
    Qo,  /* Quiet observer         */
    Lo,  /* Leaving observer       */

    Von, /* Very anxious observer, non-participant      */
    Aon, /* Anxious observer,      non-participant      */
    Qon  /* Quiet_observer,        non-participant      */
};

enum Registrar_states
{  /* In, Leave, Empty, but with Leave states implementing a countdown for the
    * leave timer.
    */

    Inn,
    Lv,L3,L2,L1,
    Mt,

    Inr,             /* In, registration fixed     */
    Lvr,L3r,L2r,L1r,
    Mtr,

    Inf,             /* In, registration forbidden */
    Lvf,L3f,L2f,L1f,
    Mtf
};
```

```
enum {Number_of_applicant_states = Qon + 1}; /* for array sizing */
enum {Number_of_registrar_states = Mtf + 1}; /* for array sizing */

enum Timers
{
    Nt = 0,  /* No timer action    */
    Jt = 1,  /* cstart_join_timer  */
    Lt = 1   /* cstart_leave_timer */
};

enum Applicant_msg
{
    Nm = 0, /* No message to transmit */
    Jm,     /* Transmit a Join        */
    Lm,     /* Transmit a Leave       */
    Em      /* Transmit an Empty      */
};

enum Registrar_indications
{
    Ni = 0,
    Li = 1,
    Ji = 2
};

/*****************************************************************************
 * GIDTT : GID PROTOCOL TRANSITION TABLES : TRANSITION TABLE STRUCTURE
 *****************************************************************************
 */

typedef struct                      /* Applicant_tt_entry */
{
    unsigned new_app_state      : 5;  /* {Applicant_states} */

    unsigned cstart_join_timer  : 1;

} Applicant_tt_entry;

typedef struct                      /* Registrar_tt_entry */
{
    unsigned new_reg_state      : 5;

    unsigned indications        : 2;

    unsigned cstart_leave_timer : 1;

} Registrar_tt_entry;

typedef struct                      /* Applicant_txtt_entry */
{
    unsigned new_app_state      : 5;

    unsigned msg_to_transmit    : 2; /* Applicant_msgs */

    unsigned cstart_join_timer  : 1;

} Applicant_txtt_entry;

typedef struct                      /* Registrar_leave_timer_entry */
{
    unsigned new_reg_state      : 5; /* Registrar_states           */

    unsigned leave_indication   : 1;
```

```
    unsigned cstart_leave_timer : 1;

} Registrar_leave_timer_entry;


/*****************************************************************************
 * GIDTT : GID PROTOCOL: MAIN APPLICANT TRANSITION TABLE
 *****************************************************************************
 */

static Applicant_tt_entry
       applicant_tt[Number_of_gid_rcv_events  + Number_of_gid_req_events +
                    Number_of_gid_amgt_events + Number_of_gid_rmgt_events]
                   [Number_of_applicant_states]=
{  /*
    * General applicant transition table. See description above.
    */
    {   /* Gid_null */
        /*Va */{Va, Nt},/*Aa */{Aa, Nt},/*Qa */{Qa, Nt},/*La */{La, Nt},
        /*Vp */{Vp, Nt},/*Ap */{Ap, Nt},/*Vp */{Vp, Nt},
        /*Vo */{Vo, Nt},/*Ao */{Ao, Nt},/*Qo */{Qo, Nt},/*Lo */{Lo, Nt},

        /*Von*/{Von,Nt},/*Aon*/{Aon,Nt},/*Qon*/{Qon,Nt}
    },
    {   /* Gid_rcv_leaveempty */
        /*Va */{Vp, Nt},/*Aa */{Vp, Nt},/*Qa */{Vp, Jt},/*La */{Vo, Nt},
        /*Vp */{Vp, Nt},/*Ap */{Vp, Nt},/*Qp */{Vp, Jt},
        /*Vo */{Lo, Nt},/*Ao */{Lo, Nt},/*Qo */{Lo, Jt},/*Lo */{Vo, Nt},

        /*Von*/{Von,Nt},/*Aon*/{Von,Nt},/*Qon*/{Von,Nt}
    },
    {   /* Gid_rcv_leavein */
        /*Va */{Va, Nt},/*Aa */{Va, Nt},/*Qa */{Vp, Jt},/*La */{La, Nt},
        /*Vp */{Vp, Nt},/*Ap */{Vp, Nt},/*Qp */{Vp, Jt},
        /*Vo */{Lo, Nt},/*Ao */{Lo, Nt},/*Qo */{Lo, Jt},/*Lo */{Vo, Nt},

        /*Von*/{Von,Nt},/*Aon*/{Von,Nt},/*Qon*/{Von,Nt}
    },
    {   /* Gid_rcv_empty */
        /*Va */{Va, Nt},/*Aa */{Va, Nt},/*Qa */{Va, Jt},/*La */{La, Nt},
        /*Vp */{Vp, Nt},/*Ap */{Vp, Nt},/*Qp */{Vp, Jt},
        /*Vo */{Vo, Nt},/*Ao */{Vo, Nt},/*Qo */{Vo, Nt},/*Lo */{Vo, Nt},

        /*Von*/{Von,Nt},/*Aon*/{Von,Nt},/*Qon*/{Von,Nt}
    },
    {   /* Gid_rcv_joinempty */
        /*Va */{Va, Nt},/*Aa */{Va, Nt},/*Qa */{Va, Jt},/*La */{Vo, Nt},
        /*Vp */{Vp, Nt},/*Ap */{Vp, Nt},/*Qp */{Vp, Jt},
        /*Vo */{Vo, Nt},/*Ao */{Vo, Nt},/*Qo */{Vo, Jt},/*Lo */{Vo, Nt},

        /*Von*/{Von,Nt},/*Aon*/{Von,Nt},/*Qon*/{Von,Jt}
    },
    {   /* Gid_rcv_joinin */
        /*Va */{Aa, Nt},/*Aa */{Qa, Nt},/*Qa */{Qa, Nt},/*La */{La, Nt},
        /*Vp */{Ap, Nt},/*Ap */{Qp, Nt},/*Qp */{Qp, Nt},
        /*Vo */{Ao, Nt},/*Ao */{Qo, Nt},/*Qo */{Qo, Nt},/*Lo */{Ao, Nt},

        /*Von*/{Aon,Nt},/*Aon*/{Qon,Nt},/*Qon*/{Qon,Nt}
    },
    {   /* Gid_join, join request. Handles repeated joins, i.e., joins for
         * states that are already in. Does not provide feedback for joins
         * that are forbidden by management controls; the expectation is
         * that this table will not be directly used by new management
         * requests.
         */
```



```
        /*Va */{Va, Nt},/*Aa */{Aa, Nt},/*Qa */{Qa, Nt},/*La */{Va, Nt},
        /*Vp */{Vp, Nt},/*Ap */{Ap, Nt},/*Qp */{Qp, Nt},
        /*Vo */{Vp, Jt},/*Ao */{Ap, Jt},/*Qo */{Qp, Nt},/*Lo */{Vp, Nt},

        /*Von*/{Von,Nt},/*Aon*/{Aon,Nt},/*Qon*/{Qon,Nt}
    },
    {   /* Gid_leave, leave request. See comments for join requests above. */
        /*Va */{La, Nt},/*Aa */{La, Nt},/*Qa */{La, Jt},/*La */{La, Nt},
        /*Vp */{Vo, Nt},/*Ap */{Ao, Nt},/*Qp */{Qo, Nt},
        /*Vo */{Vo, Nt},/*Ao */{Ao, Nt},/*Qo */{Qo, Nt},/*Lo */{Lo, Nt},

        /*Von*/{Von,Nt},/*Aon*/{Aon,Nt},/*Qon*/{Qon,Nt}
    },
    {   /* Gid_normal_operation */
        /*Va */{Vp, Nt},/*Aa */{Vp, Nt},/*Qa */{Vp, Jt},/*La */{La, Nt},
        /*Vp */{Vp, Nt},/*Ap */{Vp, Nt},/*Qp */{Vp, Jt},
        /*Vo */{Va, Nt},/*Ao */{Va, Nt},/*Qo */{Va, Jt},/*Lo */{Lo, Nt},

        /*Von*/{Va, Nt},/*Aon*/{Va, Nt},/*Qon*/{Va, Jt}
    },
    {   /* Gid_no_protocol */
        /*Va */{Von,Nt},/*Aa */{Aon,Nt},/*Qa */{Qon,Nt},/*La */{Von,Nt},
        /*Vp */{Von,Nt},/*Ap */{Aon,Nt},/*Qp */{Qon,Nt},
        /*Vo */{Von,Nt},/*Ao */{Aon,Nt},/*Qo */{Qon,Nt},/*Lo */{Von,Nt},

        /*Von*/{Von,Nt},/*Aon*/{Aon,Nt},/*Qon*/{Qon,Nt}
    },
    {   /* Gid_normal_registration, same as Gid_null for the Applicant */
        /*Va */{Va, Nt},/*Aa */{Aa, Nt},/*Qa */{Qa, Nt},/*La */{La, Nt},
        /*Vp */{Vp, Nt},/*Ap */{Ap, Nt},/*Vp */{Vp, Nt},
        /*Vo */{Vo, Nt},/*Ao */{Ao, Nt},/*Qo */{Qo, Nt},/*Lo */{Lo, Nt},

        /*Von*/{Von,Nt},/*Aon*/{Aon,Nt},/*Qon*/{Qon,Nt}
    },
    {   /* Gid_fix_registration, same as Gid_null for the Applicant */
        /*Va */{Va, Nt},/*Aa */{Aa, Nt},/*Qa */{Qa, Nt},/*La */{La, Nt},
        /*Vp */{Vp, Nt},/*Ap */{Ap, Nt},/*Vp */{Vp, Nt},
        /*Vo */{Vo, Nt},/*Ao */{Ao, Nt},/*Qo */{Qo, Nt},/*Lo */{Lo, Nt},

        /*Von*/{Von,Nt},/*Aon*/{Aon,Nt},/*Qon*/{Qon,Nt}
    },
    {   /* Gid_forbid_registration, same as Gid_null for the Applicant */
        /*Va */{Va, Nt},/*Aa */{Aa, Nt},/*Qa */{Qa, Nt},/*La */{La, Nt},
        /*Vp */{Vp, Nt},/*Ap */{Ap, Nt},/*Vp */{Vp, Nt},
        /*Vo */{Vo, Nt},/*Ao */{Ao, Nt},/*Qo */{Qo, Nt},/*Lo */{Lo, Nt},

        /*Von*/{Von,Nt},/*Aon*/{Aon,Nt},/*Qon*/{Qon,Nt}
    }
};


/*****************************************************************************
 * GIDTT : GID PROTOCOL: MAIN REGISTRAR TRANSITION TABLE
 *****************************************************************************
 */

static Registrar_tt_entry
       registrar_tt[Number_of_gid_rcv_events  + Number_of_gid_req_events +
                    Number_of_gid_amgt_events + Number_of_gid_rmgt_events]
                   [Number_of_registrar_states] =
{

    {   /* Gid_null      */
        /*In */{Inn,Ni,Nt},
        /*Lv */{Lv, Ni,Nt},
```

```
        /*L3 */{L3, Ni,Nt},/*L2 */{L2, Ni,Nt},/*L1 */{L1, Ni,Nt},
        /*Mt */{Mt, Ni,Nt},

        /*Inr*/{Inr,Ni,Nt},
        /*Lvr*/{Lvr,Ni,Nt},
        /*L3r*/{L3r,Ni,Nt},/*L2r*/{L2r,Ni,Nt},/*L1r*/{L1r,Ni,Nt},
        /*Mtr*/{Mtr,Ni,Nt},

        /*Inf*/{Inf,Ni,Nt},
        /*Lvf*/{Lvf,Ni,Nt},
        /*L3f*/{L3f,Ni,Nt},/*L2f*/{L2f,Ni,Nt},/*L1f*/{L1f,Ni,Nt},
        /*Mtf*/{Mtf,Ni,Nt}
    },
    {   /* Gid_rcv_leave */
        /*Inn*/{Lv, Ni,Lt},
        /*Lv */{Lv, Ni,Nt},
        /*L3 */{L3, Ni,Nt},/*L2 */{L2, Ni,Nt},/*L1 */{L1, Ni,Nt},
        /*Mt */{Mt, Ni,Nt},

        /*Inr*/{Lvr,Ni,Lt},
        /*Lvr*/{Lvr,Ni,Nt},
        /*L3r*/{L3r,Ni,Nt},/*L2r*/{L2r,Ni,Nt},/*L1r*/{L1r,Ni,Nt},
        /*Mtr*/{Mtr,Ni,Nt},

        /*Inf*/{Lvf,Ni,Lt},
        /*Lvf*/{Lvf,Ni,Nt},
        /*L3f*/{L3f,Ni,Nt},/*L2f*/{L2f,Ni,Nt},/*L1f*/{L1f,Ni,Nt},
        /*Mtf*/{Mtf,Ni,Nt}
    },
    {   /* Gid_rcv_empty */
        /*Inn*/{Inn,Ni,Nt},
        /*Lv */{Lv, Ni,Nt},
        /*L3 */{L3, Ni,Nt},/*L2 */{L2, Ni,Nt},/*L1 */{L1, Ni,Nt},
        /*Mt */{Mt, Ni,Nt},

        /*Inr*/{Inr,Ni,Nt},
        /*Lvr*/{Lvr,Ni,Nt},
        /*L3r*/{L3r,Ni,Nt},/*L2r*/{L2r,Ni,Nt},/*L1r*/{L1r,Ni,Nt},
        /*Mtr*/{Mtr,Ni,Nt},

        /*Inf*/{Inf,Ni,Nt},
        /*Lvf*/{Lvf,Ni,Nt},
        /*L3f*/{L3f,Ni,Nt},/*L2f*/{L2f,Ni,Nt},/*L1f*/{L1f,Ni,Nt},
        /*Mtf*/{Mtf,Ni,Nt}

    },
    {   /* Gid_rcv_joinempty */
        /*Inn*/{Inn,Ni,Nt},
        /*Lv */{Inn,Ni,Nt},
        /*L3 */{Inn,Ni,Nt},/*L2 */{Inn,Ni,Nt},/*L1 */{Inn,Ni,Nt},
        /*Mt */{Inn,Ji,Nt},

        /*Inr*/{Inr,Ni,Nt},
        /*Lvr*/{Inr,Ni,Nt},
        /*L3r*/{Inr,Ni,Nt},/*L2r*/{Inr,Ni,Nt},/*L1r*/{Inr,Ni,Nt},
        /*Mtr*/{Inr,Ni,Nt},

        /*Inf*/{Inf,Ni,Nt},
        /*Lvf*/{Inf,Ni,Nt},
        /*L3f*/{Inf,Ni,Nt},/*L2f*/{Inf,Ni,Nt},/*L1f*/{Inf,Ni,Nt},
        /*Mtf*/{Inf,Ni,Nt}

    },
    {   /* Gid_rcv_joinin */
        /*Inn*/{Inn,Ni,Nt},
```

```
        /*Lv */{Inn,Ni,Nt},
        /*L3 */{Inn,Ni,Nt},/*L2 */{Inn,Ni,Nt},/*L1 */{Inn,Ni,Nt},
        /*Mt */{Inn,Ji,Nt},

        /*Inr*/{Inr,Ni,Nt},
        /*Lvr*/{Inr,Ni,Nt},
        /*L3r*/{Inr,Ni,Nt},/*L2r*/{Inr,Ni,Nt},/*L1r*/{Inr,Ni,Nt},
        /*Mtr*/{Inr,Ni,Nt},

        /*Inf*/{Inf,Ni,Nt},
        /*Lvf*/{Inf,Ni,Nt},
        /*L3f*/{Inf,Ni,Nt},/*L2f*/{Inf,Ni,Nt},/*L1f*/{Inf,Ni,Nt},
        /*Mtf*/{Inf,Ni,Nt}
    },
    {   /* Gid_normal_operation, same as Gid_null for the Registrar */
        /*In */{Inn,Ni,Nt},
        /*Lv */{Lv, Ni,Nt},
        /*L3 */{L3, Ni,Nt},/*L2 */{L2, Ni,Nt},/*L1 */{L1, Ni,Nt},
        /*Mt */{Mt, Ni,Nt},

        /*Inr*/{Inr,Ni,Nt},
        /*Lvr*/{Lvr,Ni,Nt},
        /*L3r*/{L3r,Ni,Nt},/*L2r*/{L2r,Ni,Nt},/*L1r*/{L1r,Ni,Nt},
        /*Mtr*/{Mtr,Ni,Nt},

        /*Inf*/{Inf,Ni,Nt},
        /*Lvf*/{Lvf,Ni,Nt},
        /*L3f*/{L3f,Ni,Nt},/*L2f*/{L2f,Ni,Nt},/*L1f*/{L1f,Ni,Nt},
        /*Mtf*/{Mtf,Ni,Nt}
    },
    {   /* Gid_no_protocol, same as Gid_null for the Registrar */
        /*In */{Inn,Ni,Nt},
        /*Lv */{Lv, Ni,Nt},
        /*L3 */{L3, Ni,Nt},/*L2 */{L2, Ni,Nt},/*L1 */{L1, Ni,Nt},
        /*Mt */{Mt, Ni,Nt},

        /*Inr*/{Inr,Ni,Nt},
        /*Lvr*/{Lvr,Ni,Nt},
        /*L3r*/{L3r,Ni,Nt},/*L2r*/{L2r,Ni,Nt},/*L1r*/{L1r,Ni,Nt},
        /*Mtr*/{Mtr,Ni,Nt},

        /*Inf*/{Inf,Ni,Nt},
        /*Lvf*/{Lvf,Ni,Nt},
        /*L3f*/{L3f,Ni,Nt},/*L2f*/{L2f,Ni,Nt},/*L1f*/{L1f,Ni,Nt},
        /*Mtf*/{Mtf,Ni,Nt}
    },
    {   /* Gid_normal_registration */
        /*Inn*/{Inn,Ni,Nt},
        /*Lv */{Lv, Ni,Nt},
        /*L3 */{L3, Ni,Nt},/*L2 */{L2, Ni,Nt},/*L1 */{L1, Ni,Nt},
        /*Mt */{Mt, Ni,Nt},

        /*Inr*/{Inn,Ni,Nt},
        /*Lvr*/{Lv, Ni,Nt},
        /*L3r*/{L3, Ni,Nt},/*L2r*/{L2, Ni,Nt},/*L1r*/{L1, Ni,Nt},
        /*Mtr*/{Mt, Li,Nt},

        /*Inf*/{Inn,Ji,Nt},
        /*Lvf*/{Lv, Ji,Nt},
        /*L3f*/{L3, Ji,Nt},/*L2f*/{L2, Ji,Nt},/*L1f*/{L1, Ji,Nt},
        /*Mtf*/{Mt, Ni,Nt}

    },
    {   /* Gid_fix_registration */
        /*Inn*/{Inr,Ni,Nt},
```

```
        /*Lv */{Lvr,Ni,Nt},
        /*L3 */{L3r,Ni,Nt},/*L2 */{L2r,Ni,Nt},/*L1 */{L1r,Ni,Nt},
        /*Mt */{Mtr,Ji,Nt},

        /*Inr*/{Inr,Ni,Nt},
        /*Lvr*/{Lvr,Ni,Nt},
        /*L3r*/{L3r,Ni,Nt},/*L2r*/{L2r,Ni,Nt},/*L1r*/{L1r,Ni,Nt},
        /*Mtr*/{Mtr,Ni,Nt},

        /*Inf*/{Inr,Ji,Nt},
        /*Lvf*/{Lvr,Ji,Nt},
        /*L3f*/{L3r,Ji,Nt},/*L2f*/{L2r,Ji,Nt},/*L1f*/{L1r,Ji,Nt},
        /*Mtf*/{Mtr,Ji,Nt}

    },
    {   /* Gid_forbid_registration */
        /*Inn*/{Inf,Li,Nt},
        /*Lv */{Lvf,Li,Nt},
        /*L3 */{L3f,Li,Nt},/*L2 */{L2f,Li,Nt},/*L1 */{L1f,Li,Nt},
        /*Mt */{Mtf,Ni,Nt},

        /*Inr*/{Inr,Li,Nt},
        /*Lvr*/{Lvr,Li,Nt},
        /*L3r*/{L3r,Li,Nt},/*L2r*/{L2r,Li,Nt},/*L1r*/{L1r,Li,Nt},
        /*Mtr*/{Mtr,Li,Nt},

        /*Inf*/{Inf,Ni,Nt},
        /*Lvf*/{Lvf,Ni,Nt},
        /*L3f*/{L3f,Ni,Nt},/*L2f*/{L2f,Ni,Nt},/*L1f*/{L1f,Ni,Nt},
        /*Mtf*/{Mtf,Ni,Nt}
    }
};


/*****************************************************************************
 * GIDTT : GID PROTOCOL : APPLICANT TRANSMIT TABLE
 *****************************************************************************
 */

static Applicant_txtt_entry
       applicant_txtt[Number_of_applicant_states] =
{
 /*Va */{Aa, Jm,Jt},/*Aa */{Qa, Jm,Nt},/*Qa */{Qa, Nm,Nt},/*La */{Vo, Lm,Nt},
 /*Vp */{Aa, Jm,Jt},/*Ap */{Qa, Jm,Nt},/*Qp */{Qp, Nm,Nt},
 /*Vo */{Vo, Nm,Nt},/*Ao */{Ao, Nm,Nt},/*Qo */{Qo, Nm,Nt},/*Lo */{Vo, Nm,Nt},

 /*Von*/{Von,Nm,Nt},/*Aon*/{Aon,Nm,Nt},/*Qon*/{Qon,Nm,Nt}
};


/*****************************************************************************
 * GIDTT : GID PROTOCOL : REGISTRAR LEAVE TIMER TABLE
 *****************************************************************************
 */

static Registrar_leave_timer_entry
       registrar_leave_timer_table[Number_of_registrar_states] =
{
 /*Inn*/{Inn,Ni,Nt},
 /*Lv */{L3, Ni,Lt},/*L3 */{L2, Ni,Lt},/*L2 */{L1, Ni,Lt},/*L1 */{Mt, Li,Nt},
 /*Mt */{Mt, Ni,Nt},

 /*Inr*/{Inr,Ni,Nt},
 /*Lvr*/{L3r,Ni,Lt},/*L3r*/{L2r,Ni,Lt},/*L2r*/{L1r,Ni,Lt},/*L1r*/{Mtr,Ni,Nt},
 /*Mtr*/{Mtr,Ni,Nt},
```

```
 /*Inf*/{Inf,Ni,Nt},
 /*Lvf*/{L3f,Ni,Lt},/*L3f*/{L2f,Ni,Lt},/*L2f*/{L1f,Ni,Lt},/*L1f*/{Mtf,Ni,Nt},
 /*Mtf*/{Mtf,Ni,Nt}
};

/*****************************************************************************
 * GIDTT : GID PROTOCOL : STATE REPORTING TABLES
 *****************************************************************************
 */

static Gid_applicant_state applicant_state_table[Number_of_applicant_states] =
{
    /*Va */{Very_anxious},/*Aa */{Anxious},/*Qa */{Quiet},/*La */{Leaving},
    /*Vp */{Very_anxious},/*Ap */{Anxious},/*Qp */{Quiet},
    /*Vo */{Very_anxious},/*Ao */{Anxious},/*Qo */{Quiet},/*Lo */{Leaving},

    /*Von*/{Very_anxious},/*Aon*/{Anxious},/*Qon*/{Quiet}
};

static Gid_applicant_mgt applicant_mgt_table[Number_of_applicant_states] =
{
    /*Va */{Normal},/*Aa */{Normal},/*Qa */{Normal},
    /*La */{Normal},
    /*Vp */{Normal},/*Ap */{Normal},/*Qp */{Normal},
    /*Vo */{Normal},/*Ao */{Normal},/*Qo */{Normal},
    /*Lo */{Normal},

    /*Von*/{No_protocol},/*Aon*/{No_protocol},/*Qon*/{No_protocol}
};

static Gid_registrar_state registrar_state_table[Number_of_registrar_states] =
{
    /*Inn*/{In},
    /*Lv */{Leave},/*L3 */{Leave},/*L2 */{Leave},/*L1 */{Leave},/*Mt */{Empty},

    /*Inr*/{In},
    /*Lvr*/{Leave},/*L3r*/{Leave},/*L2r*/{Leave},/*L1r*/{Leave},/*Mtr*/{Empty},

    /*Inf*/{In},
    /*Lvf*/{Leave},/*L3f*/{Leave},/*L2f*/{Leave},/*L1f*/{Leave},/*Mtf*/{Empty}
};

static Gid_registrar_mgt registrar_mgt_table[Number_of_registrar_states] =
{
    /*Inn*/{Normal_registration},
    /*Lv */{Normal_registration},/*L3 */{Normal_registration},
    /*L2 */{Normal_registration},/*L1 */{Normal_registration},
    /*Mt */{Normal_registration},

    /*Inr*/{Registration_fixed},
    /*Lvr*/{Registration_fixed},/*L3r*/{Registration_fixed},
    /*L2r*/{Registration_fixed},/*L1r*/{Registration_fixed},
    /*Mtr*/{Registration_fixed},

    /*Inf*/{Registration_forbidden},
    /*Lvf*/{Registration_forbidden},/*L3f*/{Registration_forbidden},
    /*L2f*/{Registration_forbidden},/*L1f*/{Registration_forbidden},
    /*Mtf*/{Registration_forbidden}
};

static Boolean registrar_in_table[Number_of_registrar_states] =
{
    /*Inn*/{True},/*Lv */{True},/*L3 */{True},/*L2 */{True},/*L1 */{True},
    /*Mt */{False},
```

```
    /*Inr*/{True},/*Lvr*/{True},/*L3r*/{True},/*L2r*/{True},/*L1r*/{True},
    /*Mtr*/{True},

    /*Inf*/{False},/*Lvf*/{False},/*L3f*/{False},/*L2f*/{False},/*L1f*/{False},
    /*Mtf*/{False}
};

/*****************************************************************************
 * GIDTT : GID PROTOCOL : RECEIVE EVENTS, USER REQUESTS, & MGT PROCESSING
 *****************************************************************************
 */

Gid_event gidtt_event(Gid *my_port, Gid_machine *machine, Gid_event event)
{  /*
    * Handles receive events and join or leave requests.
    */

    Applicant_tt_entry *atransition;
    Registrar_tt_entry *rtransition;

    atransition = &applicant_tt[event][machine->applicant];
    rtransition = &registrar_tt[event][machine->registrar];

    machine->applicant = atransition->new_app_state;
    machine->registrar = rtransition->new_reg_state;

    if ((event == Gid_join) && (atransition->cstart_join_timer))
    my_port->cschedule_tx_now = True;

    my_port->cstart_join_timer  = my_port->cstart_join_timer
                                || atransition->cstart_join_timer;
    my_port->cstart_leave_timer = my_port->cstart_leave_timer
                                || rtransition->cstart_leave_timer;

    switch (rtransition->indications)
    {
    case Ji: return(Gid_join);
        break;
    case Li: return(Gid_leave);
        break;
    case Ni:
    default: return(Gid_null);
}   }


Boolean gidtt_in(Gid_machine *machine)
{  /*
    *
    */
    return(registrar_in_table[machine->registrar]);
}



/*****************************************************************************
 * GIDTT : GID PROTOCOL TRANSITION TABLES : TRANSMIT MESSAGES
 *****************************************************************************
 */

Gid_event gidtt_tx(Gid         *my_port,
                   Gid_machine *machine)
{  /*
    *
    */
```



```
    unsigned msg;
    unsigned rin;

    if ((msg = applicant_txtt[machine->applicant].msg_to_transmit) != Nm)
        rin = registrar_state_table[machine->registrar];

        my_port->cstart_join_timer  = my_port->cstart_join_timer
            || applicant_txtt[machine->applicant].cstart_join_timer;
    switch (msg)
    {
    case Jm: return(rin != Empty ? Gid_tx_joinin  : Gid_tx_joinempty);
        break;
    case Lm: return(rin != Empty ? Gid_tx_leavein : Gid_tx_leaveempty);
        break;
    case Em: return(Gid_tx_empty);
        break;
    case Nm:
    default: return(Gid_null);
    }
}


/*****************************************************************************
 * GIDTT : GID PROTOCOL TRANSITION TABLES : LEAVE TIMER PROCESSING
 *****************************************************************************
 */

Gid_event gidtt_leave_timer_expiry(Gid         *my_port,
                                   Gid_machine *machine)
{  /*
    *
    */
    Registrar_leave_timer_entry *rtransition;

    rtransition = &registrar_leave_timer_table[machine->registrar];

    machine->registrar = rtransition->new_reg_state;

    my_port->cstart_leave_timer = my_port->cstart_leave_timer
                               || rtransition->cstart_leave_timer;

    return((rtransition->leave_indication == Li) ? Gid_leave : Gid_null);
}


/*****************************************************************************
 * GIDTT : GID PROTOCOL TRANSITION TABLES : STATE REPORTING
 *****************************************************************************
 */

Boolean gidtt_machine_active(Gid_machine *machine)
{
    if ((machine->applicant == Vo) && (machine->registrar == Mt))
        return(False);
    else
        return(True);
}

void gidtt_states(Gid_machine *machine, Gid_states  *state)
{  /*
    *
    */

    state->applicant_state = applicant_state_table[machine->applicant];
```

```
    state->applicant_mgt   = applicant_mgt_table[machine->applicant];

    state->registrar_state = registrar_state_table[machine->registrar];

    state->registrar_mgt   = registrar_mgt_table[machine->registrar];
}
```

13.3.3 gip.c

```
/* gip.c */

#include “gid.h”
#include “gip.h”


/***************************************************************************
 * GIP : GARP INFORMATION PROPAGATION : CREATION, DESTRUCTION
 ***************************************************************************
 */

Boolean gip_create_gip(unsigned max_attributes, unsigned **gip)
{  /*
    * GIP maintains a set of propagation counts for up to max attributes.
    * It currently maintains no additional information, so the GIP instance
    * is represented directly by a pointer to the array of propagation counts.
    */
    unsigned *my_gip;

    if (!sysmalloc(sizeof(unsigned)*max_attributes, &my_gip))
        goto gip_creation_failure;

    *gip = my_gip;    return(True);
gip_creation_failure: return(False);
}

void gip_destroy_gip(void *gip)
{  /*
    *
    */

    sysfree(gip);
}

/***************************************************************************
 * GIP : GARP INFORMATION PROPAGATION : CONNECT, DISCONNECT PORTS
 ***************************************************************************
 */

static void gip_connect_into_ring(Gid *my_port)
{
    Gid *first_connected, *last_connected;

    my_port->is_connected           = True;
    my_port->next_in_connected_ring = my_port;

    first_connected = my_port;

    do      first_connected = first_connected->next_in_port_ring;
    while  (!first_connected->is_connected);

    my_port->next_in_connected_ring = first_connected;

    last_connected = first_connected;
```



```
    while (last_connected->next_in_connected_ring != first_connected)
           last_connected = last_connected->next_in_connected_ring;

    last_connected->next_in_connected_ring = my_port;
}


static void gip_disconnect_from_ring(Gid *my_port)
{
    Gid *first_connected, *last_connected;

    first_connected    = my_port->next_in_connected_ring;
    my_port->next_in_connected_ring = my_port;
    my_port->is_connected           = False;

    last_connected = first_connected;

    while (last_connected->next_in_connected_ring != my_port)
           last_connected = last_connected->next_in_connected_ring;

    last_connected->next_in_connected_ring = first_connected;
}


void gip_connect_port(Garp *application, int port_no)
{  /*
    * If a GID instance for this application and port number is found, is
    * enabled, and is not already connected, then connect that port into the
    * GIP propagation ring.
    *
    * Propagate every attribute that has been registered (i.e., the Registrar
    * appears not to be Empty) on any other connected port, and that has in
    * consequence a nonzero propagation count, to this port, generating a join
    * request.
    *
    * Propagate every attribute that has been registered on this port and not
    * on any others (having a propagation count of zero prior to connecting this
    * port) to all the connected ports, updating propagation counts.
    *
    * Action any timers required. Mark the port as connected.
    *
    */
    Gid      *my_port;
    unsigned  gid_index;

    if (gid_find_port(application->gid, port_no, &my_port))
    {
        if ((!my_port->is_enabled) || (my_port->is_connected)) return;

        gip_connect_into_ring(my_port);

        for (gid_index = 0; gid_index <= application->last_gid_used;
             gid_index++)
        {
            if (gip_propagates_to(my_port, gid_index))
                gid_join_request(my_port, gid_index);

            if (gid_registered_here(my_port, gid_index))
                gip_propagate_join(my_port, gid_index);
        }

        gip_do_actions(my_port);
        my_port->is_connected = True;
}   }
```

```
void gip_disconnect_port(Garp *application, int port_no)
{  /*
    * Reverses the operations performed by gip_connect_port().
    */
    Gid      *my_port;
    unsigned  gid_index;

    if (gid_find_port(application->gid, port_no, &my_port))
    {
        if ((!my_port->is_enabled) || (!my_port->is_connected)) return;

        for (gid_index = 0; gid_index <= application->last_gid_used;
             gid_index++)
        {
            if (gip_propagates_to(my_port, gid_index))
                gid_leave_request(my_port, gid_index);

            if (gid_registered_here(my_port, gid_index))
                gip_propagate_leave(my_port, gid_index);
        }

        gip_do_actions(my_port);
        gip_disconnect_from_ring(my_port);
        my_port->is_connected = False;
}   }


/*****************************************************************************
 * GIP : GARP INFORMATION PROPAGATION : PROPAGATE SINGLE ATTRIBUTES
 *****************************************************************************
 */

void gip_propagate_join(Gid *my_port, unsigned gid_index)
{  /*
    * Propagates a join indication, causing join requests to other ports
    * if required.
    *
    * The join needs to be propagated if either (a) this is the first port in
    * the connected group to register membership, or (b) there is one other port
    * in the group registering membership, but no further port that would cause
    * a join request to that port.
    *
    */
    unsigned  joining_members;
    Gid      *to_port;

    if (my_port->is_connected)
    {
        joining_members = (my_port->application->gip[gid_index] += 1);

        if (joining_members <= 2)
        {
            to_port = my_port;
            while ((to_port = to_port->next_in_connected_ring) != my_port)
            {
                if (   (joining_members == 1)
                    || (gid_registered_here(to_port, gid_index)))
                {
                    gid_join_request(to_port, gid_index);
                    to_port->application->join_propagated_fn(
                                                        my_port->application,
                                                       my_port, gid_index);
}   }   }   }   }
```



```
void gip_propagate_leave(Gid *my_port, unsigned gid_index)
{
   /* Propagates a leave indication for a single attribute, causing leave
    * requests to those other ports if required.
    *
    * See the comments for gip_propagate_join() before reading further.
    * This function decrements the 'dead-reckoning' membership count.
    *
    * The first step is to check that this port is connected to any others; if
    * not, the leave indication should not be propagated, nor should the joined
    * membership be decremented. Otherwise, the leave will need to be propagated
    * if this is either (a) the last port in the connected group to register
    * membership, or (b) there is one other port in the group registering
    * membership, in which case the leave request needs to be sent to that
    * port alone.
    */
    unsigned  remaining_members;
    Gid      *to_port;

    if (my_port->is_connected)
    {
        remaining_members = (my_port->application->gip[gid_index] -= 1);

        if (remaining_members <= 1)
        {
            to_port = my_port;
            while ((to_port = to_port->next_in_connected_ring) != my_port)
            {
                if (   (remaining_members == 0)
                    || (gid_registered_here(to_port, gid_index)))
                {
                    gid_leave_request(to_port, gid_index);
                    to_port->application->leave_propagated_fn(
                                                        my_port->application,
                                                        my_port, gid_index);
}   }   }   }   }


Boolean gip_propagates_to(Gid *my_port, unsigned gid_index)
{
    if (            (my_port->is_connected)
        && (        (my_port->application->gip[gid_index] == 2)
            || (    (my_port->application->gip[gid_index] == 1)
                && (!gid_registered_here(my_port, gid_index)))))
        return(True);
    else return(False);
}

/*****************************************************************************
 * GIP : GARP INFORMATION PROPAGATION : ACTION TIMERS
 *****************************************************************************
 */

void gip_do_actions(Gid *my_port)
{  /*
    * Calls GID to carry out GID 'scratchpad' actions accumulated during this
    * invocation of GARP for all the ports in the GIP ring, including my port.
    */
    Gid *this_port = my_port;

    do gid_do_actions(this_port);
    while ((this_port = this_port->next_in_connected_ring) != my_port);
}
```

14. Bridge Management

Management facilities are provided by MAC Bridges in accordance with the principles and concepts of the OSI Management Framework.

This clause

a) Introduces the Functional Areas of OSI Management to assist in the identification of the requirements placed on Bridges for the support of management facilities.
b) Establishes the correspondence between the Processes used to model the operation of the Bridge (7.3) and the managed objects of the Bridge.
c) Specifies the management operations supported by each managed object.

14.1 Management functions

The Functions of Management relate to the users' needs for facilities that support the planning, organization, supervision, control, protection, and security of communications resources, and account for their use. These facilities may be categorized as supporting the Functional Areas of Configuration, Fault, Performance, Security, and Accounting Management. Each of these is summarized in 14.1.1 through 14.1.5, together with the facilities commonly required for the management of communication resources, and the particular facilities provided in that functional area by Bridge Management.

14.1.1 Configuration Management

Configuration Management provides for the identification of communications resources, initialization, reset and closedown, the supply of operational parameters, and the establishment and discovery of the relationship between resources. The facilities provided by Bridge Management in this functional area are

a) The identification of all Bridges that together make up the Bridged LAN and their respective locations and, as a consequence of that identification, the location of specific end stations to particular individual LANs.
b) The ability to remotely reset, i.e., reinitialize, specified Bridges.
c) The ability to control the priority with which a Bridge Port transmits frames.
d) The ability to force a specific configuration of the spanning tree.
e) The ability to control the propagation of frames with specific group MAC Addresses to certain parts of the configured Bridged LAN.

14.1.2 Fault Management

Fault Management provides for fault prevention, detection, diagnosis, and correction. The facilities provided by Bridge Management in this functional area are

a) The ability to identify and correct Bridge malfunctions, including error logging and reporting.

14.1.3 Performance Management

Performance management provides for evaluation of the behavior of communications resources and of the effectiveness of communication activities. The facilities provided by Bridge Management in this functional area are

a) The ability to gather statistics relating to performance and traffic analysis. Specific metrics include network utilization, frame forward, and frame discard counts for individual Ports within a Bridge.

14.1.4 Security Management

Security Management provides for the protection of resources. Bridge Management does not provide any specific facilities in this functional area.

14.1.5 Accounting Management

Accounting Management provides for the identification and distribution of costs and the setting of charges. Bridge Management does not provide any specific facilities in this functional area.

14.2 Managed objects

Managed objects model the semantics of management operations. Operations upon an object supply information concerning, or facilitate control over, the Process or Entity associated with that object.

The managed resources of a MAC Bridge are those of the Processes and Entities established in 7.3 and 12.2. Specifically

a) The Bridge Management Entity (14.4 and 7.11).
b) The individual MAC Entities associated with each Bridge Port (14.5, 7.2, 7.5, and 7.6).
c) The Forwarding Process of the MAC Relay Entity (14.6, 7.2, and 7.7).
d) The Filtering Database of the MAC Relay Entity (14.7 and 7.9).
e) The Bridge Protocol Entity (14.8, 7.10, and Clause 8).
f) GARP Participants (Clause 12).

The management of each of these resources is described in terms of managed objects and operations below.

NOTE—The values specified in this clause, as inputs and outputs of management operations, are abstract information elements. Questions of formats or encodings are a matter for particular protocols that convey or otherwise represent this information. This standard specifies one such protocol encoding in Clause 15 (for optional remote management).

14.3 Data types

This subclause specifies the semantics of operations independent of their encoding in management protocol. The data types of the parameters of operations are defined only as required for that specification.

The following data types are used:

a) Boolean;
b) Enumerated, for a collection of named values;
c) Unsigned, for all parameters specified as “the number of” some quantity, and for values that are numerically compared. In the case of numeric comparisons of Spanning Tree priority values, the lower number represents the higher priority value;
d) MAC Address;
e) Latin1 String, as defined by ANSI X3.159-1989, for all text strings;
f) Time Interval, an Unsigned value representing a positive integral number of seconds, for all Spanning Tree protocol timeout parameters;
g) Counter, for all parameters specified as a “count” of some quantity. A counter increments and wraps with a modulus of 2 to the power of 64;
h) GARP Time Interval, an Unsigned value representing a positive integral number of centiseconds, for all GARP protocol time-out parameters.

14.4 Bridge Management Entity

The Bridge Management Entity is described in 7.11.

The objects that comprise this managed resource are

a) The Bridge Configuration.
b) The Port Configuration for each Port.

14.4.1 Bridge Configuration

The Bridge Configuration object models the operations that modify, or enquire about, the configuration of the Bridge's resources. There is a single Bridge Configuration object per Bridge.

The management operations that can be performed on the Bridge Configuration are Discover Bridge, Read Bridge, Set Bridge Name, and Reset Bridge.

14.4.1.1 Discover Bridge

14.4.1.1.1 Purpose

To solicit configuration information regarding the Bridge(s) in the Bridged LAN.

14.4.1.1.2 Inputs

a) Inclusion Range, a set of ordered pairs of specific MAC Addresses. Each pair specifies a range of MAC Addresses. A Bridge shall respond if and only if
   1) For one of the pairs, the numerical comparison of its Bridge Address with each MAC Address of the pair shows it to be greater than or equal to the first, and
   2) Less than or equal to the second, and
   3) Its Bridge Address does not appear in the Exclusion List parameter below.

The numerical comparison of one MAC Address with another, for the purpose of this operation, is achieved by deriving a number from the MAC Address according to the following procedure. The consecutive octets of the MAC Address are taken to represent a binary number; the first octet that would be transmitted on a LAN medium when the MAC Address is used in the source or destination fields of a MAC frame has the most significant value, the next octet the next most significant value. Within each octet the first bit of each octet is the least significant bit.

b) Exclusion List, a list of specific MAC Addresses.

14.4.1.1.3 Outputs

a) Bridge Address—the MAC Address for the Bridge from which the Bridge Identifier used by the Spanning Tree Algorithm and Protocol is derived.
b) Bridge Name—a text string of up to 32 characters, of locally determined significance.
c) Number of Ports—the number of Bridge Ports (MAC Entities).
d) Port Addresses—a list specifying the following for each Port:
   1) Port Number—the number of the Bridge Port.
   2) Port Address—the specific MAC Address of the individual MAC Entity associated with the Port.
e) Uptime—count in seconds of the time elapsed since the Bridge was last reset or initialized.

14.4.1.2 Read Bridge

14.4.1.2.1 Purpose

To obtain general information regarding the Bridge.

14.4.1.2.2 Inputs

None.

14.4.1.2.3 Outputs

a) Bridge Address—the MAC Address for the Bridge from which the Bridge Identifier used by the Spanning Tree Algorithm and Protocol is derived.
b) Bridge Name—a text string of up to 32 characters, of locally determined significance.
c) Number of Ports—the number of Bridge Ports (MAC Entities).
d) Port Addresses—a list specifying the following for each Port:
    1) Port Number.
    2) Port Address—the specific MAC Address of the individual MAC Entity associated with the Port.
e) Uptime—count in seconds of the time elapsed since the Bridge was last reset or initialized.

14.4.1.3 Set Bridge Name

14.4.1.3.1 Purpose

To associate a text string, readable by the Read Bridge operation, with a Bridge.

14.4.1.3.2 Inputs

a) Bridge Name—a text string of up to 32 characters.

14.4.1.3.3 Outputs

None.

14.4.1.4 Reset Bridge

14.4.1.4.1 Purpose

To reset the specified Bridge. The Filtering Database is cleared and initialized with the entries specified in the Permanent Database, and the Bridge Protocol Entity is initialized (8.8.1).

14.4.1.4.2 Inputs

None.

14.4.1.4.3 Outputs

None.

14.4.2 Port Configuration

The Port Configuration object models the operations that modify, or inquire about, the configuration of the Ports of a Bridge. There are a fixed set of Bridge Ports per Bridge (one for each MAC interface), and each is identified by a permanently allocated Port Number.

The allocated Port Numbers are not required to be consecutive. Also, some Port Numbers may be dummy entries, with no actual LAN Port (for example, to allow for expansion of the Bridge by addition of further MAC interfaces in the future). Such dummy Ports shall support the Port Configuration management operations, and other Port-related management operations in a manner consistent with the Port being permanently disabled.

The information provided by the Port Configuration consists of summary data indicating its name and type. Specific counter information pertaining to the number of packets forwarded, filtered, and in error is maintained by the Forwarding Process resource. The management operations supported by the Bridge Protocol Entity allow for controlling the states of each Port.

The management operations that can be performed on the Port Configuration are Read Port and Set Port Name.

14.4.2.1 Read Port

14.4.2.1.1 Purpose

To obtain general information regarding a specific Bridge Port.

14.4.2.1.2 Inputs

a) Port Number—the number of the Bridge Port.

14.4.2.1.3 Outputs

a) Port Name—a text string of up to 32 characters, of locally determined significance.
b) Port Type—the MAC Entity type of the Port (IEEE Std 802.3; ISO/IEC 8802-4; ISO/IEC 8802-5; ISO/IEC 8802-6; ISO/IEC 8802-9; IEEE 802.9a; ISO/IEC 8802-12 (IEEE Std 802.3 format); ISO/IEC 8802-12 (ISO/IEC 8802-5 format); IEEE Std 802.11; ISO 9314-2; other).

14.4.2.2 Set port name

14.4.2.2.1 Purpose

To associate a text string, readable by the Read Port operation, with a Bridge Port.

14.4.2.2.2 Inputs

a) Port Number.
b) Port Name—a text string of up to 32 characters.

14.4.2.2.3 Outputs

None.

14.5 MAC Entities

The Management Operations and Facilities provided by the MAC Entities are those specified in the Layer Management standards of the individual MACs. A MAC Entity is associated with each Bridge Port.

14.6 Forwarding Process

The Forwarding Process contains information relating to the forwarding of frames. Counters are maintained that provide information on the number of frames forwarded, filtered, and dropped due to error. Configuration data, defining how frame priority is handled, is maintained by the Forwarding Process.

The objects that comprise this managed resource are

a) The Port Counters;
b) The Priority Handling objects for each Port;
c) The Traffic Class Table for each Port.

14.6.1 The Port Counters

The Port Counters object models the operations that can be performed on the Port counters of the Forwarding Process resource. There are multiple instances (one for each MAC Entity) of the Port Counters object per Bridge.

The management operation that can be performed on the Port Counters is Read Forwarding Port Counters.

14.6.1.1 Read Forwarding Port Counters

14.6.1.1.1 Purpose

To read the forwarding counters associated with a specific Bridge Port.

14.6.1.1.2 Inputs

a) Port Number.

14.6.1.1.3 Outputs

a) Frames Received—count of all valid frames received (including BPDUs, frames addressed to the Bridge as an end station, and frames that were submitted to the forwarding process).
b) Discard Inbound—count of valid frames received that were discarded by the Forwarding Process.
c) Forward Outbound—count of frames forwarded to the associated MAC Entity.
d) Discard Lack of Buffers—count of frames that were to be transmitted through the associated Port but were discarded due to lack of buffers.
e) Discard Transit Delay Exceeded—count of frames that were to be transmitted but were discarded due to the maximum bridge transit delay being exceeded (buffering may have been available).
f) Discard on Error—count of frames that were to be forwarded on the associated MAC but could not be transmitted (e.g., frame would be too large).
g) Discard on Error Details—a list of 16 elements, each containing the source address of a frame and the reason why the frame was discarded (frame too large). The list is maintained as a circular buffer. The only reason for discard on error, at present, is transmissible service data unit size exceeded.

14.6.2 Priority Handling

The Priority Handling object models the operations that can be performed upon, or inquire about, the Default User Priority parameter, the User Priority Regeneration Table parameter, and the Outbound Access Priority Table parameter for each Port. The operations that can be performed on this object are Read Port Default User Priority, Set Port Default User Priority, Read Port User Priority Regeneration Table, Set Port User Priority Regeneration Table, and Read Outbound Access Priority Table.

14.6.2.1 Read Port Default User Priority

14.6.2.1.1 Purpose

To read the current state of the Default User Priority parameter (6.4) for a specific Bridge Port.

14.6.2.1.2 Inputs

a) Port number.

14.6.2.1.3 Outputs

a) Default User Priority value—Integer in range 0–7.

14.6.2.2 Set Port Default User Priority

14.6.2.2.1 Purpose

To set the current state of the Default User Priority parameter (6.4) for a specific Bridge Port.

14.6.2.2.2 Inputs

a) Port number;
b) Default User Priority value—Integer in range 0–7.

14.6.2.2.3 Outputs

None.

14.6.2.3 Read Port User Priority Regeneration Table

14.6.2.3.1 Purpose

To read the current state of the User Priority Regeneration Table parameter (7.5.1) for a specific Bridge Port.

14.6.2.3.2 Inputs

a) Port number.

14.6.2.3.3 Outputs

a) Regenerated User Priority value for Received User Priority 0—Integer in range 0–7.
b) Regenerated User Priority value for Received User Priority 1—Integer in range 0–7.
c) Regenerated User Priority value for Received User Priority 2—Integer in range 0–7.
d) Regenerated User Priority value for Received User Priority 3—Integer in range 0–7.
e) Regenerated User Priority value for Received User Priority 4—Integer in range 0–7.

f) Regenerated User Priority value for Received User Priority 5—Integer in range 0–7.
g) Regenerated User Priority value for Received User Priority 6—Integer in range 0–7.
h) Regenerated User Priority value for Received User Priority 7—Integer in range 0–7.

14.6.2.4 Set Port User Priority Regeneration Table

14.6.2.4.1 Purpose

To set the current state of the User Priority Regeneration Table parameter (7.5.1) for a specific Bridge Port.

14.6.2.4.2 Inputs

a) Port number;
b) Regenerated User Priority value for Received User Priority 0—Integer in range 0–7.
c) Regenerated User Priority value for Received User Priority 1—Integer in range 0–7.
d) Regenerated User Priority value for Received User Priority 2—Integer in range 0–7.
e) Regenerated User Priority value for Received User Priority 3—Integer in range 0–7.
f) Regenerated User Priority value for Received User Priority 4—Integer in range 0–7.
g) Regenerated User Priority value for Received User Priority 5—Integer in range 0–7.
h) Regenerated User Priority value for Received User Priority 6—Integer in range 0–7.
i) Regenerated User Priority value for Received User Priority 7—Integer in range 0–7.

14.6.2.4.3 Outputs

None.

14.6.3 Traffic Class Table

The Traffic Class Table object models the operations that can be performed upon, or inquire about, the current contents of the Traffic Class Table for a given Port. The operations that can be performed on this object are Read Port Traffic Class Table and Set Port Traffic Class Table.

14.6.3.1 Read Port Traffic Class Table

14.6.3.1.1 Purpose

To read the contents of the Traffic Class Table (7.7.3) for a given Port.

14.6.3.1.2 Inputs

a) Port Number.

14.6.3.1.3 Outputs

a) The number of Traffic Classes, in the range 1 through 8, supported on the Port;
b) For each value of Traffic Class supported on the Port, the value of the Traffic Class in the range 0 through 7, and the set of user_priority values assigned to that Traffic Class.

14.6.3.2 Set Port Traffic Class Table

14.6.3.2.1 Purpose

To set the contents of the Traffic Class Table (7.7.3) for a given Port.

14.6.3.2.2 Inputs

a) Port number;
b) For each value of Traffic Class supported on the Port, the value of the Traffic Class in the range 0 through 7, and the set of user_priority values assigned to that Traffic Class.

NOTE—If a Traffic Class value greater than the largest Traffic Class available on the Port is specified, then the value applied to the Traffic Class Table is the largest available Traffic Class.

14.6.3.2.3 Outputs

None.

14.6.3.3 Read Outbound Access Priority Table

14.6.3.3.1 Purpose

To read the state of the Outbound Access Priority Table parameter (Table 7-3) for a specific Bridge Port.

14.6.3.3.2 Inputs

a) Port number.

14.6.3.3.3 Outputs

a) Access Priority value for User Priority 0—Integer in range 0–7.
b) Access Priority value for User Priority 1—Integer in range 0–7.
c) Access Priority value for User Priority 2—Integer in range 0–7.
d) Access Priority value for User Priority 3—Integer in range 0–7.
e) Access Priority value for User Priority 4—Integer in range 0–7.
f) Access Priority value for User Priority 5—Integer in range 0–7.
g) Access Priority value for User Priority 6—Integer in range 0–7.
h) Access Priority value for User Priority 7—Integer in range 0–7.

14.7 Filtering Database

The Filtering Database is described in 7.9. It contains filtering information used by the Forwarding Process (7.7) in deciding through which Ports of the Bridge frames should be forwarded.

The objects that comprise this managed resource are

a) The Filtering Database;
b) The Static Filtering Entries;
c) The Dynamic Filtering Entries;
d) The Group Registration Entries;
e) The Permanent Database.

14.7.1 The Filtering Database

The Filtering Database object models the operations that can be performed on, or affect, the Filtering Database as a whole. There is a single Filtering Database object per Bridge.

The management operations that can be performed on the Database are Read Filtering Database, Set Filtering Database Ageing Time, and the Create Filtering Entry, Delete Filtering Entry, Read Filtering Entry, and Read Filtering Entry Range operations defined in 14.7.6.

14.7.1.1 Read Filtering Database

14.7.1.1.1 Purpose

To obtain general information regarding the Bridge's Filtering Database.

14.7.1.1.2 Inputs

None.

14.7.1.1.3 Outputs

a) Filtering Database Size—the maximum number of entries that can be held in the Filtering Database.
b) Number of Static Filtering Entries—the number of Static Filtering Entries currently in the Filtering Database;
c) Number of Dynamic Filtering Entries—the number of Dynamic Filtering Entries currently in the Filtering Database;
d) Ageing Time—for ageing out Dynamic Filtering Entries when the Port associated with the entry is in the Forwarding state;
e) If extended filtering services are supported, Number of Group Registration Entries—the number of Group Registration Entries currently in the Filtering Database.

14.7.1.2 Set Filtering Database ageing time

14.7.1.2.1 Purpose

To set the ageing time for Dynamic Filtering Entries.

14.7.1.2.2 Inputs

a) Ageing Time.

14.7.1.2.3 Outputs

None.

14.7.2 A Static Filtering Entry

A Static Filtering Entry object models the operations that can be performed on a single Static Filtering Entry in the Filtering Database. The set of Static Filtering Entry objects within the Filtering Database changes only under management control.

A Static Filtering Entry object supports the Create Filtering Entry, Delete Filtering Entry, Read Filtering Entry, and Read Filtering Entry Range operations defined in 14.7.6.

14.7.3 A Dynamic Filtering Entry

A Dynamic Filtering Entry object models the operations that can be performed on a single Dynamic Filtering Entry (i.e., one that is created by the Learning Process as a result of the observation of network traffic) in the Filtering Database.

A Dynamic Filtering Entry object supports the Delete Filtering Entry, Read Filtering Entry, and Read Filtering Entry Range operations defined in 14.7.6.

14.7.4 A Group Registration Entry

A Group Registration Entry object models the operations that can be performed on a single Group Registration Entry in the Filtering Database. The set of Group Registration Entry objects within the Filtering Database changes only as a result of GARP protocol exchanges.

A Group Registration Entry object supports the Read Filtering Entry and Read Filtering Entry Range operations defined in 14.7.6.

14.7.5 Permanent Database

The Permanent Database object models the operations that can be performed on, or affect, the Permanent Database. There is a single Permanent Database per Filtering Database.

The management operations that can be performed on the Permanent Database are Read Permanent Database, and the Create Filtering Entry, Delete Filtering Entry, Read Filtering Entry, and Read Filtering Entry Range operations defined in 14.7.6.

14.7.5.1 Read Permanent Database

14.7.5.1.1 Purpose

To obtain general information regarding the Permanent Database.

14.7.5.1.2 Inputs

None.

14.7.5.1.3 Outputs

a) Permanent Database Size—maximum number of entries that can be held in the Permanent Database.
b) Number of Static Filtering Entries—number of Static Filtering Entries currently in the Permanent Database.

14.7.6 General Filtering Database operations

14.7.6.1 Create Filtering Entry

14.7.6.1.1 Purpose

To create or update a Filtering Entry in the Filtering Database or Permanent Database. Only Static Filtering Entries may be created in the Filtering Database or Permanent Database.

14.7.6.1.2 Inputs

a) Identifier—Filtering Database or Permanent Database.
b) Address—MAC Address of the entry.
c) Inbound Port—the Inbound Port to which the operation applies. This parameter specifies either
    1) All Inbound Ports, or
    2) A Port number.
d) Port Map—a set of control indicators, one for each Port, as specified in 7.9.1.

Where the implementation does not support the creation of more than one Static Filtering Entry for the address specified, the value of the Inbound Port parameter is assumed to specify All Inbound Ports.

Where the implementation does not support the ability for static filtering entries to specify the use of dynamic filtering information (7.9.1), the use of this operation to create a Static Filtering Entry in the Filtering Database with the same MAC Address as an existing Dynamic Filtering Entry will cause the existing entry to be replaced by the (new) Static Filtering Entry.

Where the implementation supports the creation of multiple Static Filtering Entries for the same MAC Address (7.9.1), the creation of a new Static Filtering Entry will cause any existing Static Filtering Entry for the same Inbound Port and MAC Address to be replaced by the (new) Static Filtering Entry. The creation of a Static Filtering Entry for All Inbound Ports causes all existing Static Entries for the same MAC Address to be replaced by the (new) Static Filtering Entry.

14.7.6.1.3 Outputs

None.

14.7.6.2 Delete Filtering Entry

14.7.6.2.1 Purpose

To delete a Filtering Entry from the Filtering Database or Permanent Database.

14.7.6.2.2 Inputs

a) Identifier—Filtering Database or Permanent Database.
b) Address—MAC Address of the desired entry.
c) Inbound Port—the Inbound Port to which the operation applies. This parameter specifies either
    1) All Inbound Ports, or
    2) A Port number.

Where the implementation does not support the creation of more than one Static Filtering Entry for the address specified, the value of the Inbound Ports parameter is assumed to specify All Inbound Ports.

Where the implementation supports the creation of more than one Static Filtering Entry for the address specified, a value of the Inbound Ports parameter of All Inbound Ports results in deletion of all Static Filtering Entries for the MAC Address specified.

14.7.6.2.3 Outputs

None.

14.7.6.3 Read Filtering Entry

14.7.6.3.1 Purpose

To read Filtering Entry and Group Registration Entry information from the Filtering or Permanent Databases. This operation returns both the static and dynamic information held in a given database for a given MAC Address and inbound Port specification.

14.7.6.3.2 Inputs

a) Identifier—Filtering Database or Permanent Database.

b) Address—MAC Address of the desired information.
c) Type—Static or Dynamic entry.
d) If Type = Static entry, then Inbound Port—the Inbound Port to which the operation applies. This parameter specifies either
    1) All Inbound Ports, or
    2) A Port number.

Where the implementation does not support the creation of more than one Static Filtering Entry for the address specified, the value of the Inbound Ports parameter is assumed to specify All Inbound Ports.

NOTE—Dynamic entry types are Dynamic Filtering Entries and Group Registration Entries.

14.7.6.3.3 Outputs

a) Address—MAC Address of the desired entry.
b) Type—Static or Dynamic entry.
c) Port Map—A set of control indicators as appropriate to the entry type, as specified in 7.9.1 through 7.9.3.

14.7.6.4 Read Filtering Entry range

14.7.6.4.1 Purpose

To read a range of Filtering Entries and/or Group Registration Entries from the Filtering or Permanent Databases.

Since the number of values to be returned in the requested range may have exceeded the capacity of the service data unit conveying the management response, the returned entry range is identified. The indices that define the range take on values from zero up to Filtering Database Size minus one.

14.7.6.4.2 Inputs

a) Identifier—Filtering Database or Permanent Database.
b) Start Index—inclusive starting index of the desired entry range.
c) Stop Index—inclusive ending index of the desired range.

14.7.6.4.3 Outputs

a) Start Index—inclusive starting index of the returned entry range.
b) Stop Index—inclusive ending index of the returned entry range.
c) For each index of the returned entry range, the following are returned:
    1) Address—MAC Address of the desired entry.
    2) Type—Static or Dynamic entry.
    3) Port Map—A set of control indicators as appropriate to the entry type, as specified in 7.9.1 through 7.9.3.

14.8 Bridge Protocol Entity

The Bridge Protocol Entity is described in 7.10 and Clause 8. The objects that comprise this managed resource are

a) The Protocol Entity itself and
b) The Ports under its control.

14.8.1 The Protocol Entity

The Protocol Entity object models the operations that can be performed upon, or inquire about, the operation of the Spanning Tree Algorithm and Protocol. There is a single Protocol Entity per Bridge; it can, therefore, be identified as a single fixed component of the Protocol Entity resource.

The management operations that can be performed on the Protocol Entity are Read Bridge Protocol Parameters and Set Bridge Protocol Parameters.

14.8.1.1 Read Bridge Protocol parameters

14.8.1.1.1 Purpose

To obtain information regarding the Bridge's Bridge Protocol Entity.

14.8.1.1.2 Inputs

None.

14.8.1.1.3 Outputs

a) Bridge Identifier—as defined in 8.5.3.7.
b) Time Since Topology Change—count in seconds of the time elapsed since the Topology Change flag parameter for the Bridge (8.5.3.12) was last True.
c) Topology Change Count—count of the times the Topology Change flag parameter for the Bridge has been set (i.e., transitioned from False to True) since the Bridge was powered on or initialized.
d) Topology Change (8.5.3.12).
e) Designated Root (8.5.3.1).
f) Root Path Cost (8.5.3.2).
g) Root Port (8.5.3.3).
h) Max Age (8.5.3.4).
i) Hello Time (8.5.3.5).
j) Forward Delay (8.5.3.6).
k) Bridge Max Age (8.5.3.7).
l) Bridge Hello Time (8.5.3.9).
m) Bridge Forward Delay (8.5.3.10).
n) Hold Time (8.5.3.14).

14.8.1.2 Set Bridge Protocol parameters

14.8.1.2.1 Purpose

To modify parameters in the Bridge's Bridge Protocol Entity in order to force a configuration of the spanning tree and/or tune the reconfiguration time to suit a specific topology.

14.8.1.2.2 Inputs

a) Bridge Max Age—the new value (8.5.3.8).
b) Bridge Hello Time—the new value (8.5.3.9).
c) Bridge Forward Delay—the new value (8.5.3.10).
d) Bridge Priority—the new value of the priority part of the Bridge Identifier (8.5.3.7).

14.8.1.2.3 Outputs

None.

14.8.1.2.4 Procedure

The input parameter values are checked for compliance with 8.10.2. If they do not comply, or the value of Bridge Max Age or Bridge Forward Delay is less than the lower limit of the range specified in Table 8-3, then no action shall be taken for any of the supplied parameters. If the value of any of Bridge Max Age, Bridge Forward Delay, or Bridge Hello Time is outside the range specified in Table 8-3, then the Bridge need not take action.

Otherwise, the Bridge's Bridge Max Age, Bridge Hello Time, and Bridge Forward Delay parameters are set to the supplied values. The Set Bridge Priority procedure (8.8.4) is used to set the priority part of the Bridge Identifier to the supplied value.

14.8.2 Bridge Port

A Bridge Port object models the operations related to an individual Bridge Port in relation to the operation of the Spanning Tree Algorithm and Protocol. There are a fixed set of Bridge Ports per Bridge; each can, therefore, be identified by a permanently allocated Port Number, as a fixed component of the Protocol Entity resource. The management operations that can be performed on a Bridge Port are Read Port Parameters, Force Port State, and Set Port Parameters.

14.8.2.1 Read Port Parameters

14.8.2.1.1 Purpose

To obtain information regarding a specific Port within the Bridge's Bridge Protocol Entity.

14.8.2.1.2 Inputs

a) Port Number—the number of the Bridge Port.

14.8.2.1.3 Outputs

a) Uptime—count in seconds of the time elapsed since the Port was last reset or initialized.
b) State—the current state of the Port (i.e., Disabled, Listening, Learning, Forwarding, or Blocking) (8.4 and 8.5.5.2).
c) Port Identifier—the unique Port identifier comprising two parts, the Port Number and the Port Priority field (8.5.5.1).
d) Path Cost (8.5.5.3).
e) Designated Root (8.5.5.4).
f) Designated Cost (8.5.5.5).
g) Designated Bridge (8.5.5.6).
h) Designated Port (8.5.5.7).
i) Topology Change Acknowledge (8.5.5.8).

14.8.2.2 Force Port State

14.8.2.2.1 Purpose

To force the specified Port into Disabled or Blocking.

14.8.2.2.2 Inputs

a) Port Number—the number of the Bridge Port.
b) State—either Disabled or Blocking (8.4 and 8.5.5.2).

14.8.2.2.3 Outputs

None.

14.8.2.2.4 Procedure

If the selected state is Disabled, the Disable Port procedure (8.8.3) is used for the specified Port. If the selected state is Blocking, the Enable Port procedure (8.8.2) is used.

14.8.2.3 Set Port Parameters

14.8.2.3.1 Purpose

To modify parameters for a Port in the Bridge's Bridge Protocol Entity in order to force a configuration of the spanning tree.

14.8.2.3.2 Inputs

a) Port Number—the number of the Bridge Port.
b) Path Cost—the new value (8.5.5.3).
c) Port Priority—the new value of the priority field for the Port Identifier (8.5.5.1).

14.8.2.3.3 Outputs

None.

14.8.2.3.4 Procedure

The Set Path Cost procedure (8.8.6) is used to set the Path Cost parameter for the specified Port. The Set Port Priority procedure (8.8.5) is used to set the priority part of the Port Identifier (8.5.5.1) to the supplied value.

14.9 GARP Entities

The operation of GARP is described in Clause 12. The objects that comprise this managed resource are

a) The GARP Timer objects;
b) The GARP Attribute Type objects;
c) The GARP State Machine objects.

14.9.1 The GARP Timers object

The GARP Timer object models the operations that can be performed upon, or inquire about, the current settings of the timers used by the GARP protocol on a given Port. The management operations that can be performed on the GARP Timers object are Read GARP Timers and Set GARP Timers.

14.9.1.1 Read GARP Timers

14.9.1.1.1 Purpose

To read the current values of the GARP Timers for a given Port.

14.9.1.1.2 Inputs

a) The Port identifier.

14.9.1.1.3 Outputs

a) Current value of JoinTime—Centiseconds;
b) Current value of LeaveTime—Centiseconds;
c) Current value of LeaveAllTime—Centiseconds.

14.9.1.2 Set GARP Timers

14.9.1.2.1 Purpose

To set new values for the GARP Timers for a given Port.

14.9.1.2.2 Inputs

a) The Port identifier;
b) New value of JoinTime—Centiseconds;
c) New value of LeaveTime—Centiseconds;
d) New value of LeaveAllTime—Centiseconds.

14.9.1.2.3 Outputs

None.

14.9.2 The GARP Attribute Type object

The GARP Attribute Type object models the operations that can be performed upon, or inquire about, the operation of GARP for a given Attribute Type. The management operations that can be performed on a GARP Attribute Type are Read GARP Applicant Controls and Set GARP Applicant Controls.

14.9.2.1 Read GARP Applicant controls

14.9.2.1.1 Purpose

To read the current values of the GARP Applicant Administrative parameters (12.9.2) associated with all GARP Participants for a given Port, GARP Application and Attribute Type.

14.9.2.1.2 Inputs

a) The Port identifier;
b) The GARP Application address (Table 12-1);
c) The Attribute Type (12.11.2.2).

14.9.2.1.3 Outputs

a) The current Applicant Administrative Control Value (12.9.2);
b) Failed Registrations—Count of the number of times that this GARP Application has failed to register an attribute of this type due to lack of space in the Filtering Database.

14.9.2.2 Set GARP Applicant controls

14.9.2.2.1 Purpose

To set new values for the GARP Applicant Administrative control parameters (12.9.2) associated with all GARP Participants for a given Port, GARP Application and Attribute Type.

14.9.2.2.2 Inputs

a) The Port identifier;
b) The GARP Application address (Table 12-1);
c) The Attribute Type (12.11.2.2);
d) The desired Applicant Administrative Control Value (12.9.2).

14.9.2.2.3 Outputs

None.

14.9.3 The GARP State Machine object

The GARP State Machine object models the operations that can be performed upon, or inquire about, the operation of GARP for a given State Machine. The management operation that can be performed on a GARP State Machine is Read GARP State.

14.9.3.1 Read GARP State

14.9.3.1.1 Purpose

To read the current value of an instance of a GARP State Machine.

14.9.3.1.2 Inputs

a) The Port identifier;
b) The GARP Application address (Table 12-1);
c) The GIP Context (12.3.4);
d) The Attribute Type (12.11.2.2) associated with the State Machine;
e) The Attribute Value (12.11.2.6) associated with the State Machine.

14.9.3.1.3 Outputs

a) The current value of the combined Applicant and Registrar state machine for the attribute (Table 12-6);
b) Optionally, Originator address—the MAC Address of the originator of the most recent GARP PDU that was responsible for causing a state change in this state machine (12.9.1).

15. Management protocol

This clause specifies how the management facilities provided by MAC Bridges and specified in Clause 14 are realized through the use of

a) The LAN/MAN Management services (LMMS), provided by means of the LAN/MAN Management protocol (LMMP) and the Convergence Protocol Entity (CPE), as specified in ISO/IEC 15802-2; or
b) The Common Management Information Service (CMIS), provided by means of the Common Management Information Protocol (CMIP) and Presentation P-DATA services, as specified in ISO/IEC 9595 and ISO/IEC 9596-1; or
c) Other management protocol(s) capable of conveying equivalent information.

NOTE 1—With the exception of the support of association control services, the definitions of the service primitives for LMMS and CMIS are identical, and the PDU formats and procedures specified for LMMP and CMIP are identical. The difference between these two options therefore lies in the complexity of the protocol stack that is required in order to support LMMP (the CPE plus LLC, MAC. and PHY) or CMIP (presentation service carried over a "full OSI stack"), not in the definition of the services or the information that they convey. For simplicity, the remainder of this clause will therefore refer only to LMMS and LMMP.
NOTE 2—The IETF has developed an equivalent protocol specification, RFC 1493, which defines the Bridge MIB for use with the Simple Network ManagementProtocol (SNMP). RFC 1493 defines a MIB corresponding to the management functionality defined in the ISO/IEC 10038: 1993 edition of the MAC Bridge standard; it is undergoing revision in order to reflect the management functionality defined in this edition of the MAC Bridge standard.

It specifies

a) The mapping between the management operations on objects as defined in Clause 14 and the LMMS Services that the LMMP provides;
b) The managed object classes (and their components) that are instantiated in order to provide the managed objects defined in Clause 14. These managed object class definitions are documented using the template notation specified in ISO/IEC 10165-4; other aspects of the documentation are in accordance with the further guidance provided by IEEE Std. 802.1F-1993.
c) The name-bindings that are necessary in order to specify how the managed object classes may be named, and the permitted containment relationships between the managed object classes. A name-binding is also provided in order to correctly position the MAC Bridge object containment within the containment hierarchy defined for management of the Data Link layer in ISO/IEC 10742.

15.1 Mapping of operations onto LMMS Services

The operations on objects defined in Clause 14 are carried out by means of the following LMMS Services:

a) M-GET
b) M-SET
c) M-ACTION
d) M-CREATE
e) M-DELETE
f) M-CANCEL-GET

Table 15-1 through Table 15-4 show how the management operations defined in Clause 14 correspond to LMMS service primitives (defined in ISO/IEC 15802-2) and managed object classes (defined in 15.3 through 15.8). Table 15-6 shows the operations available for manipulation of the Selective Translation Table, as defined in ISO/IEC 11802-5.

Table 15-7 through Table 15-33 specify the detailed mapping of the Management Operation parameters to the parameters of the LMMS service primitives.

Table 15-1—Mapping of Bridge Management Entity Operations to LMMS Services

Management Operation	LMMS Service Element(s)	Managed Object Class
Discover Bridge (14.4.1.1)	M-GET	MAC Bridge DLE (15.3)
Read Bridge (14.4.1.2)	M-GET	MAC Bridge DLE (15.3)
Set Bridge Name (14.4.1.3)	M-SET	MAC Bridge DLE (15.3)
Reset Bridge (14.4.1.4)	M-ACTION	MAC Bridge DLE (15.3)
Read Port (14.4.2.1)	M-GET	Port (15.4)
Set Port Name (14.4.2.2)	M-SET	Port (15.4)

Table 15-2—Mapping of Forwarding Process Operations to LMMS Services

Management Operation	LMMS Service Element(s)	Managed Object Class
Read Forwarding Port Counters (14.6.1.1)	M-GET	Port (15.4)
Read Port Default User Priority (14.6.2.1)	M-GET	Port (15.4)
Set Port Default User Priority (14.6.2.2)	M-SET	Port (15.4)
Read Port User Priority Regeneration Table (14.6.2.3)	M-GET	Port (15.4)
Set Port User Priority Regeneration Table (14.6.2.4)	M-SET	Port (15.4)
Read Port Traffic Class Table (14.6.3.1)	M-GET	Port (15.4)
Set Port Traffic Class Table (14.6.3.2)	M-SET	Port (15.4)
Read Outbound Access Priority Table (14.6.3.3)	M-GET	Port (15.4)

Table 15-3—Mapping of Filtering Database Operations to LMMS Services

Management Operation	LMMS Service Element(s)	Managed Object Class
Read Filtering Database (14.7.1.1)	M-GET	Filtering Database (15.7)
Set Filtering Database Ageing Time (14.7.1.2)	M-SET	Filtering Database (15.7)
Read Permanent Database (14.7.5.1)	M-GET	Permanent Database (15.7)
Create Filtering Entry (14.7.6.1)	M-CREATE	Database Entry (15.8)
Delete Filtering Entry (14.7.6.2)	M-DELETE	Database Entry (15.8)
Read Filtering Entry (14.7.6.3)	M-GET	Database Entry (15.8)
Read Filtering Entry Range (14.7.6.4)	M-GET, M-CANCEL-GET	Database Entry (15.8)

Table 15-4—Mapping of Bridge Protocol Entity Operations to LMMS Services

Management Operation	LMMS Service Element(s)	Managed Object Class
Read Bridge Protocol Parameters (14.8.1.1)	M-GET	MAC Bridge DLE (15.3)
Set Bridge Protocol Parameters (14.8.1.2)	M-SET	MAC Bridge DLE (15.3)
Read Port Parameters (14.8.2.1)	M-GET	Port (15.4)
Force Port State (14.8.2.2)	M-ACTION	Port (15.4)
Set Port Parameters (14.8.2.3)	M-SET	Port (15.4)

The set of Management Operations identified in Table 15-1 through Table 15-6, along with the mappings in Table 15-7 through Table 15-38, define a Functional Unit Package that may be negotiated between manager and agent stations, as described in Annex A.3.2 of ISO/IEC 10040.

Table 15-5—Mapping of GARP Participant Operations to LMMS Services

Management Operation	LMMS Service Element(s)	Managed Object Class
Read GARP Timers (14.9.1.1)	M-GET	GARP Timers (15.12)
Set GARP Timers (14.9.1.2)	M-SET	GARP Timers (15.12)
Read GARP Applicant Controls (14.9.2.1)	M-GET	GARP Attribute Type (15.10)
Set GARP Applicant Controls (14.9.2.2)	M-SET	GARP Attribute Type (15.10)
Read GARP State (14.9.3.1)	M-GET	GARP Attribute (15.11)

Table 15-6—Mapping of Selective Translation Table Operations to LMMS Services

Management Operation	LMMS Service Element(s)	Managed Object Class
Read Selective Translation Table Entry Range (See ISO/IEC 11802-5)	M-GET, M-CANCEL-GET	Selective Translation Table Entry (15.5)

This standard assigns the following object identifier value:

{iso(1) member-body(2) us(840) ieee802dot1D(10009) functionalUnitPackage(1) bridgeManagementFunctionalUnit(1)}

as a value of the functionalUnitPackageId field of the ASN.1 type FunctionalUnitPackage defined in ISO/IEC 10040 to use for negotiating the use of this set of Management Operations.

The Bridge Functional Unit Package consists of three Functional Units, which are identified in the managerRoleFunctionalUnit and agentRoleFunctionalUnit fields of the ASN.1 type FunctionalUnitPackage:

Bit 0: Identifies support (value 1) or non-support (value 0) of the set of Management Operations defined in Table 15-1 through Table 15-5, and their mappings onto LMMS as described in Table 15-7 through Table 15-32;

Bit 1: Identifies support (value 1) or non-support (value 0) of the Management Operation defined in Table 15-6, and its mapping onto LMMS as described in Table 15-33;

Bit 2: Identifies support (value 1) or non-support (value 0) of the Management Operations defined in Table 15-5, and their mappings onto LMMS as described in Table 15-34 through Table 15-38.

NOTE—When using the service mappings described in Table 15-7 through Table 15-38 in conjunction with the LAN/MAN Management Protocol defined in ISO/IEC 15802-2, the Management Operations can be performed upon single Bridges, by addressing the bridge by use of its individual MAC Address, or upon sets of bridges by use of group MAC Addresses. In particular, the All LANs Bridge Management Group Address identified in Table 7-5 can be used to address Management Operations to all Bridges in a Bridged LAN. The use of group addressing is not available when using ISO/IEC 9596-1 (CMIP) over a full OSI stack.

Table 15-7 through Table 15-38 show only the LMMS service parameters for which this standard requires values to be assigned in specific ways. Any LMMS parameters not specifically identified are present, absent, optional, or conditional as described in the definitions of the LMMS service primitives themselves.

15.2 Managed object containment structure

The apex of the containment structure is the MAC Bridge DLE (Data Link Entity) managed object; there may be zero or more MAC Bridge DLE managed objects in a Data Link Subsystem. The MAC Bridge DLE managed objects are contained in an instance of the Data Link Subsystem managed object class, as defined in ISO/IEC 10742.

One or more Port managed objects are contained in each MAC Bridge DLE managed object. The Port managed object models the manageable properties of a single Port of the MAC Bridge. These entries are instantiated at initialization, and are not created or deleted dynamically.

Table 15-7—Mapping of Discover Bridge parameters (14.4.1.1)

M-GET Parameter Name	Req/Ind (Operation Inputs)	Rsp/Conf (Operation Outputs)
Base object class	Data Link Subsystem (see ISO/IEC 10742)	—
Base object instance	Name of Data Link Subsystem managed object	—
Scope	1st level subordinates	—
Filter	Managed object class = MAC Bridge DLE **AND** Bridge Address attribute value is within inclusion range **AND** Bridge Address attribute value is not within exclusion list	—
Attribute identifier list	Read or Discover Bridge attribute group name (15.3.22)	—
Managed object class	—	MAC Bridge DLE
Managed object instance	—	Name of MAC Bridge DLE managed object
Attribute list	—	Names/values of all attributes that are members of the Read Or Discover Bridge attribute group (15.3.22)

Table 15-8—Mapping of Read Bridge parameters (14.4.1.2)

M-GET Parameter Name	Req/Ind (Operation Inputs)	Rsp/Conf (Operation Outputs)
Base object class	MAC Bridge DLE (15.3)	—
Base object instance	Name of MAC Bridge DLE managed object	—
Scope	Base object alone	—
Filter	Not required	—
Attribute identifier list	Read or Discover Bridge attribute group name (15.3.22)	—
Managed object class	—	MAC Bridge DLE (15.3)
Managed object instance	—	Name of MAC Bridge DLE managed object
Attribute list	—	Names/values of all attributes that are members of the Read Or Discover Bridge attribute group (15.3.22)

Table 15-9—Mapping of Set Bridge Name parameters (14.4.1.3)

M-SET Parameter Name	Req/Ind (Operation Inputs)
Base object class	MAC Bridge DLE (15.3)
Base object instance	Name of MAC Bridge DLE managed object
Scope	Base object alone
Filter	Not used
Modification list	Bridge Name attribute (15.3.3) name and desired replacement value

Zero or more Selective Translation Table Entry managed objects may be contained within each Port managed object. These table entries model the information contained in the Selective Translation Table for a given port, as described in 5.2 of ISO/IEC 11802-5. The entries may be created or deleted dynamically by remote management request.

Table 15-10—Mapping of Reset Bridge parameters (14.4.1.4)

M-ACTION Parameter Name	Req/Ind (Operation Inputs)
Base object class	MAC Bridge DLE (15.3)
Base object instance	Name of MAC Bridge DLE managed object
Scope	Base object alone
Filter	Not required
Action Type	Reset Bridge
Action Information	Not used
Action Reply	Not used

Table 15-11—Mapping of Read Port parameters (14.4.2.1)

M-GET Parameter Name	Req/Ind (Operation Inputs)	Rsp/Conf (Operation Outputs)
Base object class	Port (15.4)	—
Base object instance	Name of Port managed object	—
Scope	Base object alone	—
Filter	Not required	—
Attribute identifier list	Read Port attribute group name (15.4.26)	—
Managed object class	—	Port (15.4)
Managed object instance	—	Name of Port managed object
Attribute list	—	Names/values of all attributes that are members of the Read Port attribute group (15.4.26)

Table 15-12—Mapping of Set Port Name parameters (14.4.2.2)

M-SET Parameter Name	Req/Ind (Operation Inputs)
Base object class	Port (15.4)
Base object instance	Name of Port managed object
Scope	Base object alone
Filter	Not used
Modification list	Port Name attribute (15.4.3) name and desired replacement value

Table 15-13—Mapping of Read Forwarding Port Counters parameters (14.6.1.1)

M-GET Parameter Name	Req/Ind (Operation Inputs)	Rsp/Conf (Operation Outputs)
Base object class	Port (15.4)	—
Base object instance	Name of Port managed object	—
Scope	Base object alone	—
Filter	Not required	—
Attribute identifier list	Read Forwarding Port Counters attribute group name (15.4.27)	—
Managed object class	—	Port (15.4)
Managed object instance	—	Name of Port managed object
Attribute list	—	Names/values of all attributes that are members of the Read Forwarding Port Counters attribute group (15.4.27)

Table 15-14—Mapping of Read Port Default User Priority parameters (14.6.2.1)

M-GET Parameter Name	Req/Ind (Operation Inputs)	Rsp/Conf (Operation Outputs)
Base object class	Port (15.4)	—
Base object instance	Name of Port managed object	—
Scope	Base object alone	—
Filter	Not required	—
Attribute identifier list	Default User Priority attribute name (15.4.12)	—
Managed object class	—	Port (15.4)
Managed object instance	—	Name of Port managed object
Attribute list	—	Name/value of the Default User Priority attribute (15.4.12)

Table 15-15—Mapping of Set Port Default User Priority parameters (14.6.2.2)

M-SET Parameter Name	Req/Ind (Operation Inputs)
Base object class	Port (15.4)
Base object instance	Name of Port managed object
Scope	Base object alone
Filter	Not used
Modification list	Default User Priority attribute name and desired replacement value (15.4.12)

Table 15-16—Mapping of Read Port User Priority Regeneration Table parameters (14.6.2.3)

M-GET Parameter Name	Req/Ind (Operation Inputs)	Rsp/Conf (Operation Outputs)
Base object class	Port (15.4)	—
Base object instance	Name of Port managed object	—
Scope	Base object alone	—
Filter	Not required	—
Attribute identifier list	User Priority Regeneration Table attribute name (15.4.13)	—
Managed object class	—	Port
Managed object instance	—	Name of Port managed object
Attribute list	—	Name/value of User Priority Regeneration Table attribute (15.4.13)

Table 15-17—Mapping of Set Port User Priority Regeneration Table parameters (14.6.2.2)

M-SET Parameter Name	Req/Ind (Operation Inputs)
Base object class	Port (15.4)
Base object instance	Name of Port managed object
Scope	Base object alone
Filter	Not used
Modification list	User Priority Regeneration Table attribute name and desired replacement value (15.4.13)

Either zero or one GARP Timers managed object may be contained within each Port managed object. This object models the timer values used by all instances of GARP Applications (see Clause 12) on a given Port.

Table 15-18—Mapping of Read Port Traffic Class Table parameters (14.6.3.1)

M-GET Parameter Name	Req/Ind (Operation Inputs)	Rsp/Conf (Operation Outputs)
Base object class	Port (15.4)	—
Base object instance	Name of Port managed object	—
Scope	Base object alone	—
Filter	Not required	—
Attribute identifier list	Traffic Class Table attribute name (15.4.14)	—
Managed object class	—	Port (15.4)
Managed object instance	—	Name of Port managed object
Attribute list	—	Name/value of Traffic Class Table attribute (15.4.14)

Table 15-19—Mapping of Set Traffic Class Table parameters (14.6.3.2)

M-SET Parameter Name	Req/Ind (Operation Inputs)
Base object class	Port (15.4)
Base object instance	Name of Port managed object
Scope	Base object alone
Filter	Not used
Modification list	Traffic Class Table attribute name and desired replacement value (15.4.14)

Table 15-20—Mapping of Read Outbound Access Priority Table parameters (14.6.3.3)

M-GET Parameter Name	Req/Ind (Operation Inputs)	Rsp/Conf (Operation Outputs)
Base object class	Port (15.4)	—
Base object instance	Name of Port managed object	—
Scope	Base object alone	—
Filter	Not required	—
Attribute identifier list	Outbound Access Priority Table attribute name (15.4.15)	—
Managed object class	—	Port (15.4)
Managed object instance	—	Name of Port managed object
Attribute list	—	Names/values of Outbound Access Priority Table attribute (15.4.15)

The GARP Timers managed object exists if GARP is supported on the Port concerned; otherwise it is not present.

Zero or more GARP Application managed objects may be contained within each Port managed object. Each GARP Application managed object models the common properties of a GARP Application (see Clause 12). A GARP Attribute Type managed object exists within each GARP Application managed object for each Attribute Type supported by the GARP Application concerned. Within each GARP Attribute Type managed object, a GARP Attribute managed object exists for each value of the Attribute Type for which that Application currently maintains state information for a given GIP Context. GARP Attribute managed objects are created, updated, and deleted in accordance with the operation of the GARP Application concerned. The GARP Attribute managed object models the operations that may be performed on an individual GARP state machine.

Table 15-21—Mapping of Read Filtering Database parameters (14.7.1.1)

M-GET Parameter Name	Req/Ind (Operation Inputs)	Rsp/Conf (Operation Outputs)
Base object class	Filtering Database (15.7)	—
Base object instance	Name of Filtering Database managed object	—
Scope	Base object alone	—
Filter	Not required	—
Attribute identifier list	Read Database attribute group name (15.6.5)	—
Managed object class	—	Filtering Database (15.7)
Managed object instance	—	Name of Filtering Database managed object
Attribute list	—	Names/values of all attributes that are members of the Read Database attribute group (15.6.5), as extended by the Filtering Database managed object class definition (15.7)

Table 15-22—Mapping of Set Filtering Database Ageing Time parameters (14.7.1.2)

M-SET Parameter Name	Req/Ind (Operation Inputs)
Base object class	Filtering Database (15.7)
Base object instance	Name of Filtering Database managed object
Scope	Base object alone
Filter	Not used
Modification list	Ageing Time attribute name and desired replacement value (15.7.3)

Table 15-23—Mapping of Read Permanent Database parameters (14.7.5.1)

M-GET Parameter Name	Req/Ind (Operation Inputs)	Rsp/Conf (Operation Outputs)
Base object class	Permanent Database (15.6)	—
Base object instance	Name of Permanent Database managed object	—
Scope	Base object alone	—
Filter	Not required	—
Attribute identifier list	Read Database attribute group name (15.6.5)	—
Managed object class	—	Permanent Database (15.6)
Managed object instance	—	Name of Permanent Database managed object
Attribute list	—	Names/values of all attributes that are members of the Read Database attribute group (15.6.5)

Table 15-24—Mapping of Create Filtering Entry parameters (14.7.6.1)

M-CREATE Parameter Name	Req/Ind (Operation Inputs)
Base object class	Database Entry (15.8)
Base object instance	Name of new Database Entry managed object
Superior object instance	Name of Filtering Database or Permanent Database managed object
Reference object instance	Not used
Attribute list	Name and desired initial value of Port Map attribute (15.8.3)

Table 15-25—Mapping of Delete Filtering Entry parameters (14.7.6.2)

M-DELETE Parameter Name	Req/Ind (Operation Inputs)
Base object class	Database Entry (15.8)
Base object instance	Name of Database Entry managed object to be deleted
Scope	Not used
Filter	Not used

Table 15-26—Mapping of Read Filtering Entry parameters (14.7.6.3)

M-GET Parameter Name	Req/Ind (Operation Inputs)	Rsp/Conf (Operation Outputs)
Base object class	Database Entry (15.8)	—
Base object instance	Name of Database Entry managed object in Filtering Database or Permanent Database	—
Scope	Base object alone	—
Filter	Not required	—
Attribute identifier list	Read Database Entry attribute group name (15.8.6)	—
Managed object class	—	Database Entry (15.8)
Managed object instance	—	Name of Database Entry managed object
Attribute list	—	Names/values of all attributes that are members of the Read Database Entry attribute group (15.8.6)

Table 15-27—Mapping of Read Filtering Entry Range parameters (14.7.6.4)

M-GET Parameter Name	Req/Ind (Operation Inputs)	Rsp/Conf (Operation Outputs)
Base object class	Filtering Database (15.7) or Permanent Database (15.6)	—
Base object instance	Name of Filtering Database or Permanent Database managed object	—
Scope	1st level subordinates	—
Filter	Entry Index attribute value (15.8.5) is within specified range	—
Attribute identifier list	Read Database Entry attribute group name (15.8.6)	—
Managed object class	—	Database Entry
Managed object instance	—	Name of Database Entry managed object
Attribute list	—	Names/values of all attributes that are members of the Read Database Entry attribute group (15.8.6)

A single Filtering Database managed object and a single Permanent Database managed object are contained within each MAC Bridge DLE managed object. These database managed objects model the common properties of each database, such as the size, number of entries, and ageing time. They also form container objects for the filtering entry managed objects in the filtering and permanent databases.

Zero or more Database Entry managed objects may be contained within each database managed object. Each Database Entry managed object models a single entry, dynamic or static, in the filtering or permanent database. Static entries (Static Filtering Entries) may be created and deleted by remote management request; dynamic entries are created by the operation of the Learning Process (Dynamic Filtering Entries) or by the operation of GMRP (Group Registration Entries), and may be deleted by remote management request.

Table 15-28—Mapping of Read Bridge Protocol Parameters parameters (14.8.1.1)

M-GET Parameter Name	Req/Ind (Operation Inputs)	Rsp/Conf (Operation Outputs)
Base object class	MAC Bridge DLE (15.3)	—
Base object instance	Name of MAC Bridge DLE managed object	—
Scope	Base object alone	—
Filter	Not required	—
Attribute identifier list	Read Bridge Parameters attribute group name (15.3.23)	—
Managed object class	—	MAC Bridge DLE (15.3)
Managed object instance	—	Name of MAC Bridge DLE managed object
Attribute list	—	Names/values of all attributes that are members of the Read Bridge Protocol Parameters attribute group (15.3.23)

Table 15-29—Mapping of Set Bridge Protocol Parameters parameters (14.8.1.2)

M-SET Parameter Name	Req/Ind (Operation Inputs)
Base object class	MAC Bridge DLE (15.3)
Base object instance	Name of MAC Bridge DLE managed object
Scope	Base object alone
Filter	Not used
Modification list	Bridge Max Age (15.3.17), Bridge Hello Time (15.3.18), Bridge Forward Delay (15.3.19), and Bridge Priority (15.3.21) attribute names and desired replacement values

Table 15-30—Mapping of Read Port Parameters parameters (14.8.2.1)

M-GET Parameter Name	Req/Ind (Operation Inputs)	Rsp/Conf (Operation Outputs)
Base object class	Port (15.4)	—
Base object instance	Name of Port managed object	—
Scope	Base object alone	—
Filter	Not required	—
Attribute identifier list	Read Port Parameters attribute group name (15.4.28)	—
Managed object class	—	Port (15.4)
Managed object instance	—	Name of Port managed object
Attribute list	—	Names/values of all attributes that are members of the Read Port Parameters attribute group (15.4.28)

Figure 15-1 shows the managed object containment structure for the MAC Bridge; elements in this figure marked with dashed lines show the Data Link layer containment structure defined in ISO/IEC 10742.

Table 15-31—Mapping of Force Port State parameters (14.8.2.2)

M-ACTION Parameter Name	Req/Ind (Operation Inputs)
Base object class	Port (15.4)
Base object instance	Name of Port managed object
Scope	Base object alone
Filter	Not used
Action Type	Force Port State (15.4.29)
Action Information	Desired state (Disabled or Blocking)
Action Reply	Not used

Table 15-32—Mapping of Set Port Parameters parameters (14.8.2.3)

M-SET Parameter Name	Req/Ind (Operation Inputs)
Base object class	Port (15.4)
Base object instance	Name of Port managed object
Scope	Base object alone
Filter	Not used
Modification list	Path Cost (15.4.20) and Port Priority (15.4.19) attribute names and desired replacement values

Table 15-33—Mapping of Read Selective Translation Table Entry Range parameters (ISO/IEC 11802-5)

M-GET Parameter Name	Req/Ind (Operation Inputs)	Rsp/Conf (Operation Outputs)
Base object class	Port (15.4)	—
Base object instance	Name of Port managed object	—
Scope	1st level subordinates	—
Filter	Type Value attribute value (15.5.2) is within specified range	—
Attribute identifier list	empty	—
Managed object class	—	Selective Translation Table Entry (15.5)
Managed object instance	—	Name of Selective Translation Table Entry managed object

Table 15-34—Mapping of Read GARP Timers parameters (14.9.1.1)

M-GET Parameter Name	Req/Ind (Operation Inputs)	Rsp/Conf (Operation Outputs)
Base object class	GARP Timers (15.12)	—
Base object instance	Name of GARP Timers managed object	—
Scope	Base object alone	—
Filter	Not required	—
Attribute identifier list	Names of Join Time (15.12.3), Leave Time (15.12.4), and Leave All Time (15.12.5) attributes	—
Managed object class	—	GARP Timers (15.12)
Managed object instance	—	Name of GARP Timers managed object
Attribute list	—	Names/values of Join Time (15.12.3), Leave Time (15.12.4), and Leave All Time (15.12.5) attributes

Table 15-35—Mapping of Set GARP Timers parameters (14.9.1.2)

M-SET Parameter Name	Req/Ind (Operation Inputs)
Base object class	GARP Timers (15.12)
Base object instance	Name of GARP Timers managed object
Scope	Base object alone
Filter	Not used
Modification list	Names/replacement values of Join Time (15.12.3), Leave Time (15.12.4), and Leave All Time (15.12.5) attributes

Table 15-36—Mapping of Read GARP Applicant Controls parameters (14.9.2.1)

M-GET Parameter Name	Req/Ind (Operation Inputs)	Rsp/Conf (Operation Outputs)
Base object class	GARP Attribute Type (15.10)	—
Base object instance	Name of GARP Attribute managed object	—
Scope	Base object alone	—
Filter	Not required	—
Attribute identifier list	Names of Applicant Administrative Control (15.10.3) and Failed Registrations (15.10.4) attributes	—
Managed object class	—	GARP Attribute Type (15.10)
Managed object instance	—	Name of GARP Attribute managed object
Attribute list	—	Names/values of Protocol Administrative Control (15.10.3) and Failed Registrations (15.10.4) attributes

Table 15-37—Mapping of Set GARP Applicant Controls parameters (14.9.2.2)

M-SET Parameter Name	Req/Ind (Operation Inputs)
Base object class	GARP Attribute Type (15.10)
Base object instance	Name of GARP Attribute managed object
Scope	Base object alone
Filter	Not used
Modification list	Name/replacement value of Applicant Administrative Control (15.10.3) attribute

Table 15-38—Mapping of Read GARP State parameters (14.9.3.1)

M-GET Parameter Name	Req/Ind (Operation Inputs)	Rsp/Conf (Operation Outputs)
Base object class	GARP Attribute (15.11)	—
Base object instance	Name of GARP Attribute managed object	—
Scope	Base object alone	—
Filter	Not required	—
Attribute identifier list	Names of State Value (15.11.3) and (optionally) Originator of Last PDU (15.11.4) attributes	—
Managed object class	—	GARP Attribute (15.11)
Managed object instance	—	Name of GARP Attribute managed object
Attribute list	—	Names/values of State Value (15.11.3) and (optionally) Originator of Last PDU (15.11.4) attributes

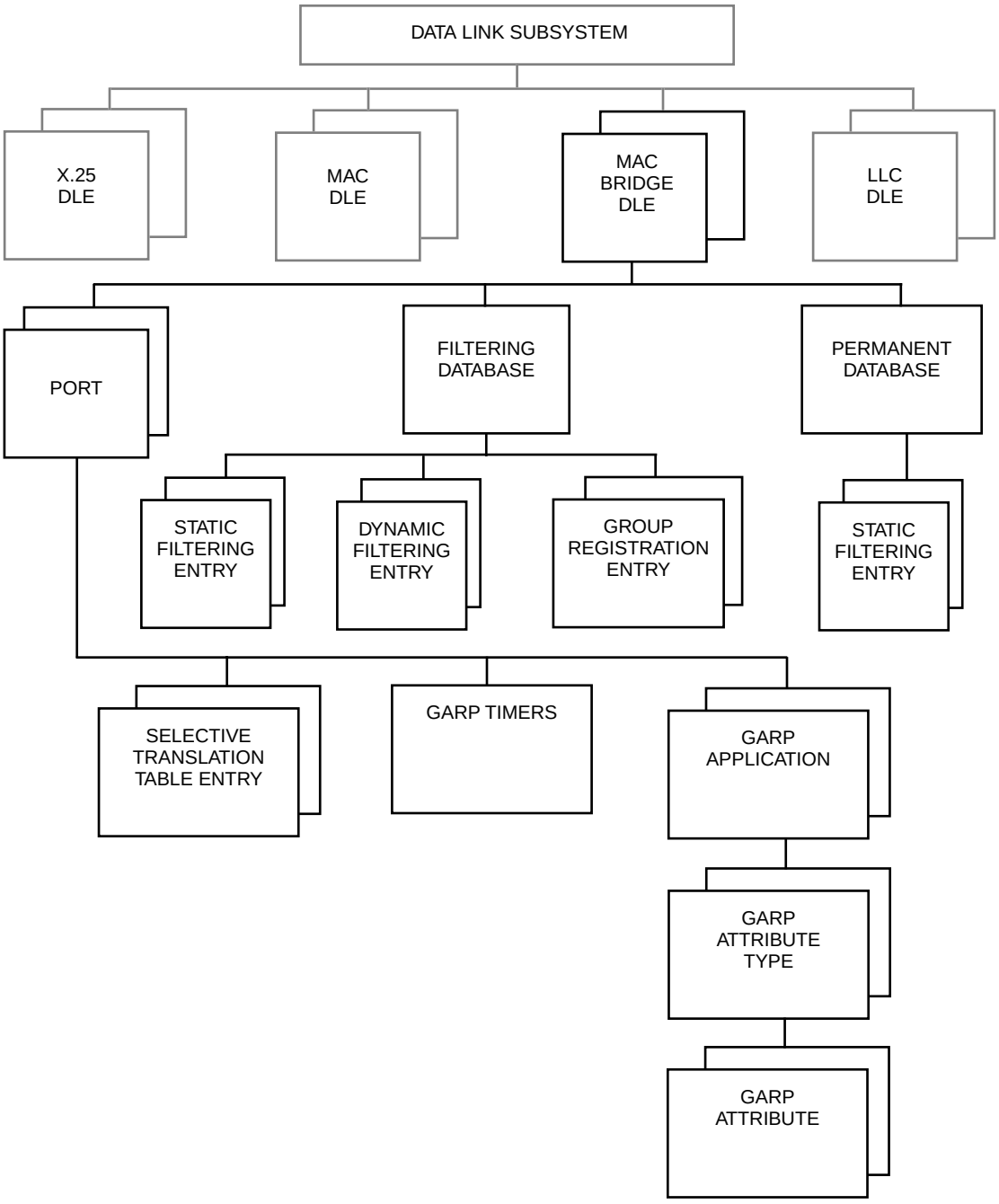


Figure 15-1—Managed object containment structure

15.3 MAC Bridge DLE Managed Object Class definition

```
oMACBridgeDLE MANAGED OBJECT CLASS
  DERIVED FROM “ISO/IEC 10742”:datalinkEntity;
  CHARACTERIZED BY
    pMACBridgeDLE PACKAGE
      BEHAVIOUR
        bMACBridgeDLE BEHAVIOUR
          DEFINED AS       !The MAC Bridge DLE managed object class combines the
                           characteristics of the Bridge Configuration object and
                           the Protocol Entity object, as defined in
                           14.4.1 and 14.8.1.!
        ;
      ;
      ATTRIBUTES           aBridgeAddress                GET, -- Naming Attribute
                           aBridgeName                   GET-REPLACE,
                           aBridgeNumPorts               GET,
                           aBridgePortAddresses          GET,
                           aUptime                       GET,
                           aBridgeIdentifier             GET,
                           aTimeSinceTopologyChange      GET,
                           aTopologyChangeCount          GET,
                           aTopologyChange               GET,
                           aDesignatedRoot               GET,
                           aRootPathCost                 GET,
                           aRootPort                     GET,
                           aMaxAge                       GET,
                           aHelloTime                    GET,
                           aForwardDelay                 GET,
                           aBridgeMaxAge                 GET-REPLACE,
                           aBridgeHelloTime              GET-REPLACE,
                           aBridgeForwardDelay           GET-REPLACE,
                           aHoldTime                     GET,
                           aBridgePriority               GET-REPLACE
      ;
      ATTRIBUTE GROUPS     agReadOrDiscoverBridge,
                           agReadBridgeProtocolParameters
      ;
      ACTIONS              acResetBridge ;
    ;
  ;
REGISTERED AS {Bridge.moClass macBridgeDLE(0)};
```

15.3.1 Name Bindings for MAC Bridge DLE

```
nbMACBridgeDLE-datalinkSubsystem NAME BINDING
  SUBORDINATE OBJECT CLASS               oMACBridgeDLE AND SUBCLASSES;
  NAMED BY SUPERIOR OBJECT CLASS         “ISO/IEC 10742”:datalinkSubsystem
                                         AND SUBCLASSES;
  WITH ATTRIBUTE                         aBridgeAddress;
REGISTERED AS {Bridge.nameBinding macBridgeDLE-datalinkSubsystem(0)};
```

15.3.2 Bridge Address Attribute definition

```
aBridgeAddress ATTRIBUTE
  WITH ATTRIBUTE SYNTAX    Bridge.BridgeAddress;
  MATCHES FOR              EQUALITY, ORDERING ;
  BEHAVIOUR
    bBridgeAddress BEHAVIOUR
      DEFINED AS           !See 14.4.1.1.3 a), 14.4.1.2.3 a), 8.5.3.7, 7.12.5.!
    ;
  ;
REGISTERED AS {Bridge.attribute bridgeAddress(0)};
```

15.3.3 Bridge Name Attribute definition

```
aBridgeName ATTRIBUTE
  WITH ATTRIBUTE SYNTAX    Bridge.BridgeName  ;
```

```
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bBridgeName BEHAVIOUR
      DEFINED AS          !See 14.4.1.1.3 b), 14.4.1.2.3 b).!
    ;
  ;
REGISTERED AS {Bridge.attribute bridgeName(1)};
```

15.3.4 Bridge Number Of Ports Attribute definition

```
aBridgeNumPorts ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.BridgeNumPorts  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bBridgeNumPorts BEHAVIOUR
      DEFINED AS          !See 14.4.1.1.3 c), 14.4.1.2.3 c)., 14.4.2.!
    ;
  ;
REGISTERED AS {Bridge.attribute bridgeNumPorts(2)};
```

15.3.5 Bridge Port Addresses Attribute definition

```
aBridgePortAddresses ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.BridgePortAddresses  ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bBridgePortAddresses BEHAVIOUR
      DEFINED AS          !See 14.4.1.1.3 d), 14.4.1.2.3 d).!
    ;
  ;
REGISTERED AS {Bridge.attribute bridgePortAddresses(3)};
```

15.3.6 Uptime Attribute definition

```
aUptime ATTRIBUTE
  DERIVED FROM            “ISO/IEC 10165-5”:nonWrapping64BitCounter ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bUptime BEHAVIOUR
      DEFINED AS          !See 14.4.1.1.3 e), 14.4.1.2.3 e).!
    ;
  ;
REGISTERED AS {Bridge.attribute uptime(4)};
```

15.3.7 Bridge Identifier Attribute definition

```
aBridgeIdentifier ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.BridgeIdentifier  ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bBridgeIdentifier BEHAVIOUR
      DEFINED AS          !See 8.5.3.7.!
    ;
  ;
REGISTERED AS {Bridge.attribute bridgeIdentifier(5)};
```

15.3.8 Time Since Topology Change Attribute definition

```
aTimeSinceTopologyChange ATTRIBUTE
  DERIVED FROM            “ISO/IEC 10165-5”:nonWrapping64BitCounter ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bTimeSinceTopologyChange BEHAVIOUR
      DEFINED AS          !See 14.8.1.1.3 b).!
    ;
  ;
REGISTERED AS {Bridge.attribute timeSinceTopologyChange(6)};
```

15.3.9 Topology Change Count Attribute definition

```
aTopologyChangeCount ATTRIBUTE
  DERIVED FROM            “ISO/IEC 10165-5”:nonWrapping64BitCounter ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bTopologyChangeCount BEHAVIOUR
      DEFINED AS          !See 14.8.1.1.3 c).!
    ;
  ;
REGISTERED AS {Bridge.attribute topologyChangeCount(7)};
```

15.3.10 Topology Change Attribute definition

```
aTopologyChange ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.TopologyChange  ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bTopologyChange BEHAVIOUR
      DEFINED AS          !See 8.5.3.12.!
    ;
  ;
REGISTERED AS {Bridge.attribute topologyChange(8)};
```

15.3.11 Designated Root Attribute definition

```
aDesignatedRoot ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.BridgeIdentifier  ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bDesignatedRoot BEHAVIOUR
      DEFINED AS          !See 8.5.3.1.!
    ;
  ;
REGISTERED AS {Bridge.attribute designatedRoot(9)};
```

15.3.12 Root Path Cost Attribute definition

```
aRootPathCost ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.RootPathCost  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bRootPathCost BEHAVIOUR
      DEFINED AS          !See 8.5.3.2.!
    ;
  ;
REGISTERED AS {Bridge.attribute rootPathCost(10)};
```

15.3.13 Root Port Attribute definition

```
aRootPort ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.RootPort  ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bRootPort BEHAVIOUR
      DEFINED AS          !See 8.5.3.3.!
    ;
  ;
REGISTERED AS {Bridge.attribute rootPort(11)};
```

15.3.14 Max Age Attribute definition

```
aMaxAge ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.MaxAge ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
```



```
    bMaxAge BEHAVIOUR
      DEFINED AS          !See 8.5.3.4.!
    ;
  ;
REGISTERED AS {Bridge.attribute maxAge(12)};
```

15.3.15 Hello Time Attribute definition

```
aHelloTime ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.HelloTime  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bHelloTime BEHAVIOUR
      DEFINED AS          !See 8.5.3.5.!
    ;
  ;
REGISTERED AS {Bridge.attribute helloTime(13)};
```

15.3.16 Forward Delay Attribute definition

```
aForwardDelay ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.ForwardDelay  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bForwardDelay BEHAVIOUR
      DEFINED AS          !See 8.5.3.6.!
    ;
  ;
REGISTERED AS {Bridge.attribute forwardDelay(14)};
```

15.3.17 Bridge Max Age Attribute definition

```
aBridgeMaxAge ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.BridgeMaxAge  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bBridgeMaxAge BEHAVIOUR
      DEFINED AS          !See 8.5.3.8.!
    ;
  ;
REGISTERED AS {Bridge.attribute bridgeMaxAge(15)};
```

15.3.18 Bridge Hello Time Attribute definition

```
aBridgeHelloTime ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.BridgeHelloTime  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bBridgeHelloTime BEHAVIOUR
      DEFINED AS          !See 8.5.3.9.!
    ;
  ;
REGISTERED AS {Bridge.attribute bridgeHelloTime(16)};
```

15.3.19 Bridge Forward Delay Attribute definition

```
aBridgeForwardDelay ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.BridgeForwardDelay  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bBridgeForwardDelay BEHAVIOUR
      DEFINED AS          !See 8.5.3.10.!
    ;
  ;
REGISTERED AS {Bridge.attribute bridgeForwardDelay(17)};
```

15.3.20 Hold Time Attribute definition

```
aHoldTime ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.HoldTime  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bHoldTime BEHAVIOUR
      DEFINED AS          !See 8.5.3.14.!
    ;
  ;
REGISTERED AS {Bridge.attribute holdTime(18)};
```

15.3.21 Bridge Priority Attribute definition

```
aBridgePriority ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.BridgePriority ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bBridgePriority BEHAVIOUR
      DEFINED AS          !See 8.5.3.7.!
    ;
  ;
REGISTERED AS {Bridge.attribute bridgePriority(19)};
```

15.3.22 Read Or Discover Bridge Attribute Group definition

```
agReadOrDiscoverBridge ATTRIBUTE GROUP
  GROUP ELEMENTS          aBridgeName,
                          aBridgeNumPorts,
                          aBridgePortAddresses,
                          aUptime
  ;
  FIXED ;
  DESCRIPTION             !This attribute group is used in the mapping of the
                          Read Bridge and Discover Bridge operations, as defined
                          in Table 15-7 and Table 15-8.! ;
REGISTERED AS {Bridge.attributeGroup readOrDiscoverBridge(0)};
```

15.3.23 Read Bridge Protocol Parameters Attribute Group definition

```
agReadBridgeProtocolParameters ATTRIBUTE GROUP
  GROUP ELEMENTS          aBridgeIdentifier,
                          aTimeSinceTopologyChange,
                          aTopologyChangeCount,
                          aTopologyChange,
                          aDesignatedRoot,
                          aRootPathCost,
                          aRootPort,
                          aMaxAge,
                          aHelloTime,
                          aForwardDelay,
                          aBridgeMaxAge,
                          aBridgeHelloTime,
                          aBridgeForwardDelay,
                          aHoldTime
  ;
  FIXED ;
  DESCRIPTION             !This attribute group is used in the mapping of the Read
                          Bridge Protocol Parameters operation, as defined in
                          Table 15-28.! ;
REGISTERED AS {Bridge.attributeGroup readBridgeProtocolParameters(1)};
```

15.3.24 Reset Bridge Action definition

```
acResetBridge ACTION
  BEHAVIOUR
    bResetBridge BEHAVIOUR
```

```
      DEFINED AS          !See 14.4.1.4.!
    ;
  ;
REGISTERED AS {Bridge.action resetBridge(0)};
```

15.4 Port Managed Object Class definition

```
oPort MANAGED OBJECT CLASS
  DERIVED FROM            "ISO/IEC 10165-2":top;
  CHARACTERIZED BY
    pPort PACKAGE
      BEHAVIOUR
        bPort BEHAVIOUR
          DEFINED AS      !The Port managed object class combines the
                          characteristics of the Port Configuration object, the
                          Port Counters object, the Transmission Priority object
                          and the Bridge Port object, as defined in
                          14.4.2, 14.6.1, 14.6.2, and 14.8.2.!
        ;
      ;
      ATTRIBUTES
                          aPortNumber                  GET, -- Naming attribute
                          aPortName                    GET-REPLACE,
                          aPortType                    GET,
                          aFramesReceived              GET,
                          aDiscardInbound              GET,
                          aForwardOutbound             GET,
                          aDiscardLackOfBuffers        GET,
                          aDiscardTransitDelayExceeded
                                                       GET,
                          aDiscardOnError              GET,
                          aDiscardOnErrorDetail        GET,
                          aDefaultUserPriority         GET-REPLACE,
                          aUserPriorityRegenerationTable
                                                       GET-REPLACE,
                          aTrafficClassTable           GET-REPLACE,
                          aOutboundAccessPriorityTable
                                                       GET,
                          aPortUptime                  GET,
                          aState                       GET,
                          aPortIdentifier              GET,
                          aPortPriority                GET-REPLACE,
                          aPathCost                    GET-REPLACE,
                          aPortDesignatedRoot          GET,
                          aDesignatedCost              GET,
                          aDesignatedBridge            GET,
                          aDesignatedPort              GET,
                          aTopologyChangeAck           GET,
                          "ISO/IEC 10742":providerEntityNames
                                                       GET
      ;
      ATTRIBUTE GROUPS    agReadPort,
                          agReadForwardingPortCounters,
                          agReadPortParameters
      ;
      ACTIONS             acForcePortState;
    ;
  ;
REGISTERED AS {Bridge.moClass port(6)};
```

15.4.1 Name Bindings for Port

```
nbPort-MACBridgeDLE NAME BINDING
  SUBORDINATE OBJECT CLASS              oPort AND SUBCLASSES;
  NAMED BY SUPERIOR OBJECT CLASS        oMACBridgeDLE AND SUBCLASSES;
  WITH ATTRIBUTE                        aPortNumber;
REGISTERED AS {Bridge.nameBinding port-MACBridgeDLE(1)};
```

15.4.2 Port Number Attribute definition

```
aPortNumber ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.PortNumber  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bPortNumber BEHAVIOUR
      DEFINED AS          !See 8.5.5.1 and 14.4.2.!
    ;
  ;
REGISTERED AS {Bridge.attribute portNumber(20)};
```

15.4.3 Port Name Attribute definition

```
aPortName ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.PortName ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bPortName BEHAVIOUR
      DEFINED AS          !See 14.4.2.1.3 a)!
    ;
  ;
REGISTERED AS {Bridge.attribute portName(21)};
```

15.4.4 Port Type Attribute definition

```
aPortType ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.PortType  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bPortType BEHAVIOUR
      DEFINED AS          !See 14.4.2.1.3 b).  This attribute contains an OBJECT
                          IDENTIFIER value that indicates the MAC Entity type.  The
                          value may be a registered value that identifies the
                          standard in which the MAC Entity type is defined, or it may
                          be a value that has been registered independently of the
                          MAC standard, with the intent of identifying a specific
                          MAC Entity type.!
    ;
  ;
REGISTERED AS {Bridge.attribute portType(22)};
```

15.4.5 Frames Received Attribute definition

```
aFramesReceived ATTRIBUTE
  DERIVED FROM            “ISO/IEC 10165-5”:nonWrapping64BitCounter ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bFramesReceived BEHAVIOUR
      DEFINED AS          !See 14.6.1.1.3 a)!
    ;
  ;
REGISTERED AS {Bridge.attribute framesReceived(23)};
```

15.4.6 Discard Inbound Attribute definition

```
aDiscardInbound ATTRIBUTE
  DERIVED FROM            “ISO/IEC 10165-5”:nonWrapping64BitCounter ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bDiscardInbound BEHAVIOUR
      DEFINED AS          !See 14.6.1.1.3 b).!
    ;
  ;
REGISTERED AS {Bridge.attribute discardInbound(24)};
```

15.4.7 Forward Outbound Attribute definition

```
aForwardOutbound ATTRIBUTE
  DERIVED FROM            “ISO/IEC 10165-5”:nonWrapping64BitCounter ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bForwardOutbound BEHAVIOUR
      DEFINED AS          !See 14.6.1.1.3 c).!
    ;
  ;
REGISTERED AS {Bridge.attribute forwardOutbound(25)};
```

15.4.8 Discard Lack Of Buffers Attribute definition

```
aDiscardLackOfBuffers ATTRIBUTE
  DERIVED FROM            “ISO/IEC 10165-5”:nonWrapping64BitCounter ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bDiscardLackOfBuffers BEHAVIOUR
      DEFINED AS          !See 14.6.1.1.3 d)!
    ;
  ;
REGISTERED AS {Bridge.attribute discardLackOfBuffers(26)};
```

15.4.9 Discard Transit Delay Exceeded Attribute definition

```
aDiscardTransitDelayExceeded ATTRIBUTE
  DERIVED FROM            “ISO/IEC 10165-5”:nonWrapping64BitCounter ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bDiscardTransitDelayExceeded BEHAVIOUR
      DEFINED AS          !See 14.6.1.1.3 e)!
    ;
  ;
REGISTERED AS {Bridge.attribute discardTransitDelayExceeded(27)};
```

15.4.10 Discard On Error Attribute definition

```
aDiscardOnError ATTRIBUTE
  DERIVED FROM            “ISO/IEC 10165-5”:nonWrapping64BitCounter ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bDiscardOnError BEHAVIOUR
      DEFINED AS          !See 14.6.1.1.3 f).!
    ;
  ;
REGISTERED AS {Bridge.attribute discardOnError(28)};
```

15.4.11 Discard On Error Detail Attribute definition

```
aDiscardOnErrorDetail ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.DiscardOnErrorDetail  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bDiscardOnErrorDetail BEHAVIOUR
      DEFINED AS          !See 14.6.1.1.3 g).!
    ;
  ;
REGISTERED AS {Bridge.attribute discardOnErrorDetail(29)};
```

15.4.12 Default User Priority Attribute definition

```
aDefaultUserPriority ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.UserPriority ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
```

```
    bDefaultUserPriority BEHAVIOUR
      DEFINED AS          !See 14.6.2.1.3 a) and 14.6.2.2.2 b).!
    ;
  ;
REGISTERED AS {Bridge.attribute defaultUserPriority(52)};
```

15.4.13 User Priority Regeneration Table Attribute definition

```
aUserPriorityRegenerationTable ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.UserPriorityRegenerationTable ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bUserPriorityRegenerationTable BEHAVIOUR
      DEFINED AS          !See 14.6.2.3.3 and 14.6.2.4.2.!
    ;
  ;
REGISTERED AS {Bridge.attribute userPriorityRegenerationTable(53)};
```

15.4.14 Traffic Class Table Attribute definition

```
aTrafficClassTable ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.TrafficClassTable ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bTrafficClassTable BEHAVIOUR
      DEFINED AS          !See 14.6.3.1.3 a) and 14.6.3.2.2 b).!
    ;
  ;
REGISTERED AS {Bridge.attribute trafficClassTable(54)};
```

15.4.15 Outbound Access Priority Table Attribute definition

```
aOutboundAccessPriorityTable ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.OutboundAccessPriorityTable ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bOutboundAccessPriority BEHAVIOUR
      DEFINED AS          !See 14.6.3.3.3.!
    ;
  ;
REGISTERED AS {Bridge.attribute outboundAccessPriorityTable(55)};
```

15.4.16 Port Uptime Attribute definition

```
aPortUptime ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.Uptime  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bPortUptime BEHAVIOUR
      DEFINED AS          !See 14.8.2.1.3 a).!
    ;
  ;
REGISTERED AS {Bridge.attribute portUptime(32)};
```

15.4.17 State Attribute definition

```
aState ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.State  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bState BEHAVIOUR
      DEFINED AS          !See 14.8.2.1.3 b) and 8.5.5.2.!
    ;
  ;
REGISTERED AS {Bridge.attribute state(33)};
```


15.4.18 Port Identifier Attribute definition

```
aPortIdentifier ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.PortIdentifier  ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bPortIdentifier BEHAVIOUR
      DEFINED AS          !See 14.8.2.1.3 c) and 8.5.5.1.!
    ;
  ;
REGISTERED AS {Bridge.attribute portIdentifier(34)};
```

15.4.19 Port Priority Attribute definition

```
aPortPriority ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.PortPriority  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bPortPriority BEHAVIOUR
      DEFINED AS          !See 14.8.2.3.2 c).!
    ;
  ;
REGISTERED AS {Bridge.attribute portPriority(35)};
```

15.4.20 Path Cost Attribute definition

```
aPathCost ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.PathCost ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bPathCost BEHAVIOUR
      DEFINED AS          !See 8.5.5.3.!
    ;
  ;
REGISTERED AS {Bridge.attribute pathCost(36)};
```

15.4.21 Port Designated Root Attribute definition

```
aPortDesignatedRoot ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.DesignatedRoot  ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bPortDesignatedRoot BEHAVIOUR
      DEFINED AS          !See 8.5.5.4.!
    ;
  ;
REGISTERED AS {Bridge.attribute portDesignatedRoot(37)};
```

15.4.22 Designated Cost Attribute definition

```
aDesignatedCost ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.DesignatedCost  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bDesignatedCost BEHAVIOUR
      DEFINED AS          !See 8.5.5.5.!
    ;
  ;
REGISTERED AS {Bridge.attribute designatedCost(38)};
```

15.4.23 Designated Bridge Attribute definition

```
aDesignatedBridge ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.DesignatedBridge  ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
```

```
    bDesignatedBridge BEHAVIOUR
      DEFINED AS          !See 8.5.5.6.!
    ;
  ;
REGISTERED AS {Bridge.attribute designatedBridge(39)};
```

15.4.24 Designated Port Attribute definition

```
aDesignatedPort ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.DesignatedPort  ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bDesignatedPort BEHAVIOUR
      DEFINED AS          !See 8.5.5.7.!
    ;
  ;
REGISTERED AS {Bridge.attribute designatedPort(40)};
```

15.4.25 Topology ChangeAck Attribute definition

```
aTopologyChangeAck ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.TopologyChangeAck  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bTopologyChangeAck BEHAVIOUR
      DEFINED AS          !See 8.5.5.8.!
    ;
  ;
REGISTERED AS {Bridge.attribute topologyChangeAck(41)};
```

15.4.26 Read Port Attribute Group definition

```
agReadPort ATTRIBUTE GROUP
  GROUP ELEMENTS          aPortName,
                          aPortType
  ;
  FIXED ;
  DESCRIPTION             !This attribute group is used in the mapping of the Read
                          Port operation, as defined in Table 15-11.! ;
REGISTERED AS {Bridge.attributeGroup readPort(2)};
```

15.4.27 Read Forwarding Port Counters Attribute Group definition

```
agReadForwardingPortCounters ATTRIBUTE GROUP
  GROUP ELEMENTS          aFramesReceived,
                          aDiscardInbound,
                          aForwardOutbound,
                          aDiscardLackOfBuffers,
                          aDiscardTransitDelayExceeded,
                          aDiscardOnError,
                          aDiscardOnErrorDetail
  ;
  FIXED ;
  DESCRIPTION             !This attribute group is used in the mapping of the Read
                          Forwarding Port Counters operation, as defined in
                          Table 15-13.! ;
REGISTERED AS {Bridge.attributeGroup readForwardingPortCounters(3)};
```

15.4.28 Read Port Parameters Attribute Group definition

```
agReadPortParameters ATTRIBUTE GROUP
  GROUP ELEMENTS          aPortUptime,
                          aState,
                          aPortIdentifier,
                          aPathCost,
                          aPortDesignatedRoot,
                          aDesignatedCost,
```



```
                          aDesignatedBridge,
                          aDesignatedPort,
                          aTopologyChangeAck
  ;
  FIXED ;
  DESCRIPTION             !This attribute group is used in the mapping of the Read
                          Port Parameters operation, as defined in Table 15-30.! ;
REGISTERED AS {Bridge.attributeGroup readPortParameters(5)};
```

15.4.29 Force Port State Action definition

```
acForcePortState ACTION
  BEHAVIOUR
    bForcePortState BEHAVIOUR
      DEFINED AS          !Models the Force Port State operation, as described in
                          14.8.2.2.!
    ;
  ;
  WITH INFORMATION SYNTAX               Bridge.ForcePortState ;
REGISTERED AS {Bridge.action forcePortState(1)};
```

15.5 Selective Translation Table Entry Managed Object Class definition

```
oSelectiveTranslationTableEntry MANAGED OBJECT CLASS
  DERIVED FROM            “ISO/IEC 10165-2”:top;
  CHARACTERIZED BY
    pSelectiveTranslationTableEntry PACKAGE
      BEHAVIOUR
        bSelectiveTranslationTableEntry BEHAVIOUR
          DEFINED AS      !The Selective Translation Table Entry managed object
                          class models the properties of a single entry in the
                          Selective Translation Table, as described in 5.2 of
                          ISO/IEC 11802-5!
        ;
      ;
      ATTRIBUTES          aTypeValue     GET -- Naming attribute
      ;
    ;
  ;
REGISTERED AS {Bridge.moClass selectiveTranslationTableEntry(2)};
```

15.5.1 Name Bindings for Selective Translation Table Entry

```
nbSelectiveTranslationTableEntry-Port NAME BINDING
  SUBORDINATE OBJECT CLASS              oSelectiveTranslationTableEntry
                                        AND SUBCLASSES;
  NAMED BY SUPERIOR OBJECT CLASS        oPort AND SUBCLASSES;
  WITH ATTRIBUTE                        aTypeValue;
  CREATE;
  DELETE;
REGISTERED AS {Bridge.nameBinding selectiveTranslationTableEntry-Port(2)};
```

15.5.2 Type Value Attribute definition

```
aTypeValue ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.TypeValue  ;
  MATCHES FOR EQUALITY ;
  BEHAVIOUR
    bTypeValue BEHAVIOUR
      DEFINED AS          !An integer value that represents the value of a
                          registered Ethernet Protocol Type.  The encoding of the
                          type value as an integer, and the corresponding bit/octet
                          transmission order when encoded in an Ethernet type
                          field, are as defined in ISO/IEC 11802-5.!
    ;
  ;
REGISTERED AS {Bridge.attribute typeValue(42)};
```

15.6 Permanent Database Managed Object Class definition

```
oPermanentDatabase MANAGED OBJECT CLASS
  DERIVED FROM            “ISO/IEC 10165-2”:top ;
  CHARACTERIZED BY
    pPermanentDatabase PACKAGE
      BEHAVIOUR
        bPermanentDatabase BEHAVIOUR
          DEFINED AS
                          !An instance of the Permanent Database managed object
                          class is a container managed object for the Database
                          Entry managed objects that comprise the Permanent
                          Database. The functionality of the Permanent Database is
                          as described in 14.7.5; the Permanent Database managed
                          object models only those aspects of the Permanent
                          Database that are necessary in order to support the Read
                          Permanent Database operation, as described in 14.7.5.1.!
        ;
      ;
      ATTRIBUTES          aDatabaseName  INITIAL VALUE Bridge.permanentDatabaseName
                                                       GET,  -- Naming Attribute
                          aDatabaseSize                GET,
                          aNumberOfEntries             GET
      ;
      ATTRIBUTE GROUPS    agReadDatabase
      ;
    ;
  ;
REGISTERED AS {Bridge.moClass permanentDatabase(3)};
```

15.6.1 Name Bindings for Permanent Database

```
nbPermanentDatabase-MACBridgeDLE NAME BINDING
  SUBORDINATE OBJECT CLASS              oPermanentDatabase AND SUBCLASSES;
  NAMED BY SUPERIOR OBJECT CLASS        oMACBridgeDLE AND SUBCLASSES;
  WITH ATTRIBUTE                        aDatabaseName;
REGISTERED AS {Bridge.nameBinding permanentDatabase-MACBridgeDLE(3)};
```

15.6.2 Database Name Attribute definition

```
aDatabaseName ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.DatabaseName  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bDatabaseName BEHAVIOUR
      DEFINED AS          !Naming attribute for databases!
    ;
  ;
REGISTERED AS {Bridge.attribute databaseName(43)};
```

15.6.3 Database Size Attribute definition

```
aDatabaseSize ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.DatabaseSize  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bDatabaseSize BEHAVIOUR
      DEFINED AS          !See 14.7.1.1.3 a) and 14.7.5.1.3 a).!
    ;
  ;
REGISTERED AS {Bridge.attribute databaseSize(44)};
```

15.6.4 Number Of Entries Attribute definition

```
aNumberOfEntries ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.NumberOfEntries  ;
  MATCHES FOR             EQUALITY, ORDERING ;
```



```
  BEHAVIOUR
    bNumberOfEntries BEHAVIOUR
      DEFINED AS          !See 14.7.1.1.3 a) and 14.7.5.1.3 a).!
    ;
  ;
REGISTERED AS {Bridge.attribute numberOfEntries(45)};
```

15.6.5 Read Database Attribute Group definition

```
agReadDatabase ATTRIBUTE GROUP
  GROUP ELEMENTS          aDatabaseSize,
                          aNumberOfEntries
  ;
  DESCRIPTION             !This attribute group is used in the mapping of the Read
                          Filtering Database and Read Permanent Database
                          operations, as defined in Table 15-21 and
                          Table 15-23.! ;
REGISTERED AS {Bridge.attributeGroup readDatabase(6)};
```

15.7 Filtering Database Managed Object Class definition

```
oFilteringDatabase MANAGED OBJECT CLASS
  DERIVED FROM            “ISO/IEC 10165-2”:top ;
  CHARACTERIZED BY
    pFilteringDatabase PACKAGE
      BEHAVIOUR
        bFilteringDatabase BEHAVIOUR
          DEFINED AS      !An instance of the Filtering Database managed object
                          class is a container managed object for the Database
                          Entry managed objects that comprise the Filtering
                          Database. The functionality of the Filtering Database is
                          as described in 14.7.1; the Filtering Database managed
                          object models only those aspects of the Filtering Database
                          that are necessary in order to support the Read Filtering
                          Database and Set Filtering Database AgeingTime
                          operations, as described in 14.7.1.1 and 14.7.1.2.!
        ;
      ;
      ATTRIBUTES          aDatabaseName  INITIAL VALUE Bridge.filteringDatabaseName
                                                       GET,  -- Naming Attribute
                          aDatabaseSize                GET,
                          aNumberOfEntries             GET,
                          aNumberOfDynamicEntries      GET,
                          aAgeingTime                  GET-REPLACE
      ;
      ATTRIBUTE GROUPS    agReadDatabase               aNumberOfDynamicEntries
                                                       aAgeingTime
                                                       -- Add to the group definition
      ;
    ;
  ;
  CONDITIONAL PACKAGES
    pFDGroupSupport PACKAGE
      ATTRIBUTES          aNumberOfGroupRegistrationEntries GET
      ;
      ATTRIBUTE GROUPS    agReadDatabase
                                       aNumberOfGroupRegistrationEntries
                                       -- Add to the group definition
      ;
    REGISTERED AS {Bridge.package fdGroupSupport (0)} ;
  PRESENT IF !Extended Filtering Services (6.6) are supported on this Port.!;
REGISTERED AS {Bridge.moClass filteringDatabase(7)};
```

15.7.1 Name Bindings for Filtering Database

```
nbFilteringDatabase-MACBridgeDLE NAME BINDING
  SUBORDINATE OBJECT CLASS            oFilteringDatabase AND SUBCLASSES;
  NAMED BY SUPERIOR OBJECT CLASS      oMACBridgeDLE AND SUBCLASSES;
```

```
  WITH ATTRIBUTE                          aDatabaseName;
REGISTERED AS {Bridge.nameBinding filteringDatabase-MACBridgeDLE(4)};
```

15.7.2 Number Of Dynamic Entries Attribute definition

```
aNumberOfDynamicEntries ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.NumberOfDynamicEntries  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bNumberOfDynamicEntries BEHAVIOUR
      DEFINED AS          !See 14.7.1.1.3 c).!
    ;
  ;
REGISTERED AS {Bridge.attribute numberOfDynamicEntries(46)};
```

15.7.3 Ageing Time Attribute definition

```
aAgeingTime ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.AgeingTime  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bAgeingTime BEHAVIOUR
      DEFINED AS          !See 14.7.1.1.3 d) and 7.9.2.!
    ;
  ;
REGISTERED AS {Bridge.attribute ageingTime(47)};
```

15.7.4 Number Of Group Registration Entries Attribute definition

```
aNumberOfGroupRegistrationEntries ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.NumberOfGroupRegistrationEntries  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bNumberOfGroupRegistrationEntries BEHAVIOUR
      DEFINED AS          !See 14.7.1.1.3 c).!
    ;
  ;
REGISTERED AS {Bridge.attribute numberOfGroupRegistrationEntries(56)};
```

15.8 Database Entry Managed Object Class definition

```
oDatabaseEntry MANAGED OBJECT CLASS
  DERIVED FROM            “ISO/IEC 10165-2”:top ;
  CHARACTERIZED BY
    pDatabaseEntry PACKAGE
      BEHAVIOUR
        bDatabaseEntry BEHAVIOUR
          DEFINED AS      !The Database Entry managed object class models dynamic
                          entries (Dynamic Filtering Entries and Group Registration
                          Entries) and static entries (Static Filtering Entries)
                          in the filtering and permanent databases, described in
                          14.7.1 and 14.7.5.  The Database Entry managed object class
                          provides the capabilities described in 14.7.6.!
        ;
      ;
      ATTRIBUTES          aAddress                    GET,  -- Naming Attribute
                          aPortMap                    GET-REPLACE,
                          aEntryType                  GET,
                          aEntryIndex                 GET
      ;
      ATTRIBUTE GROUPS    agReadDatabaseEntry
      ;
    ;
  ;
REGISTERED AS {Bridge.moClass databaseEntry(5)};
```

15.8.1 Name Bindings for Database Entry

```
nbPermanentEntry-PermanentDatabase NAME BINDING
  SUBORDINATE OBJECT CLASS               oDatabaseEntry AND SUBCLASSES;
  NAMED BY SUPERIOR OBJECT CLASS         oPermanentDatabase AND SUBCLASSES;
  WITH ATTRIBUTE                         aAddress;
  CREATE;
  DELETE;
REGISTERED AS {Bridge.nameBinding permanentEntry-PermanentDatabase(5)};

nbStaticEntry-FilteringDatabase NAME BINDING
  SUBORDINATE OBJECT CLASS               oDatabaseEntry AND SUBCLASSES;
  NAMED BY SUPERIOR OBJECT CLASS         oFilteringDatabase AND SUBCLASSES;
  WITH ATTRIBUTE                         aAddress;
  CREATE;
  DELETE;
REGISTERED AS {Bridge.nameBinding staticEntry-FilteringDatabase(6)};

nbDynamicEntry-FilteringDatabase NAME BINDING
  SUBORDINATE OBJECT CLASS               oDatabaseEntry AND SUBCLASSES;
  NAMED BY SUPERIOR OBJECT CLASS         oFilteringDatabase AND SUBCLASSES;
  WITH ATTRIBUTE                         aAddress;
  DELETE;
REGISTERED AS {Bridge.nameBinding dynamicEntry-FilteringDatabase(7)};

nbGroupEntry-FilteringDatabase NAME BINDING
  SUBORDINATE OBJECT CLASS               oDatabaseEntry AND SUBCLASSES;
  NAMED BY SUPERIOR OBJECT CLASS         oFilteringDatabase AND SUBCLASSES;
  WITH ATTRIBUTE                         aAddress;
REGISTERED AS {Bridge.nameBinding groupEntry-FilteringDatabase(8)};
```

15.8.2 Address Attribute definition

```
aAddress ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.Address ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bAddress BEHAVIOUR
      DEFINED AS          !See 14.7.6.1.2 b), 14.7.6.2.2 b), 14.7.6.3.2 b),
                          14.7.6.3.3 a), and 14.7.6.4.3 c)1). The attribute is either a
                          MAC Address (Dynamic and Group entries) or a sequence
                          consisting of a Port Number and a MAC Address (Static
                          entries), where the Port number indicates the inbound
                          Port. In the latter case, a Port number of 256
                          is used to indicate “all Ports”.!
    ;
  ;
REGISTERED AS {Bridge.attribute address(57)};
```

15.8.3 Port Map Attribute definition

```
aPortMap ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.PortMap ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bPortMap BEHAVIOUR
      DEFINED AS          !See 14.7.6.3.3 c) and 14.7.6.4.3 c) 3).
                          The PortNumber syntax is used in instances of the Dynamic
                          Filtering Entry managed object class to indicate the
                          Port for which Forwarding is specified.
                          The PortMap syntax is used in Group Registration
                          Entries, and in Static Filtering Entries that are unable
                          to indicate forwarding/filtering based on the use of
                          dynamic filtering information (see 7.9.1). The BITSTRING
                          is used to represent the outbound port forwarding/
                          filtering indicators.  The bit number of a given bit in
                          the BIT-STRING corresponds to the Port Number of the Port
                          to which it refers; a value of 1 in the bit indicates
                          Forwarding, 0 indicates Filtering.
```

```
                          The PortExtendedMap syntax is used in in Static Filtering
                          Entries that are able to indicate forwarding/filtering
                          based on the use of dynamic filtering information (see
                          7.9.1). Pairs of bits in the BITSTRING are used to
                          represent the outbound port forwarding/filtering
                          indicators.  Bits 0 and 1 in the BITSTRING correspond
                          to Port Number 0, Bits 2 and 3 correspond to Port number
                          1; and so on. In each pair of bits, a value of 1 in the
                          lower-numbered bit indicates forwarding or filtering
                          based on dynamic filtering information. A value of 0 in
                          the lower-numbered bit indicates forwarding or filtering
                          based on the value of the higher-numbered bit; a value of
                          1 in the higher-numbered bit indicates Forwarding,
                          0 indicates Filtering.!
    ;
  ;
REGISTERED AS {Bridge.attribute portMap(58)};
```

15.8.4 Entry Type Attribute definition

```
aEntryType ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.EntryType  ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bEntryType BEHAVIOUR
      DEFINED AS          !See 14.7.6.3.3 b), 14.7.6.4.3 c) 2).!
    ;
  ;
REGISTERED AS {Bridge.attribute entryType(50)};
```

15.8.5 Entry Index Attribute definition

```
aEntryIndex ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.EntryIndex  ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bEntryIndex BEHAVIOUR
      DEFINED AS          !The index number of a database entry within a given
                          database (Filtering or Permanent). Its value ranges from
                          0 through (DataBase Size) –1. In a given database, all
                          database entries shall be assigned unique index numbers.!
    ;
  ;
REGISTERED AS {Bridge.attribute entryIndex(51)};
```

15.8.6 Read Database Entry Attribute Group definition

```
agReadDatabaseEntry ATTRIBUTE GROUP
  GROUP ELEMENTS          aPortMap,
                          aEntryType,
                          aEntryIndex
  ;
  FIXED ;
  DESCRIPTION             !This attribute group is used in the mapping of the Read
                          Filtering Entry and Read Filtering Entry Range
                          operations, as defined in Table 15-26 and
                          Table 15-27.! ;
REGISTERED AS {Bridge.attributeGroup readDatabaseEntry(7)};
```

15.9 GARP Application Managed Object Class definition

```
oGARPApplication MANAGED OBJECT CLASS
  DERIVED FROM            “ISO/IEC 10165-2”:top ;
  CHARACTERIZED BY
    pGARPApplication PACKAGE
      BEHAVIOUR
        bGARPApplication BEHAVIOUR
```



```
        DEFINED AS      !The GARP Application managed object class acts as a container
                        object for the GARP Attribute Type object(s)(14.9.2) that relate
                        to a particular GARP Application.!
      ;
    ;
    ATTRIBUTES          aApplicationName            GET -- Naming Attribute
    ;
  ;
;
REGISTERED AS {Bridge.moClass garpApplication(8)};
```

15.9.1 Name Bindings for GARP Application

```
nbGARPApplication-Port NAME BINDING
  SUBORDINATE OBJECT CLASS              oGARPApplication AND SUBCLASSES;
  NAMED BY SUPERIOR OBJECT CLASS        oPort AND SUBCLASSES;
  WITH ATTRIBUTE                        aApplicationName;
REGISTERED AS {Bridge.nameBinding garpApplication-Port(9)};
```

15.9.2 Application Name Attribute definition

```
aApplicationName ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.ApplicationName ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bApplicationName BEHAVIOUR
      DEFINED AS          !The name of a GARP Application, consisting of the GARP
                          Application address of the Application concerned (see 12.3.1).!
  ;
;
REGISTERED AS {Bridge.attribute applicationName(59)};
```

15.10 GARP Attribute Type Managed Object Class definition

```
oGARPAttributeType MANAGED OBJECT CLASS
  DERIVED FROM            "ISO/IEC 10165-2":top ;
  CHARACTERIZED BY
    pGARPAttributeType PACKAGE
      BEHAVIOUR
        boGARPAttributeType BEHAVIOUR
          DEFINED AS      !The GARP Attribute Type managed object class is a
                          container object for managed objects of the GARP
                          Attribute managed object class. An instance of this
                          class is contained in the GARP Application managed object
                          for each Attribute Type supported by that Application.
                          The object also models the ability to modify the Applicant
                          Administrative Control value for the Attribute type, and to
                          read a count of failed registrations for that Attribute type
                          (see 14.9.2).!
        ;
      ;
      ATTRIBUTES          aGARPAttributeType          GET, -- Naming Attribute
                          aApplicantAdministrativeControl
                                                      GET-REPLACE,
                          aFailedRegistrations        GET
      ;
    ;
  ;
REGISTERED AS {Bridge.moClass garpAttributeType(9)};
```

15.10.1 Name Bindings for GARP Attribute Type

```
nbGARPAttributeType-GARPParticipant NAME BINDING
  SUBORDINATE OBJECT CLASS              oGARPAttributeType AND SUBCLASSES;
  NAMED BY SUPERIOR OBJECT CLASS        oGARPParticipant AND SUBCLASSES;
  WITH ATTRIBUTE                        aGARPAttributeType;
REGISTERED AS {Bridge.nameBinding garpAttributeType-GARPParticipant(10)};
```

15.10.2 GARP Attribute Type Attribute definition

```
aGARPAttributeType ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.GARPAttributeType ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bGARPAttributeType BEHAVIOUR
      DEFINED AS          !This attribute carries the value of a GARP Attribute
                          Type for a given GARP Application (see 12.11.2.2).!
    ;
  ;
REGISTERED AS {Bridge.attribute garpAttributeType(64)};
```

15.10.3 Applicant Administrative Control Attribute definition

```
aApplicantAdministrativeControl ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.ApplicantAdministrativeControl ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bApplicantAdministrativeControl BEHAVIOUR
      DEFINED AS          !This attribute carries the value of the Applicant
                          Administrative Control parameter for a
                          GARP Attribute type (see 12.9.2).!
    ;
  ;
REGISTERED AS {Bridge.attribute applicantAdministrativeControl (66)};
```

15.10.4 Failed Registrations Attribute definition

```
aFailedRegistrations ATTRIBUTE
  DERIVED FROM            “ISO/IEC 10165-5”:nonWrapping64BitCounter ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bFailedRegistrations BEHAVIOUR
      DEFINED AS          !The current value of the Failed Registrations counter
                          maintained for a GARP Attribute type (see 14.9.2).!
    ;
  ;
REGISTERED AS {Bridge.attribute failedRegistrations(63)};
```

15.11 GARP Attribute Managed Object Class definition

```
oGARPAttribute MANAGED OBJECT CLASS
  DERIVED FROM            “ISO/IEC 10165-2”:top ;
  CHARACTERIZED BY
    pGARPAttribute PACKAGE
      BEHAVIOUR
        boGARPAttribute BEHAVIOUR
          DEFINED AS      !The GARP Attribute managed object class models the
                          operations that can be performed on the state machine for
                          an instance of a GARP Attribute. An instance of this
                          class is contained in the GARP Attribute Type managed
                          object for each instance of the Attribute type that exists in
                          any GIP Context supported by the Application concerned.!
        ;
      ;
      ATTRIBUTES          aGARPAttribute              GET,  -- Naming Attribute
                          aStateValue                 GET
      ;
    ;
  ;
  CONDITIONAL PACKAGES
    pGARPOriginatorSupport PACKAGE
      ATTRIBUTES          aOriginatorOfLastPDU GET
      ;
    REGISTERED AS {Bridge.package garpOriginatorSupport (1)} ;
```



```
  PRESENT IF              !Recording of last GARP PDU originator address is
                          supported by GARP (14.9.2.1).!;
REGISTERED AS {Bridge.moClass garpAttribute (10)};
```

15.11.1 Name Bindings for GARP Attribute

```
nbGARPAttribute-GARPAttributeType NAME BINDING
  SUBORDINATE OBJECT CLASS              oGARPAttribute AND SUBCLASSES ;
  NAMED BY SUPERIOR OBJECT CLASS        oGARPAttributeType AND SUBCLASSES ;
  WITH ATTRIBUTE                        aGARPAttribute ;
REGISTERED AS {Bridge.nameBinding garpAttribute-GARPAttributeType(11)};
```

15.11.2 GARP Attribute Attribute definition

```
aGARPAttribute ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.GARPAttribute ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bGARPAttribute BEHAVIOUR
      DEFINED AS          !This is the naming attribute for the GARP Attribute managed
                          object class. It consists of a GIP Context identifier value
                          (12.3.3), concatenated with a GARP Attribute value (12.11.2.6).!
    ;
  ;
REGISTERED AS {Bridge.attribute garpAttribute (65)};
```

15.11.3 State Value Attribute definition

```
aStateValue ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.StateValue ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bStateValue BEHAVIOUR
      DEFINED AS          !This attribute carries the value of the combined
                          Applicant and Registrar state for an instance of a
                          GARP Attribute (see 12.8.4).!
    ;
  ;
REGISTERED AS {Bridge.attribute stateValue (67)};
```

15.11.4 OriginatorOfLastPDU Attribute definition

```
aOriginatorOfLastPDU ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.OriginatorOfLastPDU ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bOriginatorOfLastPDU BEHAVIOUR
      DEFINED AS          !This attribute carries the value of source MAC Address
                          of the originator of the last GARP PDU that caused a
                          state change to the GARP State Machine for this instance
                          of the GARP Attribute managed object class
                          (see 14.9.2.1.3).!
    ;
  ;
REGISTERED AS {Bridge.attribute originatorOfLastPDU (68)};
```

15.12 GARP Timers Managed Object Class definition

```
oGARPTimers MANAGED OBJECT CLASS
  DERIVED FROM            “ISO/IEC 10165-2”:top ;
  CHARACTERIZED BY
    pGARPTimers PACKAGE
      BEHAVIOUR
        boGARPTimers BEHAVIOUR
          DEFINED AS      !The GARP Timers managed object class models the
                          operations that can be performed on the timer values used by all
```

```
                          GARP Applications associated with a given Port. An instance of
                          this managed object class is contained in the Port managed
                          object of any Port that supports any GARP Applications.!
        ;
      ;
      ATTRIBUTES          aGARPTimers                  GET,  -- Naming Attribute
                          aJoinTime                    GET-REPLACE,
                          aLeaveTime                   GET-REPLACE,
                          aLeaveAllTime                GET-REPLACE
      ;
    ;
  ;
REGISTERED AS {Bridge.moClass garpTimers (11)};
```

15.12.1 Name Bindings for GARP Timers

```
nbGARPTimers-Port NAME BINDING
  SUBORDINATE OBJECT CLASS              oGARPTimers AND SUBCLASSES ;
  NAMED BY SUPERIOR OBJECT CLASS        oPort AND SUBCLASSES ;
  WITH ATTRIBUTE                        aGARPTimers ;
REGISTERED AS {Bridge.nameBinding garpTimers-Port(12)};
```

15.12.2 GARP Timers Attribute definition

```
aGARPTimers ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.GARPTimers ;
  MATCHES FOR             EQUALITY ;
  BEHAVIOUR
    bGARPTimers BEHAVIOUR
      DEFINED AS          !This is the naming attribute for the GARP Timers object.!
    ;
  ;
REGISTERED AS {Bridge.attribute garpTimers (69)};
```

15.12.3 Join Time Attribute definition

```
aJoinTime ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.JoinTime ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bJoinTime BEHAVIOUR
      DEFINED AS          !The current value of Join Time in use by a GARP
                          Participant (see 12.12.1).!
    ;
  ;
REGISTERED AS {Bridge.attribute joinTime(60)};
```

15.12.4 Leave Time Attribute definition

```
aLeaveTime ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.LeaveTime ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bLeaveTime BEHAVIOUR
      DEFINED AS          !The current value of Leave Time in use by a GARP
                          Participant (see 12.12.1).!
    ;
  ;
REGISTERED AS {Bridge.attribute leaveTime(61)};
```

15.12.5 Leave All Time Attribute definition

```
aLeaveAllTime ATTRIBUTE
  WITH ATTRIBUTE SYNTAX   Bridge.LeaveAllTime ;
  MATCHES FOR             EQUALITY, ORDERING ;
  BEHAVIOUR
    bLeaveAllTime BEHAVIOUR
```



```
      DEFINED AS          !The current value of Leave All Time in use by a GARP
                          Participant (see 12.12.1).!
    ;
  ;
REGISTERED AS {Bridge.attribute leaveAllTime(62)};
```

15.13 ASN.1 definitions

The following ASN.1 module defines the data types and data values necessary to complete the managed object class definitions for the MAC Bridge.

```
Bridge     {iso(1) member-body(2) us(840) ieee802dot1D(10009) asn1Module(2) bridgeDefinitions(0) version2(1)}
DEFINITIONS ::= BEGIN

IMPORTS
-- Import MACAddress from IEEE Std. 802.1F
MACAddress
FROM IEEE802CommonDefinitions      {iso(1) member-body(2) us(840) ieee802-1F(10011)
                                   asn1module(2) commondefinitions(0) version1(0) }
; -- End of IMPORTS

-- Define short-form identifiers for OID prefixes
bridge              OBJECT IDENTIFIER ::={iso(1) member-body(2) us(840) ieee802dot1D(10009)}
action              OBJECT IDENTIFIER ::={bridge action(9)}
asn1Module          OBJECT IDENTIFIER ::={bridge asn1Module(2)}
attribute           OBJECT IDENTIFIER ::={bridge attribute(7)}
attributeGroup      OBJECT IDENTIFIER ::={bridge attributeGroup(8)}
extensions          OBJECT IDENTIFIER ::={bridge standardSpecificExtensions(0)}
moClass             OBJECT IDENTIFIER ::={bridge managedObjectClass(3)}
nameBinding         OBJECT IDENTIFIER ::={bridge nameBinding(6)}
notification        OBJECT IDENTIFIER ::={bridge notification(10)}
package             OBJECT IDENTIFIER ::={bridge package(4)}
parameter           OBJECT IDENTIFIER ::={bridge parameter(5)}

-- Define other Types and Values used by the GDMO definitions
AccessPriority                  ::= INTEGER                 -- Range 0 through 7
Address                         ::= CHOICE          {[0] MACAddress,            -- Dynamic entries
                                                     [1] SEQUENCE                {PortNumber,
                                                                                  MACAddress}
                                                     }                          -- Static entries
AgeingTime                      ::= INTEGER                 -- Units of 1/256 seconds
ApplicantAdministrativeControl  ::= BOOLEAN         {disabled FALSE,
                                                     enabled TRUE}
ApplicationName                 ::= GARPApplicationAddress
BridgeAddress                   ::= MACAddress
BridgeForwardDelay              ::= ForwardDelay            -- Range 4.0 - 30.0 seconds
BridgeHelloTime                 ::= HelloTime               -- Range 1.0 - 10.0 seconds
BridgeIdentifier                ::= SEQUENCE        {BridgePriority,
                                                     BridgeAddress }
BridgeMaxAge                    ::= MaxAge                  -- Range 6.0 - 40.0 seconds
BridgeName                      ::= GraphicString           -- Max 32 characters long
BridgeNumPorts                  ::= INTEGER                 -- Range 1 - 255
BridgePortAddresses             ::= SET OF SEQUENCE         {PortNumber,
                                                             PortAddress }
BridgePriority                  ::= INTEGER                 -- Range 0 - 65535
DatabaseName                    ::= GraphicString
DatabaseSize                    ::= INTEGER
DesignatedBridge                ::= BridgeIdentifier
DesignatedCost                  ::= INTEGER
DesignatedPort                  ::= PortIdentifier
DesignatedRoot                  ::= BridgeIdentifier
DiscardOnErrorDetail            ::= SET OF SEQUENCE         {sourceAddress      MACAddress,
                                                             reason              DiscardReason }
```

```
                                                --16 elements in the SET
DiscardReason                       ::= INTEGER     {frameTooLarge(1)
                                                    -- No other reasons at present --}
EntryIndex                          ::= INTEGER
EntryType                           ::= INTEGER     {static(1),
                                                    dynamic(0) }
filteringDatabaseName GraphicString ::= "fdb"
ForwardDelay                        ::= INTEGER     -- Units of 1/256 seconds
GARPApplicationAddress              ::= MACAddress
GARPAttribute                       ::= SEQUENCE    {GIPContextID,
                                                    attributeValue OCTETSTRING}
-- The Attribute Value field carries an nAttribute Value encoded as a sequence of octets, as defined by the GARP Application
-- within which the Attribute is used. The encoding and interpretation of the octet string is as defined for such attribute
-- values when carried as a sequence of octets in GARP PDUs (12.11).
GARPAttributeType                   ::= INTEGER
GARPTimers                          ::= GraphicString ("GT")
GIPContextID                        ::= INTEGER
HelloTime                           ::= INTEGER     -- Units of 1/256 seconds
HoldTime                            ::= INTEGER     -- Units of 1/256 seconds
JoinTime                            ::= INTEGER     -- Units of centiseconds
LeaveTime                           ::= INTEGER     -- Units of centiseconds
LeaveAllTime                        ::= INTEGER     -- Units of centiseconds
MaxAge                              ::= INTEGER     -- Units of 1/256 seconds
NumberOfEntries                     ::= INTEGER
NumberOfDynamicEntries              ::= INTEGER
NumberOfGroupRegistrationEntries    ::= INTEGER
OriginatorOfLastPDU                 ::= MACAddress
OutboundAccessPriorityTable         ::= SEQUENCE    {AccessPriority,    -- for user priority = 0
                                                    AccessPriority,     -- for user priority = 1
                                                    AccessPriority,     -- for user priority = 2
                                                    AccessPriority,     -- for user priority = 3
                                                    AccessPriority,     -- for user priority = 4
                                                    AccessPriority,     -- for user priority = 5
                                                    AccessPriority,     -- for user priority = 6
                                                    AccessPriority}     -- for user priority = 7
permanentDatabaseName GraphicString ::= "pdb"
PortAddress                         ::= MACAddress
PortBitmap                          ::= BITSTRING
PortExtendedMap                     ::= BITSTRING
PortIdentifier                      ::= SEQUENCE    {PortPriority,
                                                    PortNumber }
PortName                            ::= GraphicString       -- Max 32 characters
PortNumber                          ::= INTEGER
PortMap                             ::= CHOICE      {[0] PortNumber,
                                                     [1] PortBitmap
                                                     [2] PortExtendedMap}
PortPriority                        ::= INTEGER             -- Range 0 - 255
PathCost                            ::= INTEGER             -- Range 1 - 65535
PortType                            ::= OBJECT IDENTIFIER
RootPathCost                        ::= INTEGER
RootPort                            ::= PortIdentifier
ForcePortState                      ::= State -- new State is limited to either Disabled or Blocking only
State                               ::= INTEGER     {disabled (0),      -- Port State
                                                    listening (1),
                                                    learning (2),
                                                    forwarding (3),
                                                    blocking (4)}
StateValue                          ::= INTEGER     {va-mt (0),
                                                    va-lv (1),
                                                    vp-mt (2),
                                                    vp-lv (3),
                                                    vo-mt (4),
                                                    vo-lv (5),
                                                    va-in (6),
                                                    vp-in (7),
```

```
                                        vo-in (8),
                                        aa-mt (9),
                                        aa-lv (10),
                                        aa-in (11),
                                        ap-in (12),
                                        ao-in (13),
                                        qa-mt (14),
                                        qa-lv (15),
                                        qa-in (16),
                                        qp-in (17),
                                        qo-in (18),
                                        la-mt (19),
                                        la-lv (20),
                                        lo-mt (21),
                                        lo-lv (22),
                                        la-in (23)}
TopologyChange              ::= BOOLEAN
TopologyChangeAck           ::= BOOLEAN
TrafficClassTable           ::= SEQUENCE {
                                    noOfSupportedClasses        INTEGER, -- Range 0 through 7
                                    SEQUENCE OF TrafficClassTableEntry}
-- The number of elements in the SEQUENCE OF is exactly equal to the number of traffic
-- classes supported on the Port.
TrafficClassTableEntry      ::= SEQUENCE    {trafficClass       INTEGER, -- Range 0 through 7
                                            priorities          SET OF UserPriority}
-- Each possible user priority value appears exactly once within the Traffic Class Table; i.e.,
-- it is not permitted to create a table in which the mapping from user priority to traffic
-- class is ambiguous or undefined.
TypeValue                   ::= INTEGER             -- Range 0 - 65535
Uptime                      ::= INTEGER             -- Units of 1/256 secs
UserPriority                ::= INTEGER             -- Range 0 through 7
UserPriorityRegenerationTable ::= SEQUENCE  {UserPriority,      -- for received priority = 0
                                            UserPriority,       -- for received priority = 1
                                            UserPriority,       -- for received priority = 2
                                            UserPriority,       -- for received priority = 3
                                            UserPriority,       -- for received priority = 4
                                            UserPriority,       -- for received priority = 5
                                            UserPriority,       -- for received priority = 6
                                            UserPriority}       -- for received priority = 7
END
```

16. Bridge performance

This clause specifies a set of parameters that represent the performance of a Bridge. These parameters have been selected to allow a basic level of confidence to be established in a Bridge, for use in an initial determination of its suitability for a given application. They cannot be considered to provide an exhaustive description of the performance of a Bridge. It is recommended that further performance information be provided and sought concerning the applicability of a Bridge implementation.

The following set of performance parameters is defined:

a) Guaranteed Port Filtering Rate, and a related time interval T_F that together characterize the traffic for which filtering is guaranteed.
b) Guaranteed Bridge Relaying Rate, and a related time interval T_R.

16.1 Guaranteed Port Filtering Rate

For a specific Bridge Port, a valid **Guaranteed Port Filtering Rate**, in frames per second, is a value that, given any set of frames from the specific Bridge Port to be filtered during any T_F interval, the Forwarding Process shall filter all of the set as long as all of the following are true:

a) The number of frames in the set does not exceed the specific Bridge Port's **Guaranteed Port Filtering Rate** multiplied by T_F.
b) The **Guaranteed Port Filtering Rate** of each of the other Bridge Port(s) is not exceeded.
c) The **Guaranteed Bridge Relaying Rate** is not exceeded.
d) Relayed frames are not discarded due to output congestion (7.7.3).
e) The information upon which the filtering decisions are based has been configured in the Filtering Database prior to the start of time interval T_F.

16.2 Guaranteed Bridge Relaying Rate

For a Bridge, a valid **Guaranteed Bridge Relaying Rate**, in frames per second, is a value that given any set of frames from the specific Bridge Port to be relayed during any T_R interval, the Forwarding Process shall relay all of the set as long as all of the following are true:

a) The number of frames in the set does not exceed the Bridge's **Guaranteed Bridge Relaying Rate** multiplied by T_R.
b) The **Guaranteed Port Filtering Rate** of each Bridge Port is not exceeded.
c) Relayed frames are not discarded due to output congestion (7.7.3).
d) The information upon which the forwarding decisions are based has been configured in the Filtering Database prior to the start of time interval T_R.

Annex A

(normative)

PICS proforma[1]

A.1 Introduction

The supplier of a protocol implementation which is claimed to conform to this standard shall complete the following Protocol Implementation Conformance Statement (PICS) proforma. If support of Source Routing Transparent Bridge Operation is claimed, then the supplier shall also complete the Protocol Implementation Conformance Statement (PICS) proforma described in Annex D.

A completed PICS proforma is the PICS for the implementation in question. The PICS is a statement of which capabilities and options of the protocol have been implemented. The PICS can have a number of uses, including use

a) By the protocol implementor, as a checklist to reduce the risk of failure to conform to the standard through oversight;
b) By the supplier and acquirer—or potential acquirer—of the implementation, as a detailed indication of the capabilities of the implementation, stated relative to the common basis for understanding provided by the standard PICS proforma;
c) By the user—or potential user—of the implementation, as a basis for initially checking the possibility of interworking with another implementation (note that, while interworking can never be guaranteed, failure to interwork can often be predicted from incompatible PICSs);
d) By a protocol tester, as the basis for selecting appropriate tests against which to assess the claim for conformance of the implementation.

A.2 Abbreviations and special symbols

A.2.1 Status symbols

M	mandatory
O	optional
O.n	optional, but support of at least one of the group of options labelled by the same numeral *n* is required
X	prohibited
pred:	conditional-item symbol, including predicate identification: see A.3.4
¬	logical negation, applied to a conditional item's predicate

A.2.2 General abbreviations

N/A	not applicable
PICS	Protocol Implementation Conformance Statement

[1]*Copyright release for PICS proformas:* Users of this standard may freely reproduce the PICS proforma in this annex so that it can be used for its intended purpose and may further publish the completed PICS.

A.3 Instructions for completing the PICS proforma

A.3.1 General structure of the PICS proforma

The first part of the PICS proforma, implementation identification and protocol summary, is to be completed as indicated with the information necessary to identify fully both the supplier and the implementation.

The main part of the PICS proforma is a fixed-format questionnaire, divided into several subclauses, each containing a number of individual items. Answers to the questionnaire items are to be provided in the rightmost column, either by simply marking an answer to indicate a restricted choice (usually Yes or No), or by entering a value or a set or range of values. (Note that there are some items where two or more choices from a set of possible answers can apply; all relevant choices are to be marked.)

Each item is identified by an item reference in the first column. The second column contains the question to be answered; the third column records the status of the item—whether support is mandatory, optional, or conditional; see also A.3.4 below. The fourth column contains the reference or references to the material that specifies the item in the main body of this standard, and the fifth column provides the space for the answers.

A supplier may also provide (or be required to provide) further information, categorized as either Additional Information or Exception Information. When present, each kind of further information is to be provided in a further subclause of items labelled *Ai* or *Xi*, respectively, for cross-referencing purposes, where *i* is any unambiguous identification for the item (e.g., simply a numeral). There are no other restrictions on its format and presentation.

A completed PICS proforma, including any Additional Information and Exception Information, is the Protocol Implementation Conformation Statement for the implementation in question.

NOTE—Where an implementation is capable of being configured in more than one way, a single PICS may be able to describe all such configurations. However, the supplier has the choice of providing more than one PICS, each covering some subset of the implementation's configuration capabilities, in case that makes for easier and clearer presentation of the information.

A.3.2 Additional information

Items of Additional Information allow a supplier to provide further information intended to assist the interpretation of the PICS. It is not intended or expected that a large quantity will be supplied, and a PICS can be considered complete without any such information. Examples might be an outline of the ways in which a (single) implementation can be set up to operate in a variety of environments and configurations, or information about aspects of the implementation that are outside the scope of this standard but that have a bearing upon the answers to some items.

References to items of Additional Information may be entered next to any answer in the questionnaire, and may be included in items of Exception Information.

A.3.3 Exception information

It may occasionally happen that a supplier will wish to answer an item with mandatory status (after any conditions have been applied) in a way that conflicts with the indicated requirement. No pre-printed answer will be found in the Support column for this: instead, the supplier shall write the missing answer into the Support column, together with an *Xi* reference to an item of Exception Information, and shall provide the appropriate rationale in the Exception item itself.

An implementation for which an Exception item is required in this way does not conform to this standard.

NOTE—A possible reason for the situation described above is that a defect in this standard has been reported, a correction for which is expected to change the requirement not met by the implementation.

A.3.4 Conditional status

A.3.4.1 Conditional items

The PICS proforma contains a number of conditional items. These are items for which both the applicability of the item itself, and its status if it does apply—mandatory or optional—are dependent upon whether or not certain other items are supported.

Where a group of items is subject to the same condition for applicability, a separate preliminary question about the condition appears at the head of the group, with an instruction to skip to a later point in the questionnaire if the "Not Applicable" answer is selected. Otherwise, individual conditional items are indicated by a conditional symbol in the Status column.

A conditional symbol is of the form "**pred:** S" where **pred** is a predicate as described in A.3.4.2 below, and S is a status symbol, M or 0.

If the value of the predicate is true (see A.3.4.2), the conditional item is applicable, and its status is indicated by the status symbol following the predicate: the answer column is to be marked in the usual way. If the value of the predicate is false, the "Not Applicable" (N/A) answer is to be marked.

A.3.4.2 Predicates

A predicate is one of the following:

a) An item-reference for an item in the PICS proforma: the value of the predicate is true if the item is marked as supported, and is false otherwise;
b) A predicate-name, for a predicate defined as a boolean expression constructed by combining item-references using the boolean operator OR: the value of the predicate is true if one or more of the items is marked as supported;
c) The logical negation symbol "¬" prefixed to an item-reference or predicate-name: the value of the predicate is true if the value of the predicate formed by omitting the "¬" symbol is false, and vice versa.

Each item whose reference is used in a predicate or predicate definition, or in a preliminary question for grouped conditional items, is indicated by an asterisk in the Item column.

A.4 PICS proforma for IEEE Std 802.1D-1998

A.4.1 Implementation identification

Supplier	
Contact point for queries about the PICS	
Implementation Name(s) and Version(s)	
Other information necessary for full identification—e.g., name(s) and version(s) of machines and/or operating system names	
NOTE 1—Only the first three items are required for all implementations; other information may be completed as appropriate in meeting the requirement for full identification. NOTE 2—The terms Name and Version should be interpreted appropriately to correspond with a supplier's terminology (e.g., Type, Series, Model).	

A.4.2 Protocol summary, IEEE Std 802.1D-1998

Identification of protocol specification	IEEE Std 802.1D-1998, Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Common specifications—Media Access Control (MAC) Bridges
Identification of amendments and corrigenda to the PICS proforma which have been completed as part of the PICS	Amd. : Corr. : Amd. : Corr. :
Have any Exception items been required? (See A.3.3: the answer Yes means that the implementation does not conform to IEEE Std 802.1D-1998.)	No [ ] Yes [ ]

Date of Statement	

A.5 Major capabilities and options

Item	Feature	Status	References	Support
(1a)	Communications Support Which Media Access Control types are supported on Bridge Ports, implemented in conformance with the relevant MAC standards?		6.5	
(1a.1)*	CSMA/CD, IEEE Std 802.3	O.1		Yes [ ] No [ ]
(1a.2)*	Token Bus, ISO/IEC 8802-4	O.1		Yes [ ] No [ ]
(1a.3)*	Token Ring, ISO/IEC 8802-5	O.1		Yes [ ] No [ ]
(1a.4)*	FDDI, ISO 9314-2	O.1		Yes [ ] No [ ]
(1a.5)*	DQDB, ISO/IEC 8802-6	O.1		Yes [ ] No [ ]
(1a.6)*	ISLAN, ISO/IEC 8802-9	O.1		Yes [ ] No [ ]
(1a.7)*	ISLAN 16-T, IEEE Std 802.9a	O.1		Yes [ ] No [ ]
(1a.8)*	Demand Priority, ISO/IEC 8802-12 (IEEE Std 802.3 format)	O.1		Yes [ ] No [ ]
(1a.9)*	Demand Priority, ISO/IEC 8802-12 (ISO/IEC 8802-5 format)	O.1		Yes [ ] No [ ]
(1a.11)*	Wireless LAN, ISO/IEC DIS 8802-11	O.1		Yes [ ] No [ ]
(1b)	Is LLC Type 1 supported on all Bridge Ports in conformance with ISO/IEC 8802-2?	M	7.2, 7.3, 7.12, ISO/IEC 8802-2	Yes [ ]
(1c)	Is Source-Routing Transparent Bridge operation supported on any of the Bridge Ports? (If support is claimed, the PICS proforma detailed in Annex D shall also be completed.).	O	Annex C	Yes [ ] No [ ]
(2)	Relay and filtering of frames (A.6)	M	7.5, 7.6, 7.7	Yes [ ]
(2a)	Does the Bridge support Basic Filtering Services?	M	6.6.5, 7.7.2	Yes [ ]
(2b)*	Does the Bridge support Extended Filtering Services? If item (2b) is not supported, mark “N/A” and continue at (2e).	O	6.6.5, 7.7.2	Yes [ ] No [ ] N/A[ ]
(2c)*	Does the Bridge support dynamic Group forwarding and filtering behavior?	2b:M	6.6.5	Yes [ ] No [ ]
(2d)*	Does the Bridge support the ability for static filtering information for individual MAC Addresses to specify a subset of Ports for which forwarding or filtering decisions are taken on the basis of dynamic filtering information?	2b:O	6.6.5	Yes [ ] No [ ]
(2e)	Does the Bridge support expedited traffic classes on any of its Ports?	O	7.1.2, 7.7.3	Yes [ ] No [ ]
(4)*	Does the Bridge support management of the priority of relayed frames?	O	6.5, 7.5.1, 7.7.3, 7.7.5, Table 7-1, Table 7-2, Table 7-3	Yes [ ] No [ ]
(5)	Maintenance of filtering information (A.7)	M	7.8, 7.9	Yes [ ]
(7a)	Can the Filtering Database be read by management?	O	7.9	Yes [ ] No [ ]

A.5 Major capabilities and options *(Continued)*

Item	Feature	Status	References	Support
(7c)*	Can Static Filtering Entries be created and deleted?	O	7.9.1	Yes [ ] No [ ]
(7g)	Can Static Filtering Entries be created and deleted in the Permanent Database?	O	7.9.6	Yes [ ] No [ ]
(7h)	Can Static Filtering Entries be created for a given MAC Address specification with a distinct Port Map for each inbound Port?	O	7.9.1	Yes [ ] No [ ]
(7i)	Can Group Registration Entries be dynamically created, updated and deleted by GMRP?	2c:M	7.9.3, 10	Yes [ ] N/A [ ]
(10)	Addressing (A.8)	M	7.12	Yes [ ]
(9a)*	Can the Bridge be configured to use 48-bit Universal Addresses?	O.3	7.12	Yes [ ] No [ ]
(9b)*	Can the Bridge be configured to use 48-bit Local Addresses?	O.3	7.12	Yes [ ] No [ ]
(13)*	Spanning Tree algorithm and protocol (A.9)	M	8, 9	Yes [ ]
(16)*	Does the Bridge support management of the Spanning Tree topology?	O	8.2	Yes [ ] No [ ]
(17)*	Does the Bridge support management of the protocol timers?	O	8.10	Yes [ ] No [ ]
(19)*	Bridge Management Operations	O	14	Yes [ ] No [ ]
(20a)*	Are the Bridge Management Operations supported via a Remote Management Protocol?	19:O.4	5	Yes [ ] No [ ] N/A [ ]
(20b)*	Are the Bridge Management Operations supported via a local management interface?	19:O.4	5	Yes [ ] No [ ] N/A [ ]

A.6 Relay and filtering of frames

Item	Feature	Status	References	Support
(2f)	Are received frames with media access method errors discarded?	M	6.4, 7.5	Yes [ ]
(2g)	Are correctly received frames submitted to the Learning Process?	M	7.5	Yes [ ]
(2h)	Are user data frames the only type of frame relayed?	M	7.5	Yes [ ]
(2i)	Are request with no response frames the only frames relayed?	M	7.5	Yes [ ]
(2j)	Are all frames addressed to the Bridge Protocol Entity submitted to it?	M	7.5	Yes [ ]
(2k)	Are user data frames the only type of frame transmitted?	M	7.6	Yes [ ]

A.6 Relay and filtering of frames *(Continued)*

Item	Feature	Status	References	Support
(2l)	Are request with no response frames the only frames transmitted?	M	7.6	Yes [ ]
(2m)	Are relayed frames queued for transmission only under the conditions in 7.7.3?	M	7.7.3, 8.4	Yes [ ]
(2n)	Is the order of relayed frames preserved in accordance with the requirements of the forwarding process?	M	7.7.3, 7.1.1	Yes [ ]
(2o)	Is a relayed frame submitted to a MAC Entity for transmission only once?	M	7.7.4, 6.3.4	Yes [ ]
(2p)	Is a maximum bridge transit delay enforced for relayed frames?	M	7.7.3	Yes [ ]
(2q)	Are queued frames discarded if a Port leaves the Forwarding State?	M	7.7.3	Yes [ ]
(2r)	Is the user priority of relayed frames preserved where possible?	M	6.4	Yes [ ]
(2s)	Is the user priority set to the Default User Priority for the reception Port otherwise?	M	6.4	Yes [ ]
(2t)	Is the user priority regenerated by means of the User Priority Regeneration Table?	M	7.5.1, Table 7-1	Yes [ ]
(2u)	Is mapping of Regenerated User Priority to Traffic Class performed by means of the Traffic Class Table?	M	7.7.3, Table 7-2	Yes [ ]
(2v)	Is the access priority derived from the Regenerated User Priority as defined by the values in Table 7-3 for each outbound media access method supported by the Bridge?	M	7.7.5, Table 7-3	Yes [ ]
(2w)	Does the implementation introduce an undetected frame error rate greater than that achievable by preserving the FCS?	X	7.7.6, 6.3.7	No [ ]
(2x)	Is the FCS of frames relayed between Ports of the same MAC type preserved?	O	7.7.6	Yes [ ] No [ ]
(2y)	Does the Bridge generate an M_UNITDATA.indication primitive on receipt of a valid frame transmitted by the Bridge Port's local MAC entity?	MS1:X	6.5.4, ISO 9314-2	No [ ] N/A [ ]
(2z)	Is only Asynchronous service used?	MS1:M	ISO 9314-2 Clause 8.1.4	Yes [ ] N/A [ ]
(2aa)	On receiving a frame from an FDDI ring for forwarding, does the bridge set the C indicator?	MS1:O	6.5.4, ISO 9314-2 Clause 7.3.8	Yes [ ] No [ ] N/A [ ]
(2ab)	On receiving a frame from an FDDI ring for forwarding, does the bridge leave the C indicator unaltered?	MS1:O	6.5.4, ISO 9314-2 Clause 7.3.8	Yes [ ] No [ ] N/A [ ]
	If item 4 is not supported, mark "N/A" and continue at item (4d).			N/A [ ]

A.6 Relay and filtering of frames *(Continued)*

Item	Feature	Status	References	Support
(4a)*	Can the Default User Priority parameter for each Port be set to any value in the range 0 through 7?	4:O.5	6.4	Yes [ ] No [ ]
(4b)*	Can the entries in the User Priority Regeneration Table for each Port be set to the full range of values shown in Table 7-1?	4:O.5	7.5.1, Table 7-1	Yes [ ] No [ ]
(4c)	Can the entries in the Traffic Class Table for each Port be set to the full range of values shown in Table 7-2?	MS2:O	7.7.3, Table 7-2	Yes [ ] No [ ] N/A [ ]
	If item 4 is supported, mark "N/A" and continue at item (4g).			N/A [ ]
(4d)	Does the Bridge support the recommended default value of the Default User Priority parameter for each Port?	¬ 4:M	6.4	Yes [ ]
(4e)	Does the Bridge support the recommended default mappings between received user priority and Regenerated User Priority for each Port as defined in Table 7-1?	¬ 4:M	7.5.1, Table 7-1	Yes [ ]
(4f)	Does the Bridge support the recommended default user_priority to traffic class mappings shown in Table 7-2 for each Port?	MS3:M	7.7.3, Table 7-2	Yes [ ] N/A [ ]
(4g)	Is the Bridge able to use any values other than those shown in Table 7-3 when determining the access priority for the media access methods shown?	X	7.7.5, Table 7-3	No [ ]

Predicates:
MS1 = 1a.4 AND NOT (1a.1 OR 1a.2 OR 1a.3 OR 1a.5 OR 1a.6 OR 1a.7 OR 1a.8 OR 1a.9)
MS2 = 2d AND 4
MS3 = 2d AND NOT 4

A.7 Maintenance of filtering entries in the Filtering Database

Item	Feature	Status	References	Support
(5a)	Are Dynamic Filtering Entries created and updated if and only if the Port State permits?	M	7.8, 7.9.2, 8.4	Yes [ ]
(5b)	Are Dynamic Filtering Entries created on receipt of frames with a group source address?	X	7.8, 7.9.2	No [ ]
(5c)	Does the Filtering Database support Static Filtering Entries?	M	7.9.1	Yes [ ]
(5d)	Can a Dynamic Filtering Entry be created that conflicts with an existing Static Filtering Entry?	X	7.8, 7.9, 7.9.1, 7.9.2	No [ ]

A.7 Maintenance of filtering entries in the Filtering Database *(Continued)*

Item	Feature	Status	References	Support
(5e)	Does the Filtering Database support Dynamic Filtering Entries?	M	7.9.2	Yes [ ]
(5f)	Does the creation of a Static Filtering Entry remove any conflicting information in a Dynamic Filtering Entry for the same address?	M	7.9.1, 7.9.2	Yes [ ]
(5g)	Does each Static Filtering Entry specify a MAC Address specification and a Port Map?	M	7.9.1	Yes [ ]
(5h)	Are Dynamic Filtering Entries removed from the Filtering Database if not updated for the Ageing Time period?	M	7.9.2	Yes [ ]
(5i)	Does each Dynamic Filtering Entry specify a MAC Address specification and a Port Map?	M	7.9.2	Yes [ ]
(5j)	Is the Filtering Database initialized with the entries contained in the Permanent Database?	M	7.9.6	Yes [ ]
	If item (2c) is not supported, mark N/A and continue at item (6a).			N/A [ ]
(5k)	Does each Group Registration Entry specify a MAC Address specification and a Port Map?	2c:M	7.9.3	Yes [ ]
(5l)	Can the MAC Address specification in Group Registration Entries represent All Groups, All Unregistered Groups, or a specific group MAC Address?	2c:M	7.9.3	Yes [ ]
(5m)	Are Group Registration Entries created, updated and removed from the Filtering Database in accordance with the specification of GMRP?	2c:M	7.9.3, 10	Yes [ ]
(5n)	Are Group Registration Entries created, updated and removed from the Filtering Database by any means other than via the operation of GMRP?	2c:X	7.9.3, 10	No [ ]
(6a)	State the Filtering Database Size.	M	7.9	____ entries
(6b)	State the Permanent Database Size.	M	7.9	____ entries
	If item (7c) is not supported, mark N/A and continue at item (8a).			N/A [ ]
(7d)	Can Static Filtering Entries be made for individual MAC Addresses?	**7c:**M	7.9.1	Yes [ ]
(7e)	Can Static Filtering Entries be made for group MAC Addresses?	**7c:**M	7.9.1	Yes [ ]

A.7 Maintenance of filtering entries in the Filtering Database *(Continued)*

Item	Feature	Status	References	Support
(7f)	Can a Static Filtering Entry be made for the broadcast MAC Address?	**7c:**M	7.9.1	Yes [ ]
(8a)	Can the Bridge be configured to use the default value of Ageing Time recommended in Table 7-4?	O	7.9.2, Table 7-4	Yes [ ] No [ ]
(8b)	Can the Bridge be configured to use any of the range of values of Ageing Time specified in Table 7-4?	O	7.9.2, Table 7-4	Yes [ ] No [ ]

A.8 Addressing

Item	Feature	Status	References	Support
(10a)	Does each Port have a separate MAC Address?	M	7.12.2	Yes [ ]
(10b)	Are all BPDUs transmitted to the same group address?	M	7.12.3, 8.2	Yes [ ]
	If item (9a) is not supported, mark N/A and continue at item (10d1).			N/A [ ]
(10c)	Are all BPDUs transmitted to the Bridge Protocol Group Address when Universal Addresses are used?	**9a:**M	7.12.3, 8.2	Yes [ ]
(10d)	Is the source address of BPDUs the address of the transmitting Port?	**9a:**M	7.12.3	Yes [ ]
(10d1)	Is the LLC address of BPDUs the standard LLC address identified for the Spanning Tree Protocol?	M	7.12.3, Table 7-8	Yes [ ]
(10e)	Is the Bridge Address a Universal Address?	M	7.12.5, 8.2	Yes [ ]
(10f)	Are frames addressed to any of the Reserved Addresses relayed by the Bridge?	X	7.12.6	No [ ]
	If item (13) is not supported, mark N/A and continue at item (11c).			N/A [ ]
(11a)	Is Bridge Management accessible through each Port using the MAC Address of the Port and the LSAP assigned?	**13:**O	7.12.4	Yes [ ] No [ ]
(11b)	Is Bridge Management accessible through all Ports using the All LANs Bridge Management Group Address?	13:O	7.12.4	Yes [ ] No [ ]
(11c)	Is the Bridge Address the Address of Port 1?	**9a:**O	7.12.5	Yes [ ] No [ ] N/A [ ]
(11d)	Are Group Addresses additional to the Reserved Addresses preconfigured in the Permanent Database?	O	7.12.6	Yes [ ] No [ ]

A.8 Addressing *(Continued)*

Item	Feature	Status	References	Support
	If item (11d) is not supported, mark N/A and continue at item (12a).			N/A [ ]
(11e)	Can the additional preconfigured entries in the Filtering Database be deleted?	**11d:**O	7.12.6	Yes [ ] No [ ]
(12a)	Can a group MAC Address be assigned to identify the Bridge Protocol Entity?	**9b:**M	8.2	Yes [ ] N/A [ ]
(12c)	Does each Port of the Bridge have a distinct identifier?	M	8.2, 8.5.5.1	Yes [ ]

A.9 Spanning Tree Algorithm

Item	Feature	Status	References	Support
(13a)	Are all the following Bridge Parameters maintained?	M	8.5.3	Yes [ ]
	Designated Root		8.5.3.1	
	Root Cost		8.5.3.2	
	Root Port		8.5.3.3	
	Max Age		8.5.3.4	
	Hello Time		8.5.3.5	
	Forward Delay		8.5.3.6	
	Bridge Identifier		8.5.3.7	
	Bridge Max Age		8.5.3.8	
	Bridge Hello Time		8.5.3.9	
	Bridge Forward Delay		8.5.3.10	
	Topology Change Detected		8.5.3.11	
	Topology Change		8.5.3.12	
	Topology Change Time		8.5.3.13	
	Hold Time		8.5.3.14	
(13b)	Are all the following Bridge Timers maintained?	M	8.5.4	Yes [ ]
	Hello Timer		8.5.4.1	
	Topology Change Notification Timer		8.5.4.2	
	Topology Change Timer		8.5.4.3	
(13c)	Are all the following Port Parameters maintained for each Port?	M	8.5.5	Yes [ ]
	Port Identifier		8.5.5.1	
	State		8.5.5.2, 8.4	

A.9 Spanning Tree Algorithm *(Continued)*

Item	Feature	Status	References	Support
	Path Cost		8.5.5.3	
	Designated Root		8.5.5.4	
	Designated Cost		8.5.5.5	
	Designated Bridge		8.5.5.6	
	Designated Port		8.5.5.7	
	Topology Change Acknowledge		8.5.5.8	
	Configuration Pending		8.5.5.9	
	Change Detection Enabled		8.5.5.10	
(13d)	Are all the following Timers maintained for each Port?	M	8.5.6	Yes [ ]
	Message Age Timer		8.5.6.1	
	Forward Delay Timer		8.5.6.2	
	Hold Timer		8.5.6.3	
(13e)	Are Protocol Parameters and Timers maintained, and BPDUs transmitted, as required on each of the following events?	M	8.7, 8.9, 8.5.3, 8.5.4, 8.5.5, 8.5.6	Yes [ ]
	Received Configuration BPDU		8.7.1	
	Received Topology Change Notification BPDU		8.7.2	
	Hello Timer Expiry		8.7.3	
	Message Age Timer Expiry		8.7.4	
	Forward Delay Timer Expiry		8.7.5	
	Topology Change Notification Timer Expiry		8.7.6	
	Topology Change Timer Expiry		8.7.7	
	Hold Timer Expiry		8.7.8	
(13f)	Do the following operations modify Protocol Parameters and Timers, and transmit BPDUs as required?	M	8.8, 8.9, 8.5.3, 8.5.4, 8.5.5, 8.5.6	Yes [ ]
	Initialization		8.8.1	
	Enable Port		8.8.2	
	Disable Port		8.8.3	
	Set Bridge Priority		8.8.4	
	Set Port Priority		8.8.5	
	Set Path Cost		8.8.6	
(13g)	Does the implementation support the ability to set the value of the Change Detection Enabled parameter to Disabled?	O	8.5.5.10	Yes [ ] No [ ]

A.9 Spanning Tree Algorithm *(Continued)*

Item	Feature	Status	References	Support
(14a)	Does the Bridge underestimate the increment to the Message Age parameter in transmitted BPDUs?	X	8.10.1	No [ ]
(14b)	Does the Bridge underestimate Forward Delay?	X	8.10.1	No [ ]
(14c)	Does the Bridge overestimate the Hello Time interval?	X	8.10.1	No [ ]
(15a)	Does the Bridge use the specified value for Hold Time?	M	8.10.2, Table 8-3	Yes [ ]
	If item (16) is not supported, mark N/A and continue at (17a)			N/A [ ]
(16a)	Can the relative priority of the Bridge be set?	**16:**M	8.2, 8.5.3.7, 8.8.4	Yes [ ]
(16b)	Can the relative priority of the Ports be set?	**16:**M	8.2, 8.5.5.1, 8.8.5	Yes [ ]
(16c)	Can the path cost for each Port be set?	**16:**M	8.2, 8.5.5.3, 8.8.6	Yes [ ]
	If item (17) is not supported, mark N/A and continue at (18a).			N/A [ ]
(17a)	Can Bridge Max Age be set to any of the range of values specified?	**17:**M	8.10.2, 8.5.3.8, Table 8-3	Yes [ ]
(17b)	Can Bridge Hello Time be set to any of the range of values specified?	**17:**M	8.10.2, 8.5.3.9, Table 8-3	Yes [ ]
(17c)	Can Bridge Forward Delay be set to any of the range of values specified?	**17:**M	8.10.2, 8.5.3.10, Table 8-3	Yes [ ]
(18a)	Do all BPDUs contain an integral number of octets?	M	9.1.1	Yes [ ]
(18b)	Are all the following BPDU parameter types encoded as specified?	M	9.1.1, 9.2	Yes [ ]
	Protocol Identifiers		9.2.1	
	Protocol Version Identifiers		9.2.2	
	BPDU Types		9.2.3	
	Flags		9.2.4	
	Bridge Identifiers		9.2.5	
	Root Path Cost		9.2.6	
	Port Identifiers		9.2.7	
	Timer Values		9.2.8	
(18c)	Do Configuration BPDUs have the format and parameters specified?	M	9.3.1	Yes [ ]
(18d)	Do Topology Change Notification BPDUs have the format and parameters specified?	M	9.3.2	Yes [ ]
(18e)	Are received BPDUs validated as specified?	M	9.3.3	Yes [ ]

A.10 Bridge Management

Item	Feature	Status	References	Support
	If item (19) is not supported, mark N/A and continue at (20c).			N/A [ ]
(19a)	Discover Bridge	**19:**M	14.4.1.1	Yes [ ]
(19b)	Read Bridge	**19:**M	14.4.1.2	Yes [ ]
(19c)	Set Bridge Name	**19:**M	14.4.1.3	Yes [ ]
(19d)	Reset Bridge	**19:**M	14.4.1.4	Yes [ ]
(19e)	Read Port	**19:**M	14.4.2.1	Yes [ ]
(19f)	Set Port Name	**19:**M	14.4.2.2	Yes [ ]
(19g)	Read Forwarding Port Counters	**19:**M	14.6.1.1	Yes [ ]
(19h)	Read Filtering Database	**19:**M	14.7.1.1	Yes [ ]
(19i)	Set Filtering Database Ageing Time	**19:**M	14.7.1.2	Yes [ ]
(19j)	Read Permanent Database	**19:**M	14.7.5.1	Yes [ ]
(19k)	Create Filtering Entry	**19:**M	14.7.6.1	Yes [ ]
(19l)	Delete Filtering Entry	**19:**M	14.7.6.2	Yes [ ]
(19m)	Read Filtering Entry	**19:**M	14.7.6.3	Yes [ ]
(19n)	Read Filtering Entry Range	**19:**M	14.7.6.4	Yes [ ]
(19o)	Read Bridge Protocol Parameters	**19:**M	14.8.1.1	Yes [ ]
(19p)	Set Bridge Protocol Parameters	**19:**M	14.8.1.2	Yes [ ]
(19q)	Read Port Parameters	**19:**M	14.8.2.1	Yes [ ]
(19r)	Force Port State	**19:**M	14.8.2.2	Yes [ ]
(19s)	Set Port Parameters	**19:**M	14.8.2.3	Yes [ ]
(19t)	Read Port Default User Priority	MS4:M	14.6.2.1	Yes [ ] N/A [ ]
(19u)	Set Port Default User Priority	MS4:M	14.6.2.2	Yes [ ] N/A [ ]
(19v)	Read Port User Priority Regeneration Table	MS5:M	14.6.2.3	Yes [ ] N/A [ ]
(19w)	Set Port User Priority Regeneration Table	MS5:M	14.6.2.4	Yes [ ] N/A [ ]
(19x)	Read Port Traffic Class Table	MS7:M	14.6.3.1	Yes [ ] N/A [ ]
(19y)	Set Port Traffic Class Table	MS7:M	14.6.3.2	Yes [ ] N/A [ ]
(19z)	Read Outbound Access Priority Table	MS6:M	14.6.3.3	Yes [ ] N/A [ ]
(19aa)	Read GARP Timers	MS8:M	14.9.1.1	Yes [ ] N/A [ ]
(19ab)	Set GARP Timers	MS8:M	14.9.1.2	Yes [ ] N/A [ ]
(19ac)	Read GARP Applicant Controls	MS8:M	14.9.2.1	Yes [ ] N/A [ ]
(19ad)	Set GARP Applicant Controls	MS8:M	14.9.2.2	Yes [ ] N/A [ ]
(19ae)	Read GARP State	MS8:M	14.9.3.1	Yes [ ] N/A [ ]

A.10 Bridge Management *(Continued)*

Item	Feature	Status	References	Support
	If item (20a) is not supported, mark N/A and continue at (20e).			N/A [ ]
(20c)	What Management Protocol standard(s) or specification(s) are supported?	**20a:**M	5.	
(20d)	What standard(s) or specifications for Managed Objects and Encodings are supported?	**20a:**M	5.	
	If item (20b) is not supported, mark N/A and continue at A.11.			N/A [ ]
(20e)	What specification of the local management interface is supported?	**20b:**M	5.	

Predicates:
MS4=19 AND 4a
MS5=19 AND 4b
MS6=19 AND 4
MS7=19 AND 4c
MS8=19 AND 2b

A.11 Performance

Item	Feature	Status	References	Support
(21a)	Specify a Guaranteed Port Filtering Rate, and the associated measurement interval *TF*, for each Bridge Port in the format specified below.	M	16.1	
(21b)	Specify a Guaranteed Bridge Relaying Rate, and the associated measurement interval *TR*, in the format specified below. Supplementary information shall clearly identify the Ports.	M	16.2	

Guaranteed Bridge Relaying Rate	TR
_ _ _ _ _ _ _ _ _ _ frames per second	_ _ _ _ _ _ second(s)

A.11 Performance *(Continued)*

Port number(s) or other identification	Guaranteed port filtering rate (specify for all ports)	T_F (specify for all ports
	_ _ _ _ _ _ _ _ _ _ _ frames per second	_ _ _ _ _ _ second(s)
	_ _ _ _ _ _ _ _ _ _ _ frames per second	_ _ _ _ _ _ second(s)
	_ _ _ _ _ _ _ _ _ _ _ frames per second	_ _ _ _ _ _ second(s)
	_ _ _ _ _ _ _ _ _ _ _ frames per second	_ _ _ _ _ _ second(s)
	_ _ _ _ _ _ _ _ _ _ _ frames per second	_ _ _ _ _ _ second(s)
	_ _ _ _ _ _ _ _ _ _ _ frames per second	_ _ _ _ _ _ second(s)
	_ _ _ _ _ _ _ _ _ _ _ frames per second	_ _ _ _ _ _ second(s)
	_ _ _ _ _ _ _ _ _ _ _ frames per second	_ _ _ _ _ _ second(s)

A.12 GARP and GMRP

Item	Feature	Status	References	Support
	If Item 2b is not supported, mark N/A and continue at item (22i).			N/A [ ]
(22a)	Is the GMRP Application address used as the destination MAC Address in all GMRP protocol exchanges?	2b:M	10.4.1, Table 12-1	Yes [ ]
(22b)	Are GMRP protocol exchanges achieved by means of LLC Type 1 procedures, using the LLC address for Spanning Tree protocol?	2b:M	12.4, 12.5, Table 7-8	Yes [ ]
(22c)	Are GMRP protocol exchanges achieved using the GARP PDU formats, and the definition of the attribute type and value encodings defined for GMRP?	2b:M	10.3.1, 12.4, 12.5, 12.11	Yes [ ]
(22d)	Does the implementation support the operation of the Applicant, Registrar, and Leave All state machines?	2b:M	12.8, 13	Yes [ ]
(22e)	Does the Bridge propagate GMRP registration information only on Ports that are part of the active topology for the Base Spanning Tree Context?	2b:M	12.3.3, 12.3.4, 13	Yes [ ]
(22f)	Are GARP PDUs received on Ports that are in the Forwarding State forwarded, filtered or discarded in accordance with the requirements for handling GARP Application addresses?	2b:M	7.12.3, 12.5	Yes [ ]

A.12 GARP and GMRP *(Continued)*

Item	Feature	Status	References	Support
(22g)	Does the GMRP application operate as defined in Clause 10?	2b:M	10, 10.3	Yes [ ]
(22h)	Are received GARP PDUs that are not well formed for the GARP Applications supported, discarded?	2b:M	10.3.1, 12.4, 12.5, 12.10, 12.11	Yes [ ]
(22i)	Are all GARP PDUs that are (a) received on Ports that are in the Forwarding State, and are (b) destined for GARP applications that the Bridge does not support, forwarded on all other Ports that are in Forwarding?	M	7.12.3, 12.5	Yes [ ]
(22j)	Are any GARP PDUs that are (a) received on any Port, and (b) destined for GARP applications that the Bridge does not support, submitted to any GARP Participants?	X	7.12.3, 12.5	No [ ]
(22k)	Are any GARP PDUs that are (a) received on any Ports that are not in the Forwarding State, and are (b) destined for GARP applications that the Bridge does not support, forwarded on any other Ports of the Bridge?	X	7.12.3, 12.5	No [ ]
(22l)	Are any GARP PDUs that are (a) received on any Ports that are in the Forwarding State, and are (b) destined for GARP applications that the Bridge supports, forwarded on any other Ports of the Bridge?	X	7.12.3, 12.5	No [ ]
(22m)	Are all GARP PDUs that are: (a) received on any Port, and (b) destined for GARP applications that the Bridge supports, submitted to the appropriate GARP Participants?	M	7.12.3, 12.5	Yes [ ]

Annex B

(informative)

Calculating Spanning Tree parameters

This annex describes the method and rationale for calculating the recommended values and operational ranges for the essential Spanning Tree Algorithm performance parameters.

B.1 Overview

The calculation is described in a number of steps. Each of these steps establishes values for a number of the parameters that are then used as the basis for the following steps.

The description and equations given are pertinent to a homogeneous Bridged LAN, i.e., one in which all the individual LANs and Bridges are of the same type and speed. It is easy to extend this for a heterogeneous Bridged LAN.

The explanation is illustrated by recommended values for IEEE 802 LANs. All times are given in seconds.

B.2 Abbreviations and special symbols

dia	**maximum bridge diameter**
life	maximum **frame lifetime**
t_d	average frame **transit delay**
ma_d	average **medium access delay**
mma_d	**maximum medium access delay**
bt_d	**maximum bridge transit delay**
time_unit	the resolution of **Message Age**
msg_aio	**maximum Message Age increment overestimate**
msg_ao	**maximum Message Age overestimate**
pdu_d	**maximum BPDU transmission delay**
lost_msgs	maximum number of lost Bridge Protocol Messages to be tolerated prior to reconfiguration
msg_prop	**maximum Bridge Protocol Message propagation time**
hello_t	**Hello Time**
hold_t	**Hold Time**
max_age	**Max Age**
fwd_delay	**Forward Delay**

B.3 Calculation

B.3.1 Lifetime, diameter, and transit delay

B.3.1.1 Step

Choose the maximum bridge diameter for the Bridged LAN and the maximum bridge transit delay. Note that, where the individual LANs support a range of transmission priorities, the bridge transit delay may vary according to priority.

B.3.1.2 Basis of choice

The frame lifetime is equal to the maximum bridge diameter times the maximum bridge transit delay plus the maximum medium access delay for the initial transmission, i.e.,

$$life = (dia \times bt\_d) + mma\_d \quad (1)$$

The average **frame transit delay** between end systems in a Bridged LAN is greater than that experienced in a single LAN by the sum of the average **forwarding delays** and **frame transmission delays** of Bridges in the path between the end systems. These will be of the order of the **medium access delays** for lightly loaded LANs. So for systems at the extremities of the Bridged LAN there will be the following:

$$t\_d >= (dia \times ma\_d) + ma\_d \quad (2)$$

This bounds any enthusiasm for insisting on low **maximum bridge transit delays** and **high maximum bridge diameters**.

B.3.1.3 Recommended values for IEEE 802 Bridged LANs

$$mma\_d >= 0.5$$
$$life <= 7.5$$
$$dia = 7$$
$$bt\_d = 1.0$$

B.3.2 Transmission of BPDUs

B.3.2.1 Step

Select the transmission priority for BPDUs and a value for the **maximum BPDU transmission delay**.

B.3.2.2 Basis of choice

In general, a high transmission priority will be chosen, since the continued operation of the Bridged LAN depends on the successful transmission and reception of BPDUs. In some cases, other traffic native to an individual LAN may be more important.

The lowest value that could be chosen for the **maximum BPDU transmission delay** then is the **maximum medium access delay** for frames of that priority. In recognition of implementation difficulties that may arise in trying to achieve this figure, it seems more reasonable to choose the value to be equal to the **maximum bridge transit delay** for frames transmitted with that priority.

$$pdu\_d = bt\_d \quad (3)$$

B.3.2.3 Recommended values for IEEE 802 Bridged LANs

Priority transmission is not available for all IEEE 802 MAC methods. Therefore, the following has been selected:

$$pdu\_d = bt\_d$$
$$= 1.0$$

B.3.3 Accuracy of message age

B.3.3.1 Step

Select an appropriate value for the **maximum Message Age increment overestimate**.

This is the maximum overestimate of the increment made to the value of the Message Age parameter in transmitted Bridge Protocol Data Units. This parameter allows a Bridge receiving a Protocol Message to discard the information in it when it becomes too old. The transmitting Bridge should not be allowed to underestimate the value of this field.

Calculate the value of the **maximum Message Age overestimate**, which is the maximum overestimate any Bridge can make of the age of received Bridge Protocol Message information.

B.3.3.2 Basis of choice

The choice of **maximum Message Age increment overestimate** is governed by the following:

a) *time_unit*—the resolution with which the Message Age parameter is carried in Configuration Messages.
b) The granularity and accuracy of timers in the Bridge.
c) The **maximum BPDU transmission delay**.

Assuming the Bridge timers are not necessarily synchronized with received BPDUs, that they are accurate, and that they have a granularity of *time_unit*, there will be, as a best effort, the following:

$$msg\_aio = pdu\_d + time\_unit \tag{4}$$

NOTE—As time_unit is small, this term in Equation (4) has been approximated to zero in the recommended values of msg_aio and msg_ao shown in B.3.3.3.

This value should be rounded up to the nearest multiple of *time_unit*. It is worth noting here that any Bridge will always increment the value by at least one unit.

Making the same allowance for the timers in a Bridge receiving and storing Bridge Protocol Message information, the **maximum Message Age overestimate** will be equal to the **maximum Message Age increment overestimate** times the **maximum bridge diameter** minus one:

$$msg\_ao = msg\_aio \times (dia - 1) \tag{5}$$

B.3.3.3 Recommended values for IEEE 802 Bridged LANs

msg_aio = 1.0
msg_ao = 6.0

B.3.4 Hello time

B.3.4.1 Step

Provisionally select a value for the Hello Time.

B.3.4.2 Basis of choice

The choice of Hello Time is made with regard to its contribution to the maximum Bridge Protocol Message propagation time (see next step).

There is no point in transmitting Bridge Protocol Messages at intervals more frequent than the **maximum BPDU transmission delay**. In the worst case, where there is an attempt to guarantee correct operation, these messages would just run into one another.

A provisional value of twice the **maximum BPDU transmission delay** is suggested.

$$hello\_t = 2 \times pdu\_d \quad (6)$$

B.3.4.3 Recommended values for IEEE 802 Bridged LANs

$hello\_t = 2.0$

B.3.5 Bridge protocol message propagation time

B.3.5.1 Step

Calculate the **maximum Bridge Protocol Message propagation time**.

B.3.5.2 Basis of choice

The **maximum Bridge Protocol Message propagation time** is the maximum time taken for a Bridge Protocol Message information to cross the Bridged LAN, from Bridge to Bridge. This is composed of the following components:

a) The maximum propagation time for a single Bridge Protocol Message to cross the Bridged LAN, i.e., **maximum BPDU transmission delay** times the **maximum bridge diameter** minus one.
b) An allowance of **Hello Time** times the maximum number of consecutive lost Bridge Protocol Messages to be tolerated (note that losing even a single message should be a rare occurrence).
c) A further allowance of **Hello Time**, since we should not assume synchronization with the Root Bridge, and we may have to wait that long for it to transmit the next BPDU.

$$msg\_prop = ((lost\_msgs + 1) \times hello\_t) + pdu\_d \times (\mathrm{dia} - 1) \quad (7)$$

B.3.5.3 Recommended values for IEEE 802 Bridged LANs

Assuming $lost\_msgs = 3$,

$msg\_prop = 14.0$

B.3.6 Hold time

B.3.6.1 Step

Select a value for **Hold Time**.

B.3.6.2 Basis of choice

If **Hold Time** is greater than the **maximum BPDU transmission delay**, then the **Maximum Bridge Protocol Message propagation time** will be set, in the worst scenario, by a delay of **Hold Time** at each Bridge rather than by a delay of **maximum BPDU transmission delay**. This would invalidate the conclusion in B.3.5, above. Therefore, the following has been chosen:

$$hold\_t = pdu\_d \tag{8}$$

B.3.6.3 Recommended values for IEEE 802 Bridged LANs

$hold\_t = 1.0$

B.3.7 Max age

B.3.7.1 Step

Calculate the lower limit for **Max Age** for the Bridged LAN.

B.3.7.2 Basis of choice

Under stable conditions (i.e., no failure, removal or insertion of Bridges and other LAN components), Bridges on the periphery of Bridged LAN must not time out the Root. To do so would result in temporary local denial of service.

This means that **Max Age** must be adequate to cope with the worst-case propagation delays and Protocol Message Age inaccuracies as follows.

If at any time a Bridge is depending on Protocol Message information whose age has been maximally overestimated, then the sum of

a) The interval between the transmission of the next Protocol Message that it receives from the Root and the original transmission of the Protocol Message information it is currently using,
b) The overestimate of the Age of the current information, and
c) The propagation time of the next Protocol Message to be received

must be less than Max Age, or the Bridge will timeout the Protocol Message information and attempt to become the Root itself.

$$\begin{aligned} max\_age &= ((lost\_msgs + 1) \times hello\_t) + msg\_ao + (pdu\_d \times (\text{dia} - 1)) \\ &= msg\_ao + msg\_prop \end{aligned}$$

B.3.7.3 Recommended values for IEEE 802 Bridged LANs

$max\_age = 20.0$

B.3.8 Forward delay

B.3.8.1 Step

Calculate the **Forward Delay**.

B.3.8.2 Basis of choice

When the Forward Delay Timer for a Port expires and the Bridge starts forwarding received frames on that Port, it must be determined that there are no longer any frames in the system that were being forwarded on the previous active topology. If there are, then there is a risk of duplicating frames or, if remnants of the old active topology still exist while the new topology is being established, of creating data loops.

So the Listening and Learning periods during which the Forward Delay Timer runs must cover the following consecutive periods:

a) From the first Bridge Port entering the Listening State (and staying there through the subsequent reconfiguration) to the last Bridge in the Bridged LAN hearing of the change in **active topology**.
b) For the last Bridge to stop the forwarding of frames received on the previous topology and for the last frame so forwarded to disappear.

In a), above, there may be a difference of up to **maximum Message Age overestimate** in the times at which Bridges timeout old Root information and are prepared to become or listen to a new Root. Following this, it can take **maximum Bridge Protocol Message propagation time** for the news of the new topology to propagate from the new Root to all Bridges.

For b), above, the time to stop forwarding will be the **maximum transmission halt delay**, which is bounded by the maximum bridge transit delay (for all priorities); subsequently, the frame will disappear within the frame lifetime.

So there is the following:

$$2 \times fwd\_d >= msg\_ao + msg\_prop + bt\_d + life \quad (9)$$

B.3.8.3 Recommended values for IEEE 802 Bridged LANs

fwd_d =15.0

B.4 Selection of parameter ranges

B.4.1 Absolute maximum values

It might be desirable to configure a LAN or Bridge with a greater **maximum medium access delay** than assumed in the calculations for recommended values above. This could be a consequence of the type of traffic carried by the LAN or particular aspects of a Bridge implementation, designed to maximize the throughput, for example. However, it is highly desirable that absolute maximum values of **maximum bridge transit delay**, **maximum BPDU transmission delay**, and **maximum Message Age increment overestimate** be mandated by this standard in order to provide for interoperability.

A Bridge operating with absolute maximum values of these parameters should be configurable with Bridges employing recommended values in a Bridged LAN of a **bridge diameter** of at least 3. This criterion is met by the following:

$$bt\_d <= 2.0$$
$$pdu\_d <= 2.0$$
$$msg\_aio <= 2.0$$

These limits are believed to encompass the requirement for parameter values greater than those recommended in B.3.

B.4.2 Hold time

There is no benefit in reducing **Hold Time** below the recommended value of **maximum BPDU transmission delay**. Nor would any purpose be served, in terms of reduced use of bandwidth or processing capability in a Bridge, by increasing **Hold Time**. It is, therefore, appropriate to fix the value of this parameter as a constant:

hold_t =1.0

B.4.3 Range of hello time

There is no requirement for **HelloTime** to be less than **Hold Time**. Similarly, no purpose would be served by setting **Hello Time** to more than twice the absolute maximum value for **maximum BPDU transmission delay**. Therefore the following have been chosen:

1.0 <= *hello_t* <= 4.0

B.4.4 Maximum required values of max age and forward delay

The maximum required values for **Max Age** and **Forward Delay** are calculated using the equations of B.3 with the following parameter values:

dia= 7
mma_d <= 2.0
bt_d= 2.0
pdu_d= 2.0
msg_aio= 2.0
hello_t= 4.0
lost_msgs= 3

which gives the following:

max_age= 40.0
fwd_delay= 30.0

Although these are believed to be the maximum values required, there is no desire to prevent greater values being used.

B.4.5 Minimum values for max age and forward delay

The minimum realistic values for **Max Age** and **Forward Delay** are calculated using the equations of B.3 with the following parameter values:

dia= 2
mma_d<= 0.5
bt_d= 0.5
pdu_d= 0.5
hold_t= 1.0

msg_aio= 1.0
hello_t= 1.0
lost_msgs= 3

which gives the following:

max_age<= 6.0
fwd_delay= 4.0

It is suggested that Bridge implementations do not permit lower values of **Max Age** and **Forward Delay** to be used in order to guard against absurd settings.

B.4.6 Relationship between max age and forward delay

In order to further guard against bad settings of parameters that affect the correctness of operation of the Spanning Tree Algorithm and Protocol, it is suggested that Bridges enforce the relationship between **Max Age** and **Forward Delay** given in B.3.8 by ensuring that

$2 \times (fwd\_delay - 1.0) >= max\_age$

Annex C

(normative)

Source-Routing Transparent Bridge operation

C.1 Overview

A Source-Routing Transparent (SRT) Bridge is a MAC Bridge that performs source routing when frames are received with routing information (RII=1), and that performs Transparent Bridging when frames are received without routing information (RII=0). This annex specifies the protocols for the operation of source routing in an SRT Bridge. Source-routing frame formats and Bridge operations will facilitate end-station source routing of frames. Source-routing operation by end stations is specified in ISO/IEC 8802-2. Source routing provides the following:

a) Greater utilization of resources by providing multiple routes through the Bridged LAN.
b) Greater individual control of frames through the Bridged LAN.

C.1.1 Scope

For the purpose of compatible interconnection of data processing equipment using source routing on a Bridged LAN, this IEEE standard specifies the operation of MAC Bridges supporting source routing. To this end it

a) Defines the principles of operation of the MAC Bridge in providing source routing, in terms of the support and preservation of the MAC Service, and the maintenance of Quality of Service.
b) Specifies the enhanced MAC Internal Sublayer Service as it pertains to the SRT Bridge.
c) Augments the architectural model of the internal operation of a Bridge.
d) Establishes the requirements for Bridge Management of the source-routing function in the Bridged LAN, identifying the basic managed objects and management operations.
e) Specifies the requirements to be satisfied by equipment claiming conformance to this standard.

Items for future study are as follows:

— How the management operations are made available to a remote Manager using the protocol and architectural description provided by ISO/IEC 15802-2.
— Performance requirements, and recommended default values and applicable ranges for the operational parameters of a Bridge.

In addition, SRT operation is defined for only those MACs that have defined the routing information field. As of this publication they include the following:

— Token Ring (ISO/IEC 8802-5).
— FDDI MAC (ISO 9314-2).

MAC-specific matters are discussed in C.2.5.

C.1.2 Definitions

C.1.2.1 ARE Rd limit

This limit represents the maximum allowable number of route descriptors in an All Routes Explorer frame. This limit is implemented in Bridges to limit the number of Bridges traversed by All Routes Explorer frames between end stations during the route determination process.

C.1.2.2 Explorer frame

A frame to which the SRT Bridge adds routing information as it forwards the frame. The explorer frame is used to discover routes. There is an All Routes Explorer, which is intended to be forwarded by every Port; and a Spanning Tree Explorer frame, which is forwarded only by Ports designated to forward them by the spanning tree protocol.

C.1.2.3 LAN-in ID (LIN)

The identifier of the LAN from which an SRT Bridge receives a frame.

C.1.2.4 LAN-out ID (LOUT)

The identifier of the LAN to which an SRT Bridge transmits a frame.

C.1.2.5 parallel bridges

Two or more Bridges connecting the same LANs.

C.1.2.6 route

A path through a series of LANs and SRT Bridges.

C.1.2.7 route control

The first two-octet field in the routing information field of a source-routed frame. This field indicates the type and characteristics of the frame to be source routed.

C.1.2.8 route descriptor

A two-octet field in the routing information field that designates a LAN ID and Bridge number.

C.1.2.9 route discovery

The process of obtaining a route to a destination station.

C.1.2.10 routing information

A route control field and a sequence of route descriptors consisting of LAN IDs and Bridge numbers, indicating a route through the network between two stations.

C.1.2.11 source routing

A bridging mechanism to route frames through a multi-LAN network by specifying in the frame a route it will traverse.

C.1.2.12 spanning tree

A topology of Bridges such that there is one and only one data route between any two end stations.

C.1.2.13 STE Rd Limit

This limit represents the maximum allowable number of route descriptors in a Spanning Tree Explorer frame. This limit is implemented in Bridges to limit the number of Bridges traversed by Spanning Tree Explorer frames.

C.1.2.14 Transparent Bridging

A bridging mechanism, which is transparent to end stations, that interconnects LAN segments by Bridges designated to forward frames through participation in a Spanning Tree Algorithm.

C.1.2.15 Abbreviations

The following abbreviations are specific to this standard among the family of IEEE 802 standards:

ARE	All Routes Explorer: RII=1, RT=10x
ARERdLim	ARE Route Descriptor Limit
BN	Bridge Number
BPP	Bridge Port Pair
LIN	LAN-in ID
LOUT	LAN-out ID
LTH	Length Field
MSDU	MAC Service Data Unit
NSR	Non-Source Routed: RII=0, no RI Field
RC	Routing Control
RD	Route Descriptor
RI	Routing Information
RII	Routing-Information Indicator
RT	Routing Type
SRF	Specifically Routed Frame: RII=1, RT=0xx
SRT	Source Routing Transparent (Bridge)
STE	Spanning Tree Explorer: RII=1, RT=11x
STERdLim	STE Route Descriptor Limit

C.1.3 Conformance

A MAC Bridge that claims conformance to this annex

a) Shall conform to the main body of this standard, augmented by this annex, and represent a superset of the functions described in the main body of this standard for frames with routing information.
b) Shall conform to the static and dynamic requirements given in C.1.3.1 and C.1.3.2.
c) Furthermore, the supplier of an implementation that is claimed to conform to this standard shall complete a copy of the PICS proforma provided in Annex D, and shall provide the information necessary to identify both the supplier and the implementation.

C.1.3.1 Static conformance requirements

An implementation claiming conformance to this standard

a) Shall conform with the MAC Sublayer and Physical Layer of the LANs it interconnects.
b) Shall conform to the static characteristics of the SRT Bridge as described in C.3.

C.1.3.2 Dynamic conformance requirements

An implementation claiming conformance to this standard

a) Shall exhibit external behavior corresponding to the Bridging operations for the handling of source-routed frames as described in C.3.
b) Shall report errors as described in C.3.

C.2 Support of the MAC Service

C.2.1 Support of the MAC Service

The MAC Service provided to end stations attached to a Bridged LAN is supported by the Bridges in that Bridged LAN as described in 6.2.

C.2.2 Preservation of the MAC Service

SRT Bridges preserve the MAC Service as described in 6.2.

C.2.3 Quality of Service maintenance

The quality of the MAC Service supported by an SRT Bridge is not significantly inferior to that provided by a single LAN. A full set of Quality of Service parameters is included in 6.3. Parameters that are affected by the source-routing capability are listed below:

a) Frame misordering
b) Frame duplication
c) Undetected frame error rate
d) Maximum Service Data Unit Size

Frames without routing information are referred to as Non-Source-Routed Frames (NSR). Source routing introduces three new types of frames. Specifically, Source-Routed Frames (SRFs) are the primary mechanism for transporting data through the source-routed network. Spanning Tree Explorer frames (STE) are sent on the portion of the spanning tree interconnected by SRT Bridges; they collect the spanning tree routing information as they travel. STE frames can be used for management and route determination but are not required for data transmission. All Routes Explorer Frames (ARE) are used explicitly during route discovery.

C.2.3.1 Frame misordering

Source routing allows multiple routes to exist and be used simultaneously between a given pair of stations. If frames are sent alternately on different routes, they might arrive in a different order. However, misordering does not occur along any particular route within each type. Bridges receiving frames of the same type (e.g., SRFs) will transmit them in a First In First Out (FIFO) manner. Regarding the specific types of frames,

a) NSR frames cannot be misordered (see the main body of this standard) because they follow the unique path of the spanning tree.
b) STE frames cannot be misordered (see the main body of this standard) because they follow the unique path of the spanning tree.
c) ARE frames are used as control frames. Misordering does not apply for these frames since they are sent on multiple routes by definition.
d) SRFs will not be misordered provided the sender uses one and only one source route to deliver the sequence of frames; frames that use different routes between the same pair of Ports are susceptible to frame misordering.

C.2.3.2 Frame duplication

a) NSR and STE frames cannot be duplicated (see the main body of this standard).
b) ARE frames are used as control frames; the receiving side must accept multiple ARE frames (one for each route).
c) SRFs cannot be duplicated because frames are forwarded only by the Bridge sequence specified in the routing information field. Furthermore, Bridges prevent frames with duplicate LAN numbers in the routing sequence from being forwarded.

C.2.3.3 Undetected frame error rate

The service provided by the MAC Sublayer introduces a very low undetected frame error rate in transmitted frames, see 6.3.7. In particular the following should be noted:

a) The rate of undetected errors for NSR and SRFs is unchanged.
b) ARE and STE frames collect routing information as they travel to their destination; therefore, the FCS is recalculated at each intermediate Bridge. These frames can have a higher undetected frame error rate, but they are not intended to be used for data transport.

C.2.3.4 Maximum service data unit size supported

For a complete description of the Maximum Service Data Unit, see 6.3.8. The SRT Bridge meets all requirements associated with Maximum Service Data Unit Size. The Maximum Service Data Unit Size supported by a Bridge between two LANs is the smaller of that supported by the LANs. No attempt is made by a Bridge to relay a frame to a LAN that does not support the size of Service Data Unit conveyed by that frame.

Source routing reduces Bridge receipt of frames of unacceptable sizes since the sender of the data can determine the maximum size of the data unit on the chosen route during route discovery.

C.2.4 Internal sublayer service

The MAC entities provide an additional enhanced service with a new set of primitives in addition to the Internal Sublayer Service specified in 6.4.

The Internal Sublayer Service excludes procedures whose operation is confined to that of the individual LANs. The unitdata primitives that describe this service follow. The parameters and their meaning are identical to those in 6.4 except for the addition of the following parameter:

routing_information

C.2.4.1 Interactions

The following primitives are defined for the MAC relay to request the MAC to transmit a PDU.

M_UNITDATA.request
M_UNITDATA.indication

C.2.4.2 Detailed service specification

All primitives are specified as examples only. Each service names the particular primitive and the required information that is passed between MAC and the MAC relay function. Only those primitives not included in 6.4 are defined here.

C.2.4.2.1 M_UNITDATA.indication

Each MAC data indication primitive invoked corresponds to the receipt of a frame from an individual LAN.

Semantics of the service primitive

```
M_UNITDATA.indication    (
                         frame_type,
                         mac_action,
                         destination_address,
                         source_address,
                         routing_information,
                         mac_service_data_unit,
                         user_ priority,
                         frame_check_sequence
                         )
```

routing_information. The routing_information parameter specifies the routing information field of the received frame.

When generated. This primitive is generated by the MAC entity to the MAC relay function to indicate the arrival of a frame to be forwarded. Such frames are reported only if they are validly formed (as defined by the respective MACs).

Effect of receipt. The receipt of this primitive causes the MAC relay function to process the frame according to the forwarding logic specified in 3.7.

C.2.4.2.2 M_UNITDATA.request

A data request primitive is invoked to transmit a frame to an individual LAN.

Semantics of the service primitive

```
M_UNITDATA.request       (
                         frame_type,
                         mac_ action,
                         destination_address,
                         source_address,
                         routing_information,
                         mac_service_data_unit,
```

user_priority,
access priority,
frame check sequence

routing_information. The routing_information parameter specifies the routing information field of the frame to be forwarded as modified by the Bridge function.

NOTE—If the Frame Check Sequence (FCS) is supplied, then it may be forwarded without recalculation; otherwise, it should be calculated.

When generated. This primitive is generated by the MAC relay function to the MAC entity when the MAC relay passes a frame to the MAC entity for forwarding.

Effect of receipt. The receipt of this primitive causes the MAC entity to append all the appropriate fields, and pass the properly formed frame to the lower layers of the protocol for transfer to the peer MAC entity or entities.

C.2.5 Support of the Internal Sublayer Service

This subclause specifies the mapping of the Internal Sublayer Service to MAC Protocol and Procedures, and the encoding of the parameters of the service in MAC frames.

C.2.5.1 Support of token ring

The Token Ring Access Method is specified in ISO/IEC 8802-5. Clause 3 of that standard specifies Formats and Facilities, and Clause 4 specifies Token Ring protocols.

On receipt of an M_UNITDATA.request primitive, the local MAC Entity composes a frame using the parameters supplied as specified below, appending the frame control, destination address, source address, routing information, and frame check sequence fields to the user data. The Routing Information Indicator bit is set to one to indicate the presence of the routing information field. The frame is enqueued for transmission. On transmission the starting delimiter, access control field, ending delimiter and frame status fields are added.

On receipt of a valid frame, with frame_type and mac_action parameter values of user_data_frame and request_with_no_response respectively, with the Routing Information Indicator bit set to one, an M_UNITDATA.indication primitive is generated, with parameters derived from the frame fields as specified below.

NOTE—On receipt of a valid MAC frame that was not transmitted by the Bridge Port's local MAC entity with the Routing Information Indicator bit set to zero, an M_UNITDATA.indication primitive is generated as required by 6.5.3, and the frame is handled by the Transparent Bridging function.

The frame_type, mac_action parameter, destination_address, source_address, mac_service_data_unit, user_priority, and frame_check_sequence parameter are encoded as specified in 6.5.3.

The Routing Information Indicator bit is encoded in the source address field. It occupies the same position in the source address field as does the Group Address indicator bit in the destination address field.

If an M_UNITDATA.request primitive is not accompanied by the frame_check_sequence parameter, it is calculated in accordance with 3.2.7 of ISO/IEC 8802-5.

For the setting of the A and C bits, refer to 6.5.3.

In order to support the MAC Internal Sublayer Service, a Token Ring Bridge must be capable of recognizing and removing frames transmitted by itself, even though they can carry a source address different than that of the Bridge Port that transmitted them.

C.2.5.2 Support of FDDI

On receipt of a valid frame (ISO 9314-2) that was not transmitted by the Bridge Port's local MAC entity, an M_UNITDATA.indication primitive is generated. The parameters associated with the primitive are derived from the frame's fields as specified in 6.5.4.

On receipt of an M_UNITDATA.request primitive, the local MAC entity composes a frame using the parameters supplied as specified in 6.5.4.

C.3 Principles of operation

This clause uses the principles and model of operation of an SRT Bridge. It provides the following:

a) An explanation of the principal elements of source-routing Bridge operation and a list of the supporting functions
b) An architectural model that governs the provision of these functions
c) A model of the operation of the SRT Bridge in terms of the operation of processes and entities that support the functions
d) Details concerning the additional addressing requirements

C.3.1 Source-routing bridge operation

The principal elements of source-routing operation are as follows:

a) Relay of source-routed data frames
b) Relay and processing of frames to support the transfer and acquisition of routing information
c) Management of the above

C.3.1.1 Relay of data frames

A MAC Bridge relays individual MAC user data frames between the separate MACs of the Bridged LANs connected to its Ports. The order of frames of given user_priority and frame_type received on one Port and transmitted on another is preserved.

The functions that support the relay of source-routed data frames and maintain the Quality of Service supported by the Bridge are as follows:

a) Frame reception
b) Discard of received frame in error
c) Selection of source-routed data frames
d) Frame discard if transmittable service data unit size exceeded (C.2.3.4)
e) Modification of routing information
f) Frame discard when maximum Bridge transit delay is exceeded
g) Selection of outbound access priority
h) Frame transmission

C.3.1.2 Dissemination of routing information

The functions that support the relay and processing of frames to support the transfer and acquisition of routing information are as follows:

a) Handling of All Routes Explorer (ARE) frames
b) Handling of Spanning Tree Explorer (STE) frames

C.3.1.3 Bridge Management

The functions that support Bridge Management control and monitor the provision of the functions in the subclauses above are specified in C.4.

C.3.2 Bridge architecture

Each Bridge Port receives and transmits frames to and from the LAN to which it is attached using the services provided by the individual MAC entity associated with that port as described in 7.2. The MAC relay entity handles the MAC-independent functions of relaying frames between Bridge Ports. If the received frame is not source routed (RII = 0), then the Bridge frame is forwarded or discarded using the logic described in Clause 7 (Transparent Bridging logic). If the received frame is source routed (RII = 1), then the frame is handled using the logic as described in C.3.3 (see Figure C-1).

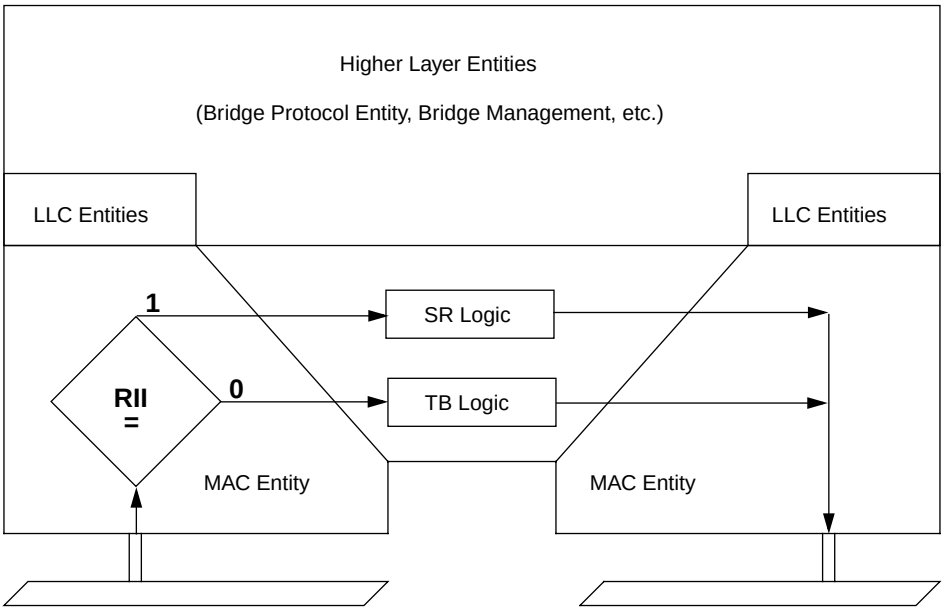


Figure C-1—SRT Bridge operation logic

In SRT bridging the port serves a specific LAN, and that LAN is assigned a unique LAN_ID. Because SRT allows multiple Bridge paths between two LANs, a Bridge Number (BN) is assigned to each Bridge path such that any two Bridge paths between the same two LANs must have a different Bridge number. From this assignment, each path between LANs can be uniquely identified by the LAN_IDs and Bridge number. A path through the Bridge is designated in the RI field by the designation of LIN, BN, LOUT where LIN is the LAN_ID of the receiving Port and LOUT is the LAN_ID of the forwarding Port.

A bridging relationship (Bridge path) therefore exists between each unique pair of Ports in a Bridge. The term Bridge Port Pair (BPP) will be used to represent this pairing of Ports forming a Bridge path, and each

BPP will be designated by both Port_Numbers and their respective BN. The Bridge path can now be specified by the set of LAN_IDs and BN assigned to that pair of Ports.

Each Port has the following attributes:

a) Port_Number
b) LAN_ID
c) Largest MSDU
d) SRT port type
e) RD Limits

With this Bridge architecture approach, it is sufficient to describe the Bridge frame handling by describing it on a Bridge Port Pair basis as it is done in C.3.7.

C.3.3 Bridge operation

The use, by the MAC relay entity, of the Internal Sublayer Service provided by the individual MAC entities associated with each Bridge Port is specified in C.3.5 and C.3.6. Bridge operations on frames without routing information (RII=0) are described in 7.3. The Bridge Protocol Entity remains unchanged. The Bridge Management Entity is enhanced to include SR managed objects as described in C.4. The MAC Relay Entity is enhanced to handle source-routed frames as well as non-source-routed frames. The source-routing forwarding process forwards received source-routed frames to other Bridge Ports on the basis of information contained in the frame and on the state of the Bridge Ports (C.3.7). Source addresses of source-routed frames (with RII=1) are not processed by the Transparent Bridge function. This is because it is not possible to associate a spanning tree Port with an address on a source-routed frame since source-routing paths do not always coincide with spanning tree paths.

C.3.3.1 Source-routing function overview

For a source-routed frame, a forwarding decision is based on information contained in a routing information field in the body of the frame, if such a field is present. If it is not present, the forwarding decision is based on the destination address field of the frame (by the Transparent Bridging logic).

After a route has been determined, the station that originates a frame designates the route that the frame will travel by embedding a description of the route in the routing information (RI) field of the transmitted frame. Bridges copy and forward these frames from one LAN segment to another without altering the frame's content provided that the MAC types are the same. If the MAC types differ, then the frame's content will be altered and the FCS will be regenerated. If data is sent in a frame in which the routing information is modified (e.g., an All Routes Explorer frame), the FCS will be recalculated and the user data integrity may be compromised. Source routing provides the capability for frames containing user data to traverse like Bridged LANs without regeneration of the FCS.

Stations initially obtain the route to a given destination station by a process called *route discovery.* This process allows the station to dynamically discover a route to the destination station, as needed. Each Bridge that forwards a route discovery (explorer) frame indicates in the RI field the Bridge's configuration (LIN, BN, LOUT) and the largest frame size the Bridge can support (through the particular LOUT), if it is smaller than the size already specified, and regenerates the FCS. In this way the routing information is built by the Bridges as the explorer frame is forwarded from LAN segment to LAN segment. When the frame reaches the destination station, the RI field indicates the route the frame traveled from source to destination and the maximum frame size supported along the entire route.

Figure C-2 depicts the elements of the source-routing function.

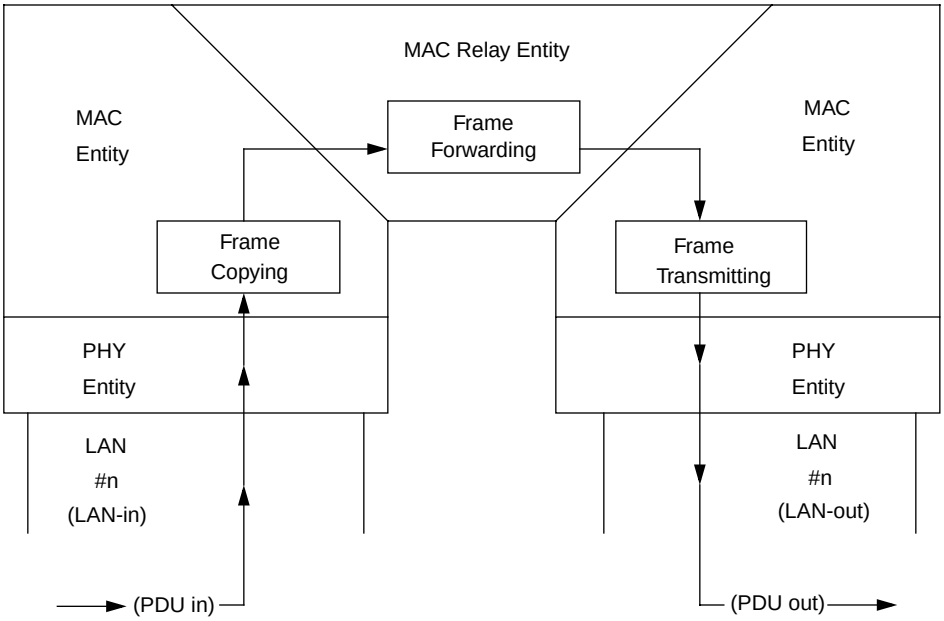
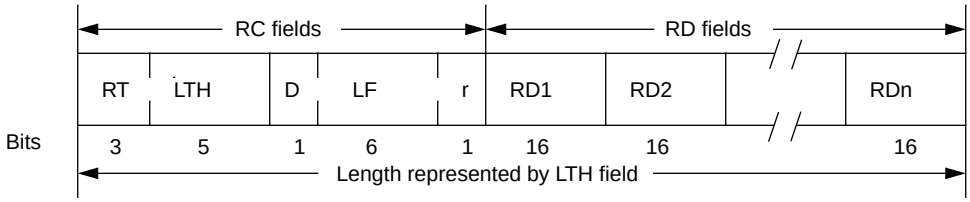


Figure C-2—Elements of a Source-Routed Bridge

C.3.3.2 Source-routing information field

The structure of the routing information field is defined in Figure C-3.



RC = Routing Control (16 bits)
RDn = Route Descriptor n (16 bits)
RT = Routing Type (3 bits)
r = reserved (1 bit)

LTH = Length (5 bits)
D = Direction bit (1 bit)
LF = Largest Frame (6 bits)

Figure C-3—Structure of the routing information field

The RI field is transmitted on the medium according to the usual practice for each MAC's treatment of data. For temporal understanding of the RI field, Figure C-4 is equivalent to the definition in Figure C-3.

Routing Type (RT) field. This field indicates whether the frame is to be forwarded through the network either along a single route, or multiple routes through multiple interconnected LANs.

Specifically Routed frame (RT=0XX). If the most significant RT bit is set to 0, the RD fields contain a specific route through the network. The XX bits are preserved in transmission throughout the Bridged LAN.

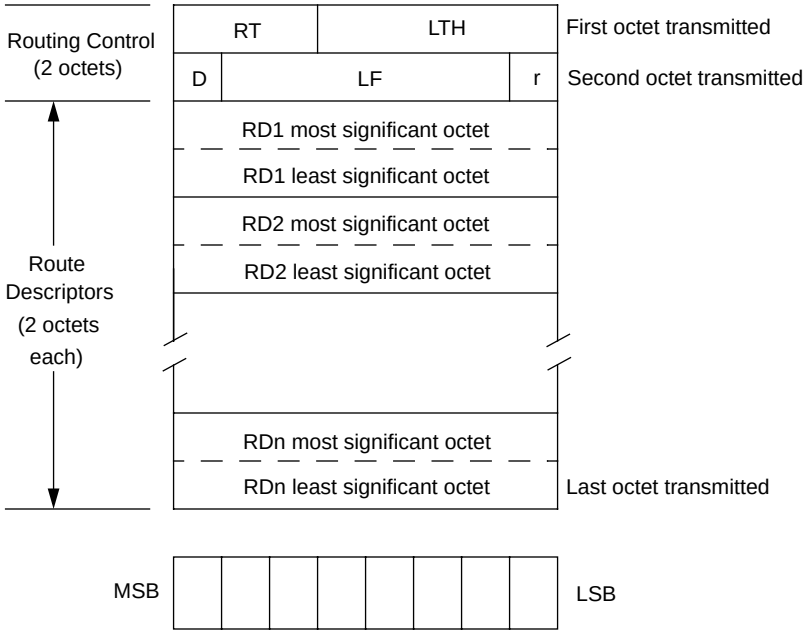


Figure C-4—Routing information field

All Routes explorer frame (RT=10X). If the RT bits are set to 10X, indicating an All Routes Explorer frame, the frame will be routed along every route in the network allowed by the ARE forwarding decision. This type will result in as many frames arriving at the destination station, as there are different routes from the source to that station. It is originated by an end station with no route descriptors. Route descriptors are added to the frame by SRT Bridges as they forward the frame. The X bit is preserved in transmission throughout the Bridged LAN.

Spanning Tree Explorer frame (RT=11X). If the RT bits are set to 11X, indicating a Spanning Tree Explorer frame, only SRT Bridges with Ports in the Transparent Bridging forwarding state relay the frame from one LAN to another with the result that the frame will be forwarded along the Bridged LAN spanning tree and appear no more than once on every LAN in the network. It is originated by an end station with no route descriptors. Route descriptors are added to the frame by SRT Bridges as they forward the frame. The X bit is preserved in transmission throughout the Bridged LAN.

Length Bits (LTH). These five bits indicate the length (in octets) of the RI field. Length field values will be even values between 2 and 30 inclusive.

Direction Bit (D). If the D bit is 0, the frame traverses the LANs in the order in which they are specified in the routing information field (RD1 to RD2 to... to RDn). Conversely, if the D bit is set to 1, the frame will traverse the LANs in the reverse order (RDn to RDn-1 to... to RD1). The D bit is meaningful only for SRFs. For STE and ARE frames, the D bit shall be 0.

Largest Frame Bits (LF). The LF bits indicate the largest size of the MAC Service Data Unit (MAC information field) that may be transmitted between two communicating stations on a specific route. The LF bits are meaningful only for STE and ARE frames. For SRFs the LF bits are ignored and shall not be altered by the Bridge. A station originating an explorer frame sets the LF bits to the maximum frame size it can handle. Forwarding Bridges shall set the LF bits to indicate the largest value that does not exceed the minimum of

a) The indicated value of the received LF bits

b) The largest MSDU size supported by the Bridge
c) The largest MSDU size supported by the Port from which the frame was received
d) The largest MSDU size supported by the Port on which the frame is to be transmitted

The destination station may further reduce the LF value to indicate its maximum frame capacity. Base LF bit encoding values and rationale are given in Figure C-6.

LF bit encodings are composed of a 3-bit base encoding and a 3-bit extended encoding (see Figure C-5). The SRT Bridge shall contain an LF mode indicator to allow for the selection of base or extended LF bits. When the LF mode indicator is set to *base mode*, the Bridge shall set the LF bits in explorer frames in accordance with the largest frame base values (see Table C-1). When the LF mode indicator is set to *extended mode*, the Bridge shall set the LF bits in explorer frames in accordance with the largest frame extended values (Table C-2).

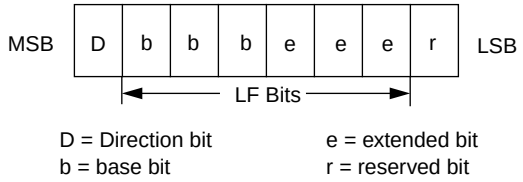


Figure C-5—Base and extended LF bits on the second octet of the RC field

000: 516 octets (ISO 8473, Connectionless Network Protocol, plus LLC)
001: 1 470 octets (ISO/IEC 8802-3, CSMA/CD LAN)
010: 2 052 octets (80 by 24 character screen with control)
011: 4 399 octets (ISO/IEC 8802-5, FDDI, 4 Mb/s token ring, ISO 9314-2)
100: 8 130 octets (ISO/IEC 8802-4 token bus LAN)
101: 11 407 octets (ISO/IEC 8802-5 4-bit burst errors unprotected)
110: 17 749 octets (ISO/IEC 8802-5 16 Mb/s token ring LAN)
111: 41 600 octets (Base for extending to 65 535 octets)

Figure C-6—Largest frame base values and rationale

Table C-1—Largest frame base values

Base	Value	Base	Value
000	516 octets	100	8 130 octets
001	1 470 octets	101	11 407 octets
010	2 052 octets	110	17 749 octets
011	4 399 octets	111	>17 749 octets

LF Value = the greatest integer in (blfv + elf × (nlfv – blfv)/8)

where

blfv = Base LF value represented by the low-order 3 bits
nlfv = Next base LF value (the next value after 111 is 65 535)
elf = Extended LF bit encoding (0 through 7)

Table C-2—Largest frame extended values

	EXTENSION								
BASE		000	001	010	011	100	101	110	111
	000	516	635	754	873	993	1 112	1 231	1 350
	001	1 470	1 542	1 615	1 688	1 761	1 833	1 906	1 979
	010	2 052	2 345	2 638	2 932	3 225	3 518	3 812	4 105
	011	4 399	4 865	5 331	5 798	6 264	6 730	7 197	7 663
	100	8 130	8 539	8 949	9 358	9 768	10 178	10 587	10 997
	101	11 407	12 199	12 992	13 785	14 578	15 370	16 163	16 956
	110	17 749	20 730	23 711	26 693	29 674	32 655	35 637	38 618
	111	41 600	44 591	47 583	50 575	53 567	56 559	59 551	>59 551

NOTE—Bridges are not required to support every value in the above list. Also, source-routing end systems on MACs with a particular maximum frame size should not send frames in which the MAC header and trailer, RI fields, and information exceed the maximum frame size for that MAC.

In general, base values in octets were chosen to allow transmission of optimally sized frames through various LAN MAC methods. These sizes were calculated by subtracting the maximum size MAC header and the maximum size Routing Information Field (30 octets) from the maximum physical frame size (see Figure C-7). Values associated with the bit encodings represent the maximum length of the MAC information field (see Figure C-7). Extended bits are used to provide seven intermediate values between each set of base values. The equation for calculating these values and the actual calculated values are included in Table C-2.

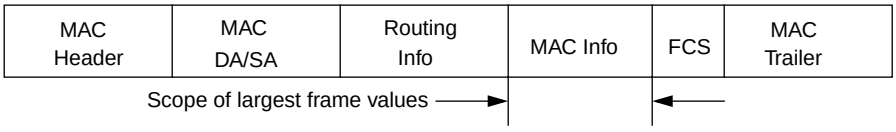


Figure C-7—Scope of the largest frame values

Route Descriptor (RD) field. This field is depicted in Figure C-8.

Figure C-8—Route descriptor field

The sequence of RD fields defines a specific route through the network. Each RD contains a network-unique 12-bit LAN ID plus a 4-bit Bridge number that is used to differentiate between two or more Bridges when they connect the same two LANs (parallel Bridges). The last Bridge number in the routing information field is reserved and set to zero.

Reserved (r) Bits. All reserved (r) bits are ignored upon receipt and transmitted as received.

C.3.3.3 Source-routing frame types

Frames are processed by Bridges based on the routing type (RT) of the frame. The RT of the frame to be processed is indicated in the RT subfield of the RI field of the frame. The following subclauses define each of the routing types.

C.3.3.3.1 Specifically routed frame type (RT=0XX)

The RD fields contain a specific route through the network. This type is used for the normal routing of data frames from source to destination.

C.3.3.3.2 All Routes Explorer frame type (RT=10X)

This frame is routed along all nonrepeating routes in the network. This type, known as an All Routes Explorer frame, will result in as many frames arriving at the destination station as there are allowable routes to that station. The Bridge adds routing information to the frame as it forwards it.

C.3.3.3.3 Spanning Tree Explorer frame type (RT=11X)

Spanning Tree Explorer frames are forwarded (by the source-routing logic) through only those Bridge Ports in the forwarding state, (i.e., along the spanning tree). The result is that the frame will appear only once at the destination station. Routing descriptors are added to this frame by the Bridges, and filtering on the destination address will not be performed. These frames traverse the adjacent portion of the spanning tree that is connected by SRT Bridges, acquiring the route as they are forwarded.

C.3.3.4 Bridge processing of source-routed frames

Processing by the Bridge is based on a two-stage procedure: frame reception and frame forwarding. Explicit function definitions for these are included in C.3.5 and C.3.7, respectively. The process of frame reception refers to the indication of a frame to the MAC Relay Entity from the MAC entity. The process of frame forwarding refers to the internal checking of frames that have been copied, the possible addition of routing information, and the actions that are performed to forward frames to another LAN segment.

C.3.4 Port state information

Port states are described in 8.4.

C.3.5 Frame reception

If a frame has no routing information (RII=0), then the Bridge shall process the frame as described in Clause 7.

If a frame has routing information (RII=1), then the frame shall be processed as indicated in C.3.7. The user_priority of such frames shall be regenerated as specified in 7.5.1.

C.3.6 Frame transmission

Frames are transmitted as described in 7.6.

C.3.7 Frame forwarding

The decision to forward a frame is based on the content of the RI field and the internal state of the Bridge. Frames of a given user_priority and frame_type shall be forwarded in the order that they were copied from the LAN segment. The forwarding process forwards received frames that are to be relayed to other Bridge Ports. Non-source-routed frames (RII=0) shall be forwarded as specified in 7.7. The processing of source-routed frames (RII=1) is included in the subclauses that follow. The following subclauses describe the actions of the SRT Bridge for a single enabled Bridge Port pair (LIN and LOUT). Multiport Bridges include multiple Bridge Port pairs and shall behave in the manner described in this clause for each Port pair. No frames shall be forwarded through disabled BPPs.

C.3.7.1 Specifically routed data frames (RT=0XX)

Bridges that find an RI field-to-bridge configuration (LIN, BN, LOUT combination) match shall forward such frames without alteration of the RI provided that both MAC entities use the same frame formats (e.g., the Bridge is from Token Ring to Token Ring, ISO/IEC 8802-5). If multiple occurrences of LOUT are in the routing information field of a frame, the Bridge shall discard the frame and, if implemented, increment the DupLout counter. Note that SRFs are forwarded or discarded based on the state tables, regardless of whether the destination is to an individual or group address. Table C-3 is explained as follows:

(SRF1). If a specifically routed frame is received by an SRT Bridge and a LIN-BN-LOUT match is not found in the routing information field, then the frame shall not be forwarded on the Port indicated by LOUT.

(SRF2). If a specifically routed frame is received by an SRT Bridge and the LIN-BN-LOUT matches, and there are no multiple occurrences of LOUT in the routing information field, and the Length field is even, the frame shall be forwarded and, if implemented, the SRFsForwarded counter shall be incremented.

(SRF3). If a specifically routed frame is received by an SRT Bridge and the LIN-BN-LOUT matches but the Length field is zero or an odd number, the frame shall be discarded and, if implemented, the Invalid RI counter shall be incremented.

(SRF4). If a specifically routed frame is received by an SRT Bridge and the LIN-BN-LOUT matches but there are multiple occurrences of LOUT in the routing information field, the frame shall be discarded and, if implemented, the DupLout counter shall be incremented.

Table C-3—Specifically routed frame forwarding state table

Ref.	Condition	Action
SRF1	LIN-BN-LOUT does not match	Do not forward frame
SRF2	(LIN-BN-LOUT match) & (occurrences of LOUT = 1) & (RI_LTH = even number)	Forward Frame on LOUT Increment (SRFsForwarded)
SRF3	(LIN-BN-LOUT match) & ((RI_LTH = odd number) | (RI_LTH = 0))	Discard Frame Increment (InvalidRi)
SRF4	(LIN-BN-LOUT match) & (occurrences of LOUT > 1)	Discard Frame Increment (DupLout)

In all cases above: RII = 1, RT = 0XX, and BPP State = enabled

Definitions: Discard = Do not forward on any LAN-out
Forward = Transmit the frame on LAN-out

C.3.7.2 All Routes Explorer frames (RT=10X)

In order to limit unnecessary traffic, SRT Bridges are allowed to filter ARE frames that are superfluous or redundant. Multiport SRT Bridges may discard (filter) ARE frames that

— Have been through the Bridge before. This can be determined by looking for a LIN-BN-LOUT combination in the RI field of the received frame.
— Have been through any LAN attached to the Bridge before. This condition is more restrictive than the condition described in a). This can be determined by examining the RI field of the received frame for a LAN ID match with any of the LANs attached to the Bridge other than the LIN.

All SRT Bridges shall forward all other frames on each LOUT except when any of the following conditions hold:

a) LOUT is already present in the RI field.
b) The last LAN ID in the RI field does not match LIN.
c) The number of route descriptors meets or exceeds the ARERdLimit.
d) The RI field length is an odd number.
e) The RI field length is zero.
f) The RI field length is four (optionally—see Table C-4).
g) The Direction bit is not 0.

If any of these conditions hold, the frame shall be discarded. If condition 2 occurs, the LanIdMismatch counter is incremented. If conditions 4, 5, or 7 occur, the Invalid RI counter is incremented.

If none of these conditions holds then,

— If the LTH = 2, the Bridge adds an RD with LIN to the RI field and,
— The Bridge adds its Bridge number and the number of LOUT to the frame.

It also increments the length field by two for every route descriptor it adds. Before forwarding the frame, the Bridge adjusts the Largest-Frame (LF) field to reflect the transfer capacity of the Bridge if the transfer capacity of the Bridge Port pair is less than the size indicated by the received LF. The transfer capacity of the Bridge is the least of

— The maximum frame size of LIN,
— The maximum frame size of LOUT, and
— The maximum frame size that can be handled by the Bridge itself.

Finally, the FCS is recomputed. ARE frames are processed as indicated in Table C-4.

Table C-4 is explained as follows:

(ARE1). If an All Routes Explorer frame is received by an SRT Bridge with the LAN-out ID already in the RI field or if the Bridge has filtered the frame, the frame shall not be forwarded on the LAN-out indicated by the LAN-out ID.

(ARE2). If an All Routes Explorer frame is received by an SRT Bridge with the last LAN ID not matching LIN, the frame shall be discarded and, if implemented, the LanIdMismatch counter shall be incremented.

(ARE3). If an All Routes Explorer frame is received by an SRT Bridge with the number of RDs meeting or exceeding AreRdLimit of the Port indicated by LOUT, the frame shall not be forwarded and, if implemented, the AreRdLmtExceeded counter of the Port indicated by LOUT shall be incremented.

Table C-4—All Routes Explorer frame forwarding state table

Ref.	Condition	Action
ARE1	(LOUT already in RI) | (bridge has filtered)	Do not forward on LOUT
ARE2	Last LAN ID in RI ¬ = LIN	Discard Frame Increment(LanIdMismatch)
ARE3	# RDs >= AreRdLimit	Do not forward Increment(ARERdLmtExceeded)
ARE4	(RI_LTH = odd number) | (RI_LTH = 0) | (D ¬ = 0)	Discard Frame Increment(InvalidRi)
ARE5	(RI_LTH = 2) & (bridge has not filtered) & (AreRdLimit > 1)	Add LIN,BN,LOUT to RI field SET RI_LTH = 6 SET LF=MIN(LF,LIN,BR,LOUT) Recalculate FCS Forward frame Increment(AREFramesForwarded)
ARE6	(RI_LTH = 4) & (bridge has not filtered) & (Last LAN ID in RI = LIN) & (AreRdLimit > 1)	Discard frame Increment(InvalidRi) OR (Optionally) Add BN,LOUT to RI field RI_LTH = RI_LTH + 2 SET LF = MIN(LF,LIN,BR,LOUT) Recalculate FCS Forward frame Increment(AREFramesFowarded)
ARE7	(RI_LTH = even number between 6 and 28 inclusive) & (LOUT not already in RI) & (bridge has not filtered) & (Last LAN ID in RI = LIN) & (# RDs < AreRdLimit)	Add BN,LOUT to RI field RI_LTH = RI_LTH + 2 SET LF = MIN(LF,LIN,BR,LOUT Recalculate FCS Forward frame Increment(AREFramesFowarded)

In all cases above: RII = 1, RT = 10X, and BPP State = enabled

Definitions: Discard = Do not forward on any LAN-out
Forward = Transmit the frame on LAN-out

(ARE4). If an All Routes Explorer frame is received by an SRT Bridge with a Length field of zero or an odd number, or the Direction bit is not 0, the frame shall be discarded and, if implemented, the InvalidRi counter shall be incremented.

(ARE5). If an SRT Bridge receives a valid All Routes Explorer with a Length field equal to 2, and the AreRdLimit is greater than 1, then the Bridge shall modify the frame being forwarded by adding its LAN-in ID (LIN), its Bridge number (BN), its LAN-out ID (LOUT), and 4 bits as 0, setting the LTH field to 6, setting the LF field of the RI to the minimum of the received LF and the transfer capacity of the Bridge, and recalculating the FCS, and then queueing the frame for transmission on its LAN-out and, if implemented, the AREFramesForwarded counter shall be incremented.

(ARE6). If an SRT Bridge receives a valid All Routes Explorer frame with RI length = 4, and the AreRdLimit is greater than 1, then it shall either

Discard the frame and, if implemented, increment the InvalidRi counter; or

Modify the frame being forwarded by setting the last 4 bits of the previous RD to its Bridge number (BN), adding the next two-octet RD consisting of the LAN-out (LOUT) and remaining 4 bits as 0, incre-

menting the Length by 2, setting the largest frame field to the minimum of the received LF and the transfer capacity of the Bridge, recalculating the FCS, queueing the frame for transmission on its LAN-out, and, if implemented, increment the AREFramesForwarded counter.

(ARE7). If an SRT Bridge receives a valid All Route Explorer with a length field of 6 to 28 (inclusive, even numbers only), and the number of RDs less than the AreRdLimit, then the Bridge shall modify the frame being forwarded by setting the last 4 bits of the previous RD to its Bridge number (BN), adding the next two-octet RD consisting of the LAN-out (LOUT) and remaining 4 bits as 0, incrementing the length by 2, setting the largest frame field to the minimum of the received LF and the transfer capacity of the Bridge, recalculating the FCS, queueing the frame for transmission on its LAN-out and, if implemented, increment the ARE-FramesForwarded counter.

C.3.7.3 Spanning Tree Explorer (STE) frames

The conditions for forwarding STE frames are identical to those for ARE frames with the following exceptions:

a) No explorer filtering is necessary because of the spanning tree.
b) Forwarding is also conditional on the state of the LAN-out port and the state of the spanning tree.

Filtering on the destination address of STE frames shall not be performed. These frames shall traverse the entire spanning tree. STE frames are processed as indicated in Table C-5.

Table C-5 is explained as follows:

(STE1). If a Spanning Tree Explorer frame is received by an SRT Bridge and the LAN-in Port and LAN-out Port are in forwarding state and LOUT is already in the RI field, then the frame shall be discarded and, if implemented, the Bridge shall increment the DuplicateLanIdOrTreeError counter.

(STE2). If a Spanning Tree Explorer frame is received by an SRT Bridge, the LAN-in Port is in forwarding state, and the last LAN ID does not match LIN, the frame shall be discarded and, if implemented, the LanId-Mismatch counter shall be incremented.

(STE3). If a Spanning Tree Explorer frame is received by an SRT Bridge, the LAN-in Port and LAN-out Port are in forwarding state and the number of RDs meets or exceeds the SteRdLimit of the Port indicated by LOUT, the frame shall not be forwarded on LOUT and, if implemented, the SteRdLimit Exceeded counter of the Port indicated by LOUT, shall be incremented.

(STE4). If a Spanning Tree Explorer frame is received by an SRT Bridge, the LAN-in Port is in forwarding state, and the length field is zero or an odd number or the Direction bit is not 0, the frame shall be discarded and, if implemented, the InvalidRi counter shall be incremented.

(STE5). If an SRT Bridge with LAN-in and LAN-out Ports in forwarding state receives a valid Spanning Tree Explorer with a length field equal to 2, and the SteRdLimit is greater than 1, then the SRT Bridge shall modify the frame being forwarded by adding its LAN-in ID (LIN), its Bridge number (BN), its LAN-out ID (LOUT), and 4 bits as 0. The SRT Bridge shall also set the length field to 6, set the largest-frame field of the RI to the minimum of the received LF and the transfer capacity of the Bridge, and recalculate the FCS. The SRT Bridge shall then queue the frame for transmission on its LAN-out and, if implemented, increment the STEFramesForwarded counter.

(STE6). If an SRT Bridge with LAN-in and LAN-out ports in forwarding state receives a valid Spanning Tree Explorer with a length field of 4, and the SteRdLimit is greater than 1, then the Bridge shall either

a) Discard the frame and, if implemented, increment the InvalidRi counter; or

Table C-5—Spanning Tree Explorer frame forwarding state table

Ref.	PortState LIN	PortState LOUT	Condition	Action
STE1	F	F	(LOUT already in RI)	Discard Frame, Increment(DupLanIdOrTreeError)
STE2	F	—	Last LAN ID in RI ¬ = LIN	Discard Frame Increment(LanIdMismatch)
STE3	F	F	# RDs >= SteRdLimit	Discard Frame Increment Increment(SteRdLmtExceeded)
STE4	F	—	(D ¬ = 0) | (RI_LTH = 0) | (RI_LTH=odd number)	Do not forward Increment(InvalidRi)
STE5	F	F	(RI_LTH = 2) & (SteRdLimit > 1)	Add LIN,BN,LOUT to RI field SET RI_LTH = 6 SET LF = MIN_(LF,LIN,BR,LOUT) Recalculate FCS Forward frame Increment(STEFramesForwarded)
STE6	F	F	(RI_LTH = 4) & (Last LAN ID in RI = LIN) & (SteRdLimit > 1)	Discard frame Increment (InvalidRI) OR (Optionally) Add BN,LOUT to RI field RI_LTH = RI_LTH + 2 SET LF = MIN(LF,LIN,BR,LOUT) Recalculate FCS Forward frame Increment(STEFramesFowarded)
STE7	F	F	(RI_LTH=even number between 6 and 28 inclusive) & (LOUT not already in RI) & (Last LAN ID in RI = LIN) & (# RDs < SteRdLimit)	Add BN,LOUT to RI field RI_LTH = RI_LTH + 2 SET LF = MIN(LF,LIN,BR,LOUT) Recalculate FCS Forward frame Increment(STEFramesFowarded)
STE8	¬F	—		Do not forward on LOUT
STE9	F	¬F		Do not forward on LOUT

In all cases above: RII = 1, RT = 11X, and BPP State = enabled
Definitions:
Discard = Do not forward on any LAN-out
Forward = Transmit the frame on LAN-out
F = Forwarding Port State
¬F = Not Forwarding Port State
— = Any Port State

b) Modify the frame being forwarded by setting the last 4 bits of the previous RD to its Bridge number (BN), adding the next two-octet RD consisting of the LAN-out (LOUT) and remaining 4 bits as 0. The SRT Bridge shall also increment the length by 2, set the largest frame field to the minimum of the received LF and the transfer capacity of the Bridge, and recalculate the FCS. The SRT Bridge shall then queue the frame for transmission on its LAN-out and, if implemented, increment the STE-FramesForwarded counter.

(STE7). If an SRT Bridge with LAN-in and LAN-out Ports in forwarding state receives a valid Spanning Tree Explorer with a length field of 6 to 28 (inclusive, even numbers only), and the number of RDs is less than the SteRdLimit, then the SRT Bridge shall modify the frame being forwarded by setting the last 4 bits

of the previous RD to its Bridge number (BN), adding the next two-octet RD consisting of the LAN-out (LOUT) and remaining 4 bits as 0. The SRT Bridge shall also increment the length by 2, set the largest frame field to the minimum of the received LF and the transfer capacity of the Bridge, and recalculate the FCS. The SRT Bridge shall then queue the frame for transmission on its LAN-out and, if implemented, increment the STEFramesForwarded counter.

(STE8). SRT Bridges shall only forward Spanning Tree Explorer frames received on a Port if the Port is in forwarding state.

(STE9). SRT Bridges shall only forward Spanning Tree Explorer frames on a Port if the Port is in forwarding state.

C.3.7.4 Duplicate Bridge number test

Bridge management may check for parallel Bridges assigned the same number as its Bridge number by using the method below.

For every two LANs the Bridge interconnects, Bridge management initiates transmission of an LLC TEST PDU with the destination address set to the MAC Address of the Port connected to one LAN and the source address set to the MAC Address of the Port connected to the other LAN. The TEST PDU should contain a route consisting of the LAN-in relative to the source address, the Bridge number, and the LAN-out relative to the source address. If Bridge management receives more than one response back from this TEST, a parallel Bridge exists that is assigned the same Bridge number and network management should be notified. See Figure C-9.

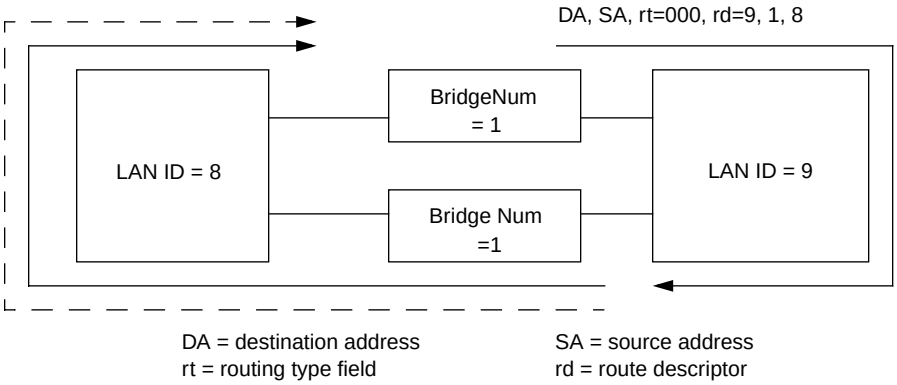


Figure C-9—Duplicate bridge number test illustration

C.3.7.5 Queued frames

The forwarding process provides storage for queued frames, awaiting an opportunity to submit these for transmission to the individual MAC entities associated with each Bridge Port. The order of frames received on the same Bridge Port shall be preserved for

a) Unicast frames with a given user_priority for a given combination of destination_address and source_address;
b) Multicast frames with a given user_priority for a given destination_address.

The Forwarding Process may provide more than one transmission queue for a given Bridge Port. Frames are assigned to storage queue(s) on the basis of their user_priority using a traffic class table that is part of the state information associated with each Port. The table indicates, for each possible value of user_priority, the

corresponding value of traffic class that shall be assigned. Priority value 0 represents the lowest value of user_priority, and 7 the highest value. Queues correspond one-to-one with traffic classes.

For management purposes, up to eight traffic classes are supported by the traffic class tables in order to allow for separate queues for each level of user_priority. Traffic classes are numbered 0 through N-1, where N is the number of traffic classes associated with a given outbound Port. Management of traffic class information is optional. Traffic class 0 corresponds to non-expedited traffic; non-zero traffic classes are expedited classes of traffic.

NOTE—In a given Bridge, it is permissible to implement different numbers of traffic classes for each Port. Ports associated with MAC methods that support a single transmission priority, such as CSMA/CD, can support more than one traffic class.

Where the Forwarding Process does not support expedited classes of traffic for a given Port, in other words, where there is a single traffic class associated with the Port, all values of user_priority map to traffic class 0. In Bridges that support expedited traffic, the default mapping of user_priority to traffic class, for the number of traffic classes implemented, is as shown in Table 7-2.

A frame queued by the forwarding process for transmission on a Port shall be removed from that queue on submission to the individual MAC entity for that Port; no further attempt shall be made to transmit the frame on that Port even if the transmission is known to have failed.

A frame queued by the forwarding process for transmission on a Port can be removed from that queue, and not subsequently transmitted, if the time for which buffering is guaranteed has been exceeded for that frame.

A frame queued for transmission on a Port shall be removed from that queue, and not subsequently submitted to the individual MAC entity for that Port, if that is necessary to ensure that the maximum Bridge transit delay (see 6.3.6) will not be exceeded at the time at which the frame would be subsequently transmitted.

An STE frame queued for transmission on a Port shall be removed from that queue if the associated Port leaves the forwarding state.

Removal of a frame from a queue for transmission on any particular Port does not of itself imply that it shall be removed from a queue for transmission on any other Port.

C.3.7.6 Selecting frames for transmission

Frames shall be selected for transmission in accordance with the provisions of 7.7.4.

C.3.7.7 Priority mapping

The source-routing forwarding process determines the value of the user_priority and access_priority parameters used to relay frames according to 7.7.5.

C.3.8 Addressing

In a source-routing Bridged LAN, each LAN and each Bridge shall be assigned a number as indicated in the following subclauses.

C.3.8.1 LAN ID

Each individual LAN in a multiple-LAN network shall be assigned a unique 12-bit non-zero LAN ID that is consistently known by all Bridges connecting to the given LAN.

C.3.8.2 Bridge number

Each Bridge shall have a Bridge number (BN). The route descriptor sequence (LIN-BN-LOUT) shall be unique for each path between two LANs connected by a Bridge.

C.3.8.3 Route descriptor

The route descriptor is a two-octet field (16 bits). The first 12 bits represent the LAN ID and the 4 bits following represent the Bridge number (see C.3.3.2 for frame format details). A sequence of route descriptors defines a unique route through the Bridged LAN. Because the end of a route is a LAN and not a Bridge, the individual Bridge portion of the last route descriptor in the routing information field is reserved and shall be set to zero. C.3.7 describes the method for creating, building, and forwarding frames with source routes.

C.4 Bridge Management

The enhancements in the following subclauses are made to the management facilities defined in Clause 14. These include the ability to control the source-routing paths through the Bridged LAN.

C.4.1 Bridge Management Entity

The following enhancements are made to the Bridge Management Entities defined in 14.4.

C.4.1.1 Bridge configuration

C.4.1.1.1 Discover Bridge

To enable the Discover Bridge management operations defined in 14.4.1.1 to report SRT capability in the Bridge, the parameter that follows is added to the outputs of the operation.

C.4.1.1.1.1 Purpose

To indicate the SRT transfer capability of the Bridge.

C.4.1.1.1.2 Additional inputs

None.

C.4.1.1.1.3 Additional outputs (14.4.1.1.3)

— SRT Transfer Capacity—a value field that indicates the largest MSDU field of a frame that can be handled by the SRT Bridge MAC relay entity. If SRT is not supported in the Bridge, this value shall be reported as zero.

C.4.1.1.2 Read Bridge

To enable the Read Bridge management operations defined in 14.4.1.2 to report SRT capability in the Bridge, the parameter that follows is added to the outputs of the operation.

C.4.1.1.2.1 Purpose

To indicate the SRT transfer capability of the Bridge.

C.4.1.1.2.2 Additional inputs

None.

C.4.1.1.2.3 Additional outputs (14.4.1.2.3)

— SRT Transfer Capacity—a value field that indicates the largest MSDU field of a frame that can be handled by the SRT Bridge MAC relay entity. If SRT is not supported in the Bridge, this value shall be reported as zero.

C.4.2 Forwarding Process

The following enhancements are made to the Forwarding Process as defined in 14.6.

C.4.2.1 The port counters

Reference 14.6.1.

C.4.2.1.1 Read Forwarding Port Counters

C.4.2.1.1.1 Purpose

The Port Counter objects for the SRT Bridge consist of the counters defined in 14.6.1.1, along with the SRT-specific counters in the following subclauses.

C.4.2.1.1.2 Additional inputs

None.

C.4.2.1.1.3 Additional outputs

The following port counters are added to the list of outputs defined by 14.6.1.1.3. If the counter is not supported, the value shall be reported as zero.

h) InvalidRI—count of frames that were discarded due to a formatting error (i.e., an odd RI length, or a 0 RI length).
i) DupLout—count of frames that were discarded due to a duplicate LOUT on SRFs.
j) LanIdMismatch—count of ARE and STE frames that were discarded because the last LAN ID in the routing information field did not equal the LAN-in ID.
k) DupLanIdOrTreeError—count of STE frames that were discarded because the LAN-out ID already existed in the routing information field.
l) STERDlimitExceeded—count of STE frames discarded due to STERD Limit exceeded.
m) ARERDlimitExceeded—count of ARE frames discarded due to ARERD Limit exceeded.
n) SRFs Forwarded—inbound count of SRFs forwarded from this Port to another Port on the Bridge.
o) STEFramesForwarded—inbound count of STE frames forwarded from this Port to another Port on the Bridge.
p) AREFramesForwarded—inbound count of ARE frames forwarded from this Port to another Port on the Bridge.

C.4.3 SRT Bridge Management Entity

C.4.3.1 SRT Bridge Configuration

The SRT Bridge Configuration operations for the SRT Bridge Management Entity allow management to Read the Bridge RD Limits, Set the Bridge RD Limits, Read the Bridge LF Mode, and Set the Bridge LF Mode.

C.4.3.1.1 Read LF Mode

C.4.3.1.1.1 Purpose

To report the mode that the Bridge uses to set LF bits in ARE and STE frames.

C.4.3.1.1.2 Inputs

None.

C.4.3.1.1.3 Outputs

a) LF Mode—a value field that indicates whether the Bridge uses base mode or extended mode to set LF bits.

C.4.3.1.2 Set LF Mode

C.4.3.1.2.1 Purpose

To set the mode that the Bridge will use to set the LF bits in ARE and STE frames.

C.4.3.1.2.2 Inputs

a) LF Mode—a value field that indicates whether the Bridge will use the base or extended mode for LF encodings

C.4.3.1.2.3 Outputs

None.

C.4.3.2 SRT Port Configuration

The SRT Port Configuration operations for the SRT Bridge Management Entity allow management to Read and Set the SRT attributes for each Port (LAN ID and Largest MSDU Size). Comparable to the operations defined in 14.8.2, the operations in the following subclauses can be performed on an SRT Port.

C.4.3.2.1 Read Port parameters

C.4.3.2.1.1 Purpose

To obtain information regarding a specific Port within the SRT Bridge Protocol Entity.

C.4.3.2.1.2 Inputs

a) Port number

C.4.3.2.1.3 Outputs

a) SRT Port type—a value field that indicates the port type available. Types include TB and SRT.
b) LAN ID—a value field that indicates the LAN ID corresponding to the LAN to which the Port is attached.
c) Largest MSDU Size—a value field that indicates the largest MSDU of a frame that can be handled by this Port.
d) ARE RD Limit—a value field that indicates the maximum number of route descriptors (RDs) that an ARE frame forwarded by the Bridge is allowed to contain.
e) STE RD Limit—a value field that indicates the maximum number of route descriptors (RDs) that an STE frame forwarded by the Bridge is allowed to contain.

C.4.3.2.2 Set LAN ID

C.4.3.2.2.1 Purpose

To associate a LAN ID with a specific Bridge Port.

C.4.3.2.2.2 Inputs

a) Port Number—the number of the Bridge Port.
b) Lan ID—the Lan ID corresponding to the LAN into which the Port is inserted made up of 12 bits.

C.4.3.2.2.3 Outputs

None.

C.4.3.2.3 Set Largest MSDU Size

C.4.3.2.3.1 Purpose

To associate a largest MSDU size with a Bridge Port.

C.4.3.2.3.2 Inputs

a) Port Number—the number of the Bridge Port.
b) Largest MSDU Size—a value field that specifies the maximum size MSDU that can be handled by this Port.

C.4.3.2.3.3 Outputs

None.

C.4.3.2.4 Set RD limit

C.4.3.2.4.1 Purpose

To associate a route descriptor limit, readable by the Read RD Limits operation, with ARE and STE frames forwarded by the Bridge.

C.4.3.2.4.2 Inputs

a) Port number—the number of the Bridge Port.

b) RD Limit Type—a value field that indicates whether the limit input is to be associated with ARE or STE frames.
c) RD Limit Value—a value field that specifies the maximum number of RDs that may be contained in the RI field of the frame type specified by the RD Limit Type. This field takes values from 0 to 14.

C.4.3.2.4.3 Outputs

None.

C.4.4 SRT Bridge Port Pair Database

The operations that follow can be performed on an SRT Bridge.

C.4.4.1 SRT Bridge Port Pair Configuration

C.4.4.1.1 Read SRT Bridge Port Pair Database Size

C.4.4.1.1.1 Purpose

To read the size of the Bridge Port Pair Database.

C.4.4.1.1.2 Inputs

None.

C.4.4.1.1.3 Outputs

a) Database Size—a value field that indicates the number of entries in the Bridge Port Pair Database.

C.4.4.1.2 Read SRT Bridge Port Pair Database Entry

C.4.4.1.2.1 Purpose

To read the attributes of an SRT Bridge Port pair database entity.

C.4.4.1.2.2 Inputs

a) Low Port number
b) High Port number

C.4.4.1.2.3 Outputs

a) Source-routing Bridge number
b) Bridge state (enabled or disabled)

C.4.4.1.3 Set SRT Bridge Port Pair Database Entry

C.4.4.1.3.1 Purpose

To set the attributes of an SRT Port pair database entry.

C.4.4.1.3.2 Inputs

a) Low Port number
b) High Port number
c) Source-routing Bridge number
d) Bridge state (enabled or disabled)

C.4.4.1.3.3 Outputs

None.

C.5 Management protocol

This matter is the subject to further ongoing study and resolution.

Annex D

(normative)

PICS proforma for Source-Routing Transparent Bridge operation[1]

D.1 Introduction

The supplier of a protocol implementation that is claimed to conform to Annex C of this standard shall complete the following PICS proforma, in addition to the PICS Proforma defined in Annex A. The provisions of A.1 through A.3 apply to the PICS proforma defined in this annex.

D.2 Relay and filtering of frames

Item	Feature	Status	References	Support
(2a)	Is the order of relayed frames of given user priority preserved?	M	C.2.3.1	Yes [ ]
(2b)	Are STE, ARE, and SRF Frames submitted to a MAC entity for transmission only once?	M	C.2.3.2	Yes [ ]
(2c)	Is the Largest Frame field set by the Bridge as described in C.3.3.2?	M	C.3.3.2	Yes [ ]
(2d)	Are frames received with no routing information (NSR) processed as described in IEEE Std 802.1D-1998?	M	C.3.5	Yes [ ]
(2e)	Are received Specifically Routed Frames (SRF) processed as described in C.3.7.1?	M	C.3.7.1	Yes [ ]
(2f)	Are received All Routes Explorer (ARE) Frames processed as described in C.3.7.2?	M	C.3.7.2	Yes [ ]
(2g)	Does the implementation support option 1 that allows filtering of frames that have been through the Bridge before?	O	C.3.7.2	Yes [ ] No [ ]
(2h)	Does the implementation support option 2 that allows filtering of frames that have been through the Bridge before?	O	C.3.7.2	Yes [ ] No [ ]
(2i)	Are received Spanning Tree Explorer (STE) frames processed as described in C.3.7.3?	M	C.3.7.3	Yes [ ]
(2j)	Does the implementation forward explorer frames (STE, ARE) with LTH = 4 as described in C.3.7.2 and C.3.7.3?	O	C.3.7.2, C.3.7.3	Yes [ ] No [ ]

[1]*Copyright release for PICS proformas:* Users of this standard may freely reproduce the PICS proforma in this annex so that it can be used for its intended purpose and may further publish the completed PICS.

D.3 Bridge numbers and LAN IDs

Item	Feature	Status	References	Support
(3a)	Can the LAN IDs of the attached LANs be configured in the Bridge?	M	C.3.8.1	Yes [ ]
(3b)	Can a Bridge Number be assigned to the Bridge?	M	C.3.8.2	Yes [ ]
(3c)	Can Bridge management use the duplicate Bridge number test to check for parallel Bridges that are assigned the same number?	O	C.3.7.4	Yes [ ] No [ ]

D.4 Bridge Management

This matter is subject to further ongoing study and resolution.

Item	Feature	Status	References	Support
(4a)	Bridge Management Operations	O	C.4	Yes [ ] No [ ]
(4b)	Read LF Mode.	M	C.4.3.1.1	Yes [ ]
(4c)	Set LF Mode.	M	C.4.3.1.2	Yes [ ]
(4d)	Read Port Parameters.	M	C.4.3.2.1	Yes [ ]
(4e)	Set ARE RD Limit.	M	C.4.3.2.4	Yes [ ]
(4f)	Specify the max value of the ARE RD Limit in this implementation.	M	C.4.3.2.4	A___
(4g)	Set STE RD Limit.	M	C.4.3.2.4	Yes [ ]
(4h)	Specify the max value of the STE RD Limit in this implementation.	M	C.4.3.2.4	A___
(4i)	Set LAN Id.	M	C.4.3.2.2	Yes [ ]
(4j)	Set Largest MSDU Size.	M	C.4.3.2.3	Yes [ ]
(4k)	Read SRT Bridge Port Pair Database Size.	M	C.4.4.1.1	Yes [ ]
(4l)	Read SRT Bridge Port Pair Database Entity.	M	C.4.4.1.2	Yes [ ]
(4m)	Set SRT Bridge Port Pair Database Entity.	M	C.4.4.1.3	Yes [ ]

Annex E

(normative)

Allocation of Object Identifier values

E.1 Introduction

This Annex contains a summary of all object identifier values that have been allocated by this standard, both in this revision and in previous revisions.

Each table shows allocations related to a specific category of information object. The heading of the table identifies the category of information object, and shows the invariant part of the object identifier value allocated to the entries in the table. The column marked Arc shows the value allocated to the arc subsequent to the invariant part, which completes the object identifier value allocated. The column marked Purpose contains a text description of the information object, and, in the case of current allocations, a reference to the location of the definition of the information object in the standard. The column marked Status shows the status of the allocated values, using the following convention:

R *Reserved.* The object identifier value is reserved for future use by this standard.

C *Current.* The object identifier value has been allocated to an information object that is defined within the current revision of the standard.

D *Deprecated.* The object identifier value has been allocated to an information object that was defined in a previous revision of the standard, and whose use is now deprecated.

E.2 Allocation Tables

Allocations for Standard-specific extensions
Invariant part of object identifier value =
{iso(1) member-body(2) us(840) ieee802dot1D(10009)standardsSpecificExtensions(0)}

ARC	PURPOSE	STATUS
N/A	N/A	N/A

Allocations for Functional Unit Packages
Invariant part of object identifier value =
{iso(1) member-body(2) us(840) ieee802dot1D(10009) functionalUnitPackage(1)}

ARC	PURPOSE	STATUS
bridgeManagementFunctionalUnit(0)	Bridge Functional Unit Package identifier (7.1 of IEEE Std 802.1j-1996)	D
bridgeManagementFunctionalUnit(1)	Bridge Functional Unit Package identifier (15.1)	C

Allocations for ASN.1 module identifiers
Invariant part of object identifier value =
{iso(1) member-body(2) us(840) ieee802dot1D(10009) asn1Module(2)}

ARC	PURPOSE	STATUS
bridgeDefinitions(0) version1(0)	Identifier for Version 1 of the Bridge ASN.1 definitions module	D
bridgeDefinitions(0) version2(1)	Identifier for Version 2 of the Bridge ASN.1 definitions module	C

Allocations for Managed object classes
Invariant part of object identifier value =
{iso(1) member-body(2) us(840) ieee802dot1D(10009) managedObjectClass(3)}

ARC	PURPOSE	STATUS
macBridgeDLE(0)	MAC Bridge DLE managed object class name (15.3)	C
port(1)	Port managed object class name (7.4 of IEEE Std 802.1j-1996)	D
selectiveTranslationTableEntry(2)	Selective Translation Table Entry managed object class name (15.5)	C
permanentDatabase(3)	Permanent Database managed object class name (15.6)	C
filteringDatabase(4)	Filtering Database managed object class name (7.7 of IEEE Std 802.1j-1996)	D
databaseEntry(5)	Database Entry managed object class name (15.8)	C
port(6)	Port managed object class name (15.4)	C
filteringDatabase(7)	Filtering Database managed object class name (15.7)	C
garpApplication(8)	GARP Application managed object class name (15.9)	C
garpAttributeType(9)	GARP Attribute Type managed object class name (15.10)	C
garpAttribute(10)	GARP Attribute managed object class name (15.11)	C
garpTimers (11)	GARP Timers managed object class name (15.12)	C

Allocations for Packages
Invariant part of object identifier value =
{iso(1) member-body(2) us(840) ieee802dot1D(10009) package(4)}

ARC	PURPOSE	STATUS
fdGroupSupport (0)	Conditional package name for support of Group Registration Entry counter by Filtering Database managed object class (15.7)	C
garpOriginatorSupport (1)	Conditional package name for support of GARP PDU originator address recording (15.11)	C

Allocations for Parameters
Invariant part of object identifier value =
{iso(1) member-body(2) us(840) ieee802dot1D(10009) parameter(5)}

ARC	PURPOSE	STATUS
N/A	N/A	N/A

Allocations for Name Binding identifiers
Invariant part of object identifier value =
{iso(1) member-body(2) us(840) ieee802dot1D(10009) nameBinding(6)}

ARC	PURPOSE	STATUS
macBridgeDLE-datalinkSubsystem(0)	Name binding identifier for MAC Bridge DLE when contained in Data Link Subsystem (15.3.1)	C
port-MACBridgeDLE(1)	Name binding identifier for Port when contained in MAC Bridge DLE (15.4.1)	C
selectiveTranslationTableEntry-Port(2)	Name binding identifier for Selective Translation Table Entry when contained in Port (15.5.1)	C
permanentDatabase-MACBridgeDLE(3)	Name binding identifier for Permanent Database when contained in MAC Bridge DLE (15.6.1)	C
filteringDatabase-MACBridgeDLE(4)	Name binding identifier for Filtering Database when contained in MAC Bridge DLE (15.7.1)	C
permanentEntry-PermanentDatabase(5)	Name binding identifier for Static Filtering Entry when contained in Permanent Database (15.8.1)	C
staticEntry-FilteringDatabase(6)	Name binding identifier for Static Filtering Entry when contained in Filtering Database (15.8.1)	C
dynamicEntry-FilteringDatabase(7)	Name binding identifier for Dynamic Filtering Entry when contained in Filtering Database (15.8.1)	C
groupEntry-FilteringDatabase(8)	Name binding identifier for Group Registration Entry when contained in Filtering Database (15.8.1)	C
garpApplication-Port(9)	Name binding identifier for GARP Application when contained in Port (15.9.1)	C
garpAttributeType-GARPApplication(10)	Name binding identifier for GARP Attribute Type when contained in GARP Application (15.10.1)	C
garpAttribute-GARPAttributeType(11)	Name binding identifier for GARP Attribute when contained in GARP Attribute Type (15.11.1)	C
garpTimers-Port(12)	Name binding identifier for GARP Timers when contained in Port (15.12.1)	C

Allocations for Attribute identifiers
Invariant part of object identifier value =
{iso(1) member-body(2) us(840) ieee802dot1D(10009) attribute(7)}

ARC	PURPOSE	STATUS
bridgeAddress(0)	Bridge Address attribute type name (15.3.2)	C
bridgeName(1)	Bridge Name attribute type name (15.3.3)	C
bridgeNumPorts(2)	Bridge Number of Ports attribute type name (15.3.4)	C
bridgePortAddresses(3)	Bridge Port Addresses attribute type name (15.3.5)	C
uptime(4)	Uptime attribute type name (15.3.6)	C
bridgeIdentifier(5)	Bridge Identifier attribute type name (15.3.7)	C
timeSinceTopologyChange(6)	Time Since Topology Change attribute type name (15.3.8)	C
topologyChangeCount(7)	Topology Change Count attribute type name (15.3.9)	C
topologyChange(8)	Topology Change attribute type name (15.3.10)	C
designatedRoot(9)	Designated Root attribute type name (15.3.11)	C
rootPathCost(10)	Root Path Cost attribute type name (15.3.12)	C
rootPort(11)	Root Port attribute type name (15.3.13)	C
maxAge(12)	Max Age attribute type name (15.3.14)	C
helloTime(13)	Hello Time attribute type name (15.3.15)	C

Allocations for Attribute identifiers *(Continued)*
Invariant part of object identifier value =
{iso(1) member-body(2) us(840) ieee802dot1D(10009) attribute(7)}

ARC	PURPOSE	STATUS
forwardDelay(14)	Forward Delay attribute type name (15.3.16)	C
bridgeMaxAge(15)	Bridge Max Age attribute type name (15.3.17)	C
bridgeHelloTime(16)	Bridge Hello Time attribute type name (15.3.18)	C
bridgeForwardDelay(17)	Bridge Forward Delay attribute type name (15.3.19)	C
holdTime(18)	Hold Time attribute type name (15.3.20)	C
bridgePriority(19)	Bridge Priority attribute type name (15.3.21)	C
portNumber(20)	Port Number attribute type name (15.4.2)	C
portName(21)	Port Name attribute type name (15.4.3)	C
portType(22)	Port Type attribute type name (15.4.4)	C
framesReceived(23)	Frames Received attribute type name (15.4.5)	C
discardInbound(24)	Discard Inbound attribute type name (15.4.6)	C
forwardOutbound(25)	Forward Outbound attribute type name (15.4.7)	C
discardLackOfBuffers(26)	Discard Lack of Buffers attribute type name (15.4.8)	C
discardTransitDelayExceeded(27)	Discard Transit Delay Exceeded attribute type name (15.4.9)	C
discardOnError(28)	Discard on Error attribute type name (15.4.10)	C
discardOnErrorDetail(29)	Discard on Error Detail attribute type name (15.4.11)	C
outboundUserPriority(30)	Outbound User Priority attribute type name (7.4.11 of IEEE Std 802.1j-1996)	D
outboundAccessPriority(31)	Outbound Access Priority (7.4.12 of IEEE Std 802.1j-1996)	D
portUptime(32)	Port Uptime attribute type name (15.4.16)	C
state(33)	State attribute type name (15.4.17)	C
portIdentifier(34)	Port Identifier attribute type name (15.4.18)	C
portPriority(35)	Port Priority attribute type name (15.4.19)	C
pathCost(36)	Path Cost attribute type name (15.4.20)	C
portDesignatedRoot(37)	Port Designated Root attribute type name (15.4.21)	C
designatedCost(38)	Designated Cost attribute type name (15.4.22)	C
designatedBridge(39)	Designated Bridge attribute type name (15.4.23)	C
designatedPort(40)	Designated Port attribute type name (15.4.24)	C
topologyChangeAck(41)	Topology Change Ack attribute type name (15.4.25)	C
typeValue(42)	Type Value attribute type name (15.5.2)	C
databaseName(43)	Database Name attribute type name (15.6.2)	C
databaseSize(44)	Database Size attribute type name (15.6.3)	C
numberOfEntries(45)	Number Of Entries attribute type name (15.6.4)	C
numberOfDynamicEntries(46)	Number Of Dynamic Entries attribute type name (15.7.2)	C
ageingTime(47)	Ageing Time attribute type name (15.7.3)	C
address(48)	Address attribute type name (7.8.1 of IEEE Std. 802.1f)	D
portNumberOrMap(49)	Port Number Or Map attribute type name (7.8.2 of IEEE Std 802.1j-1996)	D
entryType(50)	Entry Type attribute type name (15.8.4)	C

Allocations for Attribute identifiers *(Continued)*
Invariant part of object identifier value =
{iso(1) member-body(2) us(840) ieee802dot1D(10009) attribute(7)}

ARC	PURPOSE	STATUS
entryIndex(51)	Entry Index attribute type name (15.8.5)	C
defaultUserPriority(52)	Default User Priority attribute type name (15.4.12)	C
userPriorityRegenerationTable(53)	User Priority Regeneration Table attribute type name (15.4.13)	C
trafficClassTable(54)	Traffic Class Table attribute type name (15.4.12)	C
outboundAccessPriorityTable(55)	Outbound Access Priority Table attribute type name (15.4.12)	C
numberOfGroupRegistrationEntries(56)	Number Of Group Registration Entries attribute type name (15.7.4)	C
address(57)	Address attribute type name (15.8.2)	C
portMap(58)	Port Map attribute type name (15.8.3)	C
applicationName(59)	Application Name attribute type name (15.9.2)	C
joinTime(60)	Join Time attribute type name (15.12.3)	C
leaveTime(61)	Leave Time attribute type name (15.12.4)	C
leaveAllTime(62)	Leave All Time attribute type name (15.12.5)	C
failedRegistrations(63)	Failed Registrations attribute type name (15.10.4)	C
garpAttributeType(64)	GARP Attribute Type attribute type name (15.10.2)	C
garpAttribute(65)	GARP Attribute attribute type name (15.11.2)	C
applicantAdministrativeControl(66)	Applicant Administrative Control attribute type name (15.10.3)	C
stateValue(67)	State Value attribute type name (15.11.3)	C
originatorOfLastPDU(68)	Originator Of Last PDU attribute type name (15.11.4)	C
garpTimers (69)	Garp Timers attribute type name (15.12.2)	C

Allocations for Attribute Group identifiers
Invariant part of object identifier value =
{iso(1) member-body(2) us(840) ieee802dot1D(10009) attributeGroup(8)}

ARC	PURPOSE	STATUS
readOrDiscoverBridge(0)	Read or Discover Bridge attribute group name (15.3.22)	C
readBridgeProtocolParameters(1)	Read Bridge Protocol Parameters attribute group name (15.3.23)	C
readPort(2)	Read Port attribute group name (15.4.26)	C
readForwardingPortCounters(3)	Read Forwarding Port Counters attribute group name (15.4.27)	C
readTransmissionPriority(4)	Read Transmission Priority attribute group name (7.4.25 of IEEE Std 802.1j-1996)	D
readPortParameters(5)	Read Port Parameters attribute group name (15.4.28)	C
readDatabase(6)	Read Database attribute group name (15.6.5)	C
readDatabaseEntry(7)	Read Database Entry attribute group name (7.8.5 of IEEE Std 802.1j-1996)	D
readDatabaseEntry(8)	Read Database Entry attribute group name (15.8.6)	C

Allocations for Action types
Invariant part of object identifier value =
{iso(1) member-body(2) us(840) ieee802dot1D(10009) action(9)}

ARC	PURPOSE	STATUS
resetBridge(0)	Reset Bridge action type name (15.3.24)	C
forcePortState(1)	Force Port State action type name (15.4.29)	C

Allocations for Notification types
Invariant part of object identifier value =
{iso(1) member-body(2) us(840) ieee802dot1D(10009) notification(10)}

ARC	PURPOSE	STATUS
N/A	N/A	N/A

Annex F

(informative)

Target topology, migration, and interoperability

F.1 Target topology

Figure F-1 gives an example Bridged LAN configuration that is an illustration of the simple combinations of dual-port/multiport bridges, redundant paths, etc., that may appear in a Bridged LAN. The example was produced at a time when the focus was on dual-port Bridges interconnecting shared-media LANs, and the primary purpose of the example was to illustrate the capability of Spanning Tree to cope with redundant paths.

With the current and future focus on hub architectures incorporating multiport Bridges and switching technology, it is appropriate to include in the standard examples of such topologies. This is particularly important, given that in order to derive the full benefit of the dynamic multicast filtering capability, it will be necessary to employ just such LAN topologies. The following diagram therefore presents a more representative Spanning Tree topology, including both shared media elements and hub architectures.

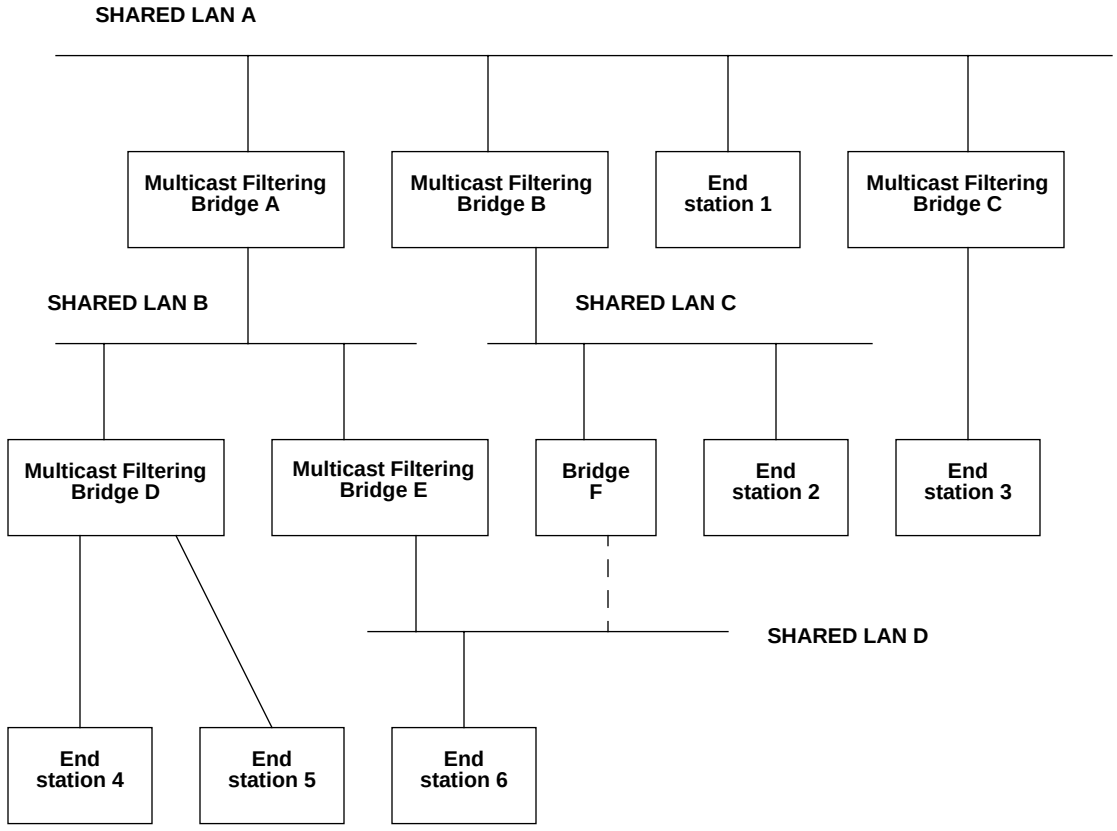


Figure F-1—Example Spanning Tree Topology

F.2 Migration considerations

This subclause examines some of the issues, and their possible solution, that will be encountered in mixed mode LANs; i.e., LANs that contain

a) A mixture of Legacy Bridges (Bridges that support only Basic Filtering Services) and Enhanced Bridges (Bridges that support Extended Filtering Services);
b) A mixture of legacy end stations, that are GMRP-unaware, and GMRP-aware end stations.

Given the range of uses, configurations, traffic patterns, and performance requirements that are to be found in LAN environments is impossibly large, this text is not intended to be definitive; it is essential that the migration strategy in a given Bridged LAN be defined in terms of its own particular environment and requirements.

F.2.1 Heterogeneous Bridge environments

The operation of GMRP is entirely transparent to Legacy Bridges; i.e., all GMRP PDUs are forwarded transparently by Legacy Bridges. It is therefore possible to intermix Legacy Bridges and Enhanced Bridges in a Bridged LAN in an arbitrary manner without affecting the integrity of the GMRP-generated directed graphs.

NOTE—This is true only if the Legacy Bridges do not contain any Static Filtering Entries that might interfere with the transmission of PDUs destined for any of the GARP Application addresses.

In a heterogeneous Bridged LAN, placing any Enhanced Bridges at the periphery of the network, rather than within the core of the network, is likely to produce the maximum benefit in terms of traffic segregation. However, such benefit may also depend on the mix and distribution of GMRP-aware and GMRP-unaware end stations (see F.2.2). In general, the nearer to the “center” of the Bridged LAN that a particular Bridge is placed, the more likely it is that any Groups defined in its Filtering Database will have registered members on all Ports; the Bridge will therefore be filtering relatively little Group traffic. Placing a Legacy Bridge at a similar location in the Bridged LAN therefore produces relatively little additional load on adjacent LAN segments. Conversely, the nearer the periphery of the Bridged LAN, the more likely it is that the Groups defined in the Filtering Database will have different membership patterns on each Port and Extended Filtering Services will therefore be of more value.

F.2.2 Heterogeneous end station environments

F.2.2.1 Use of Basic Filtering Mode and Legacy Bridges

As legacy Bridges are transparent to GARP protocol exchanges, the placement of end stations on segments served only by Legacy Bridges is of little interest to this discussion.

F.2.2.2 Use of Forward All Groups

Forward All Groups is the default Group filtering behavior (7.9.4) adopted by a given Port if the static and dynamic information held in the Filtering Database for that Port indicates a service requirement of Forward All Groups. Registering Forward All Groups as a service requirement (6.6.7) is intended to be used for two purposes:

a) To ensure that regions of the Bridged LAN containing only legacy devices can receive all multicast frames that originate in any other region of the Bridged LAN (except those that are explicitly disallowed by means of static configuration);
b) To allow successful operation of devices that require promiscuous reception, such as routers or network monitors, as discussed in F.3.

Ideally, all GMRP-unaware devices would be attached to LAN segments served by Bridge Ports operating with Forward All Groups as their default Group filtering behavior. As these segments will attract all multicast traffic originating in any other LAN segment, such legacy segments are ideally located in the same region of the Bridged LAN. As indicated in F.2.1, it may also be appropriate to attach these segments to a Bridge that is close to the center of the Bridged LAN.

F.2.2.3 Use of Forward Unregistered Groups

Forward Unregistered Groups is the default Group filtering behavior (7.9.4) adopted by a given Port if the static and dynamic information held in the Filtering Database for that Port indicates a service requirement of Forward Unregistered Groups, but does not indicate a service requirement of Forward All Groups. One of the intended uses of Forward Unregistered Groups registration is to provide for migration from GMRP-unaware environments to GMRP-aware environments. The “pass-through” behavior for unregistered Groups will allow GMRP-unaware end stations to continue to operate on segments serviced by Forward Unregistered Groups filtering behavior, as long as the multicast addresses they wish to receive are distinct from those used by GMRP-aware end stations (F.2.2.5 looks at the situation where the addresses used are not distinct). In general, it is more appropriate to place GMRP-unaware end stations on segments served by Ports providing Forward All Groups as their default Group filtering behavior.

Forward Unregistered Groups could also be useful in circumstances where GMRP-aware devices are able to distinguish between a set of “legacy” multicast addresses, for which they do not register, and a set of “new” multicast addresses, for which they register. Again, this scenario is only workable if the GMRP-unaware devices use only the “legacy” multicast addresses.

F.2.2.4 Use of Filter Unregistered Groups

Filter Unregistered Groups, which requires explicit registration of Group membership in order for an end station to receive frames destined for a Group, is the default Group filtering behavior (7.9.4) adopted by a given Port if the static and dynamic information held in the Filtering Database for that Port does not indicate a service requirement of Forward All Groups or Forward Unregistered Groups. It is essentially designed to operate with GMRP-aware end stations only. It is largely unsuitable for operation on any LAN segment that contains GMRP-unaware devices, unless the network administrator is prepared to manually configure the GARP administrative control parameters of all such Ports to meet the known requirements of all connected GMRP-unaware devices.

An alternative approach would be to include on any shared media segment that contained GMRP-unaware devices a “proxy member” that joined an appropriate set of Groups on behalf of the end stations on its segment. However, the network administrator must again be prepared to manually configure the “proxy member” in order for this to work, so this solution is little different from manually configuring the Filtering Databases.

In general, therefore, it would appear to be inappropriate to design a migration strategy around the use of Filter Unregistered Groups, unless the physical configuration of the LAN allows segregation of GMRP-aware and GMRP-unaware end stations.

F.2.2.5 Use of a common set of addresses

A major danger in migration from GMRP-unaware environments to GMRP-aware environments is that, if both types of end stations are recipients of a common set of MAC Addresses, there is the possibility that the GMRP-unaware end stations will become disenfranchised as a result of the GMRP-aware end stations registering membership of a Group with that MAC Address. How likely this is to happen depends greatly upon the environment concerned. Given the fact that the use of the GMRP registration mechanisms will, in many cases, be tightly coupled to new applications and software implementations, the real impact of this problem may prove to be smaller in practice than it might appear to be in theory.

However, given the danger, any migration environment needs to be analyzed in order to determine whether the problem actually exists. Strategies for addressing the problem are again likely to revolve around the approaches discussed in F.2.2.2, F.2.2.3, and F.2.2.4, namely:

a) Segregation of legacy end stations on Forward All Groups mode segments;
b) Manual configuration of the Filtering Database;
c) Use of “proxy member” end stations.

F.3 Interoperability with higher-layer multicast protocols and related issues

It is clearly necessary for GMRP to coexist and interoperate with devices in a Bridged LAN that may implement higher-layer multicast protocols. The primary example here will be an IP Router that supports IP multicast and IGMP (IP Group Management Protocol); however, other examples may be encountered that exhibit similar characteristics.

Although not a higher-layer protocol issue per se, it may also be necessary to accommodate devices that monitor the multicast traffic in a Bridged LAN, which rely for their operation on promiscuous reception.

F.3.1 IP multicast

An IP router needs to receive multicast frames sent to any of the addresses in the multicast address range designated for IP multicast use.

A workable approach is to ensure that all Bridge Ports connected to the same LAN segment as the router adopt a default Group filtering behavior of Forward All Groups. This can be achieved either by static configuration of the Filtering Database for the Ports concerned, or by the router issuing a REGISTER_SERVICE_REQUIREMENT primitive, specifying Forward All Groups, to its GMRP Participant, which in turn will send a GMRP PDU containing a JoinIn or JoinEmpty message (12.10.4.2) for Forward All Groups registration. For all Bridge Ports whose default Group filtering behavior is Forward Unregistered Groups or Filter Unregistered Groups that are attached to the same LAN segment (including any Bridges indirectly attached, via Bridges providing only Basic Filtering Services), this has the effect of forcing them to change their default Group filtering behavior to Forward All Groups. This has the effect of ensuring that all Bridges in the Bridged LAN will forward multicast frames in the direction of the Router, regardless of whether those frames are destined for registered Groups. This approach will work regardless of the number of routers that may exist in the Bridged LAN; however, it has the characteristic that the LAN segment that the router is attached to will “see” all multicast traffic generated anywhere in the Bridged LAN, regardless of whether it is of interest to the router.

Ideally the router would be connected to a Bridge by a LAN segment with only GMRP-unaware end stations attached to it, thus ensuring that the GARP-aware devices in the Bridged LAN obtain the maximum benefit from the Group filtering services. It may also be appropriate to attach this segment to a Bridge that is close to the center of the Bridged LAN, as indicated in the discussion of legacy devices in F.2.1.

GMRP-aware end stations involved in IP multicast reception must be capable of using GMRP to manage their membership of Groups as required by their use of IP multicast. For example, in order to join a given IP multicast Group using IGMP, the end station must perform a GMRP Join for that Group before issuing the IGMP Join.

NOTE—Although described in the context of IP multicast, the approach described here may be appropriate for routers implementing other protocol architectures that support multicast.

F.3.2 Monitoring multicast traffic

The addition of the dynamic multicast filtering capability defined in this standard to a Bridged LAN means that network monitors that would hitherto have been aware of all multicast traffic in the Bridged LAN (other than traffic constrained by static filters) may now be unaware of some multicast traffic, depending upon the point of attachment of the monitoring device and the pattern of Group registration that exists at that point in the network. The situation for multicasts will therefore be very similar to the situation for unicasts, where the dynamic entries in the Filtering Database results in some unicast traffic not being visible on some LAN segments.

Should it be desired to monitor all multicast traffic in the Bridged LAN, the simplest approach, which results in reception of all traffic except traffic destined for Groups for which static filtering information prevents membership, is for the monitor to ensure that all Bridge Ports connected to the same LAN segment as the monitor adopt a default Group filtering behavior of Forward All Groups, as described in F.3.1.

The ideal configuration for this scenario is either that the monitor is connected to a LAN segment that is already served by Ports that have adopted Forward All Groups filtering behavior, or that it is connected to a Bridge by a LAN segment with no other end stations, Bridges, or routers connected to it. It may also be appropriate to attach this segment to a Bridge that is close to the center of the Bridged LAN, as indicated in the discussion of legacy devices in F.2.1.

It should be noted that unless the monitor is attached to a LAN segment where all Bridge Ports serving the segment normally adopt Forward All Groups behavior (for example, a legacy LAN segment, as discussed in F.2.1), this approach may result in changes to the normal distribution of multicast frames in the Bridged LAN (i.e., changes to the distribution that would occur in the absence of the filtering mode changes introduced by the monitor itself).

Annex G

(informative)

Preserving the integrity of FCS fields in MAC Bridges

G.1 Background

When relaying frames between Ports of a MAC Bridge, one of the functions of the Bridge is to regenerate the FCS field in accordance with the MAC procedures that apply to the medium access method through which the frame will be relayed. One of the requirements of the MAC Bridge standard is that any such regeneration of the FCS shall not increase the level of undetected FCS errors that are experienced on the transmitting MAC over that which would be experienced by preserving the FCS (see 6.3.7 and 7.7.6). The point at issue here is only the *undetected* error rate; a conformant Bridge will discard frames that are received with an FCS error; however, there is a finite possibility of undetectable corruption taking place, i.e., the frame is corrupted, but the FCS algorithm does not reveal the error.

This annex looks at the various cases that apply to the operation of the MAC Bridge with respect to FCS regeneration and the alternative approaches that can be taken in order to address the cases. The following cases need to be considered:

a) The source and destination MAC methods are identical from the point of view of the operation of the FCS algorithm;

   NOTE—This is the case, for example, where the source and destination MAC methods are identical. It could also happen with dissimilar MAC methods if the FCS coverage, the FCS algorithm, the definitions of the fields covered by the FCS, and the bit/octet ordering are the same in both MAC methods.

b) The source and destination MAC methods differ from the point of view of the operation of the FCS algorithm;
c) The data covered by the FCS is not modified by the operation of the relay function of the Bridge;
d) The data covered by the FCS is modified by the operation of the relay function of the Bridge.

In each case, it is important to ensure that there is a low probability of additional undetected errors being generated by corruption of the data portion of the frame between removal of the old FCS and computation of the new one, which would result in the transmitted frame carrying invalid data and a valid FCS. Such corruption might occur, for example, as a result of the effect of environmental noise on the operation of the Bridge hardware.

If both a) and c) are true, then the FCS carried in the received frame is still valid, and the ideal approach would be to reuse this value as the FCS for the transmitted frame.

If either b) or d) are true, then it will be necessary for the Bridge to recalculate the FCS on transmission, and may therefore need to take additional precautions against in-memory corruption causing increased undetected FCS error levels.

G.2 Basic mathematical ideas behind CRC and FCS

The standard Cyclic Redundancy Check (CRC) algorithm is based on the following ideas:

a) An n-bit message is regarded as a polynomial, M(x), of degree n-1;

b) In order to generate a CRC value of length r bits, a generator polynomial, G(x), is used, of degree r;

c) The value of the last r bits of M(x) are chosen such that M(x) ÷ G(x) has a remainder of 0 (i.e., M(x) = 0 mod G(x)).

Messages can be added, in which each coefficient is 0 or 1:

d) M3(x) = M1(x) + M2(x). Messages are added together by bit-wise addition (XOR) of coefficients with the same bit position (i.e., coefficients with the same exponent).

Subtraction of messages:

e) Addition and subtraction are equivalent operations (as A = A XOR B XOR B); hence, using the messages in example d) above, M1(x) can be regenerated from M3(x) by adding M2(x).

Linearity:

f) If M(x) = 0 mod G(X), then (M(x) + E(x)) = 0 mod G(x) IFF E(x) = 0 mod G(x). In other words, if an error pattern, E(x), is added to a message, M(x), then the resultant message will give no remainder when divided by the generator polynomial if both M(x) and E(x) gave no remainder when divided by the generator polynomial.

g) $x^m M(x) = 0 \bmod G(x)$ IFF M(x) = 0 mod G(x). A message can be shifted by adding zero padding without affecting the integrity of the CRC.

h) Thus, the CRC algorithm has the property that it will detect any burst error, of length r bits or less, applied to the message, as such an error must result in a nonzero remainder when the message is divided by the generator polynomial. Alternatively, adding two messages together, both of which have valid CRCs, results in a message with a valid CRC.

In Ethernet, the algorithm used differs from the standard CRC, and is known as a Frame Check Sequence (FCS). The FCS is based on the CRC but with the following differences:

— The first 32 bits of M(x) are complemented before the FCS value is calculated;

— A CRC value is calculated, as described above, by dividing the first n-32 bits of M(x) by G(x) and taking the remainder as the CRC value;

— The CRC value is complemented and inserted as the last 32 bits of M(x).

By application of the addition and linearity rules, d) and f) above, the Ethernet frame with its FCS can be viewed as consisting of the following component messages, added together:

— An n-bit message, M(x), carrying a standard CRC as the last 32 bits;

— An n-bit “FCS adjustment factor”, $A_n(x)$, which adjusts the CRC to form an FCS.

Figure G-1 illustrates the conversion of a message with a CRC to the equivalent message with an FCS, by the addition of the adjustment factor. The adjustment factor consists of two components:

— The first component, when added to M(x), contributes an adjustment to the CRC which is equivalent to the effect of complementing the first 32 bits of M(x) before calculating the CRC bits, and complements the first 32 bits of M(x).

— The second component, when added to the partially adjusted M(x), restores the first 32 bits of M(x) to their original value, and complements the (adjusted) CRC to form the final FCS.

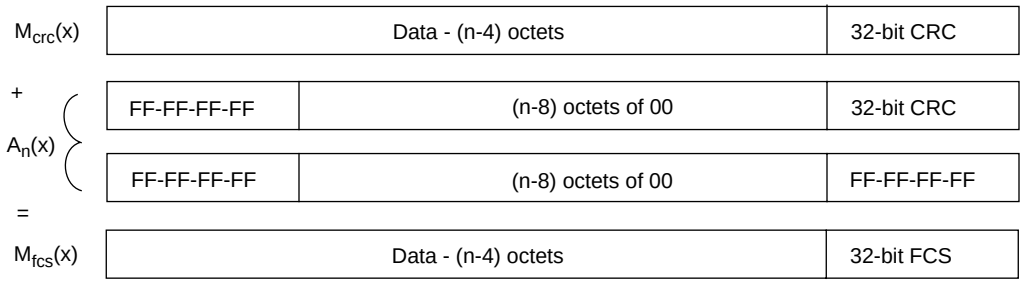


Figure G-1—Converting a CRC to an FCS

G.3 Detection Lossless Circuit approach

This approach is illustrated in Figure G-2. The basis of this approach is that the FCS used in the modified message is always recomputed from scratch using the normal FCS generation algorithm (Check Generator B); however, the original message data and FCS are also used as an input to an FCS checker (Checker A) in order to check that the inputs to the FCS generator were correct. Hence, if errors are introduced into the message while it is in memory in the Bridge in the time period since the message was received and FCS-checked, then Checker A will detect the error at the same time as the new FCS for the modified message is being generated.

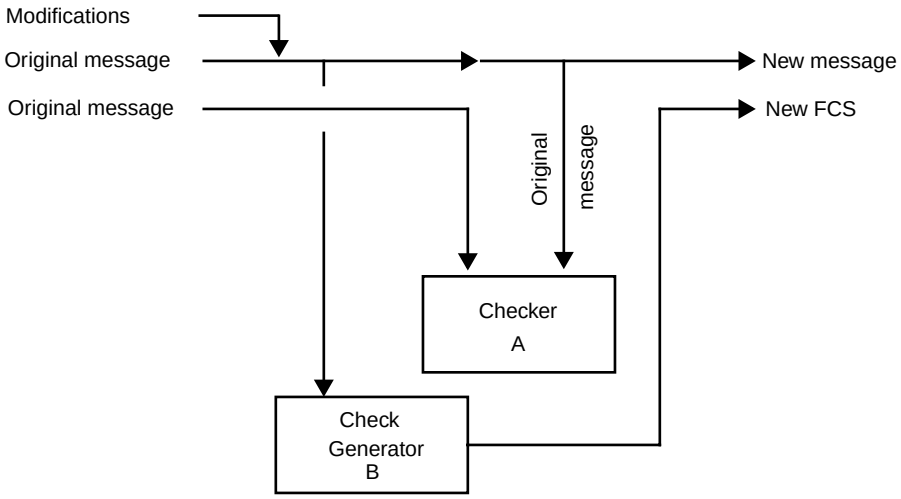


Figure G-2—Detection Lossless Circuit

Checker A is fed both with the original message in its unmodified form, and with the original message as used in the construction of the new message; this occurs after (or simultaneously with) the generation of the new FCS by Check Generator B. Any discrepancies detected by Checker A indicate that the information used to generate the new message and its FCS are suspect, and that the message should therefore be discarded.

G.4 Algorithmic modification of an FCS

In cases where the need to recalculate the FCS comes about as a result of changes to the data rather than by changes in the operation of the FCS algorithm, one option, rather than recalculating an FCS from scratch for a given message, is to examine the changes that have been made to the message and to calculate an FCS adjustment factor that reflects those changes. There are two cases:

a) The overall length of the message is unchanged, but the contents of one or more octets of data covered by the FCS have changed;
b) The overall length of the message has increased or decreased, as a result of the addition or removal of octets covered by the FCS, but the contents of the original octets is unchanged (although some may have been shifted in position as a result of the insertion/deletion).

G.4.1 Data changed, length unchanged

Adjustment of a message, $MF_1(x)$, in order to change the value of a field F from current value F_1 to new value F_2 consists of the following steps:

— Remove the FCS adjustment factor for a message of length n, $A_n(x)$;
— Remove the contribution to the message and its FCS provided by the current value of the field, $F_1(x)$;
— Add the contribution to the message and its FCS provided by the new value of the field, $F_2(x)$;
— Add the FCS adjustment factor for a message of length n, $A_n(x)$.

In other words,

$$MF_2(x) = MF_1(x) – A_n(x) – F_1(x) + F_2(x) + A_n(x)$$

The addition and subtraction of $A_n(x)$ cancel each other out, so,

$$MF_2(x) = MF_1(x) – F_1(x) + F_2(x)$$

Also, the correction factors $F_1(x)$ and $F_2(x)$ can be replaced by a single factor, $F_3(x)$, which is the contribution to M(x) and its FCS that would be provided by the value of Field F that would be produced by F_1 XOR F_2. So,

$$MF_2(x) = MF_1(x) + F_3(x)$$

This is shown in Figure G-3.

Alternatively,

$$\text{FCS of } MF_2(x) = (\text{FCS of } MF_1(x)) + \text{adjustment}$$

where

$$\text{Adjustment} = (\text{FCS of } F_1(x)) + (\text{FCS of } F_2(x))$$

or equivalently,

$$\text{FCS of } MF_2(x) = (\text{FCS of } MF_1(x)) + (\text{FCS of } F_3(x))$$

The adjustment factor (FCS of $F_3(x)$) is incidentally the same factor that would be used if the value of F were to be changed back from F_2 to F_1.

$MF_1(x)$	x data octets	F_1	y data octets	32-bit FCS
$– F_1(x)$	x octets of 00	F_1	y octets of 00	32-bit CRC
$+ F_2(x)$	x octets of 00	F_2	y octets of 00	32-bit CRC
$= MF_2(x)$	x data octets	F_2	y data octets	32-bit FCS

Note also that:

$F_1(x)$	x octets of 00	F_1	y octets of 00	32-bit CRC
$+ F_2(x)$	x octets of 00	F_2	y octets of 00	32-bit CRC
$= F_3(x)$	x octets of 00	F_1 XOR F_2	y octets of 00	32-bit FCS

Figure G-3—Field change adjustment

The relative simplicity of this case (relative to the case where the length changes) is a consequence of linearity and the fact that the FCS adjustment factor, $A_n(x)$, is constant for any given message length n, not a function of the data carried in the message.

G.4.2 Length changed, original data unchanged

Adjustment of a message M(x) of length n to cater for an inserted field, I, of length i involves the following steps:

— Remove the FCS adjustment factor for a message of length n, $A_n(x)$;
— Remove the contribution to the message and FCS that is provided by the header portion of the message (the portion before the inserted field), H(x);
— Add the contribution to the message and FCS that is provided by the header plus the inserted field I, HI(x);
— Add the FCS adjustment factor for a message of length n+i, $A_{n+i}(x)$.

So,

$$MI(x) = M(x) – A_n(x) – H(x) + HI(x) + A_{n+i}(x)$$

This is shown in Figure G-4.

Alternatively,

FCS of MI(x) = (FCS of M(x)) + adjustment

where

adjustment = (CRC of $A_n(x)$) + (CRC of H(x)) + (CRC of HI(x)) + (CRC of ($A_{n+i}(x)$)

As with the field change example, the value of *adjustment* is the same adjustment factor that would be used to adjust the FCS of MI(x) if field I were to be removed from the message.

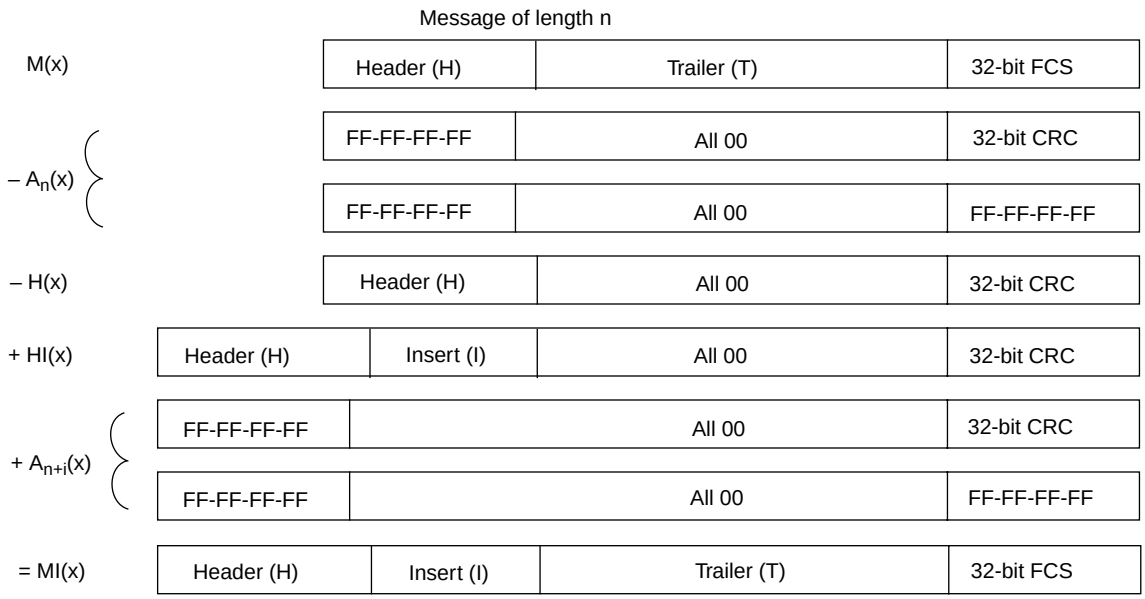


Figure G-4—Field insertion adjustment

G.4.3 Preservation of detectability

With either of the algorithmic adjustment methods described here, the important question is whether the ability of the FCS to detect errors in the message is preserved across these transformations; in other words, if a detectable error pattern, E(x), is added to the message M(x) before the modification, addition, or removal of a field, is that error still detectable after the message has been adjusted.

NOTE—The following analysis assumes that E(x) is a detectable error pattern, of length 32 bits or less. Analysis of the performance of this method with longer error patterns has not been attempted here.

In the cases where the error component is in those parts of the message that are not affected (changed or shifted) by the transformation, it is reasonably apparent that, as the transformation takes no account of the portion of the message that is in error, the detectability of that error is unchanged.

In the case of a field change, where field F_1 is to be replaced with F_2, and E(x) had been applied in some part of the message before changing the field, the new FCS value would be calculated as follows:

$$\text{FCS of } MF_2(x) = (\text{FCS of } MF_1(x)) + (\text{FCS of } F_1(x)) + (\text{FCS of } F_2(x))$$

However, the correction factor that is applied to the FCS contains components based only on F_1 and F_2, and no contribution related to the error pattern E(x). Therefore, after the value of the field has been changed to F_2, the new FCS value as calculated above will still be capable of detecting error pattern E(x), regardless of where the error pattern appears in the message.

In the case of a field insertion (or removal, as insertion and removal have already been shown to be equivalent), where E(x) has been applied to H(x), the correction factor that is applied to the FCS will contain two components related to E(x); the first based on E(x) in its original position, and the second based on E(x) shifted by i, the length of the inserted field, as follows:

FCS of MI(x) = (FCS of M(x)) + adjustment

where

adjustment = (CRC of $A_n(x)$) + (CRC of H(x)) + (CRC of HI(x)) + (CRC of ($A_{n+i}(x)$)
+ (CRC of E(x)) + (CRC of (E(x) shifted by i))

The presence of the (CRC of E(x)) component ensures that the final message will still indicate an FCS error. (Note that, even if (CRC of E(x)) and (CRC of (E(x) shifted by i)) turn out to be the same value, the final message will still indicate an FCS error, as its FCS is now correct for the value of MI(x) without the error component.)

The above does not exhaustively analyze all the cases of field insertion and removal, and in particular, does not analyze the cases where the error pattern crosses the insertion/removal boundaries; however, it can be shown that the error detection characteristics of the FCS are preserved in these cases as well.

The conclusion from the above examples is that, as long as the field values used in the calculation of these FCS correction factors are the same as the field values present in the original and final messages (e.g., that the same value of H+E is used both in its original and shifted forms), then the properties of the FCS are preserved. However, if different values are used for calculation of the correction factor from those present in the messages (as could happen, for example, as a result of in-memory corruption), then the properties of the FCS are no longer preserved. Hence, if these algorithmic approaches are used, it is necessary to design the implementation in a manner that ensures that these conditions are met.

G.5 Conclusions

Where it is necessary to recalculate the FCS before transmission, then either the algorithmic FCS modification approaches or the Detection Lossless Circuit approach offer a basis for this to be achieved while still preserving the error detection coverage that was provided by the original FCS.

The Detection Lossless Circuit approach requires access to the entire contents of the original message, to compute the necessary FCS adjustments or to check the integrity of the data that is being used to create the new message and its FCS. In contrast, the algorithmic approach requires access only to the header, insert, and FCS of the original message.

With the algorithmic approach, care needs to be taken in the implementation in order to ensure that the inputs to the correction algorithms are consistent with the contents of the original and modified messages, whereas in the Detection Lossless Circuit, such consistency checking is inherent in the way that the original FCS is used to check the generation of the new FCS.

Annex H

(informative)

Design considerations for Traffic Class Expediting and Dynamic Multicast Filtering

The following is intended to document some of the more significant design decisions taken during the development of this standard. The main purpose of this annex is to ensure that the history of such decisions is clear, thus avoiding repeated reexamination of issues that have already been resolved, and reinvention of solutions to them.

H.1 Generic Attribute Registration Protocol

H.1.1 Use of an unconfirmed protocol

The operation of GARP essentially relies upon the assumption that as long as the state machines are designed to ensure that more than one of the critical messages (JoinIn, JoinEmpty and Empty) are sent in each set of related messages, there is a high probability that all intended recipients will receive them. Early attempts at designing a GARP that involved explicit confirmations revealed a number of problems, particularly:

a) In the case where there are multiple GARP Participants attached to a segment, it is undesirable for all to respond to a request from one Participant. Limiting the responses to a single Participant is feasible; however, the value of a response indicating “I got that” from a single Participant is dubious, given that there is no guarantee that the other Participants involved have also seen the same message;
b) Developing a distribution mechanism that gives a high level of guarantee of correct operation is clearly possible (e.g., X.500’s distributed directory protocol); however, the complexity of such an approach would conflict with the fundamental design goals of MAC Bridges, i.e., that they are based on simple mechanisms and can therefore be engineered for high performance/cost ratios;
c) If a basic design goal of handling single packet loss is assumed, strategies based on multiple transmission of messages can provide the appropriate level of reliability while maintaining simplicity;
d) The users of the MAC Service have other means at their disposal to determine whether a service is being provided. For example, if the user requests receipt of a video channel, it will soon become apparent as to whether it is being received. The user/application concerned can therefore adopt “higher layer” recovery strategies to deal with the situation, without impacting the complexity of the Bridge. This is in any case probably appropriate, as the Bridge would not be in a position to determine appropriate courses of action in different combinations of user requirement and error condition.

H.1.2 Design of the Applicant state machine

The Applicant state machine has been designed to implement the requirements of GARP Participants in both end stations and Bridges. Strictly speaking, the requirements of the Applicant are slightly simpler in the end station than those of the Bridge; however, the two have been combined into a single description for reasons of overall clarity and simplicity. The main differences are as follows.

For an end station that does not need to know about the registration status of other Participants (e.g., a recipient of traffic for a Attribute) its main concern is with ensuring that all other Participants attached to its LAN segment see its Join messages. If one of its Leave messages goes astray, the Leave All garbage collection

mechanism will eventually fix the problem, so reliable reception of Leaves by other Participants is not a major issue. Therefore, for the Applicant in such an end station, the state machine could be further simplified, to one in which it simply sends two JoinIns to enter the Active Member state, and sends a single Leave to enter the Observer state.

For end stations or Bridge Ports that are concerned not only about their own membership state but also about the registration state of other Participants, the Registrar's situation is more involved. The major difference this creates is that if there are other Members or would-be Members on the same segment it is important that at least two of the members are heard by all Participants. This then ensures that each Participant, including those that actually generate the messages, will hear at least one Join on the LAN segment concerned. Hearing two different Participants is not the same thing as seeing two Join messages; both could potentially have been generated by the same Participant. Hence, the Join messages sent in GARP carry an indication of the Participant's registration state. JoinIn indicates "I wish to Join and I have seen one or more other Joins for this Attribute"; JoinEmpty indicates "I wish to Join and I have not seen any other Joins for this Attribute." In effect, the flavor of Join message sent reflects the registration state held by the Registrar for that Attribute. With this refinement, a Participant that considers itself to be in a Member state can respond to a JoinEmpty (indicating that the originator of the JoinEmpty has not registered its membership) by sending a JoinIn.

There are the following possibilities that apply when a Participant attempts to become a Member:

a) There are no other Participants in membership for that Attribute;
b) There is one other Participant in membership for that Attribute;
c) There are two or more other Participants in membership for that Attribute.

In case a), the new Member will send two JoinEmpty messages, and enter the Quiet/Active Member state. All other Participants on the LAN segment will see one or both of the JoinEmpty messages, and their Registrar state will therefore be IN for that Attribute.

In case b), the new Member sends two JoinIn messages, as the Registrar state is IN (one other member exists); this will ensure that the other Member registers the Attribute.

In case c), the new Member has seen two other Participants Join, so it can therefore enter the Passive Member state without sending a Join message at all, and by a similar argument, can become an Observer again without issuing a Leave.

H.1.3 Design of the Registrar state machine

The Registrar state machine is entirely passive, i.e., it does not transmit GARP PDUs; its job is simply to record the current registration state of an Attribute. The IN and LV states indicate a current registration; the MT state indicates de-registration. The LV to MT transition occurs if LeaveTime elapses since the last Leave or LeaveAll was seen and no Join has been received in response.

H.1.4 Analysis of GARP state machine operation

This subclause shows some example GARP protocol sequences, in order to

a) Illustrate normal operation of GARP;
b) Illustrate GARP operation under some failure conditions.

Each illustration consists of the following elements:

c) Text and a diagram, describing the scenario, the topology concerned, and the component stations/bridges;

d) A sequence showing the state transitions that occur in the state machines in each component involved in the scenario. These sequences are documented using *state sequence tables;* each row of the table shows the combined state of all components at an instant in time, plus any "system state," that is to say, state that is in the system as a whole, but that is not visible simply by looking at the state of the component end stations/Bridges. An example of "system state" would be a Join PDU that has been sent, but that has not yet been received at its destination. Rows earlier in the table represent combined states that occurred earlier in the time sequence. The table contains columns for each state machine type, plus a final column for descriptive commentary. Where appropriate, the abbreviations introduced in 12.8 are used.

H.1.4.1 Initial Join Scenarios

Topology: a leaf LAN segment, with a single station, A, attached; see Figure H-1.

Figure H-1—Topology: Leaf LAN Scenarios

First Scenario: A requests membership for Attribute AT, which hitherto has had no members; Bridge B responds normally. No packet loss occurs during the exchange. This scenario is analyzed in Table H-1. As can be seen, B considers the Attribute to be registered after the first JoinEmpty has been received from A, at step 4.

Table H-1—Initial Join: No Packet Loss

Step	State Of:	A	B	Commentary
1	Applicant Registrar "System"	VO — —	VO MT —	Starting conditions: Attribute AT unregistered in all state machines. No "system" state; A has no Registrar state machine.
2	Applicant Registrar "System"	VP — —	VO MT —	A's Applicant receives a ReqJoin primitive from the GARP Application for Attribute AT. This causes it to enter VP and wait for a transmission opportunity.
3	Applicant Registrar "System"	AA — JE(AT)	VO MT —	A sends a JoinEmpty, and enters AA.
4	Applicant Registrar "System"	AA — —	VO IN —	B receives the JoinEmpty; the Registrar registers the Attribute (IN) and the Applicant state is unchanged.
5	Applicant Registrar "System"	QA — JE(AT)	VO IN —	A sends a second JoinEmpty, and enters QA.
6	Applicant Registrar "System"	QA — —	VO IN —	The second JoinEmpty has no effect on the state of B.

Second Scenario: A requests membership for Attribute AT, which hitherto has had no members; B fails to see A's first JoinEmpty PDU. This scenario is analyzed in Table H-2. B does not register the Attribute until it receives A's second JoinEmpty PDU, at step 6.

Table H-2—Initial Join: Packet Loss on First Join PDU

Step	State Of:	A	B	Commentary
1	Applicant Registrar "System"	VO — —	VO MT —	Starting conditions: Attribute AT unregistered in all state machines. No "system" state; A has no Registrar state machine.
2	Applicant Registrar "System"	VP — —	VO MT —	A's Applicant receives a ReqJoin primitive from the GARP Application for Attribute AT. This causes it to enter VP and wait for a transmission opportunity.
3	Applicant Registrar "System"	AA — JE(AT)	VO MT —	A sends a JoinEmpty, and enters AA.
4	Applicant Registrar "System"	AA — —	VO MT —	B fails to see the JoinEmpty; the Registrar and Applicant state is unchanged.
5	Applicant Registrar "System"	QA — JE(AT)	VO MT —	A sends a second JoinEmpty, and enters QA.
6	Applicant Registrar "System"	QA — —	VO IN —	The second JoinEmpty causes B's Registrar to enter the IN state.

Third Scenario: A requests membership of Attribute AT, which hitherto has had no members; Bridge B fails to see A's second Join PDU. This scenario is analyzed in Table H-3. For practical purposes, indistinguishable from the First Scenario.

Table H-3—Initial Join: Packet Loss on Second Join PDU

Step	State Of:	End Station A	Bridge B	Commentary
1	Applicant Registrar "System"	VO — —	VO MT —	Starting conditions: Attribute AT unregistered in all state machines. No "system" state; A has no Registrar state machine.
2	Applicant Registrar "System"	VP — —	VO MT —	A's Applicant receives a ReqJoin primitive from the GARP Application for Attribute AT. This causes it to enter VP and wait for a transmission opportunity.
3	Applicant Registrar "System"	AA — JE(AT)	VO MT —	A sends a JoinEmpty, and enters AA.
4	Applicant Registrar "System"	AA — —	VO IN —	B receives the JoinEmpty; the Registrar registers the Attribute (IN) and the Applicant state is unchanged.
5	Applicant Registrar "System"	QA — JE(AT)	VO IN —	A sends a second JoinEmpty, and enters QA.
6	Applicant Registrar "System"	QA — —	VO IN —	The second JoinEmpty is lost, and has no effect on the state of B.

H.1.4.2 Last To Leave Scenarios

Topology: as described for H.1.4.1, Initial Join Scenario.

First Scenario: A requests to de-register membership for Attribute AT, which hitherto had only A as a member; Bridge B responds normally. No packet loss occurs during the exchange. This scenario is analyzed in Table H-4. As can be seen, B considers the Attribute still to be registered until the Leave Timer has expired. Other members get two opportunities to declare the Attribute, as both a Leave Empty and an Empty message are transmitted before the Attribute is finally de-registered by B.

Table H-4—Last To Leave: No Packet Loss

Step	State Of:	A	B	Commentary
1	Applicant Registrar "System"	QA — —	VO IN —	Starting conditions: Attribute AT active in A's Applicant and B's Registrar state machines. A has no Registrar state machine.
2	Applicant Registrar "System"	LA — —	VO IN —	A's Applicant receives a ReqLeave primitive from the GARP Application. A prepares to issue a Leave by entering the LA state.
3	Applicant Registrar "System"	VO — LE(AT)	VO IN —	A transmission opportunity occurs; A transmits Leave Empty and enters the VO state.
4	Applicant Registrar "System"	VO — —	LO LV —	B sees the Leave Empty. The Registrar starts Leave Timer; the Applicant enters LO and waits for a transmission opportunity.
5	Applicant Registrar "System"	VO — —	VO LV E(AT)	B transmits an Empty message for AT, to prompt any remaining members to rejoin.
6	Applicant Registrar "System"	VO — —	VO LV —	The Empty message has no effect on A.
7	Applicant Registrar "System"	VO — —	VO MT —	The Leave Timer expires; B's Registrar de-registers the Attribute.

Second Scenario: A requests to de-register membership for Attribute AT, which hitherto has had only A as a member; Bridge B fails to see A's Leave Empty PDU. This scenario is analyzed in Table H-5. On average, this scenario takes B half a Leave All Period plus a Leave Period before it finally de-registers the Attribute.

Table H-5—Last To Leave: Packet Loss on First Leave PDU

Step	State Of:	A	B	Commentary
1	Applicant Registrar "System"	QA — —	VO IN —	Starting conditions: Attribute AT active in A's Applicant and B's Registrar state machines. A has no Registrar state machine.
2	Applicant Registrar "System"	LA — —	VO IN —	A's Applicant receives a ReqLeave primitive from the GARP Application. A prepares to issue a Leave by entering the LA state.
3	Applicant Registrar "System"	VO — LE(AT)	VO IN —	A transmission opportunity occurs; A transmits Leave Empty and enters the VO state.
4	Applicant Registrar "System"	VO — —	VO IN —	B fails to see the Leave Empty.
5	Applicant Registrar "System"	VO — —	LO LV LeaveAll	B transmits a LeaveAll message. B's Registrar enters LV and starts the Leave Timer. The Applicant enters LO.
6	Applicant Registrar "System"	VO — —	LO LV —	The Leave All message has no effect on A.
7	Applicant Registrar "System"	VO — —	VO LV E(AT)	A transmission opportunity occurs; B transmits an Empty message for AT, and the Applicant returns to VO.
8	Applicant Registrar "System"	VO — —	VO LV —	The Empty message has no effect on A.
9	Applicant Registrar "System"	VO — —	VO MT —	The Leave Timer expires. B's Registrar de-registers the attribute.

Third Scenario: Strictly speaking, there are further single packet loss scenarios to be considered, involving loss of B's Leave All and Empty messages, but the sequential analysis is identical to the second scenario, as the Empty or Leave All messages would have had no effect on A's state in either case.

H.1.4.3 Leave/Rejoin Scenarios—Single Member

Topology: as described for H.1.4.1, Initial Join Scenario.

First Scenario: A is a member for Attribute AT, which has no other members. Bridge B issues a Leave All; A rejoins. No packet loss occurs during the exchange. This scenario is analyzed in Table H-6. As can be seen, A remains registered at all times. A slight variant of this scenario occurs if B manages to transmit Empty before A transmits the first Join Empty (between steps 3 and 4); however, this has no effect on steps 4 through 7.

Table H-6—Single Member LeaveAll/Rejoin: No Packet Loss

Step	State Of:	A	B	Commentary
1	Applicant Registrar "System"	QA — —	VO IN —	Starting conditions: Attribute AT active in A's Applicant and B's Registrar state machines. A has no Registrar state machine.
2	Applicant Registrar "System"	QA — —	LO LV LeaveAll	B sends a LeaveAll. This causes B to enter LO and LV states for Applicant and Registrar, and to start the Leave Timer.
3	Applicant Registrar "System"	VP — —	LO LV —	A becomes very anxious, and awaits an opportunity to send a JoinEmpty.
4	Applicant Registrar "System"	AA — JE(AT)	LO LV —	A transmits JoinEmpty, and becomes less anxious.
5	Applicant Registrar "System"	AA — —	VO IN —	B sees the Join, registers the membership and re-enters VO.
6	Applicant Registrar "System"	QA — JE(AT)	VO IN —	A transmits a second JoinEmpty, and becomes Quiet.
7	Applicant Registrar "System"	QA — —	VO IN —	The second Join Empty has no effect on B.

Second Scenario: A is a member of Attribute AT, which has no other members. Bridge B issues a Leave All; A fails to see it, and does not rejoin until B issues the Empty message. This scenario is analyzed in Table H-7. As can be seen, A remains registered at all times, despite the loss of the LeaveAll.

Table H-7—Single Member Leave/Rejoin: Packet Loss on LeaveAll PDU

Step	State Of:	A	B	Commentary
1	Applicant Registrar "System"	QA — —	VO IN —	Starting conditions: Attribute AT active in A's Applicant and B's Registrar state machines. A has no Registrar state machine.
2	Applicant Registrar "System"	QA — —	LO LV LeaveAll	B sends a LeaveAll. This causes B to enter LO and LV states for Applicant and Registrar, and to start the Leave Timer.
3	Applicant Registrar "System"	QA — —	LO LV —	A does not see the LeaveAll.
4	Applicant Registrar "System"	QA — —	VO LV E(AT)	A transmission opportunity occurs for B; it sends an Empty.
5	Applicant Registrar "System"	VA — —	VO LV —	A sees the Empty and becomes Very Anxious.
6	Applicant Registrar "System"	AA — JE(AT)	VO LV —	A transmission opportunity occurs for A. It transmits a JoinEmpty, and becomes Anxious.
7	Applicant Registrar "System"	AA — —	VO IN —	B sees the JoinEmpty, and re-registers the Attribute.
8	Applicant Registrar "System"	QA — JE(AT)	VO IN —	A transmits a second JoinEmpty, and becomes Quiet.
9	Applicant Registrar "System"	AA — —	VO IN —	The second JoinEmpty has no effect.

Third Scenario: A is a member of Attribute AT, which has no other members. Bridge B issues a Leave All; A responds with a JoinEmpty which B does not see, so B issues an Empty message. This scenario is analyzed in Table H-8. As can be seen, A remains registered at all times, despite the loss of the first JoinEmpty. The effect of the Empty is to reset A's count of Joins, so three Join Empty messages are sent before A becomes Quiet.

Table H-8—Single Member Leave/Rejoin: Packet Loss on First Join PDU

Step	State Of:	A	B	Commentary
1	Applicant Registrar "System"	QA — —	VO IN —	Starting conditions: Attribute AT active in A's Applicant and B's Registrar state machines. A has no Registrar state machine.
2	Applicant Registrar "System"	QA — —	LO LV LeaveAll	B sends a LeaveAll. This causes B to enter LO and LV states for Applicant and Registrar, and to start the Leave Timer.
3	Applicant Registrar "System"	VP — —	LO LV —	A becomes very anxious, and awaits an opportunity to send a JoinEmpty.
4	Applicant Registrar "System"	AA — JE(AT)	LO LV —	A transmits JoinEmpty, and becomes less anxious.
5	Applicant Registrar "System"	AA — —	LO LV —	B fails to see the Join Empty.
6	Applicant Registrar "System"	AA — —	VO LV E(AT)	A transmission opportunity occurs for B; it sends an Empty.
7	Applicant Registrar "System"	VA — —	VO LV —	A sees the Empty and becomes Very Anxious.
8	Applicant Registrar "System"	AA — JE(AT)	VO LV	A transmits a second JoinEmpty, and becomes Anxious.
9	Applicant Registrar "System"	AA — —	VO IN —	B sees the JoinEmpty, and re-registers the Attribute.
10	Applicant Registrar "System"	QA — JE(AT)	VO IN	A transmits a third JoinEmpty, and becomes Quiet.
11	Applicant Registrar "System"	AA — —	VO IN —	The third JoinEmpty has no effect.

H.1.4.4 Backbone LAN Initial Join Scenarios

Topology: a backbone LAN segment, with two Bridges, A and B, attached; see Figure H-2.

The scenarios look only at the behavior of the Bridge Ports that are connected to the backbone segment.

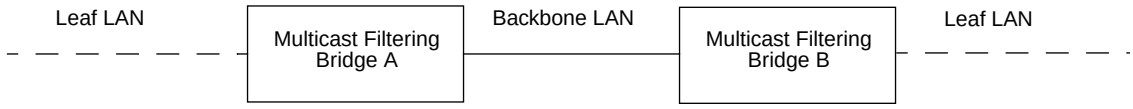


Figure H-2—Topology: Backbone LAN Scenarios

First Scenario: A requests membership for Attribute AT on the backbone, as a result of initial registrations by Applicants on its leaf LAN. B registers the same Attribute some time later. Both Bridges respond normally. No packet loss occurs during the exchange. This scenario is analyzed in Table H-9. As can be seen, B considers the Attribute to be registered after the first Join has been received from A; similarly, A considers that the Attribute is registered after B's first JoinIn is received.

Table H-9—Backbone Initial Join: No Packet Loss

Step	State Of:	A	B	Commentary
1	Applicant Registrar "System"	VO MT —	VO MT —	Starting conditions: Attribute AT unregistered in all state machines. No "system" state.
2	Applicant Registrar "System"	VP MT —	VO MT —	A's Applicant receives a ReqJoin primitive from the GARP Application for Attribute AT. This causes it to enter VP and wait for a transmission opportunity.
3	Applicant Registrar "System"	AA MT JE(AT)	VO MT —	A sends a JoinEmpty, and enters AA.
4	Applicant Registrar "System"	AA MT —	VO IN —	B receives the JoinEmpty; the Registrar registers the Attribute (IN) and the Applicant state is unchanged.
5	Applicant Registrar "System"	AA MT —	VP IN —	B's Applicant receives a ReqJoin primitive from the GARP Application for Attribute AT. This causes it to enter VP and wait for a transmission opportunity.
6	Applicant Registrar "System"	AA MT —	AA IN JI(AT)	B issues a JoinIn.
7	Applicant Registrar "System"	QA IN —	AA IN —	B's JoinIn causes A to register the attribute, and also suppresses A's second Join.
8	Applicant Registrar "System"	QA IN —	QA IN JI(AT)	B issues a second JoinIn and becomes Quiet.
9	Applicant Registrar "System"	QA IN —	QA IN —	B's second JoinIn has no effect.

Second Scenario: A requests membership of Attribute AT on the backbone, as a result of initial registrations by Applicants on its leaf LAN. Bridge B fails to see A's first Join PDU. A issues a second Join after Join Time. B subsequently joins the same Attribute. This scenario is analyzed in Table H-10. B does not register the Attribute for A until it sees A's second Join, one JoinTime after A's initial Join attempt.

Table H-10—Backbone Initial Join: Packet Loss on First Join PDU

Step	State Of:	A	B	Commentary
1	Applicant Registrar "System"	VO MT —	VO MT —	Starting conditions: Attribute AT unregistered in all state machines. No "system" state.
2	Applicant Registrar "System"	VP MT —	VO MT —	A's Applicant receives a ReqJoin primitive from the GARP Application for Attribute AT. This causes it to enter VP and wait for a transmission opportunity.
3	Applicant Registrar "System"	AA MT JE(AT)	VO MT —	A sends a JoinEmpty, and enters AA.
4	Applicant Registrar "System"	AA MT —	VO MT —	B fails to see the JoinEmpty.
5	Applicant Registrar "System"	QA MT JE(AT)	VO MT —	A sends a second JoinEmpty, and enters QA.
6	Applicant Registrar "System"	QA MT —	VO IN —	B receives the JoinEmpty; the Registrar registers the Attribute (IN) and the Applicant state is unchanged.
7	Applicant Registrar "System"	QA MT —	VP IN —	B's Applicant receives a ReqJoin primitive from the GARP Application for Attribute AT. This causes it to enter VP and wait for a transmission opportunity.
8	Applicant Registrar "System"	QA MT —	AA IN JI(AT)	B issues a JoinIn.
9	Applicant Registrar "System"	QA IN —	AA IN —	B's JoinIn causes A to register the attribute.
10	Applicant Registrar "System"	QA IN —	QA IN JE(AT)	B issues a second JoinIn and becomes Quiet.
11	Applicant Registrar "System"	QA IN —	QA IN —	B's second JoinIn has no effect.

Third Scenario: A requests membership of Attribute AT on the backbone, as a result of initial registrations by Applicants on its leaf LAN. Bridge B fails to see A's first Join PDU, but issues a Join itself for the same Attribute. This scenario is analyzed in Table H-11. B does not register the Attribute for A until after A's second Join attempt.

Table H-11—Backbone Initial Join: simultaneous Join from two Bridges with packet loss on first Join

Step	State Of:	Bridge A	Bridge B	Commentary
1	Applicant Registrar "System"	VO MT —	VO MT —	Starting conditions: Attribute AT unregistered in all state machines. No members. No "system" state.
2	Applicant Registrar "System"	VP MT —	VO MT —	A's Applicant receives a ReqJoin primitive from the GARP Application for Attribute AT. This causes it to enter VP and wait for a transmission opportunity.
3	Applicant Registrar "System"	AA MT JE(AT)	VO MT —	A sends a JoinEmpty, and enters AA.
4	Applicant Registrar "System"	AA MT —	VO MT —	B fails to see the JoinEmpty.
5	Applicant Registrar "System"	AA MT —	VP MT —	B's Applicant receives a ReqJoin primitive from the GARP Application for Attribute AT. This causes it to enter VP and wait for a transmission opportunity.
6	Applicant Registrar "System"	AA MT —	AA MT JE(AT)	B sends a JoinEmpty; and enters AA.
7	Applicant Registrar "System"	VA IN —	AA MT —	A receives the JoinEmpty. This causes it to enter VA, and to register the Attribute.
8	Applicant Registrar "System"	AA IN JI(AT)	AA MT —	A sends a JoinIn and enters AA.
9	Applicant Registrar "System"	AA IN —	QA IN —	B sees the JoinIn; it registers the attribute and further Joins from B are suppressed.
10	Applicant Registrar "System"	QA IN JE(AT)	QA IN —	A issues a second JoinIn and becomes Quiet.
11	Applicant Registrar "System"	QA IN —	QA IN —	A's second JoinIn has no effect on B.

H.1.4.5 Shared media LAN Segment Scenarios

Topology: a backbone LAN segment, with two Bridges, A and B, attached; see Figure H-3.

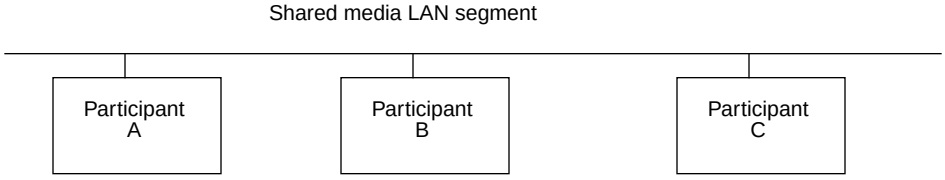


Figure H-3—Topology: Shared media LAN segment scenarios

First scenario: Illustrates the point, mentioned in H.1.2, that if there are already two or more Members on a given LAN segment, then further Members can join and leave without issuing any GARP messages. Three Participants are attached to the segment. A and B Join for Attribute AT (an identical sequence to Table H-9). Subsequently, C can Join and Leave without issuing any GARP messages. This scenario is analyzed in Table H-12.

Table H-12—Shared media: Third party can Join/Leave without sending GARP messages

Step	State Of:	A	B	C	Commentary
1	Applicant Registrar "System"	VO MT —	VO MT —	VO MT —	Starting conditions: Attribute AT unregistered in all state machines. No "system" state.
2	Applicant Registrar "System"	VP MT —	VO MT —	VO MT —	A's Applicant receives a ReqJoin primitive from the GARP Application for Attribute AT. This causes it to enter VP and wait for a transmission opportunity.
3	Applicant Registrar "System"	AA MT JE(AT)	VO MT —	VO MT —	A sends a JoinEmpty, and enters AA.
4	Applicant Registrar "System"	AA MT —	VO IN —	VO IN —	B and C receive the JoinEmpty; the Registrar registers the Attribute (IN) and the Applicant state is unchanged.
5	Applicant Registrar "System"	AA MT —	VP IN —	VO IN —	B's Applicant receives a ReqJoin primitive from the GARP Application for Attribute AT. This causes it to enter VP and wait for a transmission opportunity.
6	Applicant Registrar "System"	AA MT —	AA IN JI(AT)	VO IN —	B issues a JoinIn.
7	Applicant Registrar "System"	QA IN —	AA IN —	AO IN —	B's JoinIn causes A to register the attribute, and also suppresses A's second Join. It also causes C's Applicant to enter the AO (Anxious Observer) state.
8	Applicant Registrar "System"	QA IN —	QA IN JI(AT)	AO IN —	B issues a second JoinIn and becomes Quiet.
9	Applicant Registrar "System"	QA IN —	QA IN —	QO IN —	B's second JoinIn has no effect on A but causes C to enter the QO (Quiet Observer) state.
10	Applicant Registrar "System"	QA IN —	QA IN —	QP IN —	C's Applicant receives a ReqJoin primitive from the GARP Application for Attribute AT. Because C has seen two JoinIn's for AT, it is able to directly enter the QP (Quiet/Passive member) state, without issuing any GARP messages.
11	Applicant Registrar "System"	QA IN —	QA IN —	QO IN —	C's Applicant receives a ReqLeave primitive from the GARP Application for Attribute AT. Because it Joined AT without issuing any messages, it is able to Leave by directly entering the QO (Quiet/Observer) state, without issuing any GARP messages.

Second scenario: Shows the “last to leave” sequence for circumstances where more than two participants exist on a LAN segment. A has declared Attribute AT; B and C have registered AT in their Registrar state machines. A Leaves. The protocol exchanges occur normally. This scenario is analyzed in Table H-13.

NOTE—This second scenario is included primarily to demonstrate that the length of time taken for the last member to Leave is not a function of the number of Participants involved. This was a problem that existed with earlier versions of GARP, where the receipt of an Empty message by a registrar in the LV state (or its equivalent) caused the Leave Timer to be restarted. In the protocol as described, the time taken for the de-registration to occur after the last member leaves is equal to one Leave Time, regardless of the number of Participants.

Table H-13—Shared media: Leave sequence for three Participants

Step	State Of:	A	B	C	Commentary
1	Applicant Registrar “System”	QA MT —	VO IN —	VO IN —	Starting conditions: Attribute AT active in A’s Applicant and in B and C’s Registrar state machines.
2	Applicant Registrar “System”	LA MT —	VO IN —	VO IN —	A’s Applicant receives a ReqLeave primitive from the GARP Application. A prepares to issue a Leave by entering the LA state.
3	Applicant Registrar “System”	VO MT LE(AT)	VO IN —	VO IN —	A transmission opportunity occurs; A transmits Leave Empty and enters the VO state.
4	Applicant Registrar “System”	VO MT —	LO LV —	LO LV —	B and C see the Leave Empty. Their Registrars start Leave Timer; their Applicants enter LO and wait for a transmission opportunity.
5	Applicant Registrar “System”	VO MT —	VO LV E(AT)	LO LV —	A transmission opportunity for B occurs; it transmits an Empty message for AT, to prompt any remaining members to rejoin.
6	Applicant Registrar “System”	VO MT —	VO LV —	VO LV —	The Empty message has no effect on A, but suppresses C’s intent to transmit an Empty; C enters VO.
7	Applicant Registrar “System”	VO MT —	VO MT —	VO MT —	The Leave Timers expire in B and C; their Registrars both de-register the Attribute.

H.2 User priorities and traffic classes

This subclause documents some of the reasoning behind the choice of default mappings between user_priority values and traffic classes, as represented in Table 7-2.

H.2.1 Goals

There are many possible ways to assign user priority semantics, and to use a limited number of supporting queues. This annex does not attempt to survey the entire range of possibilities, but aims to set out a reasonable set of defaults for use in typical LAN Bridged environments to provide integrated services, i.e., to meet the original objectives of this standard.

This standard allows priorities, queue mappings, and queue service disciplines to be managed to best support the user's goals. However, it is widely appreciated that a set of well known and easily understood defaults will greatly facilitate plug and play interworking and the deployment of integrated services. The defaults advocated here have therefore been chosen both to support the emerging integrated services mapping work in the IETF (ISSLL, and IS802 in particular), and to provide useful service without any management of the Bridges.

H.2.2 Traffic types

A full description of the quality of service needs of an application and of the network traffic generated, together with characterization of the traffic itself, can be complex—surely too complex to be represented by a simple number 0 through 7. The pragmatic aim of traffic classification is to simplify requirements radically to preserve the high-speed, low-cost handling characteristic of Bridges. At the margin, potential bandwidth efficiency is traded for simplicity—historically a good decision in the LAN.

The following list of traffic types, each of which can benefit from simple segregation from the others, seems to command widespread support:

a) Network Control—characterized by a "must get there" requirement to maintain and support the network infrastructure.
b) "Voice"—characterized by less than 10 ms delay, and hence maximum jitter (one way transmission through the LAN infrastructure of a single campus).
c) "Video"—characterized by less than 100 ms delay.
d) Controlled Load—important business applications subject to some form of "admission control," be that pre-planning of the network requirement at one extreme to bandwidth reservation per flow at the time the flow is started at the other.
e) Excellent Effort—or "CEO's best effort," the best-effort type services that an information services organization would deliver to its most important customers.
f) Best Effort—LAN traffic as we know it today.
g) Background—bulk transfers and other activities that are permitted on the network but that should not impact the use of the network by other users and applications.

H.2.3 What are we managing?

User priorities and traffic classes facilitate management of

a) Latency, to support new applications;
b) Throughput, to meet service level agreements centered around bandwidth for types of traffic.

With distinct traffic classification and Bridge queuing, latency and bandwidth guarantees can be supported at higher levels of network loading. With few classes the focus is on meeting latency requirements—the bandwidth surplus required in a bursty data environment to guarantee sub-10 ms delays without distinct traffic classification is uneconomically large. As the number of traffic classes that can be used increases, the focus can shift to managing throughput.

The simple default queue servicing policy defined in this standard, strict priority, supports latency management. Active management of bandwidth sharing necessarily requires some Bridge management, to determine shares, etc.

H.2.4 Traffic type to traffic class mapping

Table H-14 describes a grouping of the traffic types introduced above as the number of Bridge queues increases. Each grouping of types is shown as {*Distinguishing type*, Type, Type, . . .}. The "distinguishing type" is not treated in any way differently in a Bridge, but is italicized here to illustrate, for any given number of queues, which traffic types have driven the allocation of types to classes.

Table H-14—Traffic type to traffic class mapping

Number of queues	Traffic types
1	{*Best Effort*, Excellent effort, Background, Voice, Controlled Load, Video, Network Control}
2	{*Best Effort*, Excellent effort, Background} {*Voice*, Controlled Load, Video, Network Control}
3	{*Best Effort*, Excellent effort, Background} {*Controlled Load*, Video} {*Voice*, Network Control}
4	{*Background*} {*Best Effort*, Excellent effort} {*Controlled Load*, Video} {*Voice*, Network Control}
5	{*Background*} {*Best Effort*, Excellent effort} {*Controlled Load*} {*Video*} {*Voice*, Network Control}
6	{*Background*} {*Best Effort* {*Excellent effort*} {*Controlled Load*} {*Video*} {*Voice*, Network Control}
7	{*Background*} {*Best Effort*} {*Excellent effort*} {*Controlled Load*} {*Video*} {*Voice*} {*Network Control*}

This grouping is proposed as the default user priority to traffic class mapping, with the correspondence of traffic types to user priority numbers as shown further below. It is designed to work with default queue handling. The remaining user priority value (only seven instead of eight traffic types have been discussed) is also assigned within the priority hierarchy.

The logic behind the step by step breaking out of traffic types as more classes become available runs as follows:

a) With a single queue, there are no choices, everything functions as with Bridges conformant to ISO/IEC 10038: 1993. If a distinguishing traffic type were to be identified, it would be Best Effort.
b) In order to support integrated services (voice, video, and data) in the presence of bursty best effort data, it is necessary to segregate all the time-critical traffic. In addition, further traffic that is to receive superior service and that is operating under admission control also needs to be separated from the uncontrolled traffic. The amount of priority traffic will be restricted by the need to support low latency (better than 10 ms one way for the local LAN infrastructure) for Voice, which becomes the defining type for the additional queue.
c) A further queue is best used to separate Controlled Load, i.e., well-behaved traffic with admission control or policy constraints, from the very latency sensitive Voice traffic. This allows rather more controlled load (and video) traffic than with two queues, while still allowing voice latencies to be met. However, all the traffic in the new queue will still have to meet interactive video latencies (if any such traffic is present), so there may still be artificially low limits on the throughput of well-behaved traffic.
d) Up to this point, additional queues have been used first to address latency issues, and next to recover throughput for well-behaved traffic with less onerous latency constraints. This has now been largely achieved for the major bandwidth users, and the focus can shift to distinguishing between the different types of uncontrolled bursty traffic, distinguishing between them on the basis of business importance. The first step is to deal with high throughput background traffic, which may cause TCP windows and timers for normal best effort (and excellent effort) traffic to slow, by segregating Background traffic.
e) Allocating the next queue to (interactive) Video, traffic with latency and jitter less than 100 ms, frees the controlled load traffic from the latency constraint, raising the achievable throughput.
f) The next queue is devoted to a little extra job security by separating business critical applications that are not sufficiently well behaved to have been classified as controlled load into their own Excellent Effort category.
g) Finally, Network Control can be segregated from Voice, though the benefits of this are unlikely to be felt with default queue handling, since network control is likely to be less delay sensitive than Voice (but probably more than background), but requires a higher delivery guarantee.
h) Eight queues are not shown, since only seven distinct types of traffic are shown. From the point of view of defining defaults at the present time it seems reasonable to allocate an additional priority around best effort to support bandwidth sharing management for bursty data applications.

Table H-15 shows the correspondence between traffic types and user_priority values used to select the defaults in Table 7-2, and defines a set of acronyms for the traffic types:

Table H-15—Traffic type acronyms

user_priority	Acronym	Traffic type
1	BK	Background
2	—	Spare
0 (Default)	BE	Best Effort
3	EE	Excellent Effort
4	CL	Controlled Load
5	VI	“Video,” < 100 ms latency and jitter
6	VO	“Voice,” < 10 ms latency and jitter
7	NC	Network Control

Table H-16 compresses the lists of traffic types for queues shown in Table H-14, using the acronyms defined in Table H-15, and shows just the defining traffic types for the given number of queues.

Table H-16—Defining traffic types

Number of queues	Defining traffic type							
1	BE							
2	BE				VO			
3	BE				CL		VO	
4	BK		BE		CL		VO	
5	BK		BE		CL	VI	VO	
6	BK		BE	EE	CL	VI	VO	
7	BK		BE	EE	CL	VI	VO	NC
8	BK	—	BE	EE	CL	VI	VO	NC

H.2.5 Traffic types and user priority values

The default user priority used for transmission by token ring stations is 0, as is the default assumed by ISO/IEC 10038: 1993-conformant Bridges for frames currently received from an Ethernet. Changing this default would result in some confusion, and likely some lack of interoperability. At the same time the default traffic type today is definitely Best Effort. The proposed solution to this dilemma is to continue to use zero both for default user priority and for best effort data transmission, and associate Background, the spare value, and Excellent Effort with user priority values of 1, 2, and 3, respectively. This means that the values 1 and 2 effectively communicate a lower priority than 0.