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

Sponsor

LAN MAN Standards Committee of the IEEE Computer Society

Approved 26 June 1997

IEEE Standards Board

Abstract: The medium access control (MAC) and physical characteristics for wireless local area networks (LANs) are specified in this standard, part of a series of standards for local and metropolitan area networks. The medium access control unit in this standard is designed to support physical layer units as they may be adopted dependent on the availability of spectrum. This standard contains three physical layer units: two radio units, both operating in the 2400–2500 MHz band, and one baseband infrared unit. One radio unit employs the frequency-hopping spread spectrum technique, and the other employs the direct sequence spread spectrum technique.
Keywords: ad hoc network, infrared, LAN, local area network, mobility, radio frequency, wireless

---

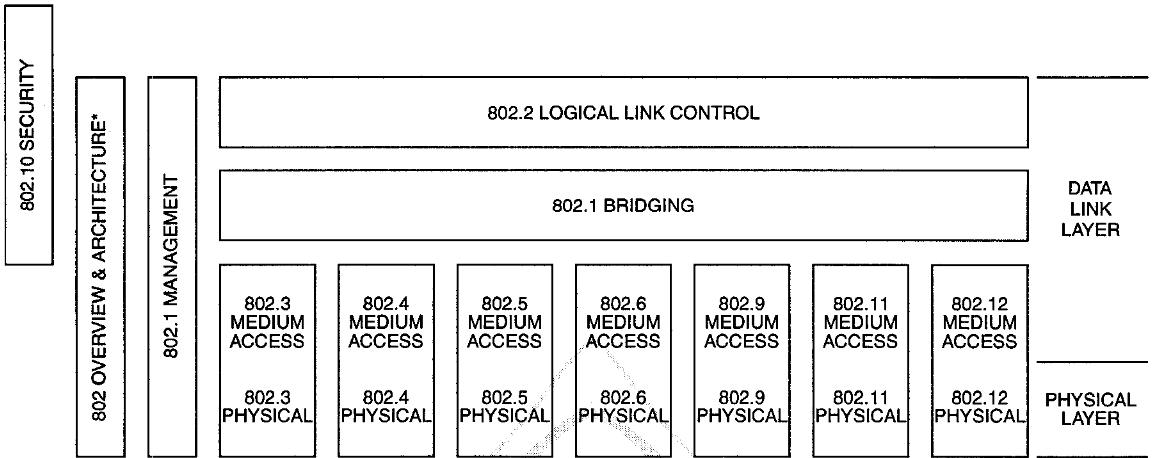
The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

ISBN 1-55937-935-9

Introduction

(This introduction is not part of IEEE Std 802.11-1997, but is included for information only.)

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/International Electrotechnical Commission (ISO/IEC) Open Systems Interconnection Basic Reference Model (ISO/IEC 7498-1: 1994). The access standards define several 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 access technologies are as follows:

• IEEE Std 802	*Overview and Architecture.* This standard provides an overview to the family of IEEE 802 Standards. This document forms part of the 802.1 scope of work.
• ANSI/IEEE Std 802.1B and 802.1k [ISO/IEC 15802-2]	*LAN/MAN Management.* Defines an Open Systems Interconnection (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 10038]	*MAC Bridging.* 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 802.2 [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 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*
- 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 DIS 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*

The reader of this standard is urged to become familiar with the complete family of standards.

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.

IEEE Std 802.11-1997

This standard defines the protocol and compatible interconnection of data communication equipment via the "air," radio or infrared, in a local area network (LAN) using the carrier sense multiple access protocol with collision avoidance (CSMA/CA) medium sharing mechanism. The medium access control (MAC) supports operation under control of an access point as well as between independent stations. The protocol includes authentication, association, and reassociation services, an optional encryption/decryption procedure, power management to reduce power consumption in mobile stations, and a point coordination function for time-bounded transfer of data. The standard includes the definition of the management information base (MIB) using Abstract Syntax Notation 1 (ASN.1) and specifies the MAC protocol in a formal way, using the Specification and Description Language (SDL).

The infrared implementation of the PHY supports 1 Mbit/s data rate with an optional 2 Mbit/s extension. The radio implementations of the PHY specify either a frequency-hopping spread spectrum (FHSS) supporting 1 Mbit/s and an optional 2 Mbit/s data rate or a direct sequence spread spectrum (DSSS) supporting both 1 and 2 Mbit/s data rates.

This standard contains state-of-the-art material. The area covered by this standard is undergoing evolution. Revisions are anticipated to this standard within the next few years to clarify existing material, to correct possible errors, and to incorporate new related material. Information on the current revision state of this and other IEEE 802 standards may be obtained from

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

Participants

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

Victor Hayes, *Chair* **Stuart J. Kerry** and **Chris Zegelin,** *Vice Chairs*
Bob O'Hara and **Greg Ennis,** *Chief Technical Editors*
George Fishel and **Carolyn L. Heide,** *Secretaries*

David Bagby, *MAC Group Chair* **C. Thomas Baumgartner,** *Infrared Chair and Editor*
Jan Boer, *Direct Sequence Chair* **Michael Fischer,** *State Diagram Editor*
Dean M. Kawaguchi, *PHY Group and FH Chair* **Mike Trompower,** *Direct Sequence Editor*

Jeff Abramowitz
Keith B. Amundsen
Phil Belanger
Manuel J. Betancor*
John Biddick
Simon Black
Alessandro M. Bolea
Pablo Brenner
Peter E. Chadwick
Naftali Chayat
Jonathon Y. Cheah
Hae Wook Choi
Wim Diepstraten
Robert J. Egan
Darwin Engwer
John Fakatselis
Matthew Fischer
Keith S. Furuya
Rich Gardner
Ian Gifford
Howard J. Hall
Bill Huhn
Donald C. Johnson
Mikio Kiyono
Joseph J. Kubler
Arthur Lashbrook
Francisco J. Lopez-Hernandez
Ronald Mahany
Bob Marshall
Jim McDonald
Akira Miura
Wayne D. Moyers
Ravi P. Nalamati
Mitsuji Okada
Al Petrick
Miri Ratner
James A. Renfro
William Roberts
Jon Walter Rosdahl
Michael Rothenberg
Chandos A. Rypinski
Anil K. Sanwalka
Roy Sebring
Glen Sherwood
Thomas Siep
Nathan Silberman
Don Sloan
Greg Smith
Marvin L. Sojka
Dave Strohschein
Bert Sullam
Mack Sullivan
Tom Tsoulogiannis
Jeanine Valadez
Sarosh Vesuna
Richard E. White
Donna A. Woznicki
Timothy M. Zimmerman
Johnny Zweig

Major contributions were received from the following individuals:

Robert Achatz
Ken Biba
Paul Eastman
Ed Geiger
Larry van der Jagt
Richard Lee
Kerry Lynn
Michael Masleid
John McKown
K. S. Natarajan
Jim Neally
Richard Ozer
Thomas Phinney
Leon S. Scaldeferri*
Jim Schuessler
François Y. Simon

*Deceased

The following persons were on the balloting committee:

Bernhard Albert
Jon M. Allingham
Jack S. Andresen
Kit Athul
Anthony L. Barnes
Robert T. Bell
Manuel J. Betancor
Simon Black
Alan L. Bridges
Graham Campbell
James T. Carlo
David E. Carlson
Peter E. Chadwick
Naftali Chayat
Alan J. Chwick
Ken Clements
Robert S. Crowder
Rifaat Dayem
Wim Diepstraten
Edward A. Dunlop
Sourav K. Dutta
Paul S. Eastman
Peter Ecclesine
Gregory Elkmann
John E. Emrich
Philip H. Enslow
Changxin Fan
Michael A. Fischer
Harvey A. Freeman
Robert J. Gagliano
Patrick S. Gonia
N. Douglas Grant
Govert M. Griffioen
Joseph L. Hammond
Victor Hayes
Kenneth C. Heck
Jan Hoogendoorn
Russell D. Housley
Walter K. Hurwitz
Richard J. Iliff
Tomoaki Ishifuji
Carlos Islas-Perez
Raj Jain
A. Kamerman
Peter M. Kelly
Yongbum Kim
Mikio Kiyono
Thaddeus Kobylarz
Stephen B. Kruger
Joseph J. Kubler
David J. Law
Jai Yong Lee
Jungtae Lee
Daniel E. Lewis
Randolph S. Little
Ming T. Liu
Joseph C. J. Loo
Donald C. Loughry
Robert D. Love
Ronald Mahany
Jim L. Mangin
Peter Martini
P. Takis Mathiopoulos
Steve Messenger
Bennett Meyer
Ann Miller
David S. Millman
Hiroshi Miyano
Stig Frode Mjolsnes
W. Melody Moh
John E. Montague
Wayne D. Moyers
Paul Nikolich
Ellis S. Nolley
Robert O'Hara
Donal O'Mahony
Roger Pandanda
Lalit Mohan Patnaik
Lucy W. Person
Thomas L. Phinney
Vikram Prabhu
Alberto Profumo
David L. Propp
Vikram Punj
Andris Putnins
Fernando Ramos
James W. Romlein
Floyd E. Ross
Michael Rothenberg
Christoph Ruland
Chandos A. Rypinski
Anil K. Sanwalka
Gregory D. Schumacher
Rich Seifert
Lee A. Sendelbach
Michael Serrone
Adarshpal S. Sethi
Donald A. Sheppard
Nathan Silberman
Joseph S. Skorupa
Michael A. Smith
Marvin L. Sojka
Efstathios D. Sykas
Geoffrey O. Thompson
Robert C. Tripi
Mike Trompower
David B. Turner
Mark-Rene Uchida
James Vorhies
Yun-Che Wang
Raymond P. Wenig
Earl J. Whitaker
David W. Wilson
Jerry A. Wyatt
Qian-Li Yang
Iwen Yao
Oren Yuen
Jonathan M. Zweig

IEEE

Contents

ANNEXES

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

1. Overview

1.1 Scope

The scope of this standard is to develop a medium access control (MAC) and physical layer (PHY) specification for wireless connectivity for fixed, portable, and moving stations within a local area.

1.2 Purpose

The purpose of this standard is to provide wireless connectivity to automatic machinery, equipment, or stations that require rapid deployment, which may be portable or hand-held, or which may be mounted on moving vehicles within a local area. This standard also offers regulatory bodies a means of standardizing access to one or more frequency bands for the purpose of local area communication.

Specifically, this standard

— Describes the functions and services required by an IEEE 802.11 compliant device to operate within ad hoc and infrastructure networks as well as the aspects of station mobility (transition) within those networks.
— Defines the MAC procedures to support the asynchronous MAC service data unit (MSDU) delivery services.
— Defines several PHY signaling techniques and interface functions that are controlled by the IEEE 802.11 MAC.
— Permits the operation of an IEEE 802.11 conformant device within a wireless local area network (LAN) that may coexist with multiple overlapping IEEE 802.11 wireless LANs.
— Describes the requirements and procedures to provide privacy of user information being transferred over the wireless medium (WM) and authentication of IEEE 802.11 conformant devices.

2. Normative references

The following standards contain provisions which, through references in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below.

IEEE Std 802-1990, IEEE Standards for Local and Metropolitan Area Networks: Overview and Architecture (ANSI).[1]

IEEE Std C95.1-1991, IEEE Standard Safety Levels with Respect to Human Exposure to Radio Frequency Electromagnetic Fields, 3 kHz to 300 GHz (ANSI).

ISO/IEC 7498-1: 1994, Information technology—Open Systems Interconnection—Basic Reference Model: The Basic Model.[2]

ISO/IEC 8802-2: 1994, Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 2: Logical link control.

ISO/IEC 8824-1: 1995, Information technology—Abstract Syntax Notation One (ASN.1): Specification of basic notation.

ISO/IEC 8824-1: 1995/Amd.1: 1996, Information technology—Abstract Syntax Notation One (ASN.1): Specification of basic notation, Amendment 1: Rules of extensibility.

ISO/IEC 8824-2: 1995, Information technology—Abstract Syntax Notation One (ASN.1): Information object specification.

ISO/IEC 8824-2: 1995/Amd.1: 1996, Information technology—Abstract Syntax Notation One (ASN.1): Information object specification, Amendment 1: Rules of extensibility.

ISO/IEC 8824-3: 1995, Information technology—Abstract Syntax Notation One (ASN.1): Constraint specification.

ISO/IEC 8824-4: 1995, Information technology—Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications.

ISO/IEC 8825-1: 1995, Information technology—ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).

ISO/IEC 8825-2: 1996, Information technology—ASN.1 encoding rules: Specification of Packed Encoding Rules (PER).

ISO/IEC 10039: 1991, Information technology—Open systems interconnection—Local area networks—Medium Access Control (MAC) service definition.

---

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

[2]ISO publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse. ISO publications are also available in the United States from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, 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.

ITU Radio Regulations, volumes 1–4.[3]

ITU-T Recommendation X.210 (11/93), Information technology—Open systems interconnection—Basic Reference Model: Conventions for the definition of OSI services (*common text with ISO/IEC*).

ITU-T Recommendation Z.100 (03/93), CCITT specification and description language (SDL).

ITU-T Recommendation Z.105 (03/95), SDL combined with ASN.1 (SDL/ASN.1).

3. Definitions

3.1 access control: The prevention of unauthorized usage of resources.

3.2 access point (AP): Any entity that has station functionality and provides access to the distribution services, via the wireless medium (WM) for associated stations.

3.3 ad hoc network: A network composed solely of stations within mutual communication range of each other via the wireless medium (WM). An ad hoc network is typically created in a spontaneous manner. The principal distinguishing characteristic of an ad hoc network is its limited temporal and spatial extent. These limitations allow the act of creating and dissolving the ad hoc network to be sufficiently straightforward and convenient so as to be achievable by nontechnical users of the network facilities; i.e., no specialized "technical skills" are required and little or no investment of time or additional resources is required beyond the stations that are to participate in the ad hoc network. The term *ad hoc* is often used as slang to refer to an independent basic service set (IBSS).

3.4 association: The service used to establish access point/station (AP/STA) mapping and enable STA invocation of the distribution system services (DSSs).

3.5 authentication: The service used to establish the identity of one station as a member of the set of stations authorized to associate with another station.

3.6 basic service area (BSA): The conceptual area within which members of a basic service set (BSS) may communicate.

3.7 basic service set (BSS): A set of stations controlled by a single coordination function (CF).

3.8 basic service set (BSS) basic rate set: The set of data transfer rates that all the stations in a BSS will be capable of using to receive frames from the wireless medium (WM). The BSS basic rate set data rates are preset for all stations in the BSS.

3.9 broadcast: The broadcast address is a unique multicast address that specifies all stations.

3.10 channel: An instance of medium use for the purpose of passing protocol data units (PDUs) that may be used simultaneously, in the same volume of space, with other instances of medium use (on other channels) by other instances of the same physical layer (PHY), with an acceptably low frame error ratio due to mutual

[3]ITU publications are available from the International Telecommunications Union, Sales Section, Place des Nations, CH-1211, Genève 20, Switzerland/Suisse. They are also available in the United States from the U.S. Department of Commerce, Technology Administration, National Technical Information Service (NTIS), Springfield, VA 22161, USA.

interference. Some PHYs provide only one channel, whereas others provide multiple channels. Examples of channel types are as shown in the following table:

Single channel	n-channel
Narrowband radio-frequency (RF) channel	Frequency division multiplexed channels
Baseband infrared	Direct sequence spread spectrum (DSSS) with code division multiple access

3.11 clear channel assessment (CCA) function: That logical function in the physical layer (PHY) that determines the current state of use of the wireless medium (WM).

3.12 confidentiality: The property of information that is not made available or disclosed to unauthorized individuals, entities, or processes.

3.13 coordination function: The logical function that determines when a station operating within a basic service set (BSS) is permitted to transmit and may be able to receive protocol data units (PDUs) via the wireless medium (WM). The coordination function within a BSS may have one point coordination function (PCF) and will have one distributed coordination function (DCF).

3.14 coordination function (CF)-Pollable: A station able (1) to respond to a CF Poll with a data frame, if such a frame is queued and able to be generated, and (2) to interpret acknowledgments in frames sent to or from the point coordinator.

3.15 deauthentication: The service that voids an existing authentication relationship.

3.16 directed address: *See:* **unicast frame.**

3.17 disassociation: The service that removes an existing association.

3.18 distributed coordination function (DCF): A class of coordination function where the same coordination function logic is active in every station in the basic service set (BSS) whenever the network is in operation.

3.19 distribution: The service that, by using association information, delivers medium access control (MAC) service data units (MSDUs) within the distribution system (DS).

3.20 distribution system (DS): A system used to interconnect a set of basic service sets (BSSs) and integrated local area networks (LANs) to create an extended service set (ESS).

3.21 distribution system medium (DSM): The medium or set of media used by a distribution system (DS) for communications between access points (APs) and portals of an extended service set (ESS).

3.22 distribution system service (DSS): The set of services provided by the distribution system (DS) that enable the medium access control (MAC) to transport MAC service data units (MSDUs) between stations that are not in direct communication with each other over a single instance of the wireless medium (WM). These services include transport of MSDUs between the access points (APs) of basic service sets (BSSs) within an extended service set (ESS), transport of MSDUs between portals and BSSs within an ESS, and transport of MSDUs between stations in the same BSS in cases where the MSDU has a multicast or broadcast destination address or where the destination is an individual address, but the station sending the MSDU chooses to involve DSS. DSSs are provided between pairs of IEEE 802.11 MACs.

3.23 extended rate set (ERS): The set of data transfer rates supported by a station (if any) beyond the extended service set (ESS) basic rate set. This set may include data transfer rates that will be defined in future physical layer (PHY) standards.

3.24 extended service area (ESA): The conceptual area within which members of an extended service set (ESS) may communicate. An ESA is larger than or equal to a basic service area (BSA) and may involve several basic service sets (BSSs) in overlapping, disjointed, or both configurations.

3.25 extended service set (ESS): A set of one or more interconnected basic service sets (BSSs) and integrated local area networks (LANs) that appear as a single BSS to the logical link control layer at any station associated with one of those BSSs.

3.26 Gaussian frequency shift keying (GFSK): A modulation scheme where the data is first filtered by a Gaussian filter in the baseband and then modulated with a simple frequency modulation.

3.27 independent basic service set (IBSS): A BSS that forms a self-contained network, and in which no access to a distribution system (DS) is available.

3.28 infrastructure: The infrastructure includes the distribution system medium (DSM), access point (AP), and portal entities. It is also the logical location of distribution and integration service functions of an extended service set (ESS). An infrastructure contains one or more APs and zero or more portals in addition to the distribution system (DS).

3.29 integration: The service that enables delivery of medium access control (MAC) service data units (MSDUs) between the distribution system (DS) and an existing, non-IEEE 802.11 local area network (via a portal).

3.30 medium access control (MAC) management protocol data unit (MMPDU): The unit of data exchanged between two peer MAC entities to implement the MAC management protocol.

3.31 medium access control (MAC) protocol data unit (MPDU): The unit of data exchanged between two peer MAC entities using the services of the physical layer (PHY).

3.32 medium access control (MAC) service data unit (MSDU): Information that is delivered as a unit between MAC service access points (SAPs).

3.33 minimally conformant network: An IEEE 802.11 network in which two stations in a single basic service area (BSA) are conformant with IEEE 802.11.

3.34 mobile station: A type of station that uses network communications while in motion.

3.35 multicast: A medium access control (MAC) address that has the group bit set. A multicast medium access control (MAC) service data unit (MSDU) is one with a multicast destination address. A multicast MPDU or control frame is one with a multicast receiver address.

3.36 network allocation vector (NAV): An indicator, maintained by each station, of time periods when transmission onto the wireless medium (WM) will not be initiated by the station whether or not the station's clear channel assessment (CCA) function senses the WM is busy.

3.37 point coordination function (PCF): A class of possible coordination functions where the coordination function logic is active in only one station in a basic service set (BSS) at any given time that the network is in operation.

3.38 portable station: A type of station that may be moved from location to location, but only uses network communications while at a fixed location.

3.39 portal: The logical point at which medium access control (MAC) service data units (MSDUs) from a non-IEEE 802.11 local area network (LAN) enter the distribution system (DS) of an extended service set (ESS).

3.40 privacy: The service used to prevent the content of messages from being read by other than the intended recipients.

3.41 reassociation: The service that enables an established association [between access point (AP) and station (STA)] to be transferred from one AP to another (or the same) AP.

3.42 station (STA): Any device that contains an IEEE 802.11 conformant medium access control (MAC) and physical layer (PHY) interface to the wireless medium (WM).

3.43 station basic rate: A data transfer rate belonging to the extended service set (ESS) basic rate set that is used by a station for specific transmissions. The station basic rate may change dynamically as frequently as each medium access control (MAC) protocol data unit (MPDU) transmission attempt, based on local considerations at that station.

3.44 station service (SS): The set of services that support transport of medium access control (MAC) service data units (MSDUs) between stations within a basic service set (BSS).

3.45 time unit (TU): A measurement of time equal to 1024 µs.

3.46 unauthorized disclosure: The process of making information available to unauthorized individuals, entities, or processes.

3.47 unauthorized resource use: Use of resource not consistent with the defined security policy.

3.48 unicast frame: A frame that is addressed to a single recipient, not a broadcast or multicast frame. *Syn:* directed address.

3.49 wired equivalent privacy (WEP): The optional cryptographic confidentiality algorithm specified by IEEE 802.11 used to provide data confidentiality that is subjectively equivalent to the confidentiality of a wired local area network (LAN) medium that does not employ cryptographic techniques to enhance privacy.

3.50 wireless medium (WM): The medium used to implement the transfer of protocol data units (PDUs) between peer physical layer (PHY) entities of a wireless local area network (LAN).

4. Abbreviations and acronyms

ACK	acknowledgment
AID	association identifier
AP	access point
ATIM	announcement traffic indication message
BSA	basic service area
BSS	basic service set
BSSID	basic service set identification
CCA	clear channel assessment
CF	contention free
CFP	contention-free period

CID	connection identifier
CP	contention period
CRC	cyclic redundancy code
CS	carrier sense
CTS	clear to send
CW	contention window
DA	destination address
DBPSK	differential binary phase shift keying
DCE	data communication equipment
DCF	distributed coordination function
DCLA	direct current level adjustment
DIFS	distributed (coordination function) interframe space
DLL	data link layer
Dp	desensitization
DQPSK	differential quadrature phase shift keying
DS	distribution system
DSAP	destination service access point
DSM	distribution system medium
DSS	distribution system service
DSSS	direct sequence spread spectrum
DTIM	delivery traffic indication message
ED	energy detection
EIFS	extended interframe space
EIRP	equivalent isotropically radiated power
ERS	extended rate set
ESA	extended service area
ESS	extended service set
FC	frame control
FCS	frame check sequence
FER	frame error ratio
FH	frequency hopping
FHSS	frequency-hopping spread spectrum
GFSK	Gaussian frequency shift keying
IBSS	independent basic service set
ICV	integrity check value
IDU	interface data unit
IFS	interframe space
IMp	intermodulation protection
IR	infrared (PHY)
ISM	industrial, scientific, and medical
IV	initialization vector
LLC	logical link control
LME	layer management entity
LRC	long retry count
lsb	least significant bit
MAC	medium access control
MDF	management-defined field
MIB	management information base
MLME	MAC sublayer management entity
MMPDU	MAC management protocol data unit
MPDU	MAC protocol data unit
msb	most significant bit
MSDU	MAC service data unit
NAV	network allocation vector

PC	point coordinator
PCF	point coordination function
PDU	protocol data unit
PHY	physical (layer)
PHY-SAP	physical layer service access point
PIFS	point (coordination function) interframe space
PLCP	physical layer convergence protocol
PLME	physical layer management entity
PMD	physical medium dependent
PMD-SAP	physical medium dependent service access point
PN	pseudonoise (PN code sequence)
PPDU	PLCP protocol data unit
ppm	parts per million
PPM	pulse position modulation
PRNG	pseudorandom number generator
PS	power save (mode)
PSDU	PLCP SDU
RA	receiver address
RF	radio frequency
RSSI	received signal strength indication
RTS	request to send
RX	receive or receiver
SA	source address
SAP	service access point
SDU	service data unit
SFD	start frame delimiter
SIFS	short interframe space
SLRC	station long retry count
SME	station management entity
SMT	station management
SQ	signal quality (PN code correlation strength)
SRC	short retry count
SS	station service
SSAP	source service access point
SSID	service set identifier
SSRC	station short retry count
STA	station
TA	transmitter address
TBTT	target beacon transmission time
TIM	traffic indication map
TSF	timing synchronization function
TU	time unit
TX	transmit or transmitter
TXE	transmit enable
WAN	wide area network
WDM	wireless distribution media
WDS	wireless distribution system
WEP	wired equivalent privacy
WM	wireless medium

5. General description

5.1 General description of the architecture

This subclause presents the concepts and terminology used within IEEE Std 802.11-1997 (referred to throughout the text as IEEE 802.11). Specific terms are defined in Clause 3. Illustrations convey key IEEE 802.11 concepts and the interrelationships of the architectural components. IEEE 802.11 uses an architecture to describe functional components of an IEEE 802.11 LAN. The architectural descriptions are not intended to represent any specific physical implementation of IEEE 802.11.

5.1.1 How wireless LAN systems are different

Wireless networks have fundamental characteristics that make them significantly different from traditional wired LANs.

5.1.1.1 Destination address does not equal destination location

In wired LANs, an address is equivalent to a physical location. This is implicitly assumed in the design of wired LANs. In IEEE 802.11, the addressable unit is a station (STA). The STA is a message destination, but not (in general) a fixed location.

5.1.1.2 The media impact the design

The physical layers used in IEEE 802.11 are fundamentally different from wired media. Thus IEEE 802.11 PHYs

a) Use a medium that has neither absolute nor readily observable boundaries outside of which stations with conformant PHY transceivers are known to be unable to receive network frames.
b) Are unprotected from outside signals.
c) Communicate over a medium significantly less reliable than wired PHYs.
d) Have dynamic topologies.
e) Lack full connectivity, and therefore the assumption normally made that every STA can hear every other STA is invalid (that is, STAs may be "hidden" from each other).
f) Have time-varying and asymmetric propagation properties.

Because of limitations on wireless PHY ranges, wireless LANs intended to cover reasonable geographic distances may be built from basic coverage building blocks.

5.1.1.3 The impact of handling mobile stations

One of the requirements of IEEE 802.11 is to handle *mobile* as well as *portable* stations. A *portable* station is one that is moved from location to location, but is only used while at a fixed location. *Mobile* stations actually access the LAN while in motion.

For technical reasons, it is not sufficient to handle only portable stations. Propagation effects blur the distinction between portable and mobile stations; stationary stations often appear to be mobile due to propagation effects.

Another aspect of mobile stations is that they may often be battery powered. Hence power management is an important consideration. For example, it cannot be presumed that a station's receiver will always be powered on.

5.1.1.4 Interaction with other IEEE 802 layers

IEEE 802.11 is required to appear to higher layers [logical link control (LLC)] as a current style 802 LAN. This requires that the IEEE 802.11 network handle station mobility within the MAC sublayer. To meet reliability assumptions (that LLC makes about lower layers), it is necessary for IEEE 802.11 to incorporate functionality that is untraditional for MAC sublayers.

5.2 Components of the IEEE 802.11 architecture

The IEEE 802.11 architecture consists of several components that interact to provide a wireless LAN that supports station mobility transparently to upper layers.

The *basic service set* (BSS) is the basic building block of an IEEE 802.11 LAN. Figure 1 shows two BSSs, each of which has two stations that are members of the BSS.

It is useful to think of the ovals used to depict a BSS as the coverage area within which the member stations of the BSS may remain in communication. (The concept of area, while not precise, is often good enough.) If a station moves out of its BSS, it can no longer directly communicate with other members of the BSS.

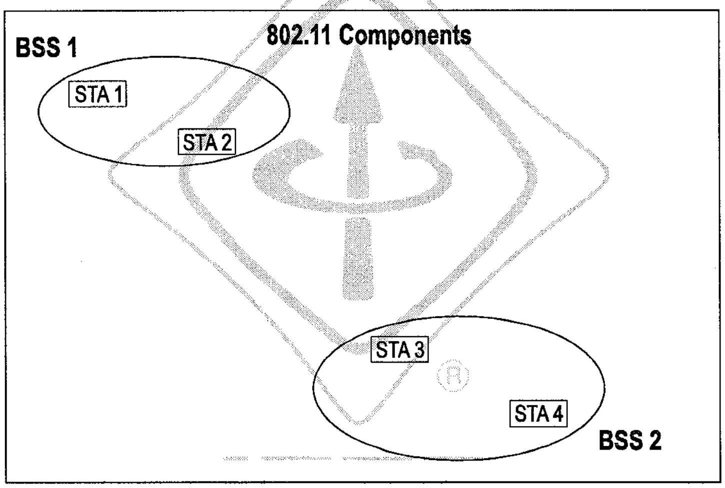


Figure 1—Basic service sets

5.2.1 The independent BSS as an ad hoc network

The independent BSS (IBSS) is the most basic type of IEEE 802.11 LAN. A minimum IEEE 802.11 LAN may consist of only two stations.

Figure 1 shows two IBSSs. This mode of operation is possible when IEEE 802.11 stations are able to communicate directly. Because this type of IEEE 802.11 LAN is often formed without pre-planning, for only as long as the LAN is needed, this type of operation is often referred to as an *ad hoc network*.

5.2.1.1 STA to BSS association is dynamic

The association between a STA and a BSS is dynamic (STAs turn on, turn off, come within range, and go out of range). To become a member of an infrastructure BSS, a station shall become "associated." These associations are dynamic and involve the use of the distribution system service (DSS), which is described later.

5.2.2 Distribution system concepts

PHY limitations determine the direct station-to-station distance that may be supported. For some networks this distance is sufficient; for other networks, increased coverage is required.

Instead of existing independently, a BSS may also form a component of an extended form of network that is built with multiple BSSs. The architectural component used to interconnect BSSs is the *distribution system* (DS).

IEEE 802.11 logically separates the wireless medium (WM) from the distribution system medium (DSM). Each logical medium is used for different purposes, by a different component of the architecture. The IEEE 802.11 definitions neither preclude, nor demand, that the multiple media be either the same or different.

Recognizing that the multiple media are *logically* different is key to understanding the flexibility of the architecture. The IEEE 802.11 LAN architecture is specified independently of the physical characteristics of any specific implementation.

The DS enables mobile device support by providing the logical services necessary to handle address to destination mapping and seamless integration of multiple BSSs.

An *access point* (AP) is a STA that provides access to the DS by providing DS services in addition to acting as a STA.

Figure 2 adds the DS and AP components to the IEEE 802.11 architecture picture.

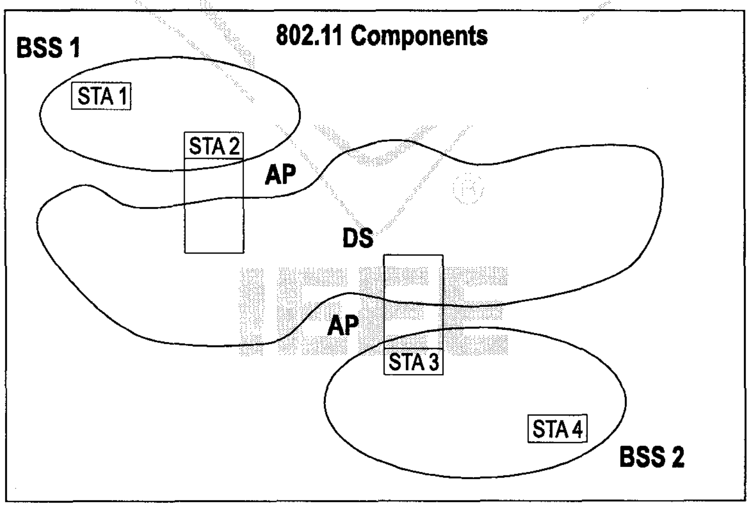


Figure 2—Distribution systems and access points

Data move between a BSS and the DS via an AP. Note that all APs are also STAs; thus they are addressable entities. The addresses used by an AP for communication on the WM and on the DSM are not necessarily the same.

5.2.2.1 Extended service set (ESS): The large coverage network

The DS and BSSs allow IEEE 802.11 to create a wireless network of arbitrary size and complexity. IEEE 802.11 refers to this type of network as the *extended service set* (ESS) network.

The key concept is that the ESS network appears the same to an LLC layer as an IBSS network. Stations within an ESS may communicate and mobile stations may move from one BSS to another (within the same ESS) transparently to LLC.

Nothing is assumed by IEEE 802.11 about the relative physical locations of the BSSs in Figure 3.

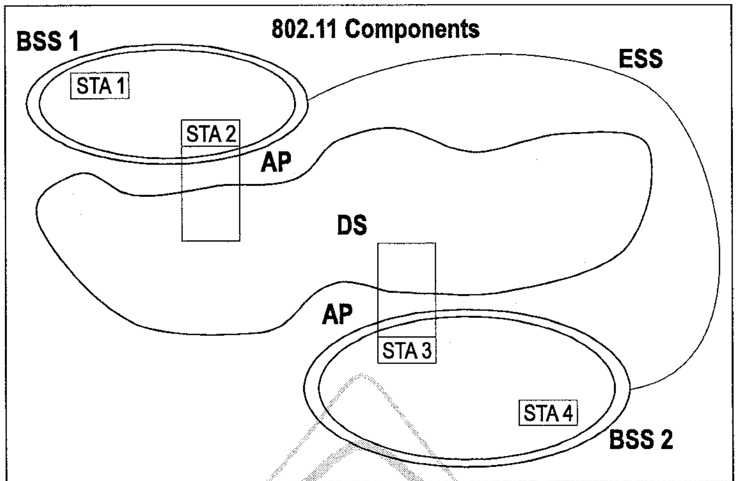


Figure 3—Extended service set

All of the following are possible:

a) The BSSs may partially overlap. This is commonly used to arrange contiguous coverage within a physical volume.
b) The BSSs could be physically disjointed. Logically there is no limit to the distance between BSSs.
c) The BSSs may be physically collocated. This may be done to provide redundancy.
d) One (or more) IBSS or ESS networks may be physically present in the same space as one (or more) ESS networks. This may arise for a number of reasons. Two of the most common are when an ad hoc network is operating in a location that also has an ESS network, and when physically overlapping IEEE 802.11 networks have been set up by different organizations.

5.2.3 Area concepts

For wireless PHYs, well-defined coverage areas simply do not exist. Propagation characteristics are dynamic and unpredictable. Small changes in position or direction may result in dramatic differences in signal strength. Similar effects occur whether a station is stationary or mobile (as moving objects may impact station-to-station propagation).

Figure 4 shows a signal strength map for a simple square room with a standard metal desk and an open doorway. Figure 4 is a static snapshot; the propagation patterns change dynamically as stations and objects in the environment move. In Figure 4 the dark (solid) blocks in the lower left are a metal desk and there is a doorway at the top right of the figure. The figure indicates relative differences in field strength with different intensities and indicates the variability of field strength even in a static environment.

While the architecture diagrams show sharp boundaries for BSSs, this is an artifact of the pictorial representation, not a physical reality. Since dynamic three-dimensional field strength pictures are difficult to draw, well-defined shapes are used by IEEE 802.11 architectural diagrams to represent the coverage of a BSS.

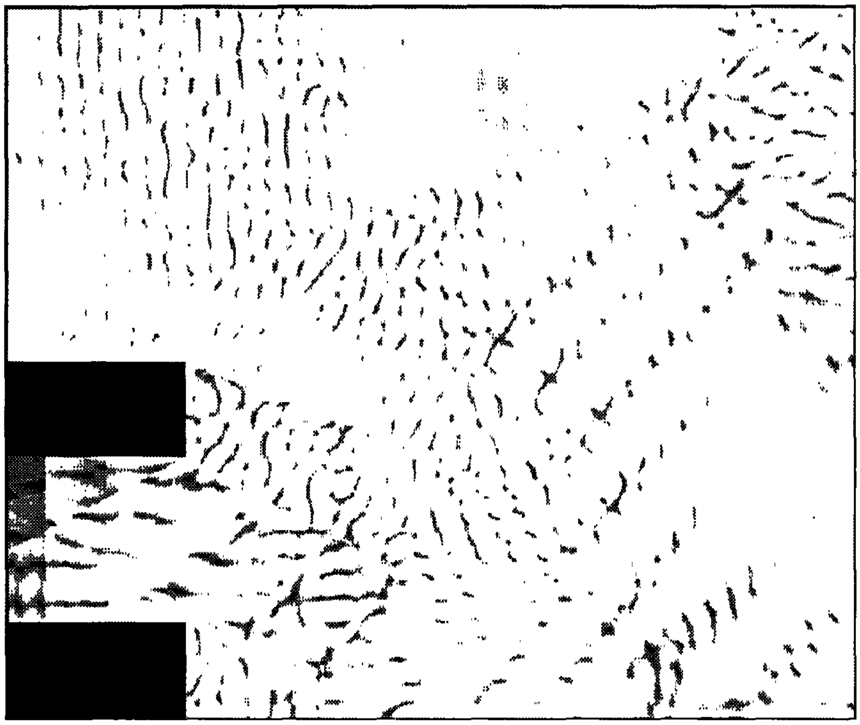


Figure 4—A representative signal intensity map

Further description difficulties arise when attempting to describe collocated coverage areas. Consider Figure 5, in which STA 6 could belong to BSS 2 or BSS 3.

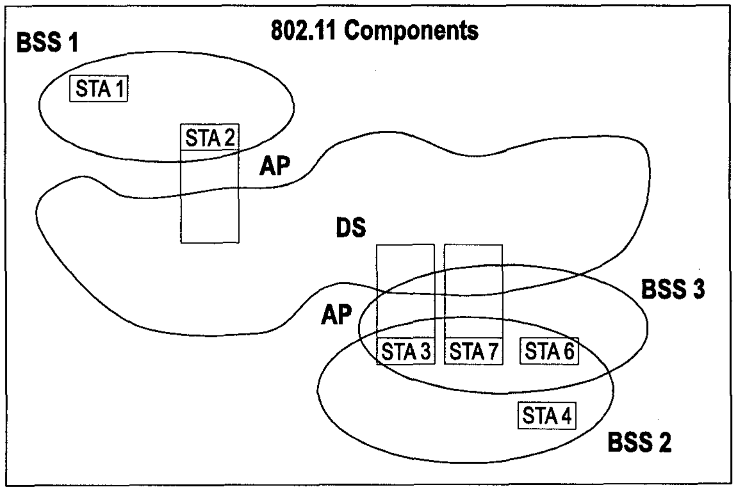


Figure 5—Collocated coverage areas

While the concept of sets of stations is correct, it is often convenient to talk about areas. For many topics the concept of area is sufficient. *Volume* is a more precise term than area, though still not technically correct. For historical reasons and convenience, this standard uses the common term *area*.

5.2.4 Integration with wired LANs

To integrate the IEEE 802.11 architecture with a traditional wired LAN, a final *logical* architectural component is introduced—a *portal.*

A portal is the logical point at which MSDUs from an integrated non-IEEE 802.11 LAN enter the IEEE 802.11 DS. For example, a portal is shown in Figure 6 connecting to a wired 802 LAN.

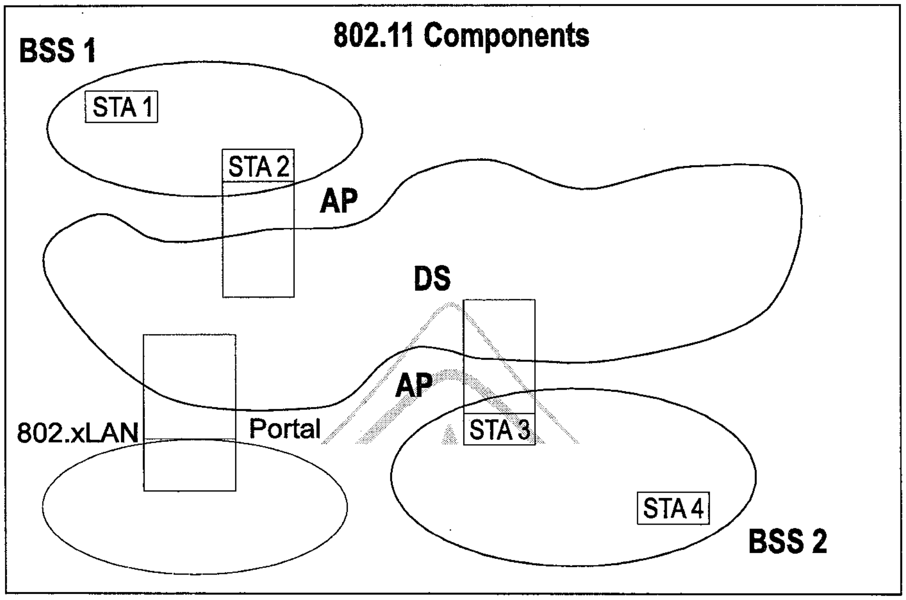


Figure 6—Connecting to other IEEE 802 LANs

All data from non-IEEE 802.11 LANs enter the 802.11 architecture via a portal. The portal provides logical integration between the IEEE 802.11 architecture and existing wired LANs. It is possible for one device to offer both the functions of an AP and a portal; this could be the case when a DS is implemented from 802 LAN components.

In IEEE 802.11, the ESS architecture (APs and the DS) provides traffic segmentation and range extension. Logical connections between IEEE 802.11 and other LANs are via the portal. Portals connect between the DSM and the LAN medium that is to be integrated.

5.3 Logical service interfaces

The IEEE 802.11 architecture allows for the possibility that the DS may not be identical to an existing wired LAN. A DS may be created from many different technologies including current 802 wired LANs. IEEE 802.11 does not constrain the DS to be either data link or network layer based. Nor does IEEE 802.11 constrain a DS to be either centralized or distributed in nature.

IEEE 802.11 explicitly does not specify the details of DS implementations. Instead, IEEE 802.11 specifies *services*. The services are associated with different components of the architecture. There are two categories of IEEE 802.11 service—the station service (SS) and the distribution system service (DSS). Both categories of service are used by the IEEE 802.11 MAC sublayer.

The complete set of IEEE 802.11 architectural services are as follows:

a) Authentication
b) Association
c) Deauthentication
d) Disassociation
e) Distribution
f) Integration
g) Privacy
h) Reassociation
i) MSDU delivery

This set of services is divided into two groups: those that are part of every station, and those that are part of a DS.

5.3.1 Station service (SS)

The service provided by stations is known as the *station service.*

The SS is present in every IEEE 802.11 station (including APs, as APs include station functionality). The SS is specified for use by MAC sublayer entities. All conformant stations provide SS.

The SS is as follows:

a) Authentication
b) Deauthentication
c) Privacy
d) MSDU delivery

5.3.2 Distribution system service (DSS)

The service provided by the DS is known as the *distribution system service.*

These services are represented in the IEEE 802.11 architecture by arrows within the APs, indicating that the services are used to cross media and address space logical boundaries. This is the convenient place to show the services in the picture. The physical embodiment of various services may or may not be within a physical AP.

The DSSs are provided by the DS. They are accessed via a STA that also provides DSSs. A STA that is providing access to DSS is an AP.

The DSSs are as follows:

a) Association
b) Disassociation
c) Distribution
d) Integration
e) Reassociation

DSSs are specified for use by MAC sublayer entities.

Figure 7 combines the components from previous figures with both types of services to show the complete IEEE 802.11 architecture.

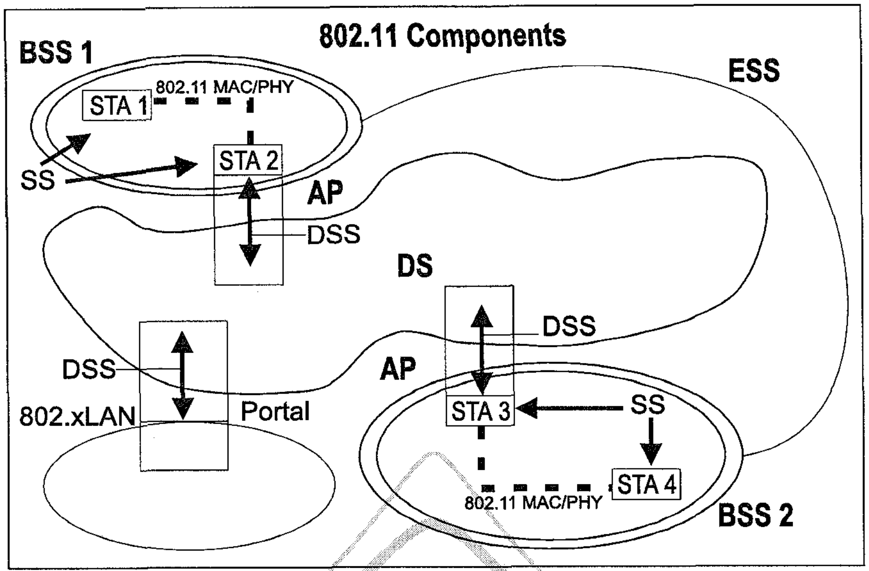


Figure 7—Complete IEEE 802.11 architecture

5.3.3 Multiple logical address spaces

Just as the IEEE 802.11 architecture allows for the possibility that the WM, DSM, and an integrated wired LAN may all be different physical media, it also allows for the possibility that each of these components may be operating within different address spaces. IEEE 802.11 only uses and specifies the use of the WM address space.

Each IEEE 802.11 PHY operates in a single medium—the WM. The IEEE 802.11 MAC operates in a single address space. MAC addresses are used on the WM in the IEEE 802.11 architecture. Therefore, it is unnecessary for the standard to explicitly specify that its addresses are "WM addresses." This is assumed throughout the IEEE 802.11 standard.

IEEE 802.11 has chosen to use the IEEE 802 48-bit address space (see 7.1.3.3.1). Thus IEEE 802.11 addresses are compatible with the address space used by the 802 LAN family.

The IEEE 802.11 choice of address space implies that for many instantiations of the IEEE 802.11 architecture, the wired LAN MAC address space and the IEEE 802.11 MAC address space may be the same. In those situations where a DS that uses MAC level 802 addressing is appropriate, all three of the logical address spaces used within a system could be identical. While this is a common case, it is not the only combination allowed by the architecture. The IEEE 802.11 architecture allows for all three logical address spaces to be distinct.

A multiple address space example is one where the DS implementation uses network layer addressing. In this case, the WM address space and the DS address space would be different.

The ability of the architecture to handle multiple logical media and address spaces is key to the ability of IEEE 802.11 to be independent of the DS implementation and to interface cleanly with network layer mobility approaches. The implementation of the DS is unspecified and is beyond the scope of this standard.

5.4 Overview of the services

There are nine services specified by IEEE 802.11. Six of the services are used to support MSDU delivery between STAs. Three of the services are used to control IEEE 802.11 LAN access and confidentiality.

This subclause presents the services, an overview of how each service is used, and a description of how the service relates to other services and the IEEE 802.11 architecture. The services are presented in an order designed to help build an understanding of the operation of an IEEE 802.11 ESS network. As a result, the SSs and DSSs are intermixed in order (rather than being grouped by category).

Each of the services is supported by one or more MAC frame types. Some of the services are supported by MAC management messages and some by MAC data messages. All of the messages gain access to the WM via the IEEE 802.11 MAC sublayer medium access method specified in Clause 9 of this standard.

The IEEE 802.11 MAC sublayer uses three types of messages—*data*, *management*, and *control* (see Clause 7). The data messages are handled via the MAC data service path.

MAC management messages are used to support the IEEE 802.11 services and are handled via the MAC management service data path.

MAC control messages are used to support the delivery of IEEE 802.11 data and management messages.

The examples here assume an ESS network environment. The differences between the ESS and the IBSS network environments are discussed separately in 5.6.

5.4.1 Distribution of messages within a DS

5.4.1.1 Distribution

This is the primary service used by IEEE 802.11 stations. It is conceptually invoked by every data message to or from an IEEE 802.11 STA operating in an ESS (when the frame is sent via the DS). Distribution is a DSS.

Refer to the ESS network in Figure 7 and consider a data message being sent from STA 1 to STA 4. The message is sent from STA 1 and received by STA 2 (the "input" AP). The AP gives the message to the distribution service of the DS.

It is the job of the distribution service to deliver the message within the DS in such a way that it arrives at the appropriate DS destination for the intended recipient.

In this example, the message is distributed to STA 3 (the "output" AP) and STA 3 accesses the WM to send the message to STA 4 (the intended destination).

How the message is distributed within the DS is not specified by IEEE 802.11. All IEEE 802.11 is required to do is to provide the DS with enough information for the DS to be able to determine the "output" point that corresponds to the desired recipient. The necessary information is provided to the DS by the three association related services (association, reassociation, and disassociation).

The previous example was a case where the AP that invoked the distribution service was different from the AP that received the distributed message. If the message had been intended for a station that was a member of the same BSS as the sending station, then the "input" and "output" APs for the message would have been the same.

In either example, the distribution service was logically invoked. Whether the message actually had to traverse the physical DSM or not is a DS implementation matter and is not specified by IEEE 802.11.

While IEEE 802.11 does not specify DS implementations, it does recognize and support the use of the WM as the DSM. This is specifically supported by the IEEE 802.11 frame formats. (Refer to Clause 7 for details.)

5.4.1.2 Integration

If the distribution service determines that the intended recipient of a message is a member of an integrated LAN, the "output" point of the DS would be a portal instead of an AP.

Messages that are distributed to a portal cause the DS to invoke the Integration function (conceptually after the distribution service). The Integration function is responsible for accomplishing whatever is needed to deliver a message from the DSM to the integrated LAN media (including any required media or address space translations). Integration is a DSS.

Messages received from an integrated LAN (via a portal) by the DS for an IEEE 802.11 STA will invoke the Integration function before the message is distributed by the distribution service.

The details of an Integration function are dependent on a specific DS implementation and are outside the scope of this standard.

5.4.2 Services that support the distribution service

The primary purpose of a MAC sublayer is to transfer MSDUs between MAC sublayer entities. The information required for the distribution service to operate is provided by the association services. Before a data message can be handled by the distribution service, a STA shall be "associated."

To understand the concept of association, it is necessary first to understand the concept of *mobility*.

5.4.2.1 Mobility types

There are three transition types of significance to this standard that describe the mobility of stations within a network:

a) ***No-transition:*** In this type, two subclasses that are usually indistinguishable are identified:
   1) Static—no motion.
   2) Local movement—movement within the PHY range of the communicating STAs [i.e., movement within a basic service area (BSA)].
b) ***BSS-transition:*** This type is defined as a station movement from one BSS in one ESS to another BSS within the same ESS.
c) ***ESS-transition:*** This type is defined as station movement from a BSS in one ESS to a BSS in a different ESS. This case is supported only in the sense that the STA may move. Maintenance of upper-layer connections cannot be guaranteed by IEEE 802.11; in fact, disruption of service is likely to occur.

The different association services support the different categories of mobility.

5.4.2.2 Association

To deliver a message within a DS, the distribution service needs to know which AP to access for the given IEEE 802.11 STA. This information is provided to the DS by the concept of *association*. Association is necessary, but not sufficient, to support BSS-transition mobility. Association is sufficient to support "no-transition" mobility. Association is a DSS.

Before a STA is allowed to send a data message via an AP, it shall first become associated with the AP. The act of becoming associated invokes the association service, which provides the STA to AP mapping to the DS. The DS uses this information to accomplish its message distribution service. How the information provided by the association service is stored and managed within the DS is not specified by this standard.

At any given instant, a STA may be associated with no more than one AP. This ensures that the DS may determine a unique answer to the question, "which AP is serving STA X?" Once an association is completed, a STA may make full use of a DS (via the AP) to communicate. Association is always initiated by the mobile STA, not the AP.

An AP may be associated with many STAs at one time.

A STA learns what APs are present and then requests to establish an association by invoking the association service. For the details of how a station learns about what APs are present, see 11.1.3 on scanning.

5.4.2.3 Reassociation

Association is sufficient for no-transition message delivery between IEEE 802.11 stations. Additional functionality is needed to support BSS-transition mobility. The additional required functionality is provided by the reassociation service. Reassociation is a DSS.

The reassociation service is invoked to "move" a current association from one AP to another. This keeps the DS informed of the current mapping between AP and STA as the station moves from BSS to BSS within an ESS. Reassociation also enables changing association attributes of an established association while the STA remains associated with the same AP. Reassociation is always initiated by the mobile STA.

5.4.2.4 Disassociation

The disassociation service is invoked whenever an existing association is to be terminated. Disassociation is a DSS.

In an ESS, this tells the DS to void existing association information. Attempts to send messages via the DS to a disassociated STA will be unsuccessful.

The disassociation service may be invoked by either party to an association (non-AP STA or AP). Disassociation is a notification, not a request. Disassociation cannot be refused by either party to the association.

APs may need to disassociate STAs to enable the AP to be removed from a network for service or for other reasons.

STAs shall attempt to disassociate whenever they leave a network. However, the MAC protocol does not depend on STAs invoking the disassociation service. (MAC management is designed to accommodate loss of an associated STA.)

5.4.3 Access and confidentiality control services

Two services are required for IEEE 802.11 to provide functionality equivalent to that which is inherent to wired LANs. The design of wired LANs assumes the physical attributes of wire. In particular, wired LAN design assumes the physically closed and controlled nature of wired media. The physically open medium nature of an IEEE 802.11 LAN violates those assumptions.

Two services are provided to bring the IEEE 802.11 functionality in line with wired LAN assumptions; authentication and privacy. *Authentication* is used instead of the wired media physical connection. *Privacy* is used to provide the confidential aspects of closed wired media.

5.4.3.1 Authentication

In wired LANs, physical security can be used to prevent unauthorized access. This is impractical in wireless LANs since they have a medium without precise bounds.

IEEE 802.11 provides the ability to control LAN access via the authentication service. This service is used by all stations to establish their identity to stations with which they will communicate. This is true for both ESS and IBSS networks. If a mutually acceptable level of authentication has not been established between two stations, an association shall not be established. Authentication is an SS.

IEEE 802.11 supports several authentication processes. The IEEE 802.11 authentication mechanism also allows expansion of the supported authentication schemes. IEEE 802.11 does not mandate the use of any particular authentication scheme.

IEEE 802.11 provides link level authentication between IEEE 802.11 stations. IEEE 802.11 does not provide either end-to-end (message origin to message destination) or user-to-user authentication. IEEE 802.11 authentication is used simply to bring the wireless link up to the assumed physical standards of a wired link. (This use of authentication is independent of any authentication process that may be used in higher levels of a network protocol stack.) If authentication other than that described here is desired, it is recommended that IEEE Std 802.10-1992 [B3][4] be implemented.

If desired, an IEEE 802.11 network may be operated using Open System authentication (see 8.1.1). This may violate implicit assumptions made by higher network layers. In an Open System, any station may become authenticated.

IEEE 802.11 also supports Shared Key authentication. Use of this authentication mechanism requires implementation of the WEP option (see 8.2). In a Shared Key authentication system, identity is demonstrated by knowledge of a shared, secret, WEP encryption key.

Management information base (MIB) functions are provided to support the standardized authentication schemes.

IEEE 802.11 requires mutually acceptable, successful, authentication.

A STA may be authenticated with many other STAs at any given instant.

5.4.3.1.1 Preauthentication

Because the authentication process could be time-consuming (depending on the authentication protocol in use), the authentication service can be invoked independently of the association service.

Preauthentication is typically done by a STA while it is already associated with an AP (with which it previously authenticated). IEEE 802.11 does not require that STAs preauthenticate with APs. However, authentication is required before an association can be established.

If the authentication is left until reassociation time, this may impact the speed with which a STA can reassociate between APs, limiting BSS-transition mobility performance. The use of preauthentication takes the authentication service overhead out of the time-critical reassociation process.

[4]The numbers in brackets correspond to those of the bibliography in Annex E.

5.4.3.2 Deauthentication

The deauthentication service is invoked whenever an existing authentication is to be terminated. Deauthentication is an SS.

In an ESS, since authentication is a prerequisite for association, the act of deauthentication shall cause the station to be disassociated. The deauthentication service may be invoked by either authenticated party (non-AP STA or AP). Deauthentication is not a request, it is a notification. Deauthentication shall not be refused by either party. When an AP sends a deauthentication notice to an associated STA, the association shall also be terminated.

5.4.3.3 Privacy

In a wired LAN, only those stations physically connected to the wire may hear LAN traffic. With a wireless shared medium, this is not the case. Any IEEE 802.11-compliant STA may hear all like-PHY IEEE 802.11 traffic that is within range. Thus the connection of a single wireless link (without privacy) to an existing wired LAN may seriously degrade the security level of the wired LAN.

To bring the functionality of the wireless LAN up to the level implicit in wired LAN design, IEEE 802.11 provides the ability to encrypt the contents of messages. This functionality is provided by the privacy service. Privacy is an SS.

IEEE 802.11 specifies an optional privacy algorithm [wired equivalent privacy (WEP)] that is designed to satisfy the goal of wired LAN "equivalent" privacy. The algorithm is not designed for ultimate security but rather to be "at least as secure as a wire." See Clause 8 for more details.

IEEE 802.11 uses the WEP mechanism (see Clause 8) to perform the actual encryption of messages. MIB functions are provided to support WEP.

Note that privacy may only be invoked for Data frames and some Authentication Management frames. All stations initially start "in the clear" in order to set up the authentication and privacy services.

The default privacy state for all IEEE 802.11 STAs is "in the clear." If the privacy service is not invoked, all messages shall be sent unencrypted. If this default is not acceptable to one party or the other, data frames shall not be successfully communicated between the LLC entities. Unencrypted data frames received at a station configured for mandatory privacy, as well as encrypted data frames using a key not available at the receiving station, are discarded without an indication to LLC (or without indication to distribution services in the case of "To DS" frames received at an AP). These frames are acknowledged on the WM [if received without frame check sequence (FCS) error] to avoid wasting WM bandwidth on retries.

5.5 Relationships between services

A STA keeps two state variables for each STA with which direct communication via the WM is needed:

— *Authentication state:* The values are unauthenticated and authenticated.
— *Association state:* The values are unassociated and associated.

These two variables create three local states for each remote STA:

— *State 1:* Initial start state, unauthenticated, unassociated.
— *State 2:* Authenticated, not associated.
— *State 3:* Authenticated and associated.

The relationships between these station state variables and the services are given by Figure 8.

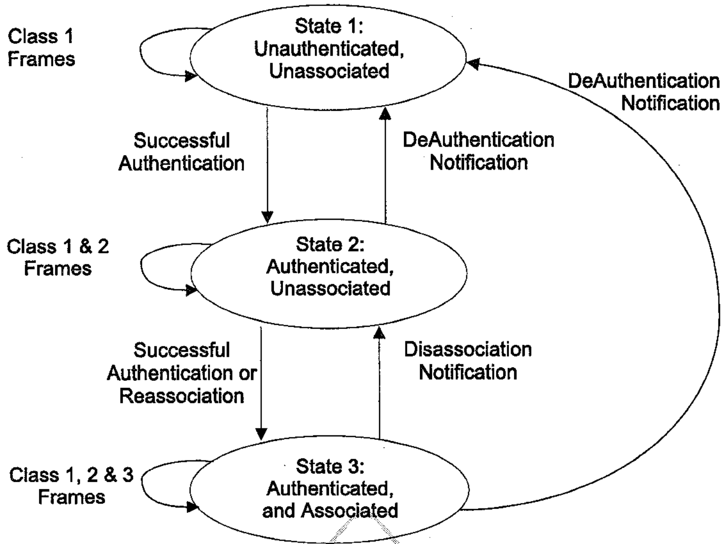


Figure 8—Relationship between state variables and services

The current state existing between the source and destination station determines the IEEE 802.11 frame types that may be exchanged between that pair of STAs (see Clause 7). The state of the sending STA given by Figure 8 is with respect to the intended receiving STA. The allowed frame types are grouped into classes and the classes correspond to the station state. In State 1, only Class 1 frames are allowed. In State 2, either Class 1 or Class 2 frames are allowed. In State 3, all frames are allowed (Classes 1, 2, and 3). The frame classes are defined as follows:

a) Class 1 frames (permitted from within States 1, 2, and 3):
   1) Control frames
      i) Request to send (RTS)
      ii) Clear to send (CTS)
      iii) Acknowledgment (ACK)
      iv) Contention-Free (CF)-End+ACK
      v) CF-End
   2) Management frames
      i) Probe request/response
      ii) Beacon
      iii) Authentication: Successful authentication enables a station to exchange Class 2 frames. Unsuccessful authentication leaves the STA in State 1.
      iv) Deauthentication: Deauthentication notification when in State 2 or State 3 changes the STA's state to State 1. The STA shall become authenticated again prior to sending Class 2 frames.
      v) Announcement traffic indication message (ATIM)
   3) Data frames
      i) Data: Data frames with frame control (FC) control bits "To DS" and "From DS" both false.
b) Class 2 frames (if and only if authenticated; allowed from within State 2 and State 3 only):
   1) Management frames:
      i) Association request/response
         — Successful association enables Class 3 frames.
         — Unsuccessful association leaves STA in State 2.
      ii) Reassociation request/response

— Successful reassociation enables Class 3 frames.
— Unsuccessful reassociation leaves the STA in State 2 (with respect to the STA that was sent the reassociation message). Reassociation frames shall only be sent if the sending STA is already associated in the same ESS.

iii) Disassociation
— Disassociation notification when in State 3 changes a Station's state to State 2. This station shall become associated again if it wishes to utilize the DS.

If STA A receives a Class 2 frame with a unicast address in the Address 1 field from STA B that is not authenticated with STA A, STA A shall send a deauthentication frame to STA B.

(The use of the word "receive" in this subclause refers to a frame that meets all of the filtering criteria specified in Clause 9.)

c) Class 3 frames (if and only if associated; allowed only from within State 3):
1) Data frames
— Data subtypes: Data frames allowed. That is, either the "To DS" or "From DS" FC control bits may be set to true to utilize DSSs.
2) Management frames
— Deauthentication: Deauthentication notification when in State 3 implies disassociation as well, changing the STA's state from 3 to 1. The station shall become authenticated again prior to another association.
3) Control frames
— PS-Poll

If STA A receives a Class 3 frame with a unicast address in the Address 1 field from STA B that is authenticated but not associated with STA A, STA A shall send a disassociation frame to STA B.

If STA A receives a Class 3 frame with a unicast address in the Address 1 field from STA B that is not authenticated with STA A, STA A shall send a deauthentication frame to STA B.

(The use of the word "receive" in this subclause refers to a frame that meets all of the filtering criteria specified in Clauses 8 and 9.)

5.6 Differences between ESS and IBSS LANs

In 5.2.1 the concept of the IBSS LAN was introduced. It was noted that an IBSS is often used to support an ad hoc network. In an IBSS network, a STA communicates directly with one or more other STAs.

Consider the full IEEE 802.11 architecture as shown in Figure 9.

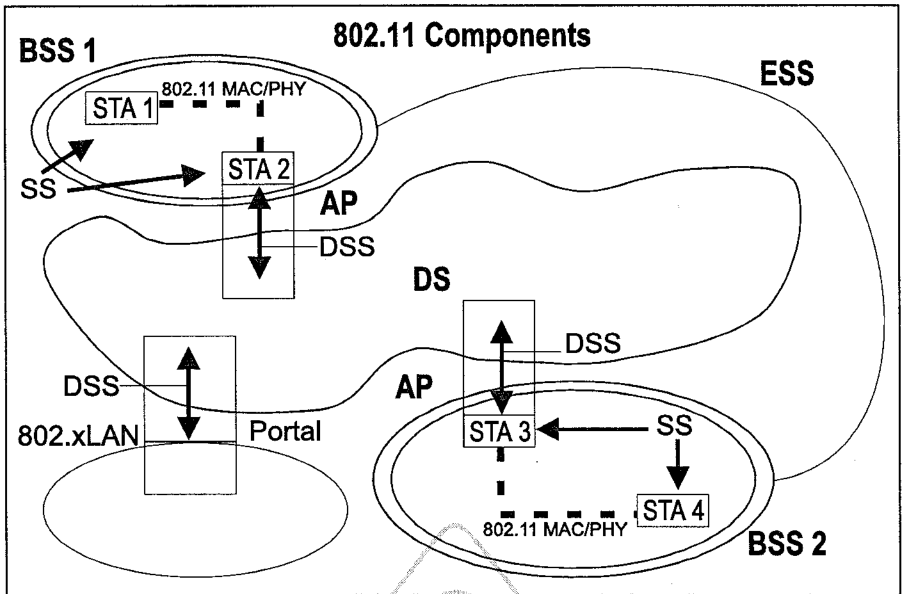


Figure 9—IEEE 802.11 architecture (again)

An IBSS consists of STAs that are directly connected. Thus there is (by definition) only one BSS. Further, since there is no physical DS, there cannot be a portal, an integrated wired LAN, or the DSSs. The logical picture reduces to Figure 10.

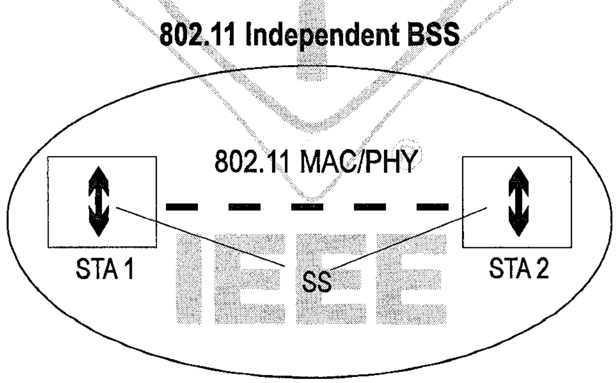


Figure 10—Logical architecture of an IBSS

Only the minimum two stations are shown in Figure 10. An IBSS may have an arbitrary number of members. In an IBSS, only Class 1 and Class 2 frames are allowed, since there is no DS in an IBSS.

The services that apply to an IBSS are the SSs.

5.7 Message information contents that support the services

Each service is supported by one or more IEEE 802.11 messages. Information items are given by name; for corresponding values, see Clause 7.

5.7.1 Data

For a STA to send data to another STA, it sends a data message, as shown below:

Data messages

- Message type: Data
- Message subtype: Data
- Information items:
  - IEEE source address of message
  - IEEE destination address of message
  - BSS ID
- Direction of message: From STA to STA

5.7.2 Association

For a STA to associate, the association service causes the following messages to occur:

Association request

- Message type: Management
- Message subtype: Association request
- Information items:
  - IEEE address of the STA initiating the association
  - IEEE address of the AP with which the initiating station will associate
  - ESS ID
- Direction of message: From STA to AP

Association response

- Message type: Management
- Message subtype: Association response
- Information items:
  - Result of the requested association. This is an item with values "successful" and "unsuccessful."
  - If the association is successful, the response shall include the association identifier (AID).
- Direction of message: From AP to STA

5.7.3 Reassociation

For a STA to reassociate, the reassociation service causes the following message to occur:

Reassociation request

- Message type: Management
- Message subtype: Reassociation request
- Information items:
  - IEEE address of the STA initiating the reassociation
  - IEEE address of the AP with which the initiating station will reassociate
  - IEEE address of the AP with which the initiating station is currently associated
  - ESSID
- Direction of message:
  - From STA to AP (The AP with which the STA is requesting reassociation)

The address of the current AP is included for efficiency. The inclusion of the current AP address facilitates MAC reassociation to be independent of the DS implementation.

Reassociation response

- Message type: Management
- Message subtype: Reassociation response
- Information items:
  - Result of the requested reassociation. This is an item with values "successful" and "unsuccessful."
  - If the reassociation is successful, the response shall include the AID.
- Direction of message: From AP to STA

5.7.4 Disassociation

For a STA to terminate an active association, the disassociation service causes the following message to occur:

Disassociation

- Message type: Management
- Message subtype: Disassociation
- Information items:
  - IEEE address of the station that is being disassociated. This shall be the broadcast address in the case of an AP disassociating with all associated stations.
  - IEEE address of the AP with which the station is currently associated.
- Direction of message: From STA to STA (e.g., STA to AP or AP to STA)

5.7.5 Privacy

For a STA to invoke the WEP privacy algorithm (as controlled by the related MIB attributes, see Clause 11), the privacy service causes MPDU encryption and sets the WEP frame header bit appropriately (see Clause 7).

5.7.6 Authentication

For a STA to authenticate with another STA, the authentication service causes one or more authentication management frames to be exchanged. The exact sequence of frames and their content is dependent on the authentication scheme invoked. For all authentication schemes, the authentication algorithm is identified within the management frame body.

In an IBSS environment, either station may be the initiating STA (STA 1). In an ESS environment, STA 1 is the mobile STA, and STA 2 is the AP.

Authentication (first frame of sequence)

- Message type: Management
- Message subtype: Authentication
- Information items:
  - Authentication algorithm identification
  - Station identity assertion
  - Authentication transaction sequence number
  - Authentication algorithm dependent information
- Direction of message: First frame in the transaction sequence is always from STA 1 to STA 2.

The first frame in an authentication sequence shall always be unencrypted.

Authentication (intermediate sequence frames)

— Message type: Management
— Message subtype: Authentication
— Information items:
  - Authentication algorithm identification
  - Authentication transaction sequence number
  - Authentication algorithm dependent information
— Direction of message:
  - Even transaction sequence numbers: From STA 2 to STA 1
  - Odd transaction sequence numbers: From STA 1 to STA 2

Authentication (final frame of sequence)

— Message type: Management
— Message subtype: Authentication
— Information items:
  - Authentication algorithm identification
  - Authentication transaction sequence number
  - Authentication algorithm dependent information
  - The result of the requested authentication. This is an item with values "successful" and "unsuccessful."
— Direction of message: STA 2 to STA 1

5.7.7 Deauthentication

For a STA to invalidate an active authentication, the following message is sent:

Deauthentication

— Message type: Management
— Message subtype: Deauthentication
— Information items:
  - IEEE address of the STA that is being deauthenticated
  - IEEE address of the STA with which the STA is currently authenticated
  - This shall be the broadcast address in the case of a STA deauthenticating all STAs currently authenticated.
— Direction of message: From STA to STA

5.8 Reference model

The standard presents the architectural view, emphasizing the separation of the system into two major parts: the MAC of the data link layer and the PHY. These layers are intended to correspond closely to the lowest layers of the ISO/IEC basic reference model of Open Systems Interconnection (OSI) (ISO/IEC 7498-1: 1994[5]). The layers and sublayers described in this standard are shown in Figure 11.

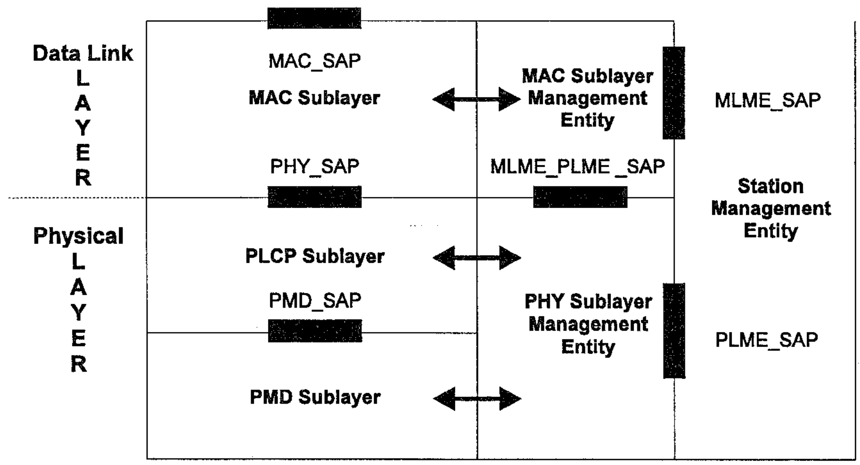


Figure 11—Portion of the ISO/IEC basic reference model covered in this standard

[5]Information on normative references can be found in Clause 2.

6. MAC service definition

6.1 Overview of MAC services

6.1.1 Asynchronous data service

This service provides peer LLC entities with the ability to exchange MAC service data units (MSDUs). To support this service, the local MAC uses the underlying PHY-level services to transport an MSDU to a peer MAC entity, where it will be delivered to the peer LLC. Such asynchronous MSDU transport is performed on a best-effort connectionless basis. There are no guarantees that the submitted MSDU will be delivered successfully. Broadcast and multicast transport is part of the asynchronous data service provided by the MAC. Due to the characteristics of the WM, broadcast and multicast MSDUs may experience a lower quality of service, compared to that of unicast MSDUs. All STAs will support the asynchronous data service. Because operation of certain functions of the MAC may cause reordering of some MSDUs, as discussed in more detail below, there are two service classes within the asynchronous data service. By selecting the desired service class, each LLC entity initiating the transfer of MSDUs is able to control whether MAC entities are or are not allowed to reorder those MSDUs.

6.1.2 Security services

Security services in IEEE 802.11 are provided by the authentication service and the wired equivalent privacy (WEP) mechanism. The scope of the security services provided is limited to station-to-station data exchange. The privacy service offered by an IEEE 802.11 WEP implementation is the encryption of the MSDU. For the purposes of this standard, WEP is viewed as a logical service located within the MAC sublayer as shown in the reference model, Figure 11. Actual implementations of the WEP service are transparent to the LLC or other layers above the MAC sublayer.

The security services provided by the WEP in IEEE 802.11 are as follows:

a) Confidentiality;
b) Authentication; and
c) Access control in conjunction with layer management.

During the authentication exchange, parties A and B exchange authentication information as described in Clause 8.

The MAC sublayer security services provided by WEP rely on information from non-Layer 2 management or system entities. Management entities communicate information to WEP through a set of MIB attributes.

6.1.3 MSDU ordering

The services provided by the MAC sublayer permit, and may in certain cases require, the reordering of MSDUs. The MAC does not intentionally reorder MSDUs except as may be necessary to improve the likelihood of successful delivery based on the current operational ("power management") mode of the designated recipient station(s). The sole effect of this reordering (if any), for the set of MSDUs received at the MAC service interface of any single station, is a change in the delivery order of broadcast and multicast MSDUs, relative to directed MSDUs, originating from a single source station address. If a higher-layer protocol using the asynchronous data service cannot tolerate this possible reordering, the optional StrictlyOrdered service class should be used. MSDUs transferred between any pair of stations using the StrictlyOrdered service class are not subject to the relative reordering that is possible when the ReorderableMulticast service class is used. However, the desire to receive MSDUs sent using the StrictlyOrdered service class at a station precludes simultaneous use of the MAC power management facilities at that station.

In order for the MAC to operate properly, the DS must meet the requirements of ISO/IEC 15802-1: 1995.

Subclause 9.8 specifies operational restrictions that ensure the appropriate ordering of MSDUs.

6.2 Detailed service specification

6.2.1 MAC data services

The IEEE 802.11 MAC supports the following service primitives as defined in ISO/IEC 8802-2: 1994:

— MA-UNITDATA.request
— MA-UNITDATA.indication
— MA-UNITDATA-STATUS.indication

The following three subclauses (6.2.1.1 through 6.2.1.3) give the LLC definitions of the primitives and specify parameter value restrictions imposed by IEEE 802.11.

6.2.1.1 MA-UNITDATA.request

6.2.1.1.1 Function

This primitive requests a transfer of an MSDU from a local LLC sublayer entity to a single peer LLC sublayer entity, or multiple peer LLC sublayer entities in the case of group addresses.

6.2.1.1.2 Semantics of the service primitive

The parameters of the primitive are as follows:

```
MA-UNITDATA.request     (
                        source address,
                        destination address,
                        routing information,
                        data,
                        priority,
                        service class
                        )
```

The source address (SA) parameter specifies an individual MAC sublayer address of the sublayer entity to which the MSDU is being transferred.

The destination address (DA) parameter specifies either an individual or a group MAC sublayer entity address.

The routing information parameter specifies the route desired for the data transfer (a null value indicates source routing is not to be used). For IEEE 802.11, the routing information parameter must be null.

The data parameter specifies the MSDU to be transmitted by the MAC sublayer entity. For IEEE 802.11, the length of the MSDU must be less than or equal to 2304 octets.

The priority parameter specifies the priority desired for the data unit transfer. IEEE 802.11 allows two values: Contention or ContentionFree.

The service class parameter specifies the service class desired for the data unit transfer. IEEE 802.11 allows two values: ReorderableMulticast or StrictlyOrdered.

6.2.1.1.3 When generated

This primitive is generated by the LLC sublayer entity whenever an MSDU is to be transferred to a peer LLC sublayer entity or entities.

6.2.1.1.4 Effect of receipt

The receipt of this primitive causes the MAC sublayer entity to append all MAC specified fields, including DA, SA, and all fields that are unique to IEEE 802.11, and pass the properly formatted frame to the lower layers for transfer to peer MAC sublayer entity or entities.

6.2.1.2 MA-UNITDATA.indication

6.2.1.2.1 Function

This primitive defines the transfer of an MSDU from the MAC sublayer entity to the LLC sublayer entity, or entities in the case of group addresses. In the absence of error, the contents of the data parameter are logically complete and unchanged relative to the data parameter in the associated MA-UNITDATA.request primitive.

6.2.1.2.2 Semantics of the service primitive

The primitive provides parameters as follows:

```
MA-UNITDATA.indication  (
                        source address,
                        destination address,
                        routing information,
                        data,
                        reception status,
                        priority,
                        service class
                        )
```

The SA parameter is an individual address as specified by the SA field of the incoming frame.

The DA parameter is either an individual or a group address as specified by the DA field of the incoming frame.

The routing information parameter specifies the route that was used for the data transfer. IEEE 802.11 will always set this field to null.

The data parameter specifies the MSDU as received by the local MAC entity.

The reception status parameter indicates the success or failure of the received frame for those frames that IEEE 802.11 reports via a MA-UNITDATA.indication. This MAC only reports “success” as all failures of reception are discarded without generating MA-UNITDATA.indication.

The priority parameter specifies the receive processing priority that was used for the data unit transfer. IEEE 802.11 allows two values: Contention or ContentionFree.

The service class parameter specifies the receive service class that was used for the data unit transfer. IEEE 802.11 allows two values: ReorderableMulticast or StrictlyOrdered.

6.2.1.2.3 When generated

The MA-UNITDATA.indication primitive is passed from the MAC sublayer entity to the LLC sublayer entity or entities to indicate the arrival of a frame at the local MAC sublayer entity. Frames are reported only if they are validly formatted at the MAC sublayer, received without error, received with valid (or null) WEP encryption, and their destination address designates the local MAC sublayer entity.

6.2.1.2.4 Effect of receipt

The effect of receipt of this primitive by the LLC sublayer is dependent on the validity and content of the frame.

6.2.1.3 MA-UNITDATA-STATUS.indication

6.2.1.3.1 Function

This primitive has local significance and provides the LLC sublayer with status information for the corresponding preceding MA-UNITDATA.request primitive.

6.2.1.3.2 Semantics of the service primitive

The primitive parameters are as follows:

```
MA-UNITDATA-STATUS.indication    (
                                 source address,
                                 destination address,
                                 transmission status,
                                 provided priority,
                                 provided service class
                                 )
```

The SA parameter is an individual MAC sublayer entity address as specified in the associated MA-UNITDATA.request primitive.

The DA parameter is either an individual or group MAC sublayer entity address as specified in the associated MA-UNITDATA.request primitive.

The transmission status parameter will be used to pass status information back to the local requesting LLC sublayer entity. IEEE 802.11 specifies the following values for transmission status:

a) Successful,
b) Undeliverable (for unacknowledged directed MSDUs when the aShortRetryMax or aLongRetryMax retry limit would otherwise be exceeded),
c) Excessive data length,
d) Non-null source routing,
e) Unsupported priority (for priorities other than Contention or ContentionFree),
f) Unsupported service class (for service classes other than ReorderableMulticast or StrictlyOrdered),
g) Unavailable priority (for ContentionFree when no point coordinator is available, in which case the MSDU is transmitted with a provided priority of Contention), and
h) Unavailable service class (for StrictlyOrdered service when the station's power management mode is other than "active").

i) Undeliverable (TransmitMSDUTimer reached aMaxTransmitMSDULifetime before successful delivery).
j) Undeliverable (no BSS available).
k) Undeliverable (the key referenced by aWEPDefaultKeyID or a specific key mapping is null).

The provided priority parameter specifies the priority that was used for the associated data unit transfer (Contention or ContentionFree).

The provided service class parameter specifies the class of service used for the associated data unit transfer (ReorderableMulticast or StrictlyOrdered).

6.2.1.3.3 When generated

The MA-UNITDATA-STATUS.indication primitive is passed from the MAC sublayer entity to the LLC sublayer entity to indicate the status of the service provided for the corresponding MA-UNITDATA.request primitive.

6.2.1.3.4 Effect of receipt

The effect of receipt of this primitive by the LLC sublayer is dependent upon the type of operation employed by the LLC sublayer entity.

7. Frame formats

The format of the MAC frames is specified in this clause. All stations shall be able to properly construct frames for transmission and decode frames upon reception, as specified in this clause.

7.1 MAC frame formats

Each frame consists of the following basic components:

a) A *MAC header*, which comprises frame control, duration, address, and sequence control information.
b) A variable length *frame body*, which contains information specific to the frame *type*.
c) A *frame check sequence* (FCS), which contains an IEEE 32-bit cyclic redundancy code (CRC).

7.1.1 Conventions

The MAC protocol data units (MPDUs) or frames in the MAC sublayer are described as a sequence of fields in specific order. Each figure in Clause 7 depicts the fields/subfields as they appear in the MAC frame and in the order in which they are passed to the physical layer convergence protocol (PLCP), from left to right.

In figures, all bits within fields are numbered, from 0 to k, where the length of the field is $k + 1$ bit. The octet boundaries within a field can be obtained by taking the bit-numbers of the field modulo 8. Octets within numeric fields that are longer than a single octet are depicted in increasing order of significance, from lowest numbered bit to highest numbered bit. The octets in fields longer than a single octet are sent to the PLCP in order from the octet containing the lowest numbered bits to the octet containing the highest numbered bits.

Any field containing a CRC is an exception to this convention and is transmitted commencing with the coefficient of the highest-order term.

MAC addresses are assigned as ordered sequences of bits. The Individual/Group bit is always transferred first and is bit 0 of the first octet.

Values specified in decimal are coded in natural binary unless otherwise stated. The values in Table 1 are in binary, with the bit assignments shown in the table. Values in other tables are shown in decimal notation.

Reserved fields and subfields are set to 0 upon transmission and are ignored on reception.

7.1.2 General frame format

The MAC frame format comprises a set of fields that occur in a fixed order in all frames. Figure 12 depicts the general MAC frame format. The fields Address 2, Address 3, Sequence Control, Address 4, and Frame Body are only present in certain frame types. Each field is defined in 7.1.3. The format of the each of the individual frame types is defined in 7.2.

Octets: 2	2	6	6	6	2	6	0 - 2312	4
Frame Control	Duration/ ID	Address 1	Address 2	Address 3	Sequence Control	Address 4	Frame Body	FCS

MAC Header (spans Frame Control through Address 4)

Figure 12—MAC frame format

7.1.3 Frame fields

7.1.3.1 Frame Control field

The Frame Control field consists of the following subfields: Protocol Version, Type, Subtype, To DS, From DS, More Fragments, Retry, Power Management, More Data, Wired Equivalent Privacy (WEP), and Order. The format of the frame control field is illustrated in Figure 13.

B0 B1	B2 B3	B4 B7	B8	B9	B10	B11	B12	B13	B14	B15
Protocol Version	Type	Subtype	To DS	From DS	More Frag	Retry	Pwr Mgt	More Data	WEP	Order
Bits: 2	2	4	1	1	1	1	1	1	1	1

Figure 13—Frame Control field

7.1.3.1.1 Protocol Version field

The Protocol Version field is 2 bits in length and is invariant in size and placement across all revisions of IEEE Std 802.11. For this standard, the value of the protocol version is 0. All other values are reserved. The revision level will be incremented only when a fundamental incompatibility exists between a new revision and the prior edition of the standard. A device that receives a frame with a higher revision level than it supports will discard the frame without indication to the sending station, or to LLC.

7.1.3.1.2 Type and Subtype fields

The Type field is 2 bits in length, and the Subtype field 4 bits in length. The Type and Subtype fields together identify the function of the frame. There are three frame types: control, data, and management. Each of the frame types have several defined subtypes. Table 1 defines the valid combinations of type and subtype.

Table 1— Valid type/subtype combinations

Type value b3 b2	Type description	Subtype value b7 b6 b5 b4	Subtype description
00	Management	0000	Association request
00	Management	0001	Association response
00	Management	0010	Reassociation request
00	Management	0011	Reassociation response
00	Management	0100	Probe request
00	Management	0101	Probe response
00	Management	0110–0111	Reserved
00	Management	1000	Beacon
00	Management	1001	Announcement traffic indication message (ATIM)
00	Management	1010	Disassociation

Table 1—Valid type/subtype combinations *(continued)*

Type value b3 b2	Type description	Subtype value b7 b6 b5 b4	Subtype description
00	Management	1011	Authentication
00	Management	1100	Deauthentication
00	Management	1101–1111	Reserved
01	Control	0000–1001	Reserved
01	Control	1010	Power Save (PS)-Poll
01	Control	1011	Request To Send (RTS)
01	Control	1100	Clear To Send (CTS)
01	Control	1101	Acknowledgment (ACK)
01	Control	1110	Contention Free (CF)-End
01	Control	1111	CF-End + CF-Ack
10	Data	0000	Data
10	Data	0001	Data + CF-Ack
10	Data	0010	Data + CF-Poll
10	Data	0011	Data + CF-Ack + CF-Poll
10	Data	0100	Null function (no data)
10	Data	0101	CF-Ack (no data)
10	Data	0110	CF-Poll (no data)
10	Data	0111	CF-Ack + CF Poll (no data)
10	Data	1000–1111	Reserved
11	Reserved	0000–1111	Reserved

7.1.3.1.3 To DS field

The To DS field is 1 bit in length and is set to 1 in data type frames destined for the DS. This includes all data type frames sent by STAs associated with an AP. The To DS field is set to 0 in all other frames.

7.1.3.1.4 From DS field

The From DS field is 1 bit in length and is set to 1 in data type frames exiting the DS. It is set to 0 in all other frames.

The permitted To/From DS bit combinations and their meanings are given in Table 2.

7.1.3.1.5 More Fragments field

The More Fragments field is 1 bit in length and is set to 1 in all data or management type frames that have another fragment of the current MSDU or current MMPDU to follow. It is set to 0 in all other frames.

Table 2—To/From DS combinations in data type frames

To/From DS values	Meaning
To DS = 0 From DS = 0	A data frame direct from one STA to another STA within the same IBSS, as well as all management and control type frames.
To DS = 1 From DS = 0	Data frame destined for the DS.
To DS = 0 From DS = 1	Data frame exiting the DS.
To DS = 1 From DS = 1	Wireless distribution system (WDS) frame being distributed from one AP to another AP.

7.1.3.1.6 Retry field

The Retry field is 1 bit in length and is set to 1 in any data or management type frame that is a retransmission of an earlier frame. It is set to 0 in all other frames. A receiving station uses this indication to aid in the process of eliminating duplicate frames.

7.1.3.1.7 Power Management field

The Power Management field is 1 bit in length and is used to indicate the power management mode of a STA. The value of this field remains constant in each frame from a particular STA within a frame exchange sequence defined in 9.7. The value indicates the mode in which the station will be after the successful completion of the frame exchange sequence.

A value of 1 indicates that the STA will be in power-save mode. A value of 0 indicates that the STA will be in active mode. This field is always set to 0 in frames transmitted by an AP.

7.1.3.1.8 More Data field

The More Data field is 1 bit in length and is used to indicate to a STA in power-save mode that more MSDUs, or MAC management PDUs (MMPDUs) are buffered for that STA at the AP. The More Data field is valid in directed data or management type frames transmitted by an AP to an STA in power-save mode. A value of 1 indicates that at least one additional buffered MSDU, or MMPDU, is present for the same STA.

The More Data field may be set to 1 in directed data type frames transmitted by a contention-free (CF)-Pollable STA to the point coordinator (PC) in response to a CF-Poll to indicate that the STA has at least one additional buffered MSDU available for transmission in response to a subsequent CF-Poll.

The More Data field is set to 0 in all other directed frames.

The More Data field is set to "1" in broadcast/multicast frames transmitted by the AP, when additional broadcast/multicast MSDUs, or MMPDUs, remain to be transmitted by the AP during this beacon interval. The More Data field is set to "0" in broadcast/multicast frames transmitted by the AP when no more broadcast/multicast MSDUs, or MMPDUs, remain to be transmitted by the AP during this beacon interval and in all broadcast/multicast frames transmitted by non-AP stations.

7.1.3.1.9 WEP field

The WEP field is 1 bit in length. It is set to 1 if the Frame Body field contains information that has been processed by the WEP algorithm. The WEP field is only set to 1 within frames of type Data and frames of type

Management, subtype Authentication. The WEP field is set to 0 in all other frames. When the WEP bit is set to 1, the Frame Body field is expanded as defined in 8.2.5.

7.1.3.1.10 Order field

The Order field is 1 bit in length and is set to 1 in any data type frame that contains an MSDU, or fragment thereof, which is being transferred using the StrictlyOrdered service class. This field is set to 0 in all other frames.

7.1.3.2 Duration/ID field

The Duration/ID field is 16 bits in length. The contents of the this field is as follows:

a) In control type frames of subtype Power Save (PS)-Poll, the Duration/ID field carries the association identity (AID) of the station that transmitted the frame in the 14 least significant bits (lsb), with the 2 most significant bits (msb) both set to 1. The value of the AID is in the range 1–2007.
b) In all other frames, the Duration/ID field contains a duration value as defined for each frame type in 7.2. For frames transmitted during the contention-free period (CFP), the duration field is set to 32 768.

Whenever the contents of the Duration/ID field are less than 32 768, the duration value is used to update the network allocation vector (NAV) according to the procedures defined in Clause 9.

The encoding of the Duration/ID field is given in Table 3.

Table 3—Duration/ID field encoding

Bit 15	Bit 14	Bits 13–0	Usage
0	0–32 767		Duration
1	0	0	Fixed value within frames transmitted during the CFP
1	0	1–16 383	Reserved
1	1	0	Reserved
1	1	1–2 007	AID in PS-Poll frames
1	1	2 008–16 383	Reserved

7.1.3.3 Address fields

There are four address fields in the MAC frame format. These fields are used to indicate the BSSID, source address, destination address, transmitting station address, and receiving station address. The usage of the four address fields in each frame type is indicated by the abbreviations BSSID, DA, SA, RA, and TA, indicating BSS identifier (BSSID), Destination Address, Source Address, Receiver Address, and Transmitter Address, respectively. Certain frames may not contain some of the address fields.

Certain address field usage is specified by the relative position of the Address field (1–4) within the MAC header, independent of the type of address present in that field. For example, receiver address matching is always performed on the contents of the Address 1 field in received frames, and the receiver address of CTS and ACK frames is always obtained from the Address 2 field in the corresponding RTS frame, or from the frame being acknowledged.

7.1.3.3.1 Address representation

Each Address field contains a 48-bit address as defined in 5.2 of IEEE Std 802-1990.

7.1.3.3.2 Address designation

A MAC sublayer address is one of two types:

a) *Individual address.* The address associated with a particular station on the network.
b) *Group address.* A multidestination address, associated with one or more stations on a given network. There are two kinds of group addresses:
   1) *Multicast-group address.* An address associated by higher-level convention with a group of logically related stations.
   2) *Broadcast address.* A distinguished, predefined multicast address that always denotes the set of all stations on a given LAN. All 1's in the Destination Address field are interpreted to be the broadcast address. This group is predefined for each communication medium to consist of all stations actively connected to that medium; it is used to broadcast to all the active stations on that medium. All stations are able to recognize the broadcast address. It is not necessary that a station be capable of generating the broadcast address.

The address space is also partitioned into locally administered and universal (globally administered) addresses. The nature of a body and the procedures by which it administers these universal (globally administered) addresses is beyond the scope of this standard. See IEEE Std 802-1990 for more information.

7.1.3.3.3 BSSID field

The BSSID is a 48-bit field of the same format as an IEEE 802 MAC address. This field uniquely identifies each BSS. The value of this field, in an infrastructure BSS, is the MAC address currently in use by the STA in the AP of the BSS.

The value of this field in an IBSS is a locally administered IEEE MAC address formed from a 46-bit random number generated according to the procedure defined in 11.1.3. The individual/group bit of the address is set to 0. The universal/local bit of the address is set to 1. This mechanism is used to provide a high probability of selecting an unique BSSID.

The value of all 1's is used to indicate the broadcast BSSID. A broadcast BSSID may only be used in the BSSID field of management frames of subtype probe request.

7.1.3.3.4 Destination Address (DA) field

The Destination Address (DA) field contains an IEEE MAC individual or group address that identifies the MAC entity or entities intended as the final recipient(s) of the MSDU (or fragment thereof) contained in the frame body field.

7.1.3.3.5 Source Address (SA) field

The Source Address (SA) field contains an IEEE MAC individual address that identifies the MAC entity from which the transfer of the MSDU (or fragment thereof) contained in the frame body field was initiated. The individual/group bit is always transmitted as a zero in the source address.

7.1.3.3.6 Receiver Address (RA) field

The receiver address (RA) field contains an IEEE MAC individual or group address that identifies the intended immediate recipient STA(s), on the wireless medium (WM), for the information contained in the frame body field.

7.1.3.3.7 Transmitter Address (TA) field

The transmitter address (TA) field contains an IEEE MAC individual address that identifies the STA that transmitted, onto the WM, the MPDU contained in the frame body field. The Individual/Group bit is always transmitted as a zero in the transmitter address.

7.1.3.4 Sequence Control field

The Sequence Control field is 16 bits in length and consists of two subfields, the Sequence Number and the Fragment Number. The format of the Sequence Control field is illustrated in Figure 14.

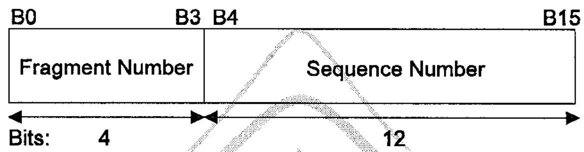


Figure 14—Sequence Control field

7.1.3.4.1 Sequence Number field

The Sequence Number is a 12-bit field indicating the sequence number of an MSDU, or MMPDU. Each MSDU or MMPDU transmitted by a STA is assigned a sequence number. Sequence numbers are assigned from a single modulo 4096 counter, starting at 0 (zero) and incrementing by 1 (one) for each MSDU or MMPDU. Each fragment of an MSDU or MMPDU contains the assigned sequence number. The sequence number remains constant in all retransmissions of an MSDU, MMPDU, or fragment thereof.

7.1.3.4.2 Fragment Number field

The Fragment Number is a 4-bit field indicating the number of each fragment of an MSDU or MMPDU. The fragment number is set to zero in the first or only fragment of an MSDU or MMPDU and is incremented by one for each successive fragment of that MSDU or MMPDU. The fragment number remains constant in all retransmissions of the fragment.

7.1.3.5 Frame Body field

The Frame Body is a variable length field and contains information specific to individual frame types and subtypes. The minimum frame body is zero octets. The maximum length frame body is defined by the maximum length (MSDU + ICV + IV); where ICV and IV are the WEP fields defined in 8.2.5.

7.1.3.6 FCS field

The FCS field is a 32-bit field containing a 32-bit CRC. The FCS is calculated over all the fields of the MAC header and the Frame Body field. These are referred to as the *calculation fields*.

The FCS is calculated using the following standard generator polynomial of degree 32:

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

The FCS is the one's complement of the sum (modulo 2) of the following:

a) The remainder of $x^k \times (x^{31} + x^{30} + x^{29} + \ldots + x^2 + x + 1)$ divided (modulo 2) by $G(x)$, where k is the number of bits in the calculation fields, and
b) The remainder after multiplication of the contents (treated as a polynomial) of the calculation fields by x^{32} and then division by $G(x)$.

The FCS field is transmitted commencing with the coefficient of the highest-order term.

As a typical implementation, at the transmitter, the initial remainder of the division is preset to all ones and is then modified by division of the calculation fields by the generator polynomial $G(x)$. The one's complement of this remainder is transmitted, with the high-order bit first, as the FCS field.

At the receiver, the initial remainder is preset to all ones and the serial incoming bits of the calculation fields and FCS, when divided by $G(x)$, results in the absence of transmission errors, in a unique nonzero remainder value. The unique remainder value is the polynomial:

$$x^{31} + x^{30} + x^{26} + x^{25} + x^{24} + x^{18} + x^{15} + x^{14} + x^{12} + x^{11} + x^{10} + x^8 + x^6 + x^5 + x^4 + x^3 + x + 1$$

7.2 Format of individual frame types

7.2.1 Control frames

In the following descriptions, "immediately previous" frame means a frame whose reception concluded within the prior short interframe space (SIFS) interval.

The subfields within the Frame Control field of control frames are set as illustrated in Figure 15.

B0 Protocol Version	Type	Subtype	To DS	From DS	More Frag	Retry	Pwr Mgt	More Data	WEP	Order B15
Protocol Version	Control	Subtype	0	0	0	0	Pwr Mgt	0	0	0
Bits: 2	2	4	1	1	1	1	1	1	1	1

Figure 15—Frame Control field subfield values within control frames

7.2.1.1 Request To Send (RTS) frame format

The frame format for the RTS frame is as defined in Figure 16.

Octets:	2	2	6	6	4
	Frame Control	Duration	RA	TA	FCS

MAC Header (Frame Control through TA)

Figure 16—RTS frame

The RA of the RTS frame is the address of the STA, on the WM, that is the intended immediate recipient of the pending directed data or management frame.

The TA is the address of the STA transmitting the RTS frame.

The duration value is the time, in microseconds, required to transmit the pending data or management frame, plus one CTS frame, plus one ACK frame, plus three SIFS intervals. If the calculated duration includes a fractional microsecond, that value is rounded up to the next higher integer.

7.2.1.2 Clear To Send (CTS) frame format

The frame format for the CTS frame is as defined in Figure 17.

Octets: 2	2	6	4
Frame Control	Duration	RA	FCS

MAC Header (Frame Control, Duration, RA)

Figure 17—CTS frame

The RA of the CTS frame is copied from the TA field of the immediately previous RTS frame to which the CTS is a response.

The duration value is the value obtained from the Duration field of the immediately previous RTS frame, minus the time, in microseconds, required to transmit the CTS frame and its SIFS interval. If the calculated duration includes a fractional microsecond, that value is rounded up to the next higher integer.

7.2.1.3 Acknowledgment (ACK) frame format

The frame format for the ACK frame is as defined in Figure 18.

Octets: 2	2	6	4
Frame Control	Duration	RA	FCS

MAC Header (Frame Control, Duration, RA)

Figure 18—ACK frame

The RA of the ACK frame is copied from the Address 2 field of the immediately previous directed data, management, or PS-Poll control frame.

If the More Fragment bit was set to 0 in the Frame Control field of the immediately previous directed data or management frame, the duration value is set to 0. If the More Fragment bit was set to 1 in the Frame Control field of the immediately previous directed data or management frame, the duration value is the value obtained from the Duration field of the immediately previous data or management frame, minus the time, in microseconds, required to transmit the ACK frame and its SIFS interval. If the calculated duration includes a fractional microsecond, that value is rounded up to the next higher integer.

7.2.1.4 Power-Save Poll (PS-Poll) frame format

The frame format for the PS-Poll frame is as defined in Figure 19.

Octets: 2	2	6	6	4
Frame Control	AID	BSS ID	TA	FCS

MAC Header

Figure 19—PS-Poll frame

The BSSID is the address of the STA contained in the AP. The TA is the address of the STA transmitting the frame. The AID is the value assigned to the STA transmitting the frame by the AP in the association response frame that established that STA's current association.

The AID value always has its 2 msb both set to 1. All STAs, upon receipt of a PS-Poll frame, update their NAV settings as appropriate under the coordination function rules using a duration value equal to the time, in microseconds, required to transmit one ACK frame plus one SIFS interval.

7.2.1.5 CF-End frame format

The frame format for the CF-End frame is as defined in Figure 20.

Octets: 2	2	6	6	4
Frame Control	Duration	RA	BSSID	FCS

MAC Header

Figure 20—CF-End frame

The BSSID is the address of the STA contained in the AP. The RA is the broadcast group address.

The Duration field is set to 0.

7.2.1.6 CF-End + CF-Ack frame format

The frame format for the contention-free-end acknowledge (CF-End + CF-Ack) frame is as defined in Figure 21.

Octets: 2	2	6	6	4
Frame Control	Duration	RA	BSSID	FCS

MAC Header

Figure 21—CF-End + CF-Ack Frame

The BSSID is the address of the STA contained in the AP. The RA is the broadcast group address.

The Duration field is set to 0.

7.2.2 Data frames

The frame format for a Data frame is independent of subtype and is as defined in Figure 22.

Octets: 2	2	6	6	6	2	6	0 - 2312	4
Frame Control	Duration / ID	Address 1	Address 2	Address 3	Sequence Control	Address 4	Frame Body	FCS

MAC Header (Frame Control through Address 4)

Figure 22—Data frame

The content of the Address fields of the data frame is dependent upon the values of the To DS and From DS bits and is defined in Table 4. Where the content of a field is shown as N/A, the field is omitted. Note that Address 1 always holds the receiver address of the intended receiver (or, in the case of multicast frames, receivers), and that Address 2 always holds the address of the station that is transmitting the frame.

Table 4 —Address field contents

To DS	From DS	Address 1	Address 2	Address 3	Address 4
0	0	DA	SA	BSSID	N/A
0	1	DA	BSSID	SA	N/A
1	0	BSSID	SA	DA	N/A
1	1	RA	TA	DA	SA

A station uses the contents of the Address 1 field to perform address matching for receive decisions. In cases where the Address 1 field contains a group address, the BSSID also is validated to ensure that the broadcast or multicast originated in the same BSS.

A station uses the contents of the Address 2 field to direct the acknowledgment if an acknowledgment is necessary.

The DA is the destination of the MSDU (or fragment thereof) in the frame body field.

The SA is the address of the MAC entity that initiated thc MSDU (or fragment thereof) in the frame body field.

The RA is the address of the STA contained in the AP in the wireless distribution system that is the next immediate intended recipient of the frame.

The TA is the address of the STA contained in the AP in the wireless distribution system that is transmitting the frame.

The BSSID of the Data frame is determined as follows:

a) If the station is an AP or is associated with an AP, the BSSID is the address currently in use by the STA contained in the AP.
b) If the station is a member of an IBSS, the BSSID is the BSSID of the IBSS.

The frame body consists of the MSDU or a fragment thereof, and a WEP IV and ICV (if and only if the WEP subfield in the frame control field is set to 1). The frame body is null (zero octets in length) in data frames of Subtype Null function (no data), CF-Ack (no data), CF-Poll (no data), and CF-Ack+CF-Poll (no data).

Within all data type frames sent during the CFP, the Duration field is set to the value 32 768. Within all data type frames sent during the contention period, the Duration field is set according to the following rules:

— If the Address 1 field contains a group address, the duration value is set to 0.
— If the More Fragments bit is set to 0 in the Frame Control field of a frame and the Address 1 field contains an individual address, the duration value is set to the time, in microseconds, required to transmit one ACK frame, plus one SIFS interval.
— If the More Fragments bit is set to 1 in the Frame Control field of a frame, and the Address 1 field contains an individual address, the duration value is set to the time, in microseconds, required to transmit the next fragment of this data frame, plus two ACK frames, plus three SIFS intervals.

The duration value calculation for the data frame is based on the rules in 9.6 that determine the data rate at which the control frames in the frame exchange sequence are transmitted. If the calculated duration includes a fractional microsecond, that value is rounded up to the next higher integer. All stations process Duration field values less than or equal to 32 767 from valid data frames to update their NAV settings as appropriate under the coordination function rules.

7.2.3 Management frames

The frame format for a Management frame is independent of frame subtype and is as defined in Figure 23.

Octets: 2	2	6	6	6	2	0 - 2312	4
Frame Control	Duration	DA	SA	BSS ID	Sequence Control	Frame Body	FCS

MAC Header (Frame Control through Sequence Control)

Figure 23—Management frame format

A STA uses the contents of the Address 1 field to perform the address matching for receive decisions. In the case where the Address 1 field contains a group address and the frame type is other than Beacon, the BSSID also is validated to ensure that the broadcast or multicast originated in the same BSS. If the frame type is Beacon, other address matching rules apply, as specified in 11.1.2.3.

The address fields for management frames do not vary by frame subtype.

The BSSID of the management frame is determined as follows:

a) If the station is an AP or is associated with an AP, the BSSID is the address currently in use by the STA contained in the AP.
b) If the station is a member of an IBSS, the BSSID is the BSSID of the IBSS.
c) In Management frames of subtype Probe Request, the BSSID is either a specific BSSID, or the broadcast BSSID as defined in the procedures specified in Clause 10.

The DA is the destination of the frame.

The SA is the address of the station transmitting the frame.

Within all management type frames sent during the CFP, the Duration field is set to the value 32 768. Within all management type frames sent during the contention period, the Duration field is set according to the following rules:

— If the DA field contains a group address, the duration value is set to 0.

— If the More Fragments bit is set to 0 in the Frame Control field of a frame and the DA contains an individual address, the duration value is set to the time, in microseconds, required to transmit one ACK frame, plus one SIFS interval.
— If the More Fragments bit is set to 1 in the Frame Control field of a frame, and the DA contains an individual address, the duration value is the time, in microseconds, required to transmit the next fragment of this management frame, plus two ACK frames, plus three SIFS intervals.

The duration value calculation for the management frame is based on the rules in 9.6 that determine the data rate at which the control frames in the frame exchange sequence are transmitted. If the calculated duration includes a fractional microsecond, that value is rounded up to the next higher integer. All stations process Duration field values less than or equal to 32 767 from valid management frames to update their NAV settings as appropriate under the coordination function rules.

The frame body consists of the fixed fields and information elements defined for each management frame subtype. All fixed fields and information elements are mandatory unless stated otherwise, and they can appear only in the specified order. Stations encountering an element type they do not understand ignore that element. Element type codes not explicitly defined in the standard are reserved, and do not appear in any frames.

7.2.3.1 Beacon frame format

The frame body of a management frame of subtype Beacon contains the information shown in Table 5.

Table 5—Beacon frame body

Order	Information	Note
1	Timestamp	
2	Beacon interval	
3	Capability information	
4	SSID	
5	Supported rates	
6	FH Parameter Set	1
7	DS Parameter Set	2
8	CF Parameter Set	3
9	IBSS Parameter Set	4
10	TIM	5

NOTES
1—The FH Parameter Set information element is only present within Beacon frames generated by STAs using frequency-hopping PHYs.
2—The DS Parameter Set information element is only present within Beacon frames generated by STAs using direct sequence PHYs.
3—The CF Parameter Set information element is only present within Beacon frames generated by APs supporting a PCF.
4—The IBSS Parameter Set information element is only present within Beacon frames generated by STAs in an IBSS.
5—The TIM information element is only present within Beacon frames generated by APs.

7.2.3.2 IBSS Announcement Traffic Indication Message (ATIM) frame format

The frame body of a management frame of subtype ATIM is null.

7.2.3.3 Disassociation frame format

The frame body of a management frame of subtype Disassociation contains the information shown in Table 6.

Table 6—Disassociation frame body

Order	Information
1	Reason code

7.2.3.4 Association Request frame format

The frame body of a management frame of subtype Association Request contains the information shown in Table 7.

Table 7 —Association Request frame body

Order	Information
1	Capability information
2	Listen interval
3	SSID
4	Supported rates

7.2.3.5 Association Response frame format

The frame body of a management frame of subtype Association Response contains the information shown in Table 8.

Table 8—Association Response frame body

Order	Information
1	Capability information
2	Status code
3	Association ID (AID)
4	Supported rates

7.2.3.6 Reassociation Request frame format

The frame body of a management frame of subtype Reassociation Request contains the information shown in Table 9.

Table 9—Reassociation Request frame body

Order	Information
1	Capability information
2	Listen interval
3	Current AP address
4	SSID
5	Supported rates

7.2.3.7 Reassociation Response frame format

The frame body of a management frame of subtype Reassociation Response contains the information shown in Table 10.

Table 10—Reassociation Response frame body

Order	Information
1	Capability information
2	Status code
3	Association ID (AID)
4	Supported rates

7.2.3.8 Probe Request frame format

The frame body of a management frame of subtype Probe Request contains the information shown in Table 11.

Table 11—Probe Request frame body

Order	Information
1	SSID
2	Supported rates

7.2.3.9 Probe Response frame format

The frame body of a management frame of subtype Probe Response contains the information shown in Table 12.

Table 12—Probe Response frame body

Order	Information	Note
1	Timestamp	
2	Beacon interval	
3	Capability information	
4	SSID	
5	Supported rates	
6	FH Parameter Set	1
7	DS Parameter Set	2
8	CF Parameter Set	3
9	IBSS Parameter Set	4

NOTES
1—The FH Parameter Set information element is only present within Probe Response frames generated by STAs using frequency-hopping PHYs.
2—The DS Parameter Set information element is only present within Probe Response frames generated by STAs using direct sequence PHYs.
3—The CF Parameter Set information element is only present within Probe Response frames generated by APs supporting a PCF.
4—The IBSS Parameter Set information element is only present within Probe Response frames generated by STAs in an IBSS.

7.2.3.10 Authentication frame format

The frame body of a management frame of subtype Authentication contains the information shown in Table 13.

Table 13—Authentication frame body

Order	Information	Note
1	Authentication algorithm number	
2	Authentication transaction sequence number	
3	Status code	1
4	Challenge text	2

NOTES
1—The status code information is reserved and set to 0 in certain Authentication frames as defined in Table 14.
2—The challenge text information is only present in certain Authentication frames as defined in Table 14.

Table 14—Presence of challenge text information

Authentication algorithm number	Authentication trans. sequence number	Status code	Challenge text
Open System	1	Reserved	Not present
Open System	2	Status	Not present
Shared Key	1	Reserved	Not present
Shared Key	2	Status	Present
Shared Key	3	Reserved	Present
Shared Key	4	Status	Not present

7.2.3.11 Deauthentication

The frame body of a management frame of subtype Deauthentication contains the information shown in Table 15.

Table 15—Deauthentication frame body

Order	Information	Note
1	Reason code	

7.3 Management frame body components

Within management frames, fixed-length mandatory frame body components are defined as fixed fields; variable length mandatory and all optional frame body components are defined as information elements.

7.3.1 Fixed fields

7.3.1.1 Authentication Algorithm Number field

The Authentication Algorithm Number field indicates a single authentication algorithm. The length of the Authentication Algorithm Number field is two octets. The Authentication Algorithm Number field is illustrated in Figure 24. The following values are defined for authentication algorithm number:

Authentication algorithm number = 0: Open System
Authentication algorithm number = 1: Shared Key
All other values of authentication number are reserved.

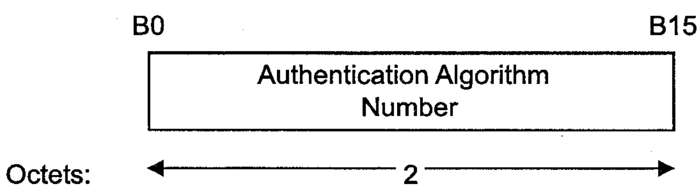


Figure 24—Authentication Algorithm Number fixed field

7.3.1.2 Authentication Transaction Sequence Number field

The Authentication Transaction Sequence Number field indicates the current state of progress through a multistep transaction. The length of the Authentication Transaction Sequence Number field is two octets. The Authentication Transaction Sequence Number field is illustrated in Figure 25.

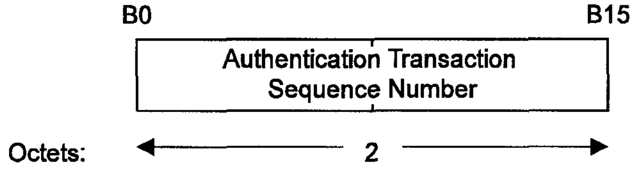


Figure 25—Authentication Transaction Sequence Number fixed field

7.3.1.3 Beacon Interval field

The Beacon Interval field represents the number of time units (TU) between target beacon transmission times (TBTTs). The length of the Beacon Interval field is two octets. The Beacon Interval field is illustrated in Figure 26.

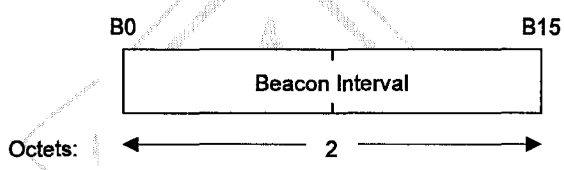


Figure 26—Beacon Interval fixed field

7.3.1.4 Capability Information field

The Capability Information field contains a number of subfields that are used to indicate requested or advertised capabilities. The length of the Capability Information field is two octets. The Capability Information field consists of the following subfields: ESS, IBSS, CF-Pollable, CF-Poll Request, and Privacy. The remaining part of the Capability Information field is reserved. The format of the Capability Information field is as illustrated in Figure 27.

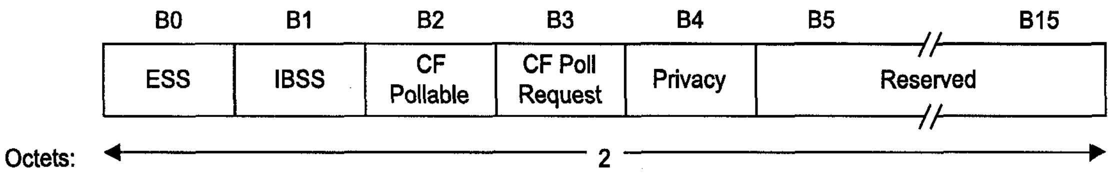


Figure 27—Capability Information fixed field

Each Capability Information subfield is interpreted only in the management frame subtypes for which the transmission rules are defined.

APs set the ESS subfield to 1 and the IBSS subfield to 0 within transmitted Beacon or Probe Response management frames. STAs within an IBSS set the ESS subfield to 0 and the IBSS subfield to 1 in transmitted Beacon or Probe Response management frames.

STAs set the CF-Pollable and CF-Poll Request subfields in Association and Reassociation Request management frames according to Table 16.

Table 16—STA usage of CF-Pollable and CF-Poll Request

CF-Pollable	CF-Poll request	Meaning
0	0	STA is not CF-Pollable
0	1	STA is CF-Pollable, not requesting to be placed on the CF-Polling list
1	0	STA is CF-Pollable, requesting to be placed on the CF-Polling list
1	1	STA is CF-Pollable, requesting never to be polled

APs set the CF-Pollable and CF-Poll Request subfields in Beacon, Probe Response, Association Response, and Reassociation Response management frames according to Table 17. An AP sets the CF-Pollable and CF-Poll Request subfield values in Association Response and Reassociation Response management frames equal to the values in the last Beacon or Probe Response frame that it transmitted.

Table 17—AP usage of CF-Pollable and CF-Poll Request

CF-Pollable	CF-Poll Request	Meaning
0	0	No point coordinator at AP
0	1	Point coordinator at AP for delivery only (no polling)
1	0	Point coordinator at AP for delivery and polling
1	1	Reserved

APs set the Privacy subfield to 1 within transmitted Beacon, Probe Response, Association Response, and Reassociation Response management frames if WEP encryption is required for all data type frames exchanged within the BSS. If WEP encryption is not required, the Privacy subfield is set to 0.

STAs within an IBSS set the Privacy subfield to 1 in transmitted Beacon or Probe Response management frames if WEP encryption is required for all data type frames exchanged within the IBSS. If WEP encryption is not required, the Privacy subfield is set to 0.

7.3.1.5 Current AP Address field

The Current AP Address field is the MAC address of the AP with which the station is currently associated. The length of the Current AP Address field is six octets. The Current AP Address field is illustrated in Figure 28.

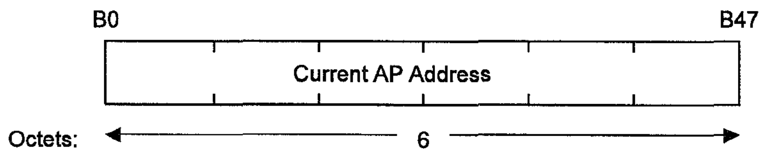


Figure 28—Current AP Address fixed field

7.3.1.6 Listen Interval field

The Listen Interval field is used to indicate to the AP how often an STA wakes to listen to Beacon management frames. The value of this parameter is the STA's aListenInterval MIB attribute and is expressed in units of Beacon Interval. The length of the Listen Interval field is two octets. The Listen Interval field is illustrated in Figure 29.

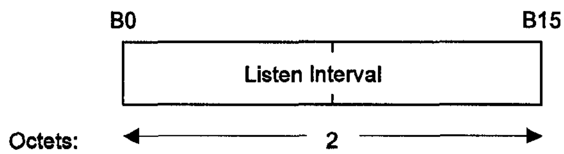


Figure 29—Listen Interval fixed field

An AP may use the Listen Interval information in determining the lifetime of frames that it buffers for an STA.

7.3.1.7 Reason Code field

This Reason Code field is used to indicate the reason that an unsolicited notification management frame of type Disassociation or Deauthentication was generated. The length of the Reason Code field is two octets. The Reason Code field is illustrated in Figure 30.

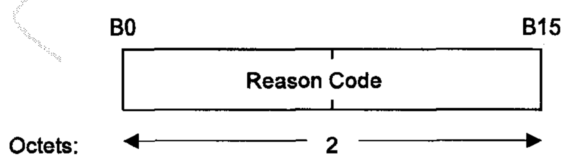


Figure 30—Reason Code fixed field

The reason codes are defined in Table 18.

Table 18—Reason codes

Reason code	Meaning
0	Reserved
1	Unspecified reason
2	Previous authentication no longer valid
3	Deauthenticated because sending station is leaving (has left) IBSS or ESS
4	Disassociated due to inactivity
5	Disassociated because AP is unable to handle all currently associated stations
6	Class 2 frame received from nonauthenticated station
7	Class 3 frame received from nonassociation station
8	Disassociated because sending station is leaving (has left) BSS
9	Station requesting (re)association is not authenticated with responding station
10–65 535	Reserved

7.3.1.8 Association ID (AID) field

The AID field is a value assigned by an AP during association and represents the 16-bit ID of a STA. The length of the AID field is two octets. The AID field is illustrated in Figure 31.

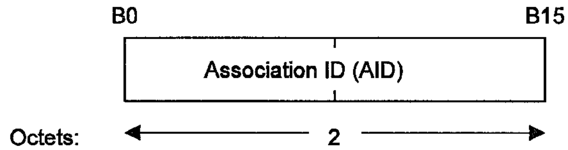


Figure 31—AID fixed field

The value assigned as the Association ID is in the range 1–2007 and is placed in the 14 lsb of the AID field, with the 2 msb of the AID field both set to 1 (see 7.1.3.2).

The AID value 0 is used to announce broadcast and multicast frames in traffic indication map information elements.

7.3.1.9 Status Code field

The Status Code field is used in a response management frame to indicate the success or failure of a requested operation. The length of the Status Code field is two octets. The Status Code field is illustrated in Figure 32.

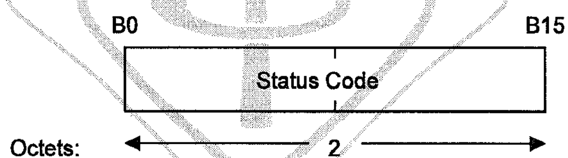


Figure 32—Status Code fixed field

If an operation is successful, then the status code is set to 0. If an operation results in failure, the status code indicates a failure cause. The failure cause codes are defined in Table 19.

Table 19—Status codes

Status code	Meaning
0	Successful
1	Unspecified failure
2–9	Reserved
10	Cannot support all requested capabilities in the Capability Information field
11	Reassociation denied due to inability to confirm that association exists
12	Association denied due to reason outside the scope of this standard
13	Responding station does not support the specified authentication algorithm
14	Received an Authentication frame with authentication transaction sequence number out of expected sequence
15	Authentication rejected because of challenge failure
16	Authentication rejected due to timeout waiting for next frame in sequence

Table 19—Status codes *(continued)*

Status code	Meaning
17	Association denied because AP is unable to handle additional associated stations
18	Association denied due to requesting station not supporting all of the data rates in the BSSBasicRateSet parameter
19–65 535	Reserved

7.3.1.10 Timestamp

This field represents the value of the TSFTIMER (see 11.1) of a frame's source. The length of the Timestamp field is eight octets. The Timestamp field is illustrated in Figure 33.

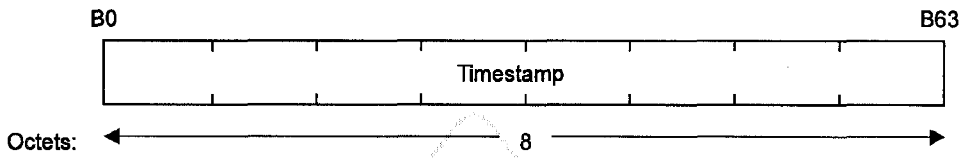


Figure 33—Timestamp fixed field

7.3.2 Information elements

Elements are defined to have a common general format consisting of a one-octet Element ID field, a one-octet length field, and a variable-length element-specific information field. Each element is assigned a unique Element ID as defined in this specification. The Length field specifies the number of octets in the Information field. See Figure 34.

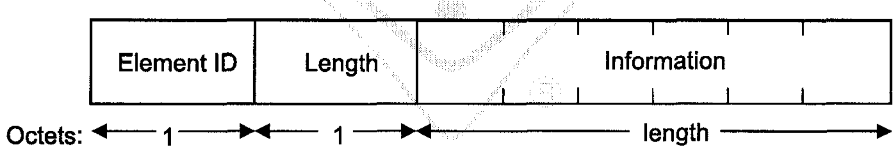


Figure 34—Element format

The set of valid elements is defined in Table 20.

Table 20—Element IDs

Information element	Element ID
SSID	0
Supported rates	1
FH Parameter Set	2
DS Parameter Set	3
CF Parameter Set	4
TIM	5
IBSS Parameter Set	6
Reserved	7–15
Challenge text	16
Reserved for challenge text extension	17–31
Reserved	32–255

7.3.2.1 Service Set Identity (SSID) element

The Service Set Identity (SSID) element indicates the identity of an extended service set (ESS) or IBSS. See Figure 35.

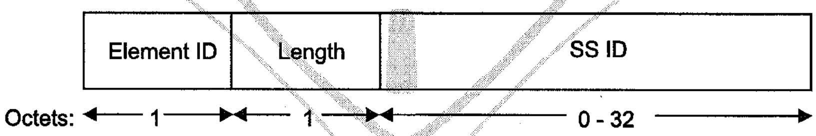


Figure 35—SSID element format

The length of the SSID information field is between 0 and 32 octets. A zero length information field indicates the broadcast SSID.

7.3.2.2 Supported Rates element

The Supported Rates element specifies all the rates that this STA is capable of receiving. The information field is encoded as 1 to 8 octets where each octet describes a single supported rate in units of 500 kbit/s.

Within Beacon, Probe Response, Association Response, and Reassociation Response management frames, each supported rate belonging to the BSSBasicRateSet as defined in 10.3.10.1, is encoded as an octet with the msb (bit 7) set to 1 (e.g., a 1 Mbit/s rate belonging to the BSSBasicRateSet is encoded as X'82'). Rates not belonging to the BSSBasicRateSet are encoded with the msb set to 0 (e.g., a 2 Mbit/s rate not belonging to the BSSBasicRate Set is encoded as X'04'). The msb of each Supported Rate octet in other management frame types is ignored by receiving STAs.

BSSBasicRateSet information in Beacon and Probe Response management frames is used by STAs in order to avoid associating with a BSS if they do not support all the data rates in the BSSBasicRateSet. See Figure 36.

	Element ID	Length	Supported Rates
Octets:	1	1	1 -8

Figure 36—Supported rates element format

7.3.2.3 FH Parameter Set element

The FH Parameter Set element contains the set of parameters necessary to allow synchronization for STAs using a frequency-hopping (FH) PHY. The information field contains Dwell Time, Hop Set, Hop Pattern, and Hop Index parameters. The total length of the information field is 5 octets. See Figure 37.

	Element ID	Length	Dwell Time (TU)	Hop Set	Hop Pattern	Hop Index
Octets:	1	1	2	1	1	1

Figure 37—FH Parameter Set element format

The Dwell Time field is two octets in length and contains the dwell time in TU.

The Hop Set field identifies the particular set of hop patterns and is a single octet.

The Hop Pattern field identifies the individual pattern within a set of hop patterns and is a single octet.

The Hop Index field selects the current channel index within a pattern and is a single octet.

7.3.2.4 DS Parameter Set element

The DS Parameter Set element contains information to allow channel number identification for STAs using a direct sequence spread spectrum (DSSS) PHY. The information field contains a single parameter containing the current channel number. The length of the current channel number parameter is one octet. See Figure 38.

	Element ID	Length	Current Channel
Octets:	1	1	1

Figure 38—DS Parameter Set element format

7.3.2.5 CF Parameter Set element

The CF Parameter Set element contains the set of parameters necessary to support the PCF. The information field contains the CFPCount, CFPPeriod, CFPMaxDuration, and CFPDurRemaining fields. The total length of the information field is 6 octets. See Figure 39.

	Element ID	Length	CFP Count	CFP Period	CFP MaxDuration (TU)	CFP DurRemaining (TU)
Octets:	1	1	1	1	2	2

Figure 39—CF Parameter Set element format

CFPCount indicates how many DTIMs (including the current frame) appear before the next CFP start. A CFPCount of 0 indicates that the current DTIM marks the start of the CFP.

CFPPeriod indicates the number of DTIM intervals between the start of CFPs. The value is an integral number of DTIM intervals.

CFPMaxDuration indicates the maximum duration, in TU, of the CFP that may be generated by this PCF. This value is used by STAs to set their NAV at the TBTT of beacons that begin CFPs.

CFPDurRemaining indicates the maximum time, in TU, remaining in the present CFP, and is set to zero in CFP Parameter elements of beacons transmitted during the contention period. The value of CFPDurRemaining is referenced to the immediately previous TBTT. This value is used by all STAs to update their NAVs during CFPs.

7.3.2.6 TIM

The TIM element contains four fields: DTIM Count, DTIM Period, Bitmap Control, and Partial Virtual Bitmap. See Figure 40.

	Element ID	Length	DTIM Count	DTIM Period	Bitmap Control	Partial Virtual Bitmap
Octets:	1	1	1	1	1	1 - 251

Figure 40—TIM element format

The Length field for this element indicates the length of the information field, which is constrained as described below.

The DTIM Count field indicates how many beacons (including the current frame) appear before the next DTIM. A DTIM Count of 0 indicates that the current TIM is a DTIM. The DTIM count field is a single octet.

The DTIM Period field indicates the number of Beacon intervals between successive DTIMs. If all TIMs are DTIMs, the DTIM Period field has the value 1. The DTIM Period value 0 is reserved. The DTIM period field is a single octet.

The Bitmap Control field is a single octet. The low-order bit contains the Traffic Indicator bit associated with Association ID 0. This bit is set to 1 in TIM elements with a value of 0 in the DTIM Count field when one or more broadcast or multicast frames are buffered at the AP. The high-order 7 bit forms the Bitmap Offset subfield. The Bitmap Offset subfield is a number between 0 and 250, formed by using the Bitmap Control field with the low-order bit set to 0, and is further described below.

The traffic-indication virtual bitmap, maintained by the AP that generates a TIM, consists of 2008 b, and is organized into 251 octets such that bit number N ($0 \leq N \leq 2007$) in the bitmap corresponds to bit number (N mod 8) in octet number $\lfloor N / 8 \rfloor$ where the low-order bit of each octet is bit number 0, and the high order bit is bit number 7. Each bit in the traffic-indication virtual bitmap corresponds to traffic buffered for a specific station within the BSS that the AP is prepared to deliver at the time the beacon frame is transmitted. Bit number N is 0 if there are no directed frames buffered for the station whose Association ID is N. If any directed frames for that station are buffered and the AP is prepared to deliver them, bit number N in the traffic-indication virtual bitmap is 1. A PC may decline to set bits in the TIM for CF-Pollable stations it does not intend to poll (see 11.2.1.5).

The Partial Virtual Bitmap field consists of octets numbered N1 through N2 of the traffic indication virtual bitmap, where N1 is the largest number such that bits numbered 1 through ($N1 \times 8$) – 1 in the bitmap are all

0 and N2 is the smallest number such that bits numbered (N2 + 1) × 8 through 2007 in the bitmap are all 0. In this case, the Bitmap Offset subfield value contains the number N1, and the Length field will be set to (N2 – N1) + 4.

In the event that all bits other than bit 0 in the virtual bitmap are 0, the Partial Virtual Bitmap field is encoded as a single octet equal to 0, and the Bitmap Offset subfield is 0.

7.3.2.7 IBSS Parameter Set element

The IBSS Parameter Set element contains the set of parameters necessary to support an IBSS. The information field contains the ATIM Window parameter. See Figure 41.

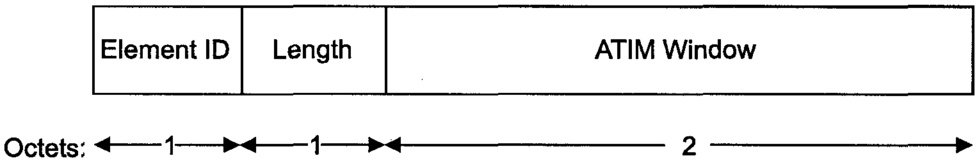


Figure 41—IBSS Parameter Set element format

The ATIM Window field is 2 octets in length and contain the ATIM Window length in TU.

7.3.2.8 Challenge Text element

The Challenge Text element contains the challenge text within Authentication exchanges. The element information field length is dependent upon the authentication algorithm and the transaction sequence number as specified in 8.1. See Figure 42.

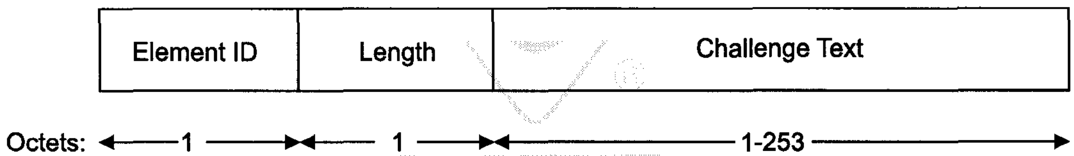


Figure 42—Challenge Text element format

8. Authentication and privacy

8.1 Authentication services

IEEE 802.11 defines two subtypes of authentication service: *Open System* and *Shared Key*. The subtype invoked is indicated in the body of authentication management frames. Thus authentication frames are self-identifying with respect to authentication algorithm. All management frames of subtype Authentication shall be unicast frames as authentication is done between pairs of stations (i.e., multicast authentication is not allowed). Management frames of subtype Deauthentication are advisory, and may therefore be sent as group-addressed frames.

A mutual authentication relationship shall exist between two stations following a successful authentication exchange as described below. Authentication shall be used between stations and the AP in an infrastructure BSS. Authentication may be used between two STAs in an IBSS.

8.1.1 Open System authentication

Open System authentication is the simplest of the available authentication algorithms. Essentially it is a null authentication algorithm. Any STA that requests authentication with this algorithm may become authenticated if aAuthenticationType at the recipient station is set to Open System authentication. Open System authentication is not required to be successful as a STA may decline to authenticate with any particular other STA. Open System authentication is the default authentication algorithm.

Open System authentication involves a two-step authentication transaction sequence. The first step in the sequence is the identity assertion and request for authentication. The second frame in the sequence is the authentication result. If the result is "successful," the STAs shall be mutually authenticated.

8.1.1.1 Open System authentication (first frame)

— Message type: Management
— Message subtype: Authentication
— Information items:
  • Authentication Algorithm Identification = "Open System"
  • Station Identity Assertion (in SA field of header)
  • Authentication transaction sequence number = 1
  • Authentication algorithm dependent information (none)
— Direction of message: From authentication initiating STA

8.1.1.2 Open System authentication (final frame)

— Message type: Management
— Message subtype: Authentication
— Information items:
  • Authentication Algorithm Identification = "Open System"
  • Authentication transaction sequence number = 2
  • Authentication algorithm dependent information (none)
  • The result of the requested authentication as defined in 7.3.1.9.
— Direction of message: From authenticating STA to initiating STA

If aAuthenticationType does not include the value "Open System," the result code shall not take the value "successful."

8.1.2 Shared Key authentication

Shared Key authentication supports authentication of STAs as either a member of those who know a shared secret key or a member of those who do not. IEEE 802.11 Shared Key authentication accomplishes this without the need to transmit the secret key in the clear; requiring the use of the WEP privacy mechanism. Therefore, this authentication scheme is only available if the WEP option is implemented. Additionally, the Shared Key authentication algorithm shall be implemented as one of aAuthenticationAlgorithms at any STA where WEP is implemented.

The required secret, shared key is presumed to have been delivered to participating STAs via a secure channel that is independent of IEEE 802.11. This shared key is contained in a write-only MIB attribute via the MAC management path. The attribute is write-only so that the key value remains internal to the MAC.

During the Shared Key authentication exchange, both the challenge and the encrypted challenge are transmitted. This facilitates unauthorized discovery of the PRN (pseudorandom number) sequence for the key/IV pair used for the exchange. Implementations should therefore avoid using the same key/IV pair for subsequent frames.

A STA shall not initiate a Shared Key authentication exchange unless its aPrivacyOptionImplemented attribute is "true."

In the following description, the STA initiating the authentication exchange is referred to as the requester, and the STA to which the initial frame in the exchange is addressed is referred to as the responder.

8.1.2.1 Shared Key authentication (first frame)

— Message type: Management
— Message subtype: Authentication
— Information Items:
  - Station Identity Assertion (in SA field of header)
  - Authentication Algorithm Identification = "Shared Key"
  - Authentication transaction sequence number = 1
  - Authentication algorithm dependent information (none)
— Direction of message: From requester to responder

8.1.2.2 Shared Key authentication (second frame)

Before sending the second frame in the Shared Key authentication sequence, the responder shall use WEP to generate a string of octets that shall be used as the authentication challenge text.

— Message type: Management
— Message subtype: Authentication
— Information Items:
  - Authentication Algorithm Identification = "Shared Key"
  - Authentication transaction sequence number = 2
  - Authentication algorithm dependent information = the authentication result.
    The result of the requested authentication as defined in 7.3.1.9.

    If the status code is not "successful," this shall be the last frame of the transaction sequence. If the status code is not "successful," the content of the challenge text field is unspecified.

    If the status code is "successful," the following additional information items shall have valid contents:

    Authentication algorithm dependent information = challenge text.

This field shall be of fixed length of 128 octets. The field shall be filled with octets generated by the WEP PRNG. The actual value of the challenge field is unimportant, but the value shall not be a single static value. The key and initialization vector (IV) used when generating the challenge text are unspecified because this key/IV value does not have to be shared and does not affect interoperability.

— Direction of message: From responder to requester

8.1.2.3 Shared Key authentication (third frame)

The requester shall copy the challenge text from the second frame into the third frame. The third frame shall be transmitted after encryption by WEP, as defined in 8.2.3 using the shared secret key.

— Message type: Management
— Message subtype: Authentication
— Information Items:
  - Authentication Algorithm Identification = "Shared Key"
  - Authentication transaction sequence number = 3
  - Authentication algorithm dependent information = challenge text from sequence two frame.
— Direction of message: From requester to responder

This frame shall be encrypted as described below.

8.1.2.4 Shared Key authentication (final frame)

The responder shall attempt to decrypt the contents of the third frame in the authentication sequence as described below. If the WEP ICV check is successful, the responder shall then compare the decrypted contents of the Challenge Text field to the challenge text that was sent in frame 2 of the sequence. If they are the same, then the responder shall respond with a successful status code in frame 4 of the sequence. If the WEP ICV check fails, the responder shall respond with an unsuccessful status code in frame 4 of the sequence as described below.

— Message type: Management
— Message subtype: Authentication
— Information Items:
  - Authentication Algorithm Identification = "Shared Key"
  - Authentication transaction sequence number = 4
  - Authentication algorithm dependent information = the authentication result.
    The result of the requested authentication.
    This is a fixed length item with values "successful" and "unsuccessful."
— Direction of message: From responder to requester

8.2 The Wired Equivalent Privacy (WEP) algorithm

8.2.1 Introduction

Eavesdropping is a familiar problem to users of other types of wireless technology. IEEE 802.11 specifies a wired LAN equivalent data confidentiality algorithm. *Wired equivalent privacy* is defined as protecting authorized users of a wireless LAN from casual eavesdropping. This service is intended to provide functionality for the wireless LAN equivalent to that provided by the physical security attributes inherent to a wired medium.

Data confidentiality depends on an external key management service to distribute data enciphering/deciphering keys. The IEEE 802.11 standards committee specifically recommends against running an IEEE 802.11

LAN with privacy but without authentication. While this combination is possible, it leaves the system open to significant security threats.

8.2.2 Properties of the WEP algorithm

The WEP algorithm has the following properties:

— *It is reasonably strong:* The security afforded by the algorithm relies on the difficulty of discovering the secret key through a brute-force attack. This in turn is related to the length of the secret key and the frequency of changing keys. WEP allows for the changing of the key (k) and frequent changing of the IV.
— *It is self-synchronizing:* WEP is self-synchronizing for each message. This property is critical for a data-link level encryption algorithm, where "best effort" delivery is assumed and packet loss rates may be high.
— *It is efficient:* The WEP algorithm is efficient and may be implemented in either hardware or software.
— *It may be exportable:* Every effort has been made to design the WEP system operation so as to maximize the chances of approval, by the U.S. Department of Commerce, of export from the U.S. of products containing a WEP implementation. However, due to the legal and political climate toward cryptography at the time of publication, no guarantee can be made that any specific IEEE 802.11 implementations that use WEP will be exportable from the United States of America.
— *It is optional:* The implementation and use of WEP is an IEEE 802.11 option.

8.2.3 WEP theory of operation

The process of disguising (binary) data in order to hide its information content is called *encryption* (see [B4]). Data that is not enciphered is called *plaintext* (denoted by P) and data that is enciphered is called *ciphertext* (denoted by C). The process of turning ciphertext back into plaintext is called *decryption.* A *cryptographic algorithm,* or cipher, is a mathematical function used for enciphering or deciphering data. Modern cryptographic algorithms use a key sequence (denoted by k) to modify their output. The encryption function E operates on P to produce C:

$$E_k(P) = C$$

In the reverse process, the decryption function D operates on C to produce P:

$$D_k(C) = P$$

As illustrated in Figure 43, note that if the same key can be used for encryption and decryption then

$$D_k(E_k(P)) = P$$

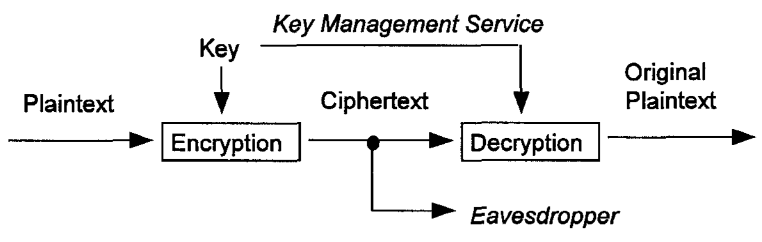


Figure 43—A confidential data channel

The WEP algorithm is a form of electronic code book in which a block of plaintext is bitwise XORed with a pseudorandom key sequence of equal length. The key sequence is generated by the WEP algorithm.

Referring to Figure 44 and viewing from left to right, encipherment begins with a *secret key* that has been distributed to cooperating STAs by an external key management service. WEP is a symmetric algorithm in which the same key is used for encipherment and decipherment.

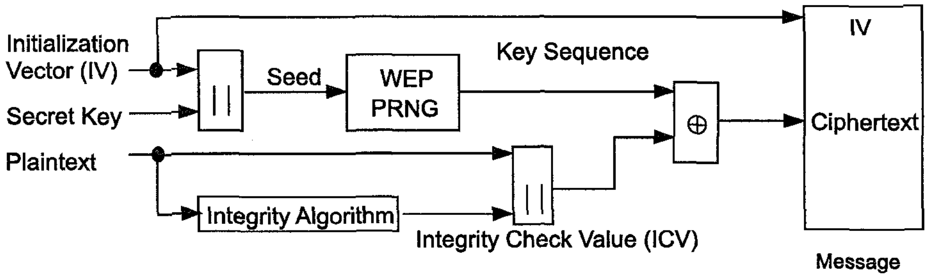


Figure 44—WEP encipherment block diagram

The secret key is concatenated with an *initialization vector* (IV) and the resulting *seed* is input to a *pseudo-random number generator* (PRNG). The PRNG outputs a *key sequence k* of pseudorandom octets equal in length to the number of data octets that are to be transmitted in the expanded MPDU plus 4 [since the key sequence is used to protect the *integrity check value* (ICV) as well as the data]. Two processes are applied to the plaintext MPDU. To protect against unauthorized data modification, an integrity algorithm operates on *P* to produce an ICV. Encipherment is then accomplished by mathematically combining the key sequence with the plaintext concatenated with the ICV. The output of the process is a *message* containing the IV and ciphertext.

The WEP PRNG is the critical component of this process, since it transforms a relatively short secret key into an arbitrarily long key sequence. This greatly simplifies the task of key distribution, as only the secret key needs to be communicated between STAs. The IV extends the useful lifetime of the secret key and provides the self-synchronous property of the algorithm. The secret key remains constant while the IV changes periodically. Each new IV results in a new seed and key sequence, thus there is a one-to-one correspondence between the IV and *k*. The IV may be changed as frequently as every MPDU and, since it travels with the message, the receiver will always be able to decipher any message. The IV is transmitted in the clear since it does not provide an attacker with any information about the secret key, and since its value must be known by the recipient in order to perform the decryption.

When choosing how often to change IV values, implementors should consider that the contents of some fields in higher-layer protocol headers, as well as certain other higher-layer information, is constant or highly predictable. When such information is transmitted while encrypting with a particular key and IV, an eavesdropper can readily determine portions of the key sequence generated by that (key, IV) pair. If the same (key, IV) pair is used for successive MPDUs, this effect may substantially reduce the degree of privacy conferred by the WEP algorithm, allowing an eavesdropper to recover a subset of the user data without any knowledge of the secret key. Changing the IV after each MPDU is a simple method of preserving the effectiveness of WEP in this situation.

The WEP algorithm is applied to the frame body of an MPDU. The {IV, frame body, ICV} triplet forms the actual data to be sent in the data frame.

For WEP protected frames, the first four octets of the frame body contain the IV field for the MPDU. This field is defined in 8.2.5. The 64-bit PRNG seed is formed using the secret key as the most significant 40 bits and the initialization vector (IV) as the least significant 24 bits. The IV is followed by the MPDU, which is followed by the ICV. The WEP ICV is 32 bits. The WEP Integrity Check algorithm is CRC-32, as defined in 7.1.3.6.

As stated previously, WEP combines *k* with *P* using bitwise XOR.

Referring to Figure 45 and viewing from left to right, decipherment begins with the arrival of a message. The IV of the incoming message shall be used to generate the key sequence necessary to decipher the incoming message. Combining the ciphertext with the proper key sequence yields the original plaintext and ICV. Correct decipherment shall be verified by performing the integrity check algorithm on the recovered plaintext and comparing the output ICV′ to the ICV transmitted with the message. If ICV′ is not equal to ICV, the received MPDU is in error and an error indication is sent to MAC management. MSDUs with erroneous MPDUs (due to inability to decrypt) shall not be passed to LLC.

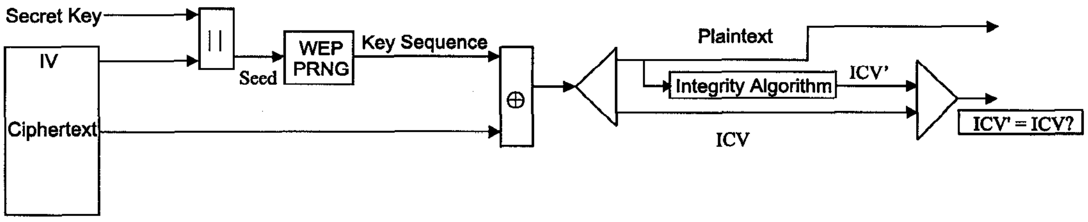


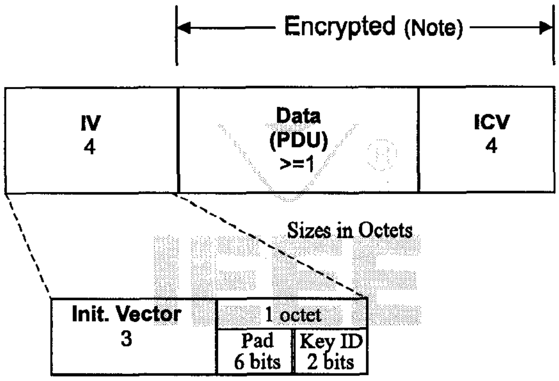
Figure 45—WEP decipherment block diagram

8.2.4 WEP algorithm specification

WEP uses the RC4 PRNG algorithm from RSA Data Security, Inc.[6]

8.2.5 WEP MPDU expansion

Figure 46 shows the encrypted MPDU as constructed by the WEP algorithm.



NOTE-The encipherment process has expanded the original MPDU by 8 octets, 4 for the Initialization Vector (IV) field and 4 for the Integrity Check Value (ICV). The ICV is calculated on the Data field only.

Figure 46—Construction of expanded WEP MPDU

The WEP ICV shall be a 32-bit field containing the 32-bit cyclic redundancy code (CRC) defined in 7.1.3.6 calculated over the Data (PDU) field as depicted in Figure 46. The expanded MPDU shall include a 32-bit IV field immediately preceding the MPDU. This field shall contain three subfields: a three-octet field that contains the initialization vector, a 2-bit key ID field, and a 6-bit pad field. The ordering conventions defined in 7.1.1 apply to the IV fields and its subfields and to the ICV field. The key ID subfield contents select one

[6]Details of the RC4 algorithm are available from RSA. Please contact RSA for algorithm details and the uniform RC4 licensee terms that RSA offers to anyone wishing to use RC4 for the purpose of implementing the IEEE 802.11 WEP option. If necessary, contact the IEEE Standards Department for details on how to communicate with RSA.

of four possible secret key values for use in decrypting this MPDU. Interpretation of these bits is discussed further in 8.3.2. The contents of the pad subfield shall be zero. The key ID occupies the two msb of the last octet of the IV field, while the pad occupies the six lsb of this octet.

The WEP mechanism is invisible to entities outside the IEEE 802.11 MAC data path.

8.3 Security-Related MIB attributes

The IEEE 802.11 security mechanisms are controlled via the MAC management path and related MIB attributes. This subclause gives an overview of the security related MIB attributes and how they are used. For details of the MIB attribute definitions, refer to 11.4.

8.3.1 Authentication-Related MIB attributes

The type of authentication invoked when authentication is attempted is controlled by the AuthenticationType parameter to the MLME-AUTHENTICATE.request primitive. The type of authentication request that may be accepted by a STA is controlled by the MIB attribute aAuthenticationType. The type of authentication is selected from the following set of values:

— Open System
— Shared Key

All other values are reserved. The numeric encoding of these values is given in 7.3.1.1.

8.3.2 Privacy-Related MIB attributes

WEP invocation is controlled by the parameters passed to the MLME-AUTHENTICATE.request primitive as well as a number of MIB attributes. An overview of the attributes and their usage is given in this clause. All MIB attributes that hold WEP keys are externally write-only; the contents shall not be readable via MAC management SAPs. See 11.4 for the formal MIB attribute definitions.

The boolean variable aPrivacyInvoked shall be set to "false" to prevent the STA from transmitting MPDUs of type Data with the WEP subfield of the Frame Control field set to 1. It does not affect MPDU or MMPDU reception.

The default value for all WEP keys shall be null. Note that encrypting a frame using WEP with a null key is not the same as failing to encrypt the frame. Any request to encrypt a frame with a null key shall result in the MSDU being discarded and an MA-UNIDATA-STATUS.indication with a transmission status indicating that the frame may not be encrypted with a null key. Decrypting a frame whose WEP subfield is set to 1 involves stripping the IV, and checking the ICV against the calculated ICV′ value computed over the data contained in the MPDU.

To support shared key configurations, the MIB contains a four-element vector called "aWEPDefaultKeys." The default value for each element of this vector is null. These elements contain the default keys to be used with WEP.

An additional attribute called "aWEPDefaultKeyID" is an integer. When set to a value of 0, 1, 2, or 3, MPDUs transmitted with the WEP subfield of the Frame Control field set to 1 shall be encrypted using the first, second, third, or fourth element, respectively, from aWEPDefaultKeys, unless the frame has an individual RA and a key mapping exists for the RA of the frame. On receive, the incoming MPDU shall be decrypted using the element from aWEPDefaultKeys specified by the received key ID field, unless the frame has an individual RA and a key mapping exists for the TA of the frame. The value in the transmitted key ID field shall be zero in all cases except when aWEPDefaultKeyID is used to encrypt a frame and is set to a value of 1, 2, or 3, in which case the transmitted key ID field shall contain the value of aWEPDefaultKeyID.

When the boolean attribute aExcludeUnencrypted is set to True, MPDUs of type Data received by the STA with the WEP subfield of the Frame Control field equal to zero shall not be indicated at the MAC service interface. When aExcludeUnencrypted is set to True, only MSDUs that have been decrypted successfully shall be indicated at the MAC service interface.

IEEE 802.11 does not require that the same WEP key be used for all STAs. The MIB supports the ability to share a separate WEP key for each RA/TA pair. Key mapping is supported by a MIB attribute that is an array called "aWEPKeyMappings." aWEPKeyMappings contains zero or one entry for each MAC address, up to an implementation-defined maximum number of entries identified by aWEPKeyMappingLength, and contains two fields for each entry: a boolean "WEPOn" and the corresponding WEPKey. In an infrastructure BSS, the AP's WEPOn value in the entry in its aWEPKeyMapping table corresponding to a STA's MAC address shall not be set to True for a STA if that STA has not successfully initiated and completed an authentication sequence using an authentication type other than "Open System." The default value for all WEPOn fields is False. aWEPKeyMappings shall be indexed by either RA or TA addresses (since WEP is applied only to the wireless link), as described below. When an entry in the table exists for a particular MAC address, the values in the aWEPKeyMappings attribute shall be used instead of the aWEPDefaultKeyID and aWEPDefaultKeys variables.

The minimal value of aWEPKeyMappingLength shall be 10. This value represents a minimum capability that may be assumed for any STA implementing the WEP option.

When transmitting a frame of type Data, the values of aPrivacyInvoked, aWEPKeyMappings, aWEPDefaultKeys, and aWEPDefaultKeyID in effect at an unspecified time between receipt by the MAC of the MAUNITDATA.request primitive and the time of transmission of that frame shall be used according to the following decision tree:

```
if aPrivacyInvoked is "false"
        the MPDU is transmitted without encryption
else
        if (the MPDU has an individual RA and
         there is an entry in aWEPKeyMappings for that RA)
                if that entry has WEPOn set to "false"
                        the MPDU is transmitted without encryption
                else
                        if that entry contains a key that is null
                                discard the entire MSDU and generate an
                                MA-UNITDATA-STATUS.indication primitive to
                                notify LLC that the MSDU was undeliverable due to
                                a null WEP key
                        else
                                encrypt the MPDU using that entry's key, setting the keyID
                                subfield of the IV field to zero
        else
                if (the MPDU has a group RA and the Privacy subfield
                        of the Capability Information field in this BSS is set to 0)
                        the MPDU is transmitted without encryption
                else
                        if aWEPDefaultKeys[aWEPDefaultKeyID] is null
                                discard the MSDU and generate an
                                MA-UNITDATA-STATUS.indication primitive to
                                notify LLC that the entire MSDU was undeliverable
```

```
                due to a null WEP key
        else
                encrypt the MPDU using
                aWEPDefaultKeys[aWEPDefaultKeyID],
                setting the KeyID subfield of the IV field to
                aWEPDefaultKeyID
```

When receiving a frame of type Data, the values of aPrivacyOptionImplemented, aWEPKeyMappings, aWEPDefaultKeys, aWEPDefaultKeyID, and aExcludeUnencrypted in effect at the time the PHY-RXSTART.indication primitive is received by the MAC shall be used according to the following decision tree:

```
if the WEP subfield of the Frame Control Field is zero
    if aExcludeUnencrypted is "true"
        discard the frame body without indication to LLC and increment
        aWEPExcludedCount
    else
        receive the frame without decryption
else
    if aPrivacyOptionImplemented is "true"
        if (the MPDU has individual RA and
         there is an entry in aWEPKeyMappings matching the MPDU's TA)
            if that entry has WEPOn set to "false"
                discard the frame body and increment
                aWEPUndecryptableCount
            else
                if that entry contains a key that is null
                    discard the frame body and increment
                    aWEPUndecryptableCount
                else
                    attempt to decrypt with that key, incrementing
                    aWEPICVErrorCount if the ICV check fails
        else
            if aWEPDefaultKeys[keyID] is null
                discard the frame body and increment
                aWEPUndecryptableCount
            else
                attempt to decrypt with aWEPDefaultKeys[keyID],
                incrementing aWEPICVErrorCount if the ICV check fails
    else
        discard the frame body and increment aWEPUndecryptableCount
```

When transmitting a frame of type Management, subtype Authentication with an Authentication Transaction Sequence Number field value of 2, the MAC shall operate according to the following decision tree:

```
if aPrivacyOptionImplemented is "false"
    the MMPDU is transmitted with a sequence
    of zero octets in the Challenge Text field and a Status Code value of 13
else
    the MMPDU is transmitted with a sequence of 128 octets generated using the
    WEP PRNG and a key whose value is unspecified and beyond the scope of this
    standard and a randomly chosen IV value (note that this will typically be selected
    by the same mechanism for choosing IV values for transmitted data MPDUs)
    in the Challenge Text field and a status code value of 0 (the IV used is
```


immaterial and is not transmitted). Note that there are cryptographic issues involved in the choice of key/IV for this process as the challenge text is sent unencrypted and therefore provides a known output sequence from the PRNG.

When receiving a frame of type Management, subtype Authentication with an Authentication Transaction Sequence Number field value of 2, the MAC shall operate according to the following decision tree:

```
if the WEP subfield of the Frame Control field is one
        respond with a status code value of 15
else
        if aPrivacyOptionImplemented is "true"
                if there is a mapping in aWEPKeyMappings matching the MSDU's TA
                        if that key is null
                                respond with a frame whose Authentication Transaction
                                Sequence Number field is 3 that contains the appropriate
                                Authentication Algorithm Number, a status code value of
                                15 and no Challenge Text field, without encrypting the
                                contents of the frame
                        else
                                respond with a frame whose Authentication Transaction
                                Sequence Number field is 3 that contains the appropriate
                                Authentication Algorithm Number, a status code value of
                                0 and the identical Challenge Text field, encrypted using
                                that key, and setting the key ID subfield in the IV field to 0
                else
                        if aWEPDefaultKeys[aWEPDefaultKeyID] is null
                                respond with a frame whose Authentication Transaction
                                Sequence Number field is 3 that contains the appropriate
                                Authentication Algorithm Number, a status code value of
                                15 and no Challenge Text field, without encrypting the
                                contents of the frame
                        else
                                respond with a frame whose Authentication Transaction
                                Sequence Number field is 3 that contains the appropriate
                                Authentication Algorithm Number, a status code value of 0
                                and the identical Challenge Text field, encrypted using
                                aWEPDefaultKeys[aWEPDefaultKeyID], setting the
                                key ID subfield in the IV field to aWEPDefaultKeyID
        else
                respond with a frame whose Authentication Transaction
                Sequence Number field is 3 that contains the appropriate Authentication
                Algorithm Number, a status code value of 13 and no Challenge Text
                field, without encrypting the contents of the frame
```

When receiving a frame of type Management, subtype Authentication with an Authentication Transaction Sequence Number field value of 3, the MAC shall operate according to the following decision tree:

```
if the WEP subfield of the Frame Control field is zero
        respond with a status code value of 15
else
        if aPrivacyOptionImplemented is "true"
                if there is a mapping in aWEPKeyMappings matching the MSDU's TA
                        if that key is null
```

respond with a frame whose Authentication Transaction Sequence Number field is 4 that contains the appropriate Authentication Algorithm Number, and a status code value of 15 without encrypting the contents of the frame

else

attempt to decrypt with that key, incrementing aWEPICVErrorCount and responding with a status code value of 15 if the ICV check fails

else

if aWEPDefaultKeys[keyID] is null

respond with a frame whose Authentication Transaction Sequence Number field is 4 that contains the appropriate Authentication Algorithm Number, and a status code value of 15 without encrypting the contents of the frame

else

attempt to decrypt with aWEPDefaultKeys[keyID], incrementing aWEPICVErrorCount and responding with a status code value of 15 if the ICV check fails

else

respond with a frame whose Authentication Transaction Sequence Number field is 4 that contains the appropriate Authentication Algorithm Number, and a status code value of 15

The attribute aPrivacyInvoked shall not take the value "true" if the attribute aPrivacyOptionImplemented is "false." Setting the attribute aWEPKeyMappings to a value that includes more than aWEPKeyMappingLength entries is illegal and shall have an implementation-specific effect on the operation of the privacy service. Note that aWEPKeyMappings may contain between zero and aWEPKeyMappingLength entries, inclusive.

It is recommended that the values of the attributes in the aPrivacygrp not be changed during the authentication sequence as unintended operation may result.

9. MAC sublayer functional description

The MAC functional description is presented in this clause. The architecture of the MAC sublayer, including the distributed coordination function (DCF), the point coordination function (PCF), and their coexistence in an IEEE 802.11 LAN are introduced in 9.1. These functions are expanded on in 9.2 and 9.3, and a complete functional description of each is provided. Fragmentation and defragmentation are covered in 9.4 and 9.5. Multirate support is addressed in 9.6. The allowable frame exchange sequences are listed in 9.7. Finally, a number of additional restrictions to limit the cases in which MSDUs are reordered or discarded are described in 9.8.

9.1 MAC architecture

The MAC architecture can be described as shown in Figure 47 as providing the PCF through the services of the DCF.

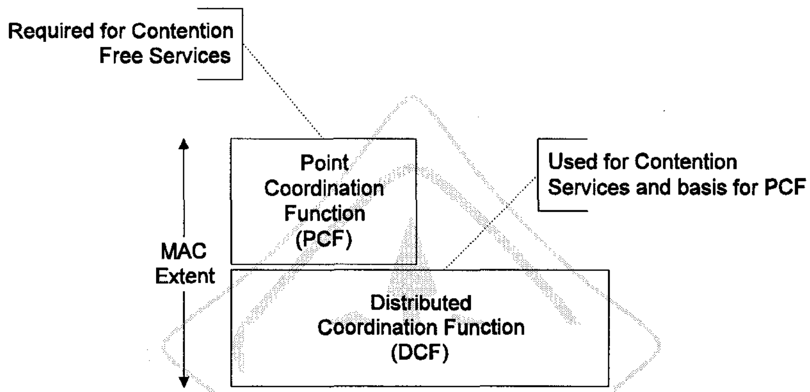


Figure 47—MAC architecture

9.1.1 Distributed coordination function (DCF)

The fundamental access method of the IEEE 802.11 MAC is a DCF known as *carrier sense multiple access with collision avoidance*, or CSMA/CA. The DCF shall be implemented in all STAs, for use within both IBSS and infrastructure network configurations.

For a STA to transmit, it shall sense the medium to determine if another STA is transmitting. If the medium is not determined to be busy (see 9.2.1), the transmission may proceed. The CSMA/CA distributed algorithm mandates that a gap of a minimum specified duration exist between contiguous frame sequences. A transmitting STA shall ensure that the medium is idle for this required duration before attempting to transmit. If the medium is determined to be busy, the STA shall defer until the end of the current transmission. After deferral, or prior to attempting to transmit again immediately after a successful transmission, the STA shall select a random backoff interval and shall decrement the backoff interval counter while the medium is idle. A refinement of the method may be used under various circumstances to further minimize collisions—here the transmitting and receiving STA exchange short control frames [request to send (RTS) and clear to send (CTS) frames] after determining that the medium is idle and after any deferrals or backoffs, prior to data transmission. The details of CSMA/CA, deferrals, and backoffs are described in 9.2. RTS/CTS exchanges are also presented in 9.2.

9.1.2 Point coordination function (PCF)

The IEEE 802.11 MAC may also incorporate an optional access method called a PCF, which is only usable on infrastructure network configurations. This access method uses a point coordinator (PC), which shall operate at the access point of the BSS, to determine which STA currently has the right to transmit. The operation is essentially that of polling with the PC performing the role of the polling master. The operation of the PCF may require additional coordination, not specified in this standard, to permit efficient operation in cases where multiple point-coordinated BSSs are operating on the same channel, in overlapping physical space.

The PCF uses a virtual carrier sense mechanism aided by an access priority mechanism. The PCF shall distribute information within Beacon management frames to gain control of the medium by setting the network allocation vector (NAV) in STAs. In addition, all frame transmissions under the PCF may use an interframe space (IFS) that is smaller than the IFS for frames transmitted via the DCF. The use of a smaller IFS implies that point-coordinated traffic shall have priority access to the medium over STAs in overlapping BSSs operating under the DCF access method.

The access priority provided by a PCF may be utilized to create a *contention-free* (CF) access method. The PC controls the frame transmissions of the STAs so as to eliminate contention for a limited period of time.

9.1.3 Coexistence of DCF and PCF

The DCF and the PCF shall coexist in a manner that permits both to operate concurrently within the same BSS. When a PC is operating in a BSS, the two access methods alternate, with a contention-free period (CFP) followed by a contention period (CP). This is described in greater detail in 9.3.

9.1.4 Fragmentation/defragmentation overview

The process of partitioning a MAC service data unit (MSDU) or a MAC management protocol data unit (MMPDU) into smaller MAC level frames, MAC protocol data units (MPDUs), is called fragmentation. Fragmentation creates MPDUs smaller than the original MSDU or MMPDU length to increase reliability, by increasing the probability of successful transmission of the MSDU or MMPDU in cases where channel characteristics limit reception reliability for longer frames. Fragmentation is accomplished at each immediate transmitter. The process of recombining MPDUs into a single MSDU or MMPDU is defined as defragmentation. Defragmentation is accomplished at each immediate recipient.

Only MPDUs with a unicast receiver address shall be fragmented. Broadcast/multicast frames shall not be fragmented even if their length exceeds aFragmentationThreshold.

When a directed MSDU is received from the LLC or a directed MMPDU is received from the MAC sublayer management entity (MLME) with a length greater than aFragmentationThreshold, the MSDU or MMPDU shall be fragmented. The MSDU or MMPDU is divided into MPDUs. Each fragment is a frame no larger than aFragmentationThreshold. It is possible that any fragment may be a frame smaller than aFragmentationThreshold. An illustration of fragmentation is shown in Figure 48.

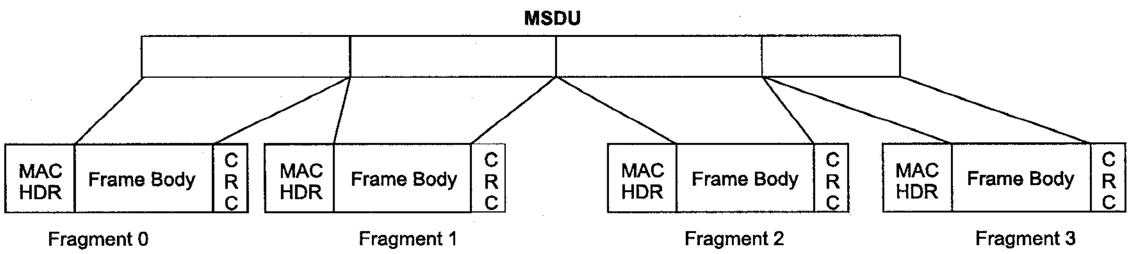


Figure 48—Fragmentation

The MPDUs resulting from the fragmentation of an MSDU or MMPDU are sent as independent transmissions, each of which is separately acknowledged. This permits transmission retries to occur per fragment, rather than per MSDU or MMPDU. Unless interrupted due to medium occupancy limitations for a given PHY, the fragments of a single MSDU or MMPDU are sent as a burst during the CP, using a single invocation of the DCF medium access procedure. The fragments of a single MSDU or MMPDU are sent during a CFP as individual frames obeying the rules of the PC medium access procedure.

9.1.5 MAC data service

The MAC data service shall translate MAC service requests from LLC into input signals utilized by the MAC state machines. The MAC data service shall also translate output signals from the MAC state machines into service indications to LLC. The translations are given in the MAC data service state machine defined in Annex C.

9.2 DCF

The basic medium access protocol is a DCF that allows for automatic medium sharing between compatible PHYs through the use of CSMA/CA and a random backoff time following a busy medium condition. In addition, all directed traffic uses immediate positive acknowledgment (ACK frame) where retransmission is scheduled by the sender if no ACK is received.

The CSMA/CA protocol is designed to reduce the collision probability between multiple STAs accessing a medium, at the point where collisions would most likely occur. Just after the medium becomes idle following a busy medium (as indicated by the CS function) is when the highest probability of a collision exists. This is because multiple STAs could have been waiting for the medium to become available again. This is the situation that necessitates a random backoff procedure to resolve medium contention conflicts.

Carrier Sense shall be performed both through physical and virtual mechanisms.

The virtual carrier sense mechanism is achieved by distributing reservation information announcing the impending use of the medium. The exchange of RTS and CTS frames prior to the actual data frame is one means of distribution of this medium reservation information. The RTS and CTS frames contain a Duration/ID field that defines the period of time that the medium is to be reserved to transmit the actual data frame and the returning ACK frame. All STAs within the reception range of either the originating STA (which transmits the RTS) or the destination STA (which transmits the CTS) shall learn of the medium reservation. Thus a STA can be unable to receive from the originating STA, yet still know about the impending use of the medium to transmit a data frame.

Another means of distributing the medium reservation information is the Duration/ID field in directed frames. This field gives the time that the medium is reserved, either to the end of the immediately following ACK, or in the case of a fragment sequence, to the end of the ACK following the next fragment.

The RTS/CTS exchange also performs both a type of fast collision inference and a transmission path check. If the return CTS is not detected by the STA originating the RTS, the originating STA may repeat the process (after observing the other medium-use rules) more quickly than if the long data frame had been transmitted and a return ACK frame had not been detected.

Another advantage of the RTS/CTS mechanism occurs where multiple BSSs utilizing the same channel overlap. The medium reservation mechanism works across the BSA boundaries. The RTS/CTS mechanism may also improve operation in a typical situation where all STAs can receive from the AP, but cannot receive from all other STAs in the BSA.

The RTS/CTS mechanism cannot be used for MPDUs with broadcast and multicast immediate address because there are multiple destinations for the RTS, and thus potentially multiple concurrent senders of the CTS in response. The RTS/CTS mechanism need not be used for every data frame transmission. Because the additional RTS and CTS frames add overhead inefficiency, the mechanism is not always justified, especially for short data frames.

The use of the RTS/CTS mechanism is under control of the aRTSThreshold attribute. This attribute may be set on a per-STA basis. This mechanism allows STAs to be configured to use RTS/CTS either always, never, or only on frames longer than a specified length.

A STA configured not to initiate the RTS/CTS mechanism shall still update its virtual carrier sense mechanism with the duration information contained in a received RTS or CTS frame, and shall always respond to an RTS addressed to it with a CTS.

The medium access protocol allows for STAs to support different sets of data rates. All STAs shall receive all the data rates in the aBasicRateSet and transmit at one or more of the aBasicRateSet data rates. To support the proper operation of the RTS/CTS and the virtual carrier sense mechanism, all STAs shall be able to detect the RTS and CTS frames. For this reason the RTS and CTS frames shall be transmitted at one of the aBasicRateSet rates. (See 9.6 for a description of multirate operation.)

Data frames sent under the DCF shall use the frame type Data and subtype Data or Null Function. STAs receiving Data type frames shall only consider the frame body as the basis of a possible indication to LLC.

9.2.1 Carrier sense mechanism

Physical and virtual carrier sense functions are used to determine the state of the medium. When either function indicates a busy medium, the medium shall be considered busy; otherwise, it shall be considered idle.

A physical carrier sense mechanism shall be provided by the PHY. See Clause 12 for how this information is conveyed to the MAC. The details of physical carrier sense are provided in the individual PHY specifications.

A virtual carrier sense mechanism shall be provided by the MAC. This mechanism is referred to as the network allocation vector (NAV). The NAV maintains a prediction of future traffic on the medium based on duration information that is announced in RTS/CTS frames prior to the actual exchange of data. The duration information is also available in the MAC headers of all frames sent during the CP other than PS-Poll Control frames. The mechanism for setting the NAV using RTS/CTS in the DCF is described in 9.2.5.4, and use of the NAV in PCF is described in 9.3.2.2.

The carrier sense mechanism combines the NAV state and the STA's transmitter status with physical carrier sense to determine the busy/idle state of the medium. The NAV may be thought of as a counter, which counts down to zero at a uniform rate. When the counter is zero, the virtual carrier sense indication is that the medium is idle; when nonzero, that it is busy. The medium shall be determined to be busy whenever the STA is transmitting.

9.2.2 MAC-Level acknowledgments

The reception of some frames, as described in 9.7, 9.2.8, and 9.3.3.4, require the receiving STA to respond with an acknowledgment, generally an ACK frame, if the FCS of the received frame is correct. This technique is known as positive acknowledgment.

Lack of reception of an expected ACK frame indicates to the source STA that an error has occurred. Note, however, that the destination STA may have received the frame correctly, and that the error occurred in the reception of the ACK frame. To the initiator of the frame exchange, this condition is indistinguishable from an error occurring in the initial frame.

9.2.3 Interframe space (IFS)

The time interval between frames is called the interframe space (IFS). A STA shall determine that the medium is idle through the use of the carrier sense function for the interval specified. Four different IFSs are defined to provide priority levels for access to the wireless media; they are listed in order, from the shortest to the longest. Figure 49 shows some of these relationships.

a) SIFS short interframe space
b) PIFS PCF interframe space
c) DIFS DCF interframe space
d) EIFS extended interframe space

The different IFSs shall be independent of the STA bit rate. The IFS timings shall be defined as time gaps on the medium, and shall be fixed for each PHY (even in multirate capable PHYs). The IFS values are determined from attributes specified in the PHY MIB.

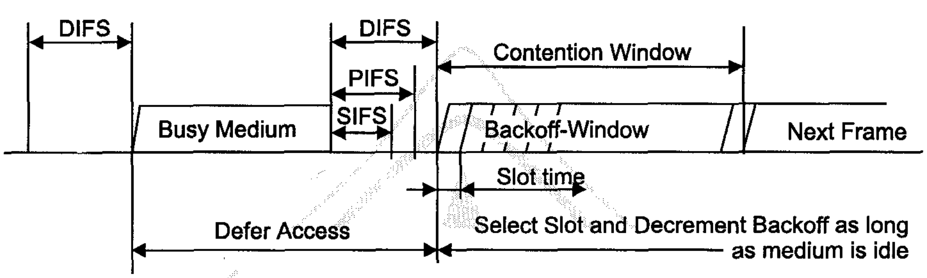


Figure 49—Some IFS relationships

9.2.3.1 Short IFS (SIFS)

The SIFS shall be used for an ACK frame, a CTS frame, the second or subsequent MPDU of a fragment burst, by a STA responding to any polling by the PCF, and may be used by a PC for any types of frames during the CFP (see 9.3). The SIFS is the time from the end of the last symbol of the previous frame to the beginning of the first symbol of the preamble of the subsequent frame as seen at the air interface. The valid cases where the SIFS may or shall be used are listed in the frame exchange sequences found in 9.7.

The SIFS timing shall be achieved when the transmission of the subsequent frame is started at the TxSIFS Slot boundary as specified in 9.2.10. An IEEE 802.11 implementation shall not allow the space between frames that are defined to be separated by a SIFS time, as measured on the medium, to vary from the nominal SIFS value by more than ±10% of aSlotTime for the PHY in use.

SIFS is the shortest of the interframe spaces. SIFS shall be used when STAs have seized the medium and need to keep it for the duration of the frame exchange sequence to be performed. Using the smallest gap between transmissions within the frame exchange sequence prevents other STAs, which are required to wait for the medium to be idle for a longer gap, from attempting to use the medium, thus giving priority to completion of the frame exchange sequence in progress.

9.2.3.2 PCF IFS (PIFS)

The PIFS shall be used only by STAs operating under the PCF to gain priority access to the medium at the start of the CFP. A STA using the PCF shall be allowed to transmit contention-free traffic after its carrier sense mechanism (see 9.2.1) determines that the medium is idle at the TxPIFS slot boundary as defined in 9.2.10. Subclause 9.3 describes the use of the PIFS by STAs operating under the PCF.

9.2.3.3 DCF IFS (DIFS)

The DIFS shall be used by STAs operating under the DCF to transmit data frames (MPDUs) and management frames (MMPDUs). A STA using the DCF shall be allowed to transmit if its carrier sense mechanism (see 9.2.1) determines that the medium is idle at the TxDIFS slot boundary as defined in 9.2.10 after a correctly received frame, and its backoff time has expired. A STA using the DCF shall not transmit within an EIFS after it determines that the medium is idle following reception of a frame for which the PHYRX-END.indication primitive contained an error or a frame for which the MAC FCS value was not correct. A STA may transmit after subsequent reception of an error-free frame, resynchronizing the STA. This allows the STA to transmit using the DIFS following that frame.

9.2.3.4 Extended IFS (EIFS)

The EIFS shall be used by the DCF whenever the PHY has indicated to the MAC that a frame transmission was begun that did not result in the correct reception of a complete MAC frame with a correct FCS value. The duration of an EIFS is defined in 9.2.10. The EIFS interval shall begin following indication by the PHY that the medium is idle after detection of the erroneous frame, without regard to the virtual carrier-sense mechanism. The EIFS is defined to provide enough time for another STA to acknowledge what was, to this STA, an incorrectly received frame before this STA commences transmission. Reception of an error-free frame during the EIFS resynchronizes the STA to the actual busy/idle state of the medium, so the EIFS is terminated and normal medium-access (using DIFS and, if necessary, backoff) continues following reception of that frame.

9.2.4 Random backoff time

A STA desiring to initiate transfer of data MPDUs and/or management MMPDUs shall invoke the carrier sense mechanism (see 9.2.1) to determine the busy/idle state of the medium. If the medium is busy, the STA shall defer until the medium is determined to be idle without interruption for a period of time equal to DIFS when the last frame detected on the medium was received correctly, or after the medium is determined to be idle without interruption for a period of time equal to EIFS when the last frame detected on the medium was not received correctly. After this DIFS or EIFS medium idle time, the STA shall then generate a random backoff period for an additional deferral time before transmitting, unless the backoff timer already contains a nonzero value, in which case the selection of a random number is not needed and not performed. This process minimizes collisions during contention between multiple STAs that have been deferring to the same event.

$$\text{Backoff Time} = \text{Random}() \times \text{aSlotTime}$$

where

Random() = Pseudorandom integer drawn from a uniform distribution over the interval [0,CW], where CW is an integer within the range of values of the MIB attributes aCWmin and aCWmax, aCWmin ≤ CW ≤ aCWmax. It is important that designers recognize the need for statistical independence among the random number streams among STAs.

aSlotTime = The value of the correspondingly named MIB attribute.

The contention window (CW) parameter shall take an initial value of aCWmin. Every STA shall maintain a STA short retry count (SSRC) as well as a STA long retry count (SLRC), both of which shall take an initial value of zero. The SSRC shall be incremented whenever any short retry count associated with any MSDU is incremented. The SLRC shall be incremented whenever any long retry count associated with any MSDU is incremented. The CW shall take the next value in the series every time an unsuccessful attempt to transmit an MPDU causes either STA retry counter to increment, until the CW reaches the value of aCWmax. A retry is defined as the entire sequence of frames sent, separated by SIFS intervals, in an attempt to deliver an MPDU, as described in 9.7. Once it reaches aCWmax, the CW shall remain at the value of aCWmax until it is reset. This improves the stability of the access protocol under high load conditions. See Figure 50.

The CW shall be reset to aCWmin after every successful attempt to transmit an MSDU or MMPDU, when SLRC reaches aLongRetryLimit, or when SSRC reaches aShortRetryLimit. The SSRC shall be reset to 0 whenever a CTS frame is received in response to an RTS frame, whenever an ACK frame is received in response to an MPDU or MMPDU transmission, or whenever a frame with a group address in the Address1 field is transmitted. The SLRC shall be reset to 0 whenever an ACK frame is received in response to transmission of an MPDU or MMPDU of length greater than aRTSThreshold, or whenever a frame with a group address in the Address1 field is transmitted.

The set of CW values shall be sequentially ascending integer powers of 2, minus 1, beginning with a PHY-specific aCWmin value, and continuing up to and including a PHY-specific aCWmax value.

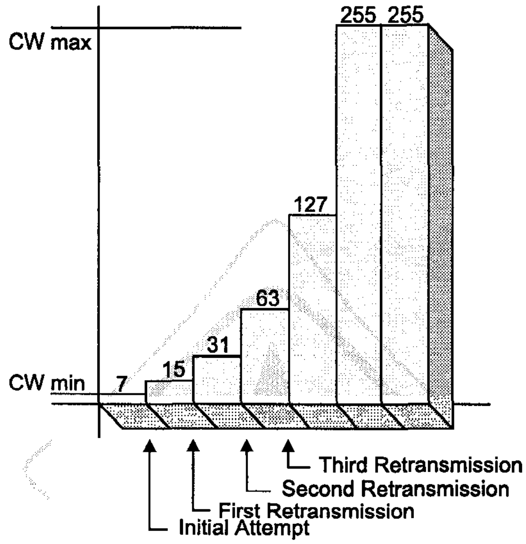


Figure 50—An example of exponential increase of CW

9.2.5 DCF access procedure

The CSMA/CA access method is the foundation of the DCF. The operational rules vary slightly between DCF and PCF.

9.2.5.1 Basic access

Basic access refers to the core mechanism a STA uses to determine whether it may transmit.

In general, a STA may transmit a pending MPDU when it is operating under the DCF access method, either in the absence of a PC, or in the CP of the PCF access method, when the STA determines that the medium is idle for greater than or equal to a DIFS period, or an EIFS period if the immediately preceding medium-busy event was caused by detection of a frame that was not received at this STA with a correct MAC FCS value. If, under these conditions, the medium is determined by the carrier sense mechanism to be busy when a STA desires to initiate the initial frame of one of the frame exchanges described in 9.7, exclusive of the CF period, the random backoff algorithm described in 9.2.5.2 shall be followed. There are conditions, specified elsewhere in Clause 9, where the random backoff algorithm shall be followed even for the first attempt to initiate a frame exchange sequence.

In a STA having an FH PHY, control of the channel is lost at a dwell time boundary and the STA shall have to contend for the channel after the dwell boundary. It is required that STAs having an FH PHY complete transmission of the entire MPDU and associated acknowledgment (if required) before the dwell time boundary. If, when transmitting or retransmitting an MPDU, there is not enough time remaining in the dwell to allow transmission of the MPDU plus the acknowledgment (if required), the STA shall defer the transmission by selecting a random backoff time, using the present CW (without advancing to the next value in the series). The short retry counter and long retry counter for the MSDU are not affected.

The basic access mechanism is illustrated in Figure 51.

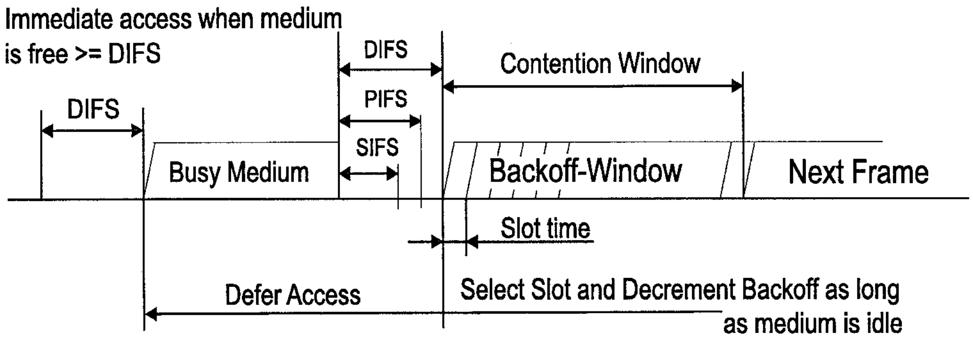


Figure 51—Basic access method

9.2.5.2 Backoff procedure

The backoff procedure shall be invoked for a STA to transfer a frame when finding the medium busy as indicated by either the physical or virtual carrier sense mechanism (see Figure 52). The backoff procedure shall also be invoked when a transmitting STA infers a failed transmission as defined in 9.2.5.7 or 9.2.8.

To begin the backoff procedure, the STA shall set its Backoff Timer to a random backoff time using the equation in 9.2.4. All backoff slots occur following a DIFS period during that the medium is determined to be idle for the duration of the DIFS period, or following an EIFS period during which the medium is determined to be idle for the duration of the EIFS period following detection of a frame that was not received correctly.

A STA performing the backoff procedure shall use the carrier sense mechanism (9.2.1) to determine whether there is activity during each backoff slot. If no medium activity is indicated for the duration of a particular backoff slot, then the backoff procedure shall decrement its backoff time by aSlotTime.

If the medium is determined to be busy at any time during a backoff slot, then the backoff procedure is suspended; that is, the backoff timer shall not decrement for that slot. The medium shall be determined to be idle for the duration of a DIFS period or EIFS, as appropriate (see 9.2.3), before the backoff procedure is allowed to resume. Transmission shall commence whenever the Backoff Timer reaches zero.

A backoff procedure shall be performed immediately after the end of every transmission with the More Fragments bit set to 0 of an MPDU of type Data, Management, or Control with subtype PS-Poll, even if no additional transmissions are currently queued. In the case of successful acknowledged transmissions, this backoff procedure shall begin at the end of the received ACK frame. In the case of unsuccessful transmissions requiring acknowledgment, this backoff procedure shall begin at the end of the ACK timeout interval. If the transmission was successful, the CW value reverts to aCWmin before the random backoff interval is chosen, and the STA short retry count and/or STA long retry count are updated as described in 9.2.4. This assures that transmitted frames from a STA are always separated by at least one backoff interval.

The effect of this procedure is that when multiple STAs are deferring and go into random backoff, then the STA selecting the smallest backoff time using the random function will win the contention.

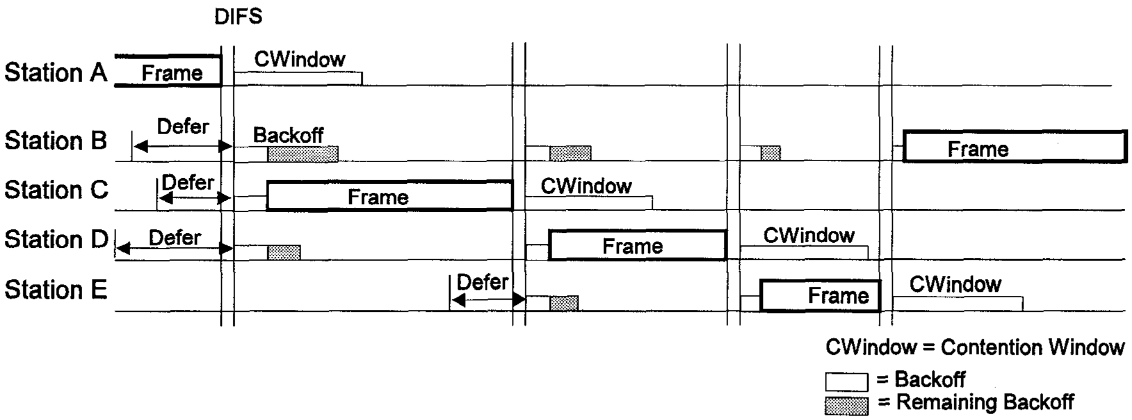


Figure 52—Backoff procedure

In an IBSS, the backoff time for a pending non-beacon or non-ATIM transmission shall not decrement in the period from the target beacon transmission time (TBTT) until the expiration of the ATIM window and the backoff time for a pending ATIM management frame shall decrement only within the ATIM window. (See Clause 11.) Within an IBSS, a separate backoff interval shall be generated to precede the transmission of a beacon, as described in 11.1.2.2.

9.2.5.3 Recovery procedures and retransmit limits

Error recovery is always the responsibility of the STA that initiates a frame exchange sequence, as defined in 9.7. Many circumstances may cause an error to occur that requires recovery. For example, the CTS frame may not be returned after an RTS frame is transmitted. This may happen due to a collision with another transmission, due to interference in the channel during the RTS or CTS frame, or because the STA receiving the RTS frame has an active virtual carrier sense condition (indicating a busy medium time period).

Error recovery shall be attempted by retrying transmissions for frame exchange sequences that the initiating STA infers have failed. Retries shall continue, for each failing frame exchange sequence, until the transmission is successful, or until the relevant retry limit is reached, whichever occurs first. STAs shall maintain a short retry count and a long retry count for each MSDU or MMPDU awaiting transmission. These counts are incremented and reset independently of each other.

After an RTS frame is transmitted, the STA shall perform the CTS procedure, as defined in 9.2.5.7. If the RTS transmission fails, the short retry count for the MSDU or MMPDU and the STA short retry count are incremented. This process shall continue until the number of attempts to transmit that MSDU or MMPDU reaches aShortRetryLimit.

After transmitting a frame that requires acknowledgment, the STA shall perform the ACK procedure, as defined in 9.2.8. The short retry count for an MSDU or MMPDU and the STA short retry count shall be incremented every time transmission of a MAC frame of length less than or equal to aRTSThreshold fails for that MSDU or MMPDU. This short retry count and the STA short retry count shall be reset when a MAC frame of length less than or equal to aRTSThreshold succeeds for that MSDU or MMPDU. The long retry count for an MSDU or MMPDU and the STA long retry count shall be incremented every time transmission of a MAC frame of length greater than aRTSThreshold fails for that MSDU or MMPDU. This long retry count and the STA long retry count shall be reset when a MAC frame of length greater than aRTSThreshold succeeds for that MSDU or MMPDU. All retransmission attempts for an MSDU or MMPDU that has failed the ACK procedure one or more times shall be made with the Retry field set to 1 in the Data or Management type frame.

Retries for failed transmission attempts shall continue until the short retry count for the MSDU or MMPDU is equal to aShortRetryLimit or until the long retry count for the MSDU or MMPDU is equal to aLongRetryLimit. When either of these limits is reached, retry attempts shall cease, and the MSDU or MMPDU shall be discarded.

A STA in power-save mode, in an ESS, initiates a frame exchange sequence by transmitting a PS-Poll frame to request data from an AP. In the event that neither an ACK frame nor a data frame is received from the AP in response to a PS-Poll frame, then the STA shall retry the sequence, by transmitting another PS-Poll frame, at its convenience. If the AP sends a data frame in response to a PS-Poll frame, but fails to receive the ACK frame acknowledging this data frame, the next PS-Poll frame from the same STA may cause a retransmission of the last MSDU. This duplicate MSDU shall be filtered at the receiving STA using the normal duplicate frame filtering mechanism. If the AP responds to a PS-Poll by transmitting an ACK frame, then responsibility for the data frame delivery error recovery shifts to the AP because the data is transferred in a subsequent frame exchange sequence, which is initiated by the AP. The AP shall attempt to deliver one MSDU to the STA that transmitted the PS-Poll, using any frame exchange sequence valid for a directed MSDU. If the power save STA that transmitted the PS-Poll returns to Doze state after transmitting the ACK frame in response to successful receipt of this MSDU, but the AP fails to receive this ACK frame, the AP will retry transmission of this MSDU until the relevant retry limit is reached. See Clause 11 for details on filtering of extra PS-Poll frames.

9.2.5.4 Setting and resetting the NAV

STAs receiving a valid frame shall update their NAV with the information received in the Duration/ID field, but only when the new NAV value is greater than the current NAV value and only when the frame is not addressed to the receiving STA. Various additional conditions may set or reset the NAV, as described in 9.3.2.2. When the NAV is reset, a PHY-CCARESET.request shall be issued.

Figure 53 indicates the NAV for STAs that may hear the RTS frame, while other STAs may only receive the CTS frame, resulting in the lower NAV bar as shown (with the exception of the STA to which the RTS was addressed).

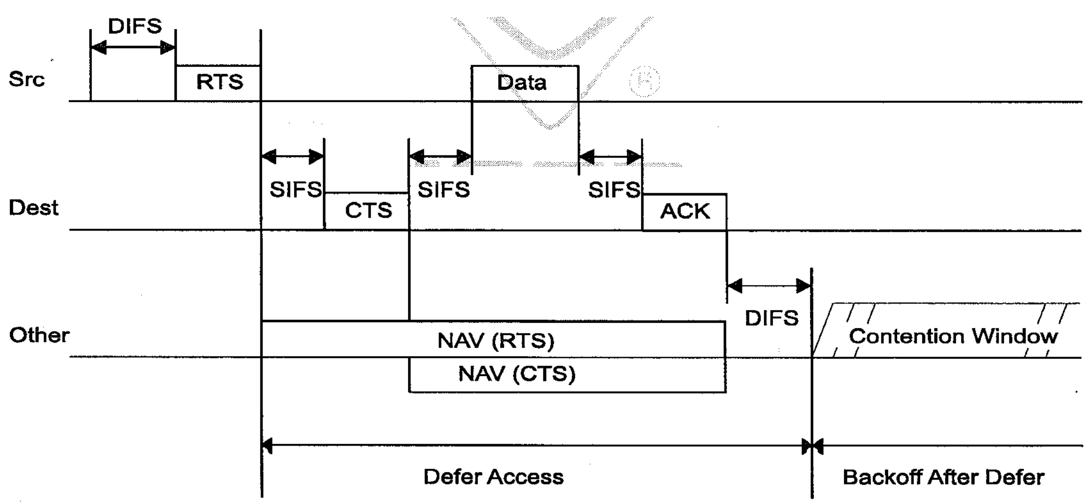


Figure 53—RTS/CTS/data/ACK and NAV setting

A STA that used information from an RTS frame as the most recent basis to update its NAV setting is permitted to reset its NAV if no PHY-RXSTART.indication is detected from the PHY during a period with a duration of (2 × aSIFSTime) + (CTS_Time) + (2 × aSlotTime) starting at the PHY-RXEND.indication corresponding to the detection of the RTS frame. The "CTS_Time" shall be calculated using the length of the CTS frame and the data rate at which the RTS frame used for the most recent NAV update was received.

9.2.5.5 Control of the channel

The SIFS is used to provide an efficient MSDU delivery mechanism. Once the STA has contended for the channel, that STA shall continue to send fragments until either all fragments of a single MSDU or MMPDU have been sent, an acknowledgment is not received, or the STA is restricted from sending any additional fragments due to a dwell time boundary. Should the sending of the fragments be interrupted due to one of these reasons, when the next opportunity for transmission occurs the STA shall resume transmission. The algorithm by which the STA decides which of the outstanding MSDUs shall next be attempted after an unsuccessful transmission attempt is beyond the scope of this standard, but any such algorithm shall comply with the restrictions listed in 9.8.

Figure 54 illustrates the transmission of a multiple-fragment MSDU using the SIFS.

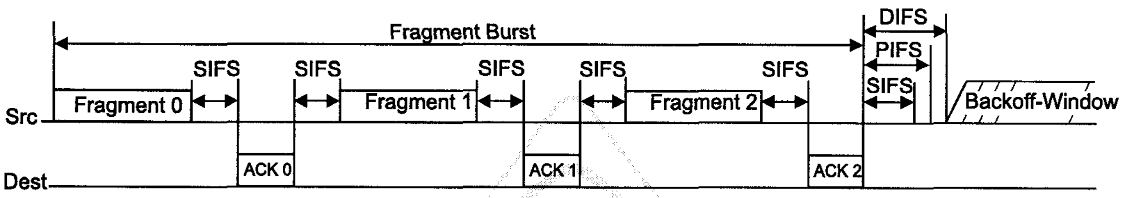


Figure 54—Transmission of a multiple-fragment MSDU using SIFS

When the source STA transmits a fragment, it shall release the channel, then immediately monitor the channel for an acknowledgment as described in 9.2.8.

When the destination STA has finished sending the acknowledgment, the SIFS following the acknowledgment shall be reserved for the source STA to continue (if necessary) with another fragment. The STA sending the acknowledgment shall not transmit on the channel immediately following the acknowledgment.

The process of sending multiple fragments after contending for the channel is defined as a fragment burst.

If the source STA receives an acknowledgment but there is not enough time to transmit the next fragment and receive an acknowledgment due to an impending dwell boundary, it shall contend for the channel at the beginning of the next dwell time.

If the source STA does not receive an acknowledgment frame, it shall attempt to retransmit the failed MPDU or another eligible MPDU, as defined in 9.8, after performing the backoff procedure and the contention process.

After a STA contends for the channel to retransmit a fragment of an MSDU, it shall start with the last fragment that was not acknowledged. The destination STA shall receive the fragments in order (since the source sends them in order, and they are individually acknowledged). It is possible, however, that the destination STA may receive duplicate fragments. It shall be the responsibility of the receiving STA to detect and discard duplicate fragments.

A STA shall transmit after the SIFS only under the following conditions during a fragment burst:

— The STA has just received a fragment that requires acknowledging.

— The source STA has received an acknowledgment to a previous fragment, has more fragment(s) for the same MSDU to transmit, and there is enough time before the next dwell boundary to send the next fragment and receive its acknowledgment.

The following rules shall also apply:

— When a STA has transmitted a frame other than an initial or intermediate fragment, that STA shall not transmit on the channel following the acknowledgment for that frame, without performing the backoff procedure.
— When an MSDU has been successfully delivered or all retransmission attempts have been exhausted, and the STA has a subsequent MSDU to transmit, then the STA shall perform a backoff procedure.
— Only unacknowledged fragments shall be retransmitted.

9.2.5.6 RTS/CTS usage with fragmentation

The following is a description of using RTS/CTS for a fragmented MSDU or MMPDU. The RTS/CTS frames define the duration of the following frame and acknowledgment. The Duration/ID field in the data and acknowledgment (ACK) frames specifies the total duration of the next fragment and acknowledgment. This is illustrated in Figure 55.

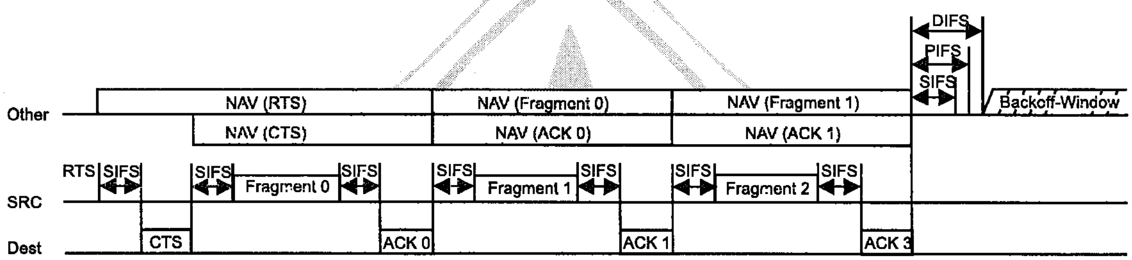


Figure 55—RTS/CTS with fragmented MSDU

Each frame contains information that defines the duration of the next transmission. The duration information from RTS frames shall be used to update the NAV to indicate busy until the end of ACK 0. The duration information from the CTS frame shall also be used to update the NAV to indicate busy until the end of ACK 0. Both Fragment 0 and ACK 0 shall contain duration information to update the NAV to indicate busy until the end of ACK 1. This shall be done by using the Duration/ID field in the Data and ACK frames. This shall continue until the last fragment, which shall have a duration of one ACK time plus one SIFS time, and its ACK, which shall have its Duration/ID field set to zero. Each fragment and ACK acts as a virtual RTS and CTS; therefore no further RTS/CTS frames need to be generated after the RTS/CTS that began the frame exchange sequence even though subsequent fragments may be larger than aRTSThreshold. At STAs using a frequency-hopping PHY, when there is insufficient time before the next dwell boundary to transmit the subsequent fragment, the STA initiating the frame exchange sequence may set the Duration/ID field in the last data or management frame to be transmitted before the dwell boundary to the duration of one ACK time plus one SIFS time.

In the case where an acknowledgment is sent but not received by the source STA, STAs that heard the fragment, or ACK, will mark the channel busy for the next frame exchange due to the NAV having been updated from these frames. This is the worst-case situation, and it is shown in Figure 56. If an acknowledgment is not sent by the destination STA, STAs that can only hear the destination STA will not update their NAV and may attempt to access the channel when their NAV updated from the previously received frame reaches zero. All STAs that hear the source will be free to access the channel after their NAV updated from the transmitted fragment has expired.

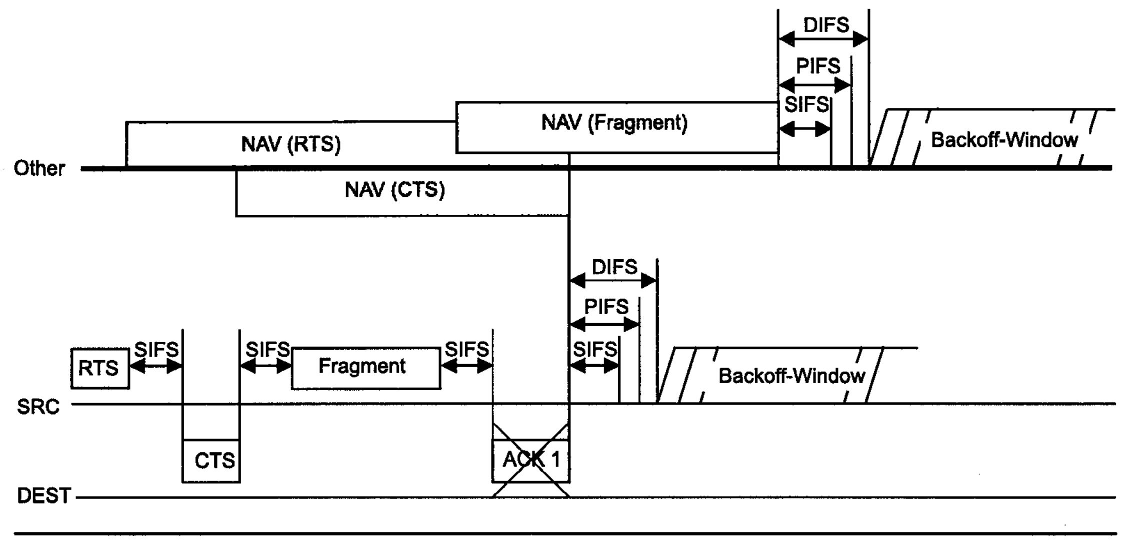


Figure 56—RTS/CTS with transmitter priority and missed acknowledgment

9.2.5.7 CTS procedure

A STA that is addressed by an RTS frame shall transmit a CTS frame after a SIFS period if the NAV at the STA receiving the RTS frame indicates that the medium is idle. If the NAV at the STA receiving the RTS indicates the medium is not idle, that STA shall not respond to the RTS frame. The RA field of the CTS frame shall be the value obtained from the TA field of the RTS frame to which this CTS frame is a response. The Duration/ID field in the CTS frame shall be the duration field from the received RTS frame, adjusted by subtraction of aSIFSTime and the number of microseconds required to transmit a CTS frame at the data rate used for the RTS frame to which this CTS frame is a response.

After transmitting an RTS frame, the STA shall wait for a CTSTimeout interval, starting at the PHY-TXEND.confirm. If a PHY-RXSTART.indication does not occur during the CTSTimeout interval, the STA shall conclude that the transmission of the RTS has failed, and this STA shall invoke its backoff procedure upon expiration of the CTSTimeout interval. If a PHY-RXSTART.indication does occur during the CTSTimeout interval, the STA shall wait for the corresponding PHY-RXEND.indication to determine whether the RTS transmission was successful. The recognition of a valid CTS frame sent by the recipient of the RTS frame, corresponding to this PHY-RXEND.indication, shall be interpreted as successful response, permitting the frame sequence to continue (see 9.7). The recognition of anything else, including any other valid frame, shall be interpreted as failure of the RTS transmission. In this instance, the STA shall invoke its backoff procedure at the PHY-RXEND.indication and may process the received frame.

9.2.6 Directed MPDU transfer procedure

A STA shall use an RTS/CTS exchange for directed frames only when the length of the MPDU is greater than the length threshold indicated by the aRTSThreshold attribute.

The aRTSThreshold attribute shall be a managed object within the MAC MIB, and its value may be set and retrieved by the MAC LME. The value 0 shall be used to indicate that all MPDUs shall be delivered with the use of RTS/CTS. Values of aRTSThreshold larger than the maximum MSDU length shall indicate that all MPDUs shall be delivered without RTS/CTS exchanges.

When an RTS/CTS exchange is used, the asynchronous data frame shall be transmitted after the end of the CTS frame and a SIFS period. No regard shall be given to the busy or idle status of the medium when transmitting this data frame.

When an RTS/CTS exchange is not used, the asynchronous data frame shall be transmitted following the success of the basic access procedure. With or without the use of the RTS/CTS exchange procedure, the STA that is the destination of an asynchronous data frame shall follow the ACK procedure.

9.2.7 Broadcast and multicast MPDU transfer procedure

In the absence of a PCF, when broadcast or multicast MPDUs are transferred from a STA with the ToDS bit clear, only the basic access procedure shall be used. Regardless of the length of the frame, no RTS/CTS exchange shall be used. In addition, no ACK shall be transmitted by any of the recipients of the frame. Any broadcast or multicast MPDUs transferred from a STA with a ToDS bit set shall, in addition to conforming to the basic access procedure of CSMA/CA, obey the rules for RTS/CTS exchange, because the MPDU is directed to the AP. The broadcast/multicast message shall be distributed into the BSS. The STA originating the message shall receive the message as a broadcast/multicast message. Therefore, all STAs shall filter out broadcast/multicast messages that contain their address as the source address. Broadcast and multicast MSDUs shall be propagated throughout the ESS.

There is no MAC-level recovery on broadcast or multicast frames, except for those frames sent with the ToDS bit set. As a result, the reliability of this traffic is reduced, relative to the reliability of directed traffic, due to the increased probability of lost frames from interference or collisions or time-varying channel properties.

9.2.8 ACK procedure

An ACK frame shall be generated as shown in the frame exchange sequences listed in 9.7.

Upon successful reception of a frame of a type that requires acknowledgment with the ToDS bit set, an AP shall generate an ACK frame. An ACK frame shall be transmitted by the destination STA that is not an AP, whenever it successfully receives a unicast frame of a type that requires acknowledgment, but not if it receives a broadcast or multicast frame of such type. After a successful reception of a frame requiring acknowledgment, transmission of the ACK frame shall commence after a SIFS period, without regard to the busy/idle state of the medium.

The source STA shall wait ACKTimeout amount of time without receiving an ACK frame before concluding that the MPDU failed. (See Figure 57.)

After transmitting an MPDU that requires an ACK frame as a response (see 9.7), the STA shall wait for an ACK-Timeout interval, starting at the PHY-TXEND.confirm. If a PHY-RXSTART.indication does not occur during the ACKTimeout interval, the STA concludes that the transmission of the MPDU has failed, and this STA shall invoke its backoff procedure upon expiration of the ACKTimeout interval. If a PHY-RXSTART.indication does occur during the ACKTimeout interval, the STA shall wait for the corresponding PHY-RXEND.indication to determine whether the MPDU transmission was successful. The recognition of a valid ACK frame sent by the recipient of the MPDU requiring acknowledgment, corresponding to this PHY-RXEND.indication, shall be interpreted as successful acknowledgment, permitting the frame sequence to continue, or to end without retries, as appropriate for the particular frame sequence in progress. The recognition of anything else, including any other valid frame, shall be interpreted as failure of the MPDU transmission. In this instance, the STA shall invoke its backoff procedure at the PHY-RXEND.indication and may process the received frame. The sole exception is that recognition of a valid data frame sent by the recipient of a PS-Poll frame shall also be accepted as successful acknowledgment of the PS-Poll frame.

9.2.9 Duplicate detection and recovery

Since MAC-level acknowledgments and retransmissions are incorporated into the protocol, there is the possibility that a frame may be received more than once. Such duplicate frames shall be filtered out within the destination MAC.

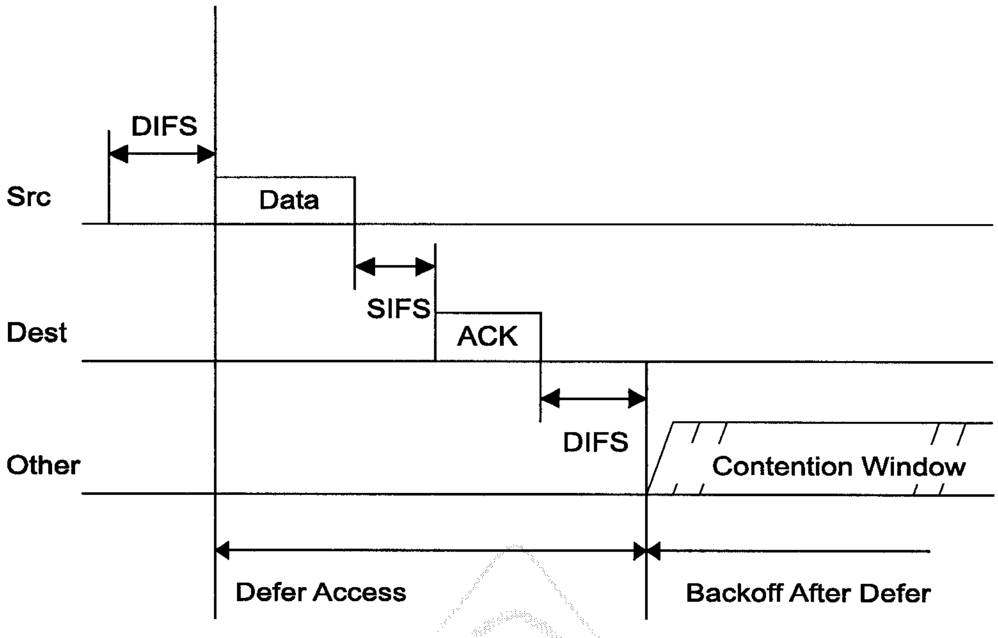


Figure 57—Directed data/ACK MPDU

Duplicate frame filtering is facilitated through the inclusion of a Sequence Control field (consisting of a sequence number and fragment number) within data and management frames. MPDUs that are part of the same MSDU shall have the same sequence number, and different MSDUs shall (with a high probability) have a different sequence number.

The sequence number is generated by the transmitting STA as an incrementing sequence of integers.

The receiving STA shall keep a cache of recently received <Address 2, sequence-number, fragment-number> tuples. A receiving STA is required to keep only the most recent cache entry per Address 2–sequence-number pair, storing only the most recently received fragment number for that pair. A receiving STA may omit tuples obtained from broadcast/multicast or ATIM frames from the cache.

A destination STA shall reject as a duplicate frame any frame that has the Retry bit set in the Frame Control field and that matches an <Address 2, sequence-number, and fragment-number> tuple of an entry in the cache.

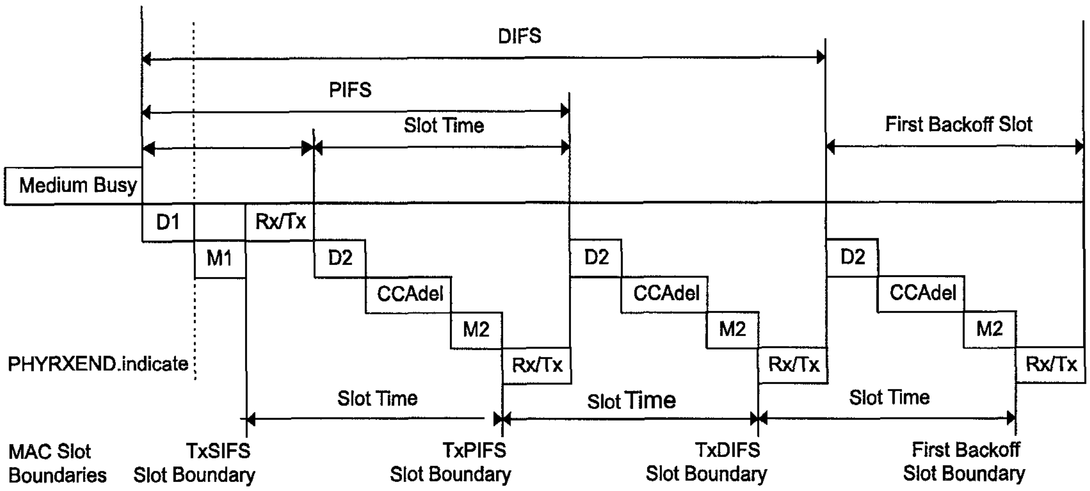
There is a small possibility that a frame may be improperly rejected due to such a match; however, this occurrence would be rare and simply results in a lost frame (similar to an FCS error in other LAN protocols).

The destination STA shall perform the ACK procedure on all successfully received frames requiring acknowledgment, even if the frame is discarded due to duplicate filtering.

9.2.10 DCF timing relations

The relationships between the IFS specifications are defined as time gaps on the medium. The associated MIB attributes are provided by the specific PHY. (See Figure 58.)

All timings that are referenced from the end of the transmission are referenced from the end of the last symbol of a frame on the medium. The beginning of transmission refers to the first symbol of the next frame on the medium.



D1 = aRxRFDelay + aRxPLCPDelay (referenced from the end of the last symbol of a frame on the medium)
D2 = D1 + Air Propagation Time
Rx/Tx = aRXTXTurnaroundTime (begins with a PHYTXSTART.request)
M1 = M2 = aMACPrcDelay
CCAdel = aCCA Time - D1

Figure 58—DCF timing relationships

aSIFSTime and aSlotTime are defined in the MIB, and are fixed per PHY.

aSIFSTime is: aRxRFDelay + aRxPLCPDelay + aMACPrcDelay + aRxTxTurnaroundTime.

aSlotTime is: aCCATime + aRxTxTurnaroundTime + aAirPropagationTime + aMACProcessingDelay.

The PIFS and DIFS are derived by the following equations, as illustrated in Figure 58.

PIFS = aSIFSTime + aSlotTime

DIFS = aSIFSTime + 2 × aSlotTime

The EIFS is derived from the SIFS and the DIFS and the length of time it takes to transmit an ACK Control frame at 1 Mbit/s by the following equation:

EIFS = aSIFSTime + (8 × ACKSize) + aPreambleLength + aPLCPHeaderLngth+ DIFS

where

ACKSize is the length, in bytes, of an ACK frame; and

(8 × ACKSize)+ aPreambleLength + aPLCPHeaderLngth is expressed in microseconds required to transmit at the PHY's lowest mandatory rate.

Figure 58 illustrates the relation between the SIFS, PIFS, and DIFS as they are measured on the medium and the different MAC slot boundaries TxSIFS, TxPIFS, and TxDIFS. These slot boundaries define when the transmitter shall be turned on by the MAC to meet the different IFS timings on the medium, after subsequent detection of the CCA result of the previous slot time.

The following equations define the MAC slot boundaries, using attributes defined in the MIB, which are such that they compensate for implementation timing variations. The starting reference of these slot boundaries is again the end of the last symbol of the previous frame on the medium.

TxSIFS = SIFS – aRxTxTurnaroundTime

TxPIFS = TxSIFS + aSlotTime

TxDIFS = TxSIFS + 2 x aSlotTime.

The tolerances are specified in the PHY MIB, and shall only apply to the SIFS specification, so that tolerances shall not accumulate.

9.3 PCF

The PCF provides contention-free frame transfer. The PC shall reside in the AP. It is an option for an AP to be able to become the PC. All STAs inherently obey the medium access rules of the PCF, because these rules are based on the DCF, and they set their NAV at the beginning of each CFP. The operating characteristics of the PCF are such that all STAs are able to operate properly in the presence of a BSS in which a PC is operating, and, if associated with a point-coordinated BSS, are able to receive all frames sent under PCF control. It is also an option for a STA to be able to respond to a contention free poll (CF-Poll) received from a PC. A STA that is able to respond to CF-Polls is referred to as being CF-Pollable, and may request to be polled by an active PC. CF-Pollable STAs and the PC do not use RTS/CTS in the CFP. When polled by the PC, a CF-Pollable STA may transmit only one MPDU, which can be to any destination (not just to the PC), and may "piggyback" the acknowledgment of a frame received from the PC using particular data frame subtypes for this transmission. If the data frame is not in turn acknowledged, the CF-Pollable STA shall not retransmit the frame unless it is polled again by the PC, or it decides to retransmit during the CP. If the addressed recipient of a CF transmission is not CF-Pollable, that STA acknowledges the transmission using the DCF acknowledgment rules, and the PC retains control of the medium. A PC may use contention-free frame transfer solely for delivery of frames to STAs, and never to poll non-CF-Pollable STAs.

A PC may perform a backoff on retransmission of an unacknowledged frame during the CFP. A PC that is maintaining a polling list may retry the unacknowledged frame the next time the particular AID is at the top of the polling list.

A PC may retransmit an unacknowledged frame during the CFP after a PIFS time.

When more than one point-coordinated BSS is operating on the same PHY channel in overlapping space, the potential exists for collisions between PCF transfer activities by the independent PCs. The rules under which multiple, overlapping point-coordinated BSSs may coexist are presented in 9.3.3.2. As shown in Figure 47, the PCF is built on top of the CSMA/CA-based DCF, by utilizing the access priority provisions provided by this scheme. An active PC shall be located at an AP, which restricts PCF operation to infrastructure networks. PCF is activated at a PC-capable AP by setting the CFPMaxDuration parameter in the CF Parameter Set of the MLMEStart.request to a non-zero value.

Data frames sent during under the DCF shall use the data subtypes Data or Null Function. Data frames sent by, or in response to polling by, the PC during the CFP shall use the appropriate data subtypes based upon the usage rules:

— Data+CF-Poll, Data+CF-Ack+CF-Poll, CF-Poll, and CF-Ack+CF-Poll shall only be sent by a PC.
— Data, Data+CF-Ack, Null Function, and CF-Ack may be sent by a PC or by any CF-Pollable STA.

STAs receiving Data type frames shall only consider the frame body as the basis of a possible indication to LLC, if the frame is of subtype Data, Data+CF-Ack, Data+CF-Poll, or Data+CF-Ack+CF-Poll. CF-Pollable STAs shall interpret all subtype bits of received Data type frames for CF purposes, but shall only inspect the frame body if the frame is of subtype Data, Data+CF-Ack, Data+CF-Poll, or Data+CF-Ack+CF-Poll.

9.3.1 CFP structure and timing

The PCF controls frame transfers during a CFP. The CFP shall alternate with a CP, when the DCF controls frame transfers, as shown in Figure 59. Each CFP shall begin with a Beacon frame that contains a DTIM element (hereafter referred to as a "DTIM"). The CFPs shall occur at a defined repetition rate, which shall be synchronized with the beacon interval as specified in the following paragraphs.

The PC generates CFPs at the *contention-free repetition rate* (CFPRate), which is defined as a number of DTIM intervals. The PC shall determine the CFPRate (depicted as a repetition interval in the illustrations in Figure 59 and Figure 60) to use from the CFPRate parameter in the CF Parameter Set. This value, in units of DTIM intervals, shall be communicated to other STAs in the BSS in the CFPPeriod field of the CF Parameter Set element of Beacon frames. The CF Parameter Set element shall only be present in Beacon and Probe Response frames transmitted by STAs containing an active PC.

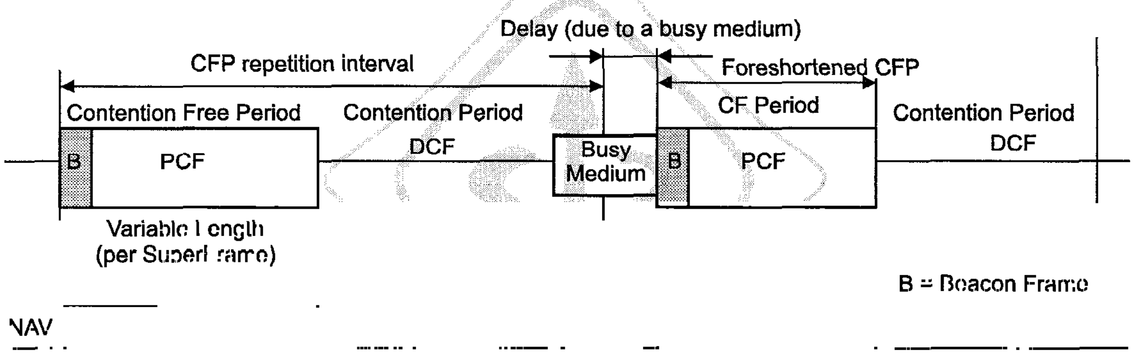


Figure 59—CFP/CP alternation

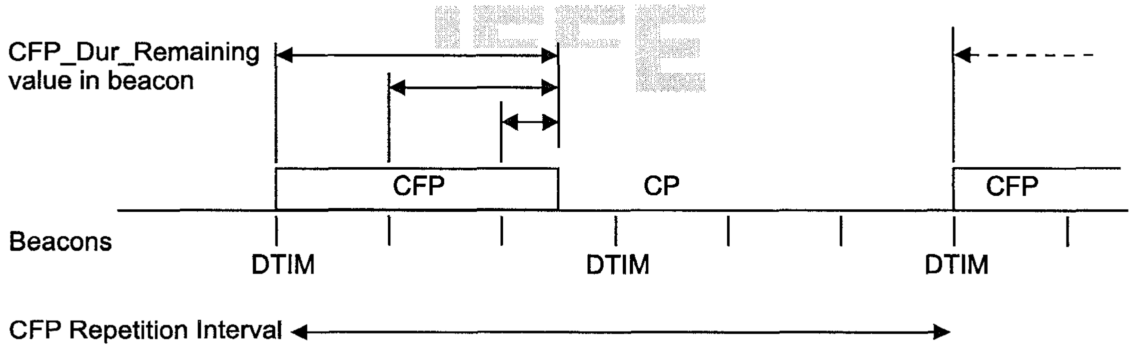


Figure 60—Beacons and CFPs

The length of the CFP is controlled by the PC, with maximum duration specified by the value of the CFP-MaxDuration Parameter in the CF Parameter Set at the PC. Neither the maximum duration nor the actual duration (signaled by transmission of a Control frame of subtype CF-End or CF-End+ACK by the PC) is constrained to be a multiple of the beacon interval. If the CFP duration is greater than the beacon interval, the PC shall transmit beacons at the appropriate times during the CFP (subject to delay due to traffic at the nominal times, as with all beacons). The CF Parameter Set element in all beacons at the start of, or within, a CFP shall contain a nonzero value in the CFPDurRemaining field. This value, in units of TU, shall specify the maximum time from the transmission of this beacon to the end of this CFP. The value of the CFPDurRemaining field shall be zero in beacons sent during the CP. An example of these relationships is illustrated in Figure 60, which shows a case where the CFP is two DTIM intervals, the DTIM interval is three beacon intervals, and the aCFPMaxDuration value is approximately 2.5 beacon intervals.

The PC may terminate any CFP at or before the aCFPMaxDuration, based on available traffic and size of the polling list. Because the transmission of any beacon may be delayed due to a medium busy condition at the nominal beacon transmission time, a CFP may be foreshortened by the amount of the delay. In the case of a busy medium due to DCF traffic, the beacon shall be delayed for the time required to complete the current DCF frame exchange. In cases where the beacon transmission is delayed, the CFPDurRemaining value in the beacon at the beginning of the CFP shall specify a time that causes the CFP to end no later than TBTT plus the value of aCFPMaxDuration. This is illustrated in Figure 61.

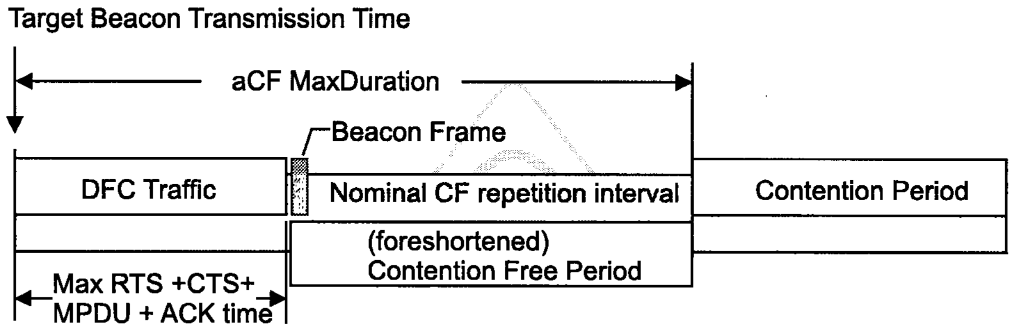


Figure 61—Example of delayed beacon and foreshortened CFP

9.3.2 PCF access procedure

The contention-free transfer protocol is based on a polling scheme controlled by a PC operating at the AP of the BSS. The PC gains control of the medium at the beginning of the CFP and attempts to maintain control for the entire CFP by waiting a shorter time between transmissions than the STAs using the DCF access procedure. All STAs in the BSS (other than the PC) set their NAVs to the CFPMaxDuration value at the nominal start time of each CFP. This prevents most contention by preventing non-polled transmissions by STAs whether or not they are CF-Pollable. Acknowledgment of frames sent during the CFP may be accomplished using Data+CF-ACK, CF-ACK, Data+CF-ACK+CF-Poll (only on frames transmitted by the PC), or CF-ACK+CF-Poll (only on frames transmitted by the PC) frames in cases where a Data (or Null) frame immediately follows the frame being acknowledged, thereby avoiding the overhead of separate ACK Control frames. Non-CF-Pollable or unpolled CF-Pollable STAs acknowledge frames during the CFP using the DCF ACK procedure.

9.3.2.1 Fundamental access

At the nominal beginning of each CFP, the PC shall sense the medium. When the medium is determined to be idle for one PIFS period, the PC shall transmit a Beacon frame containing the CF Parameter Set element and a DTIM element.

After the initial beacon frame, the PC shall wait for at least one SIFS period then transmit one of the following: a data frame, a CF-Poll frame, a Data+CF-Poll frame, or a CF-End frame. If the CFP is null, i.e., there is no traffic buffered and no polls to send at the PC, a CF-End frame shall be transmitted immediately after the initial beacon.

STAs receiving directed, error-free frames from the PC are expected to respond after a SIFS period, in accordance with the transfer procedures defined in 9.3.3. If the recipient STA is not CF-Pollable, the response to receipt of an error-free data frame shall always be an ACK frame.

9.3.2.2 NAV operation during the CFP

The mechanism for handling the NAV during the CFP is designed to facilitate the operation of overlapping CFP coordinated infrastructure BSSs. The mechanism by which infrastructure BSSs coordinate their CFPs is beyond the scope of this standard.

Each STA, except the STA with the PC, shall preset its NAV to the CFPMaxDuration value (obtained from the CF Parameter Set element in beacons from this PC) at each target beacon transmission time (TBTT) (see Clause 11) at which a CFP is scheduled to start (based on the CFPPeriod field in the CF Parameter Set element of the Beacon frames from this PC). Each non-PC STA shall update its NAV using the CFDurRemaining value in any error-free CF Parameter Set element of the Beacon frame that the STA receives. This includes CFDurRemaining values in CF Parameter Set elements from Beacon frames received from other (overlapping) BSSs.

These actions prevent STAs from taking control of the medium during the CFP, which is especially important in cases where the CFP spans multiple medium-occupancy intervals, such as dwell periods of an FH PHY. This setting of the NAV also reduces the risk of hidden STAs determining the medium to be idle for a DIFS period during the CFP and possibly corrupting a transmission in progress.

A STA joining a BSS operating with a PC shall use the information in the CFDurRemaining element of the CF parameter set of any received Beacon or Probe Response frames to update its NAV prior to initiating any transmissions.

The PC shall transmit a CF-End or CF-End+ACK frame at the end of each CFP. A STA that receives either of these frames, from any BSS, shall reset its NAV.

9.3.3 PCF transfer procedure

Frame transfers under the PCF typically consist of frames alternately sent from the AP/PC and sent to the AP/PC. During the CFP, the ordering of these transmissions, and the STA allowed to transmit frames to the PC at any given point in time, shall be controlled by the PC. Figure 62 depicts a frame transfer during a typical CFP. The rules under which this frame transfer takes place are detailed in the following subclauses.

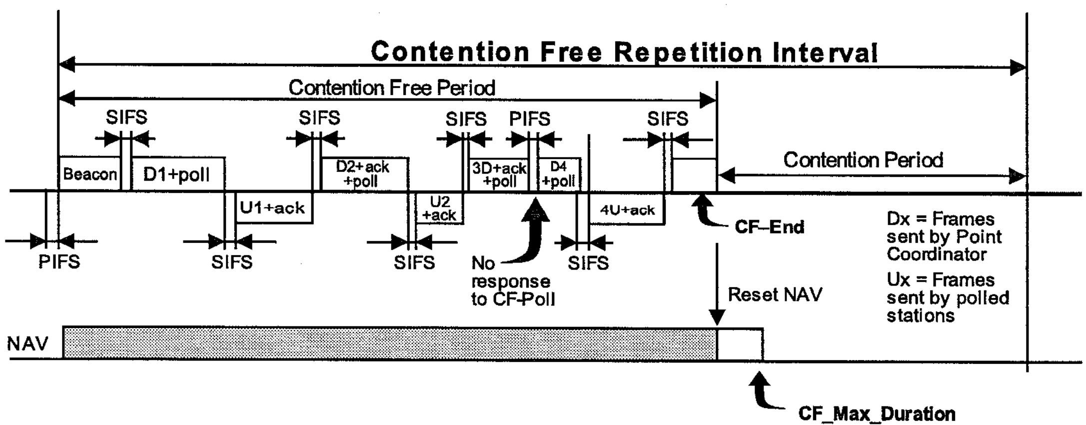


Figure 62—Example of PCF frame transfer

In a STA having an FH PHY, control of the channel is lost at a dwell time boundary. It is required that the current MPDU transmission and the accompanying acknowledgment of the MPDU be transmitted before the dwell time boundary. After having been polled by the PC, if there is not enough time remaining in the dwell to allow transmission of the MPDU plus the acknowledgment, the STA shall defer the transmission of the MPDU and shall transmit a Null frame or CF-ACK frame. The short retry counter and long retry counter for the MSDU shall not be affected.

In a STA having an FH PHY, the PC shall not transmit a CF-Poll to a STA if there is insufficient time remaining before the dwell boundary for the STA to respond with a Null frame or CF-ACK frame.

9.3.3.1 PCF transfers when the PCF STA is transmitter or recipient

The PC shall transmit frames between the Beacon that starts the CFP and the CF-End using the SIFS except in cases where a transmission by another STA is expected by the PC and a SIFS period elapses without the receipt of the expected transmission. In such cases the PC may send its next pending transmission as soon as one PIFS after the end of its last transmission. This permits the PC to retain control of the medium in the presence of an overlapping BSS. The PC may transmit any of the following frame types to CF-Pollable STAs:

— Data, used to send data from the PC when the addressed recipient is not being polled and there is no previous frame to acknowledge;
— Data+CF-ACK, used to send data from the PC when the addressed recipient is not being polled and the PC needs to acknowledge the receipt of a frame received from a CF-Pollable STA a SIFS period before starting this transmission;
— Data+CF-Poll, used to send data from the PC when the addressed recipient is the next STA to be permitted to transmit during this CFP and there is no previous frame to acknowledge;
— Data+CF-ACK+CF-Poll, used to send data from the PC when the addressed recipient is the next STA to be permitted to transmit during this CFP and the PC needs to acknowledge the receipt of a frame received from a CF-Pollable STA a SIFS period before starting this transmission;
— CF-Poll, used when the PC is not sending data to the addressed recipient, but the addressed recipient is the next STA to be permitted to transmit during this CFP and there is no previous frame to acknowledge;
— CF-ACK+CF-Poll, used when the PC is not sending data to the addressed recipient but the addressed recipient is the next STA to be permitted to transmit during this CFP and the PC needs to acknowledge the receipt of a frame from a CF-Pollable STA a SIFS period before starting this transmission;
— CF-ACK, used when the PC is not sending data to, or polling, the addressed recipient, but the PC needs to acknowledge receipt of a frame from a CF-Pollable STA a SIFS period before starting this transmission (useful when the next transmission by the PC is a management frame, such as a beacon); or
— Any management frame that is appropriate for the AP to send under the rules for that frame type.

The PC may transmit data or management frames to non-CF-Pollable, non-power-save STAs during the CFP. These STAs shall acknowledge receipt with ACK frames after a SIFS, as with the DCF. The PC may also transmit broadcast or multicast frames during the CFP. Because the Beacon frame that initiates the CFP contains a DTIM element, if there are associated STAs using power-save mode, the broadcasts and multicasts buffered shall be sent immediately after any beacon containing a TIM element with a DTIM count field with a value of 0.

A CF-Pollable STA that receives a directed data frame with any of subtype that includes CF-Poll may transmit one data frame a SIFS period after receiving the CF-Poll. CF-Pollable STAs shall ignore, but not reset, their NAV when performing transmissions in response to a CF-Poll.

Non-CF-Pollable STAs that receive a directed frame during the CFP shall transmit an ACK, but shall not reset their NAV.

For frames that require MAC-level acknowledgment, CF-Pollable STAs that received a CF-Poll (of any type) may perform this acknowledgment using the Data+CF-ACK subtype in the response to the CF-Poll. For example, the U1 frame in Figure 62 contains the acknowledgment to the preceding D1 frame. Also the D2 frame contains the acknowledgment to the preceding U1 frame. The PC may use the CF-ACK subtypes to acknowledge a received frame even if the data frame sent with the CF-ACK subtype is addressed to a different STA than the one being acknowledged. CF-Pollable STAs that are expecting an acknowledgment shall interpret the subtype of the frame (if any) sent by the PC a SIFS period after that STA's transmission to the PC. If a frame that requires MAC level acknowledgment is received by a non-CF-Pollable STA, that STA shall not interpret the CF-Poll indication (if any), and shall acknowledge the frame by sending an ACK Control frame after a SIFS period.

The lengths of the frames may be variable, only bounded by the frame and/or fragment length limitations that apply for the BSS. If a CF-Pollable STA does not respond to a CF-Poll (of any type) within the SIFS period following a transmission from the PC, or a non-CF-Pollable STA does not return the ACK frame within a SIFS period following a transmission from the PC that requires acknowledgment, then the PC shall resume control and may transmit its next frame after a PIFS period from the end of the PC's last transmission.

A CF-Pollable STA shall always respond to a CF-Poll directed to its MAC address and received without error. If the STA has no frame to send when polled, the response shall be a Null frame. If the STA has no frame to send when polled, but an acknowledgment is required for the frame that conveyed the CF-Poll, the response shall be a CF-ACK (no data) frame. The null response is required to permit a "no-traffic" situation to be distinguished from a collision between overlapping PCs.

The CFP shall end when the CFPDurRemaining time has elapsed since the Beacon frame originating the CFP or when the PC has no further frames to transmit nor STAs to poll. In either case, the end of the CFP shall be signaled by the transmission of a CF-End by the PC. If there is a received frame that requires acknowledgment at the time the CF-End is to be transmitted, the PC shall transmit a CF-End+ACK frame instead. All STAs of the BSS receiving a CF-End or CF-End+ACK shall reset their NAVs so they may attempt to transmit during the CP.

9.3.3.2 Operation with overlapping point-coordinated BSSs

Because the PCF operates without the CSMA/CA contention window randomization and backoff of the DCF, there is a risk of repeated collisions if multiple, overlapping, point-coordinated BSSs are operating on the same PHY channel, and their CFP-Rates and beacon intervals are approximately equal. To minimize the risk of significant frame loss due to CF collisions, the PC shall use a DIFS plus a random backoff delay (with CW in the range of 1 to aCWmin) to start a CFP when the initial beacon is delayed because of deferral due to a busy medium. The PC may optionally use this backoff during the CFP prior to retransmitting an unacknowledged, directed data or management frame.

To further reduce the susceptibility to inter-PC collisions, the PC shall require the medium be determined as being idle for a DIFS period plus a random (over a range of 1 to aCWmin) number of slot times once every aMediumOccupancyLimit TU during the CFP. This results in loss of control of the medium to overlapping BSS or hidden STA traffic, because the STAs in this BSS are prevented from transmitting by their NAV setting to CFPMaxDuration or CFPDurRemaining. For operation of the PCF in conjunction with an FH PHY, aMediumOccupancyLimit shall be set equal to the dwell time. For operation in conjunction with other PHY types, aMediumOccupancyLimit may be set equal to CFPMaxDuration, unless extra protection against PCF collisions is desired. The aMediumOccupancyLimit is also useful for compliance in regulatory domains that impose limits on continuous transmission time by a single STA as part of a spectrum etiquette.

9.3.3.3 CFPMaxDuration limit

The value of CFPMaxDuration shall be limited to allow coexistence between contention and contention-free traffic.

The minimum value for CFPMaxDuration is two times MaxMPDUTime plus the time required to send the initial Beacon frame and the CF-End frame of the CFP. This may allow sufficient time for the AP to send one data frame to a STA, while polling that STA, and for the polled STA to respond with one data frame.

The maximum value for CFPMaxDuration is the duration of (BeaconPeriod × DTIMPeriod × CFPRate) minus (MaxMPDUTime plus (2 × aSIFSTime) plus (2 × aSlotTime) plus (8 × ACKSize)), expressed in microseconds, when operating with a contention window of aCWmin. This allows sufficient time to send at least one data frame during the CP.

9.3.3.4 Contention-Free usage rules

A PC may send broadcast or multicast frames, and directed data or management frames to any active STA, as well as to CF-Pollable Power Save STAs. During the CFP, CF-Pollable STAs shall acknowledge after a SIFS period, the receipt of each Data+CF-Poll frame or Data+CF-ACK+CF-Poll frame using Data+CF-Ack or CF-Ack (no data) frames, the receipt of each CF_Poll (no data) using Data or Null (no data), and the receipt of all other data and management frames using ACK Control frames. Non-CF-Pollable STAs shall acknowledge receipt of data and management frames using ACK Control frames sent after a SIFS period. This non-CF-Pollable operation is the same as that already employed by such STAs for DCF operation.

When polled by the PCF (Data+CF-Poll, Data+CF-ACK+CF-Poll, CF-Poll, or CF-ACK+CF-Poll) a CF-Pollable STA may send one data frame to any destination. Such a frame directed to or through the PC STA shall be acknowledged by the PC, using the CF-ACK indication (Data+CF-ACK, Data+CF-ACK+CF-Poll, CF-ACK, CF-ACK+CF-Poll, or CF-End+ACK) sent after a SIFS. Such a frame directed to a non-CF-Pollable STA shall be acknowledged using an ACK Control frame sent after a SIFS period. A polled CF-Pollable STA with neither a data frame nor an acknowledgment to send shall respond by transmitting a Null frame after a SIFS period. A polled CF-Pollable STA with insufficient time before the end of the CFP or current medium occupancy limit, to send its queued MPDU and receive an acknowledgment, shall respond by transmitting a Null frame, or a CF-ACK frame if polled using Data+CF-Poll or Data+CF-ACK+CF-Poll, after a SIFS period. The CF-Pollable STA may set the More Data bit in its response to permit the PC to distinguish between an empty STA queue and a response due to insufficient time to transfer an MPDU.

The PC shall not issue frames with a subtype that includes CF-Polls if insufficient time remains in the current CFP to permit the polled STA to transmit a data frame containing a minimum length MPDU.

9.3.4 Contention-Free polling list

If the PC supports use of the CFP for inbound frame transfer as well as for frame delivery, the PC shall maintain a "polling list" for use in selecting STAs that are eligible to receive CF-Polls during CFPs. The polling list functional characteristics are defined below. If the PC supports the use of the CFP solely for frame delivery, the PC does not require a polling list, and shall never generate data frames with a subtype that includes CF-Poll. The form of contention-free support provided by the PC is identified in the Capability Information field of Beacon, Association Response, Reassociation Response, and Probe Response management frames, which are sent from APs. Any such frames sent by STAs, as in noninfrastructure networks, shall always have these bits set to zero.

The polling list is used to force the polling of CF-Pollable STAs, whether or not the PC has pending traffic to transmit to those STAs. The polling list may be used to control the use of Data+CF-Poll and Data+CF-ACK+CF-Poll types for transmission of data frames being sent to CF-Pollable STAs by the PC. The polling list is a *logical* construct, which is not exposed outside of the PC. A minimum set of polling list maintenance techniques are required to ensure interoperability of arbitrary CF-Pollable STAs in BSSs controlled by arbitrary access points with active PCs. APs may also implement additional polling list maintenance techniques that are outside the scope of this standard.

9.3.4.1 Polling list processing

The PC shall send a CF-Poll to at least one STA during each CFP when there are entries in the polling list. During each CFP, the PC shall issue polls to a subset of the STAs on the polling list in order by ascending AID value.

While time remains in the CFP, the delivery of all CF frames has been completed and all STAs on the polling list have been polled, the PC may generate one or more CF-Polls to *any* STAs on the polling list. While time remains in the CFP, the delivery of all CF frames has been completed and all STAs on the polling list have been polled, the PC *may* send data or management frames to *any* STAs.

In order to gain maximum efficiency from the CFP, and the ability to piggyback acknowledgments on successor data frames in the opposite direction, the PC should generally use Data+CF-Poll and Data+CF-ACK+CF-Poll types for each data frame transmitted while sufficient time for the potential response to the CF-Poll remains in the CFP.

9.3.4.2 Polling list update procedure

A STA indicates its CF-Pollability using the CF-Pollable subfield of the Capability Information field of Association Request and Reassociation Request frames. If a STA desires to change the PC's record of CF-Pollability, that STA shall perform a reassociation. During association, a CF-Pollable STA may also request to be placed on the polling list for the duration of its association, or by setting the CF-Poll Request subfield in the Capability Information field. If a CF-Pollable STA desires never to be placed on the polling list, that STA shall perform Association with both the CF-Pollable subfield false and CF-Poll Request subfield true. Never being polled is useful for CF-Pollable STAs that normally use power-save mode, permitting them to receive buffered traffic during the CFP (since they have to be awake to receive the DTIM that initiated the CFP), but not requiring them to stay awake to receive CF-Polls when they have no traffic to send. If a STA desires to be removed from the polling list, that STA shall perform a reassociation.

CF-Pollable STAs that are not on the polling list, but did not request never to be polled during their most recent association, may be dynamically placed on the polling list by the PC to handle bursts of frame transfer activity by that STA.

9.4 Fragmentation

The MAC may fragment and reassemble directed MSDUs or MMPDUs. The fragmentation and defragmentation mechanisms allow for fragment retransmission.

The length of a fragment MPDU shall be an equal number of octets for all fragments except the last, which may be smaller. The length of a fragment MPDU shall always be an even number of octets, except for the last fragment of an MSDU or MMPDU, which may be either an even or an odd number of octets. The length of a fragment shall never be larger than aFragmentationThreshold unless WEP is invoked for the MPDU. If WEP is active for the MPDU, then the MPDU shall be expanded by IV and ICV (see 8.2.5); this may result in a fragment larger than aFragmentationThreshold.

When data is to be transmitted, the number of octets in the fragment (before WEP processing) shall be determined by aFragmentationThreshold and the number of octets in the MPDU that have yet been assigned to a fragment at the instant the fragment is constructed for the first time. Once a fragment is transmitted for the first time, its frame body content and length shall be fixed until it is successfully delivered to the immediate receiving STA. A STA shall be capable of receiving fragments of arbitrary length.

If a fragment requires retransmission, its frame body content and length shall remain fixed for the lifetime of the MSDU or MMPDU at that STA. After a fragment is transmitted once, contents and length of that fragment are not allowed to fluctuate to accommodate the dwell time boundaries. Each fragment shall contain a Sequence Control field, which is comprised of a sequence number and fragment number. When a STA is transmitting an MSDU or MMPDU, the sequence number shall remain the same for all fragments of that MSDU or MMPDU. The fragments shall be sent in order of lowest fragment number to highest fragment number, where the fragment number value starts at zero, and increases by one for each successive fragment. The Frame Control field also contains a bit, the More Fragments bit, that is equal to zero to indicate the last (or only) fragment of the MSDU or MMPDU.

The source STA shall maintain a transmit MSDU timer for each MSDU being transmitted. The attribute aMaxTransmitMSDULifetime specifies the maximum amount of time allowed to transmit an MSDU. The timer starts on the attempt to transmit the first fragment of the MSDU. If the timer exceeds aMaxTransmitMSDULifetime, then all remaining fragments are discarded by the source STA and no attempt is made to complete transmission of the MSDU.

9.5 Defragmentation

Each fragment contains information to allow the complete MSDU or MMPDU to be reassembled from its constituent fragments. The header of each fragment contains the following information that is used by the destination STA to reassemble the MSDU or MMPDU:

- Frame type
- Address of the sender, obtained from the Address2 field
- Destination address
- *Sequence Control field:* This field allows the destination STA to check that all incoming fragments belong to the same MSDU or MMPDU, and the sequence in which the fragments should be reassembled. The sequence number within the Sequence Control field remains the same for all fragments of an MSDU or MMPDU, while the fragment number within the Sequence Control field increments for each fragment.
- *More Fragments indicator:* Indicates to the destination STA that this is not the last fragment of the MSDU or MMPDU. Only the last or sole fragment of the MSDU or MMPDU shall have this bit set to zero. All other fragments of the MSDU or MMPDU shall have this bit set to one.

The destination STA shall reconstruct the MSDU or MMPDU by combining the fragments in order of fragment number portion of the Sequence Control field. If WEP has been applied to the fragment, it shall be decrypted before the fragment is used for defragmentation of the MSDU or MMPDU. If the fragment with the More Fragments bit set to zero has not yet been received, then the destination STA knows that the MSDU or MMPDU is not yet complete. As soon as the STA receives the fragment with the More Fragments bit set to zero, the STA knows that no more fragments may be received for the MSDU or MMPDU.

All STAs shall support the concurrent reception of fragments of at least three MSDUs or MMPDUs. Note that a STA receiving more than three fragmented MSDUs or MMPDUs concurrently may experience a significant increase in the number of frames discarded.

The destination STA shall maintain a Receive Timer for each MSDU or MMPDU being received, for a minimum of three MSDUs or MMPDUs. The STA may implement additional timers to be able to receive additional concurrent MSDUs or MMPDUs. The receiving STA shall discard all fragments that are part of an MSDU or MMPDU for which a timer is not maintained. There is also an attribute, aMaxReceiveLifetime, that specifies the maximum amount of time allowed to receive an MSDU. The receive MSDU or MMPDU timer starts on the reception of the first fragment of the MSDU or MMPDU. If the receive MSDU timer exceeds aMaxReceiveLifetime, then all received fragments of this MSDU or MMPDU are discarded by the destination STA. If additional fragments of a directed MSDU or MMPDU are received after its aMaxReceiveLifetime is exceeded, those fragments shall be acknowledged and discarded.

To properly reassemble MPDUs into an MSDU or MMPDU, a destination STA shall discard any duplicated fragments received. A STA shall discard duplicate fragments as described in 9.2.9, duplicate detection. However, an acknowledgment shall be sent in response to a duplicate fragment of a directed MSDU.

9.6 Multirate support

Some PHYs have multiple data transfer rate capabilities that allow implementations to perform dynamic rate switching with the objective of improving performance. The algorithm for performing rate switching is beyond the scope of this standard, but in order to ensure coexistence and interoperability on multirate-capable PHYs, this standard defines a set of rules that shall be followed by all STAs.

All Control frames shall be transmitted at one of the rates in the BSSBasicRateSet (see 10.3.10.1), or at one of the rates in the PHY mandatory rate set so they will be understood by all STAs.

All frames with multicast and broadcast RA shall be transmitted at one of the rates included in the BSSBasicRateSet, regardless of their type.

Data and/or management MPDUs with a unicast immediate address shall be sent on any supported data rate selected by the rate switching mechanism (whose output is an internal MAC variable called MACCurrentRate, defined in units of 500 kbit/s, which is used for calculating the Duration/ID field of each frame). A STA shall not transmit at a rate that is known not to be supported by the destination STA, as reported in the supported rates element in the management frames. For frames of type Data+CF-ACK, Data+CF-Poll+CF-ACK and CF-Poll+CF-ACK, the rate chosen to transmit the frame must be supported by both the addressed recipient STA and the STA to which the ACK is intended.

Under no circumstances shall a STA initiate transmission of a data or management frame at a data rate higher than the greatest rate in the OperationalRateSet, a parameter of the MLME-JOIN.request primitive.

In order to allow the transmitting STA to calculate the contents of the Duration/ID field, the responding STA shall transmit its Control Response frame (either CTS or ACK) at the same rate as the immediately previous frame in the frame exchange sequence (as defined in 9.7), if this rate belongs to the PHY mandatory rates, or else at the highest possible rate belonging to the PHY rates in the BSSBasicRateSet.

9.7 Frame exchange sequences

The allowable frame exchange sequences are summarized in Table 21 and Table 22. A legend applicable to both tables follows Table 22.

Table 21—Frame sequences

Sequence	Frames in sequence	Usage
Data(bc/mc)	1	Broadcast or multicast MSDU
Mgmt(bc)	1	Broadcast MMPDU
{RTS – CTS –} [Frag – ACK –] Last – ACK	2	Directed MSDU or MMPDU
PS-Poll – ACK	2	Deferred PS-POLL response
PS-Poll – [Frag – ACK –] Last – ACK	3	Immediate PS-POLL response
DTIM(CF) – [≤CF-Sequence> –] {CF-End}	2 or more	Start of CFP
[<CF-Sequence> –] {CF-End}	2 or more	Continuation of CFP after missing ACK or medium occupancy boundary

Table 22—CF frame sequences

CF frame sequence	Frames in sequence	Usage
Beacon(CF)	1	Beacon during CFP
Data(bc/mc)	1	Broadcast or multicast MSDU
Mgmt(bc)	1 or 2	Broadcast MMPDU
Mgmt(dir) – ACK	2 or 3	Directed MMPDU
Data(dir)+CF-Poll{+CF-Ack} – Data(dir)+CF-Ack – {CF-Ack(no data)}	2	Poll and ACK sent with MPDUs
Data(dir)+CF-Poll{+CF-Ack} – CF-Ack(no data)	2	Poll of STA with empty queue, insufficient time for queued MPDU, or too little time remaining before a dwell or medium occupancy boundary to send a queued frame
CF-Poll(no data){+CF-Ack} – Data(dir) – {CF-Ack(no data)}	2	Separate poll, ACK sent with MPDU
CF-Poll(no data){+CF-Ack} – Data(dir) – ACK	3	Polled STA sends to STA in BSS
CF-Poll(no data){+CF-Ack} – Null(no data)	2	Separate poll, STA queue empty, or insufficient time for queued MPDU or too little time remaining before a dwell or medium occupancy boundary to send a queued frame
Data(dir){+CF-Ack} – ACK	2	ACK if not CF-Pollable or not polled

LEGEND (For Table 21 and Table 22)

1—Items enclosed in brackets "[...]" may occur zero or more times in the sequence.

2—Items enclosed in braces "{...}" may occur zero or one time in the sequence.

3—An isolated hyphen "-" represents a SIFS interval separating the pair of frames.

4—"Data(bc/mc)" represents any frame of type Data with a broadcast or multicast address in the Address1 field.

5—"Mgmt(bc)" represents any Management type frame with a broadcast address in the DA field.

6—"RTS" represents a Control frame of subtype RTS.

7—"CTS" represents a Control frame of subtype CTS.

8—"ACK" represents a Control frame of subtype ACK.

9—"Frag" represents an MPDU of type Data or an MMPDU of type Management with an individual address in the Address1 field that has the More Fragments field set to "1."

10—"Last" represents an MDPU of type Data or an MMPDU of type Management with an individual address in the Address1 field that has the More Fragments field set to "0."

11—"PS-Poll" represents a Control frame of subtype PS-Poll.

12—"DTIM(CF)" represents a management frame of subtype Beacon and that contains a DTIM information element with a nonzero value in the CFDurRemaining field of its Parameter Set element.

13—"CF-End" represents a Control frame of type CF-End, or (if the final frame of the immediately preceding <CF-Sequence> was a directed data or management frame requiring acknowledgment by the AP) of type CF-End+Ack.

14—"Beacon(CF)" represents a management frame of subtype Beacon with a nonzero value in the CFDurRemaining field of its CF Parameter Set element.

15—"Data(dir)" represents any MPDU of type Data with an individual address in the Address1 field.

16—"Mgmt(dir)" represents any MMPDU of type Management with an individual address in the Address1 field.

17—"CF-Ack(no data)" represents a data frame of subtype CF-ACK (no data).

18—"CF-Poll(no data)" represents a data frame of subtype CF-Poll (no data).

19—"Null(no data)" represents a data frame of subtype Null Function (no data).

20—The notation "{+CF-Ack}" indicates that the frame may or may not include a contention-free acknowledgment.

21—The notation "+CF-Ack" indicates that the frame includes a contention-free acknowledgment.

22—The notation "+CF-Poll" indicates that the frame includes a contention-free poll.

23—<CF-Sequence> represents a sequence of one or more frames sent during a CFP. A valid <CF-Sequence> shall consist of one of the frame sequences shown in Table 22. The collection of sequences of frame exchanges corresponding to the [<CF-Sequence>] from may occur in any order within the CFP.

Individual frames within each of these sequences are separated by a SIFS.

9.8 MSDU transmission restrictions

To avoid reordering MSDUs between pairs of LLC entities and/or unnecessarily discarding MSDUs, the following restrictions shall be observed by any STA that is able to concurrently process multiple outstanding MSDUs for transmission. Note that here the term "outstanding" refers to an MSDU or MMPDU that is eligible to be transmitted at a particular time. A STA may have any number (greater than or equal to one) of eligible MSDUs outstanding concurrently, subject to the restrictions below.

The STA shall ensure that no more than one MSDU or MMPDU from a particular SA to a particular individual RA is outstanding at a time. Note that a simpler, more restrictive invariant to maintain is that no more than one MSDU with a particular individual RA may be outstanding at a time.

In a STA where the optional StrictlyOrdered service class has been implemented, that STA shall ensure that there is no group-addressed (multidestination) MSDU of the StrictlyOrdered service class outstanding from the SA of any other outstanding MSDU (either directed or group-addressed). This is because a group-addressed MSDU is implicitly addressed to a collection of peer STAs that could include any individual RA.

It is recommended that the STA select a value of aMaxMSDUTransmitLifetime that is sufficiently large that the STA does not discard MSDUs due to excessive Transmit MSDU timeouts under normal operating conditions.

10. Layer management

10.1 Overview of management model

Both MAC and physical layers conceptually include management entities, called MAC sublayer management and PHY layer management entities (MLME and PLME, respectively). These entities provide the layer management service interfaces through which layer management functions may be invoked.

In order to provide correct MAC operation, a station management entity (SME) shall be present within each STA. The SME is a layer-independent entity that may be viewed as residing in a separate management plane or as residing "off to the side." The exact functions of the SME are not specified in this standard, but in general this entity may be viewed as responsible for such functions as the gathering of layer-dependent status from the various layer management entities, and similarly setting the value of layer-specific parameters. SME would typically perform such functions on behalf of general system management entities and would implement standard management protocols. Figure 11 depicts the relationship among management entities.

The various entities within this model interact in various ways. Certain of these interactions are defined explicitly within this standard, via a service access point (SAP) across which defined primitives are exchanged. Other interactions are not defined explicitly within this standard, such as the interfaces between MAC and MLME and between the PLCP and PLME, represented as double arrows within the figure. The specific manner in which these MAC and PHY management entities are integrated into the overall MAC and PHY layers is not specified within this standard.

The management SAPs within this model are the following:

— SME-MLME SAP
— SME-PLME SAP
— MLME-PLME SAP

The latter two SAPs support identical primitives, and in fact may be viewed as a single SAP (called the PLME SAP) that may be used either directly by MLME or by SME. In this fashion, the model reflects what is anticipated to be a common implementation approach in which PLME functions are controlled by the MLME (on behalf of SME). In particular, PHY implementations are not required to have separate interfaces defined other than their interfaces with the MAC and MLME.

10.2 Generic management primitives

The management information specific to each layer is represented as a management information base (MIB) for that layer. The MAC and PHY layer management entities are viewed as "containing" the MIB for that layer. The generic model of MIB-related management primitives exchanged across the management SAPs is to allow the SAP user-entity to either GET the value of a MIB attribute, or to SET the value of a MIB attribute. The invocation of a SET.request primitive may require that the layer entity perform certain defined actions.

Figure 63 depicts these generic primitives.

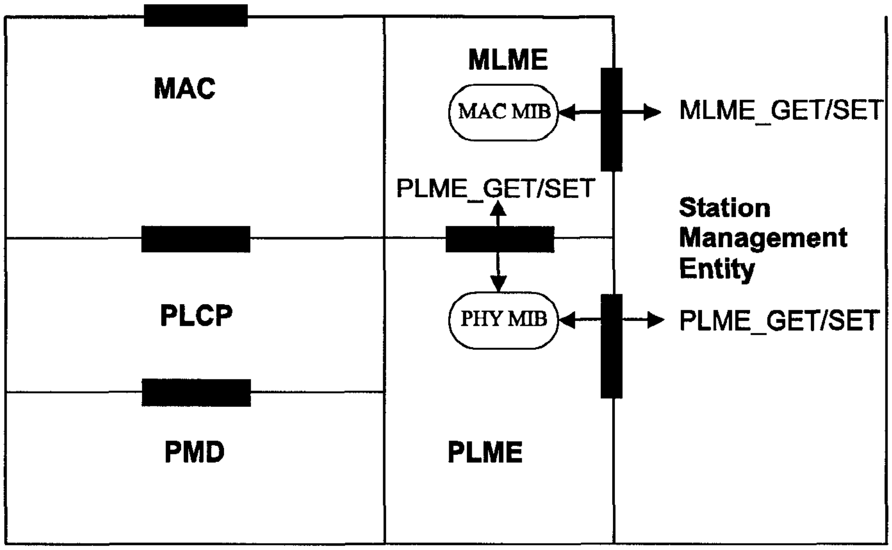


Figure 63—GET and SET operations

The GET and SET primitives in fact are represented as REQUESTs with associated CONFIRM primitives. These primitives are prefixed by MLME or PLME depending upon whether the MAC or PHY layer management SAP is involved. In the following, XX denotes MLME or PLME:

XX-GET.request (MIBattribute)
Requests the value of the given MIBattribute.

XX-GET.confirm (status, MIBattribute, MIBattributevalue)
Returns the appropriate MIB attribute value if status = "success," otherwise returns an error indication in the Status field. Possible error status values include "invalid MIB attribute" and "attempt to get write-only MIB attribute."

XX-SET.request (MIBattribute, MIBattributevalue)
Requests that the indicated MIB attribute be set to the given value. If this MIBattribute implies a specific action, then this request that the action be performed.

XX-SET.confirm (status, MIBattribute)
If status = "success," this confirms that the indicated MIB attribute was set to the requested value, otherwise returns an error condition in status field. If this MIBattribute implies a specific action, then this confirms that the action was performed. Possible error status values include "invalid MIB attribute" and "attempt to set read-only MIB attribute."

Additionally, there are certain requests (with associated confirms) that may be invoked across a given SAP which do not involve the setting or getting of a specific MIB attribute. One of these is supported by each SAP, as follows:

— XX-RESET.request: where XX is MLME or PLME as appropriate.
— XX-RESET.confirm

This service is used to initialize the management entities, the MIBs, and the datapath entities. It may include a list of attributes for items to be initialized to non-default values. The corresponding .confirm indicates success or failure of the request.

Other SAP-specific primitives are identified in 10.3.

10.3 MLME SAP interface

The services provided by the MLME to the SME are specified in this subclause. These services are described in an abstract way and do not imply any particular implementation or exposed interface. MLME SAP primitives are of the general form ACTION.request followed by ACTION.confirm. The SME uses the services provided by the MLME through the MLME SAP.

10.3.1 Power management

This mechanism supports the process of establishment and maintenance of the power management mode of a STA.

10.3.1.1 MLME-POWERMGT.request

10.3.1.1.1 Function

This primitive requests a change in the power management mode.

10.3.1.1.2 Semantics of the service primitive

The primitive parameters are as follows:

```
MLME-POWERMGT.request    (
                         PowerManagementMode,
                         WakeUp,
                         ReceiveDTIMs
                         )
```

Name	Type	Valid range	Description
PowerManagementMode	Enumeration	ACTIVE, POWER_SAVE	An enumerated type that describes the desired power management mode of the STA.
WakeUp	Boolean	True, false	When true, the MAC is forced immediately into the Awake state. This parameter has no effect if the current power management mode is ACTIVE.
ReceiveDTIMs	Boolean	True, false	When true, this parameter causes the STA to awaken to receive all DTIM frames. When false, the STA is not required to awaken for every DTIM frame.

10.3.1.1.3 When generated

This primitive is generated by the SME to implement the power savings strategy of an implementation.

10.3.1.1.4 Effect of receipt

This request sets the STA's power management parameters. The MLME subsequently issues a MLME-POWERMGT.confirm that reflects the results of the power management change request.

10.3.1.2 MLME-POWERMGT.confirm

10.3.1.2.1 Function

This primitive confirms the change in power management mode.

10.3.1.2.2 Semantics of the service primitive

The primitive parameters are as follows:

```
MLME-POWERMGT.confirm      (
                           ResultCode
                           )
```

Name	Type	Valid range	Description
ResultCode	Enumeration	SUCCESS, INVALID_PARAMETERS, NOT_SUPPORTED	Indicates the result of the MLME-POWERMGT.request

10.3.1.2.3 When generated

This primitive is generated by the MLME as a result of an MLME-POWERMGT.request to establish a new power management mode. It is not generated until the change has completed.

10.3.1.2.4 Effect of receipt

The SME is notified of the change of power management mode.

10.3.2 Scan

This mechanism supports the process of determining the characteristics of the available BSSs.

10.3.2.1 MLME-SCAN.request

10.3.2.1.1 Function

This primitive requests a survey of potential BSSs that the STA may later elect to try to join.

10.3.2.1.2 Semantics of the service primitive

The primitive parameters are as follows:

```
MLME-SCAN.request          (
                           BSSType,
                           BSSID,
                           SSID,
                           ScanType,
                           ProbeDelay,
                           ChannelList,
                           MinChannelTime,
                           MaxChannelTime
                           )
```

Name	Type	Valid range	Description
BSSType	Enumeration	INFRASTRUCTURE, INDEPENDENT, ANY_BSS	Determines whether Infrastructure BSS, Independent BSS, or both are included in the scan.
BSSID	MACAddress	Any valid individual or broadcast MAC address	Identifies a specific or broadcast BSSID
SSID	Octet string	0–32 octets	Specifies the desired SSID or the broadcast SSID
ScanType	Enumeration	ACTIVE, PASSIVE	Indicates either active or passive scanning
ProbeDelay	Integer	N/A	Delay (in μs) to be used prior to transmitting a Probe frame during active scanning
ChannelList	Ordered set of integers	Each channel will be selected from the valid channel range for the appropriate PHY and carrier set.	Specifies a list of channels that are examined when scanning for a BSS
MinChannelTime	Integer	≥ ProbeDelay	The minimum time (in TU) to spend on each channel when scanning
MaxChannelTime	Integer	≥ MinChannelTime	The maximum time (in TU) to spend on each channel when scanning

10.3.2.1.3 When generated

This primitive is generated by the SME for a STA to determine if there are other BSSs that it may join.

10.3.2.1.4 Effect of receipt

This request initiates the scan process when the current frame exchange sequence is completed.

10.3.2.2 MLME-SCAN.confirm

10.3.2.2.1 Function

This primitive returns the descriptions of the set of BSSs detected by the scan process.

10.3.2.2.2 Semantics of the service primitive

The primitive parameters are as follows:

```
MLME-SCAN.confirm          (
                           BSSDescriptionSet,
                           ResultCode
                           )
```

Name	Type	Valid range	Description
BSSDescriptionSet	Set of BSSDescriptions	N/A	The BSSDescriptionSet is returned to indicate the results of the scan request. It is a set containing zero or more instances of a BSSDescription.
ResultCode	Enumeration	SUCCESS, INVALID_ PARAMETERS	Indicates the result of the MLME-SCAN.confirm

Each BSSDescription consists of the following elements:

Name	Type	Valid range	Description
BSSID	MACAddress	N/A	The BSSID of the found BSS
SSID	Octet string	1–32 octets	The SSID of the found BSS
BSSType	Enumeration	INFRASTRUCTURE, INDEPENDENT	The type of the found BSS
Beacon Period	Integer	N/A	The Beacon period of the found BSS (in TU)
DTIM Period	Integer	As defined in frame format	The DTIM period of the BSS (in beacon periods)
Timestamp	Integer	N/A	The timestamp of the received frame (probe response/beacon) from the found BSS
Local Time	Integer	N/A	The value of the STA's TSF timer at the start of reception of the first octet of the timestamp field of the received frame (probe response or beacon) from the found BSS
PHY parameter set	As defined in frame format	As defined in frame format	The parameter set relevant to the PHY
CF parameter set	As defined in frame format	As defined in frame format	The parameter set for the CF periods, if found BSS supports CF mode
IBSS parameter set	As defined in frame format	As defined in frame format	The parameter set for the IBSS, if found BSS is an IBSS
CapabilityInformation	As defined in frame format	As defined in frame format	The advertised capabilities of the BSS
BSSBasicRateSet	Set of integers	1 through 127 inclusive (for each integer in the set)	The set of data rates (in units of 500 kb/s) that must be supported by all STAs that desire to join this BSS. The STAs must be able to receive at each of the data rates listed in the set.

10.3.2.2.3 When generated

This primitive is generated by the MLME as a result of an MLME-SCAN.request to ascertain the operating environment of the STA.

10.3.2.2.4 Effect of receipt

The SME is notified of the results of the scan procedure.

10.3.3 Synchronization

This mechanism supports the process of selection of a peer in the authentication process.

10.3.3.1 MLME-JOIN.request

10.3.3.1.1 Function

This primitive requests synchronization with a BSS.

10.3.3.1.2 Semantics of the service primitive

The primitive parameters are as follows:

MLME-JOIN.request (
BSSDescription,
JoinFailureTimeout,
ProbeDelay.
OperationalRateSet
)

Name	Type	Valid range	Description
BSSDescription	BSSDescription	N/A	The BSSDescription of the BSS to join. The BSSDescription is a member of the set of descriptions that was returned as a result of a MLME-SCAN.request.
JoinFailureTimeout	Integer	≥1	The time limit, in units of beacon intervals, after which the join procedure will be terminated
ProbeDelay	Integer	N/A	Delay (in μs) to be used prior to transmitting a Probe frame during active scanning
OperationalRateSet	Set of integers	1 through 127 inclusive (for each integer in the set)	The set of data rates (in units of 500 kbit/s) that the STA may use for communication within the BSS. The STA must be able to receive at each of the data rates listed in the set. The OperationalRateSet is a superset of the BSSBasicRateSet advertised by the BSS.

10.3.3.1.3 When generated

This primitive is generated by the SME for a STA to establish synchronization with a BSS.

10.3.3.1.4 Effect of receipt

This primitive initiates a synchronization procedure once the current frame exchange sequence is complete. The MLME synchronizes its timing with the specified BSS based on the elements provided in the BSSDescription parameter. The MLME subsequently issues a MLME-JOIN.confirm that reflects the results.

10.3.3.2 MLME-JOIN.confirm

10.3.3.2.1 Function

This primitive confirms synchronization with a BSS.

10.3.3.2.2 Semantics of the service primitive

The primitive parameters are as follows:

MLME-JOIN.confirm (
ResultCode
)

Name	Type	Valid range	Description
ResultCode	Enumeration	SUCCESS, INVALID_PARAMETERS, TIMEOUT	Indicates the result of the MLME-JOIN.request

10.3.3.2.3 When generated

This primitive is generated by the MLME as a result of an MLME-JOIN.request to establish synchronization with a BSS.

10.3.3.2.4 Effect of receipt

The SME is notified of the results of the synchronization procedure.

10.3.4 Authenticate

This mechanism supports the process of establishing an authentication relationship with a peer MAC entity.

10.3.4.1 MLME-AUTHENTICATE.request

10.3.4.1.1 Function

This primitive requests authentication with a specified peer MAC entity.

10.3.4.1.2 Semantics of the service primitive

The primitive parameters are as follows:

```
MLME-AUTHENTICATE.request    (
                              PeerSTAAddress,
                              AuthenticationType,
                              AuthenticateFailureTimeout
                              )
```

Name	Type	Valid range	Description
PeerSTAAddress	MACAddress	Any valid individual MAC address	Specifies the address of the peer MAC entity with which to perform the authentication process
AuthenticationType	Enumeration	OPEN_SYSTEM, SHARED_KEY	Specifies the type of authentication algorithm to use during the authentication process
AuthenticationFailure-Timeout	Integer	≥1	Specifies a time limit (in TU) after which the authentication procedure will be terminated

10.3.4.1.3 When generated

This primitive is generated by the SME for a STA to establish authentication with a specified peer MAC entity in order to permit Class 2 frames to be exchanged between the two STAs. During the authentication procedure, the SME may generate additional MLME-AUTHENTICATE.request primitives.

10.3.4.1.4 Effect of receipt

This primitive initiates an authentication procedure. The MLME subsequently issues a MLME-AUTHENTICATE.confirm that reflects the results.

10.3.4.2 MLME-AUTHENTICATE.confirm

10.3.4.2.1 Function

This primitive reports the results of an authentication attempt with a specified peer MAC entity.

10.3.4.2.2 Semantics of the service primitive

The primitive parameters are as follows:

MLME-AUTHENTICATE.confirm (
PeerSTAAddress,
AuthenticationType,
ResultCode
)

Name	Type	Valid range	Description
PeerSTAAddress	MACAddress	Any valid individual MAC address	Specifies the address of the peer MAC entity with which the authentication process was attempted. This value must match the peerSTAAddress parameter specified in the corresponding MLME-AUTHENTICATE.request.
AuthenticationType	Enumeration	OPEN_SYSTEM, SHARED_KEY	Specifies the type of authentication algorithm that was used during the authentication process. This value must match the authenticationType parameter specified in the corresponding MLME-AUTHENTICATE.request.
ResultCode	Enumeration	SUCCESS, INVALID_PARAMETERS, TIMEOUT, TOO_MANY_SIMULTANEOUS_REQUESTS, REFUSED	Indicates the result of the MLME-AUTHENTICATE.request.

10.3.4.2.3 When generated

This primitive is generated by the MLME as a result of an MLME-AUTHENTICATE.request to authenticate with a specified peer MAC entity.

10.3.4.2.4 Effect of receipt

The SME is notified of the results of the authentication procedure.

10.3.4.3 MLME-AUTHENTICATE.indication

10.3.4.3.1 Function

This primitive reports the establishment of an authentication relationship with a specific peer MAC entity.

10.3.4.3.2 Semantics of the service primitive

The primitive parameters are as follows:

MLME-AUTHENTICATE.indication (
PeerSTAAddress,
AuthenticationType
)

Name	Type	Valid range	Description
PeerSTAAddress	MACAddress	Any valid individual MAC address	Specifies the address of the peer MAC entity with which the authentication relationship was established
AuthenticationType	Enumeration	OPEN_SYSTEM, SHARED_KEY	Specifies the type of authentication algorithm that was used during the authentication process

10.3.4.3.3 When generated

This primitive is generated by the MLME as a result of the establishment of an authentication relationship with a specific peer MAC entity that resulted from an authentication procedure that was initiated by that specific peer MAC entity.

10.3.4.3.4 Effect of receipt

The SME is notified of the establishment of the authentication relationship.

10.3.5 De-authenticate

This mechanism supports the process of invalidating an authentication relationship with a peer MAC entity.

10.3.5.1 MLME-DEAUTHENTICATE.request

10.3.5.1.1 Function

This primitive requests that the authentication relationship with a specified peer MAC entity be invalidated.

10.3.5.1.2 Semantics of the service primitive

The primitive parameters are as follows:

```
MLME-DEAUTHENTICATE.request     (
                                PeerSTAAddress,
                                ReasonCode
                                )
```

Name	Type	Valid range	Description
PeerSTAAddress	MACAddress	Any valid individual MAC address	Specifies the address of the peer MAC entity with which to perform the deauthentication process
ReasonCode	As defined in frame format.	As defined in frame format.	Specifies the reason for initiating the deauthentication procedure

10.3.5.1.3 When generated

This primitive is generated by the SME for a STA to invalidate authentication with a specified peer MAC entity in order to prevent the exchange of Class 2 frames between the two STAs. During the deauthentication procedure, the SME may generate additional MLME-DEAUTHENTICATE.request primitives.

10.3.5.1.4 Effect of receipt

This primitive initiates a deauthentication procedure. The MLME subsequently issues a MLME-DEAUTHENTICATE.confirm that reflects the results.

10.3.5.2 MLME-DEAUTHENTICATE.confirm

10.3.5.2.1 Function

This primitive reports the results of a deauthentication attempt with a specified peer MAC entity.

10.3.5.2.2 Semantics of the service primitive

The primitive parameters are as follows:

```
MLME-DEAUTHENTICATE.confirm      (
                                 PeerSTAAddress,
                                 ResultCode
                                 )
```

Name	Type	Valid range	Description
PeerSTAAddress	MACAddress	Any valid individual MAC address	Specifies the address of the peer MAC entity with which the deauthentication process was attempted
ResultCode	Enumeration	SUCCESS, INVALID_PARAMETERS, TOO_MANY_SIMULTANEOUS_REQUESTS	Indicates the result of the MLME-DEAUTHENTICATE.request

10.3.5.2.3 When generated

This primitive is generated by the MLME as a result of an MLME-DEAUTHENTICATE.request to invalidate the authentication relationship with a specified peer MAC entity.

10.3.5.2.4 Effect of receipt

The SME is notified of the results of the deauthentication procedure.

10.3.5.3 MLME-DEAUTHENTICATE.indication

10.3.5.3.1 Function

This primitive reports the invalidation of an authentication relationship with a specific peer MAC entity.

10.3.5.3.2 Semantics of the service primitive

The primitive parameters are as follows:

```
MLME-DEAUTHENTICATE.indication   (
                                 PeerSTAAddress,
                                 ReasonCode
                                 )
```


Name	Type	Valid range	Description
PeerSTAAddress	MACAddress	Any valid individual MAC address	Specifies the address of the peer MAC entity with which the authentication relationship was invalidated
ReasonCode	As defined in frame format.	As defined in frame format.	Specifies the reason the deauthentication procedure was initiated

10.3.5.3.3 When generated

This primitive is generated by the MLME as a result of the invalidation of an authentication relationship with a specific peer MAC entity.

10.3.5.3.4 Effect of receipt

The SME is notified of the invalidation of the specific authentication relationship.

10.3.6 Associate

The following primitives describe how a STA becomes associated with an access point (AP).

10.3.6.1 MLME-ASSOCIATE.request

10.3.6.1.1 Function

This primitive requests association with a specified peer MAC entity that is acting as an AP.

10.3.6.1.2 Semantics of the service primitive

The primitive parameters are as follows:

```
MLME-ASSOCIATE.request    (
                          PeerSTAAddress,
                          AssociateFailureTimeout,
                          CapabilityInformation,
                          ListenInterval
                          )
```

Name	Type	Valid range	Description
PeerSTAAddress	MACAddress	Any valid individual MAC address	Specifies the address of the peer MAC entity with which to perform the association process
AssociateFailureTimeout	Integer	≥ 1	Specifies a time limit (in TU) after which the associate procedure will be terminated
CapabilityInformation	As defined in frame format	As defined in frame format	The operational capability definitions to be used by the MAC entity
ListenInterval	Integer	≥ 0	Specifies the number of beacon intervals that may pass before the STA awakens and listens for the next beacon

10.3.6.1.3 When generated

This primitive is generated by the SME when a STA wishes to establish association with an AP.

10.3.6.1.4 Effect of receipt

This primitive initiates an association procedure. The MLME subsequently issues a MLME-ASSOCIATE.confirm that reflects the results.

10.3.6.2 MLME-ASSOCIATE.confirm

10.3.6.2.1 Function

This primitive reports the results of an association attempt with a specified peer MAC entity that is acting as an AP.

10.3.6.2.2 Semantics of the service primitive

The primitive parameters are as follows:

MLME-ASSOCIATE.confirm (
ResultCode
)

Name	Type	Valid range	Description
ResultCode	Enumeration	SUCCESS, INVALID_ PARAMETERS, TIMEOUT, REFUSED	Indicates the result of the MLME-ASSOCIATE.request

10.3.6.2.3 When generated

This primitive is generated by the MLME as a result of an MLME-ASSOCIATE.request to associate with a specified peer MAC entity that is acting as an AP.

10.3.6.2.4 Effect of receipt

The SME is notified of the results of the association procedure.

10.3.6.3 MLME-ASSOCIATE.indication

10.3.6.3.1 Function

This primitive reports the establishment of an association with a specific peer MAC entity.

10.3.6.3.2 Semantics of the service primitive

The primitive parameters are as follows:

MLME-ASSOCIATE.indication (
PeerSTAAddress
)

Name	Type	Valid range	Description
PeerSTAAddress	MACAddress	Any valid individual MAC address	Specifies the address of the peer MAC entity with which the association was established

10.3.6.3.3 When generated

This primitive is generated by the MLME as a result of the establishment of an association with a specific peer MAC entity that resulted from an association procedure that was initiated by that specific peer MAC entity.

10.3.6.3.4 Effect of receipt

The SME is notified of the establishment of the association.

10.3.7 Reassociate

The following primitives describe how a STA becomes associated with another AP.

10.3.7.1 MLME-REASSOCIATE.request

10.3.7.1.1 Function

This primitive requests a change in association to a specified new peer MAC entity that is acting as an AP.

10.3.7.1.2 Semantics of the service primitive

The primitive parameters are as follows:

```
MLME-REASSOCIATE.request    (
                            NewAPAddress,
                            ReassociateFailureTimeout,
                            CapabilityInformation,
                            ListenInterval
                            )
```

Name	Type	Valid range	Description
NewAPAddress	MACAddress	Any valid individual MAC address	Specifies the address of the peer MAC entity with which to perform the reassociation process
ReassociateFailureTimeout	Integer	≥1	Specifies a time limit (in TU) after which the reassociate procedure will be terminated
CapabilityInformation	As defined in frame format	As defined in frame format	The operational capability definitions to be used by the MAC entity
LlistenInterval	Integer	≥0	Specifies the number of beacon intervals that may pass before the STA awakens and listens for the next beacon.

10.3.7.1.3 When generated

This primitive is generated by the SME for a STA to change association to a specified new peer MAC entity that is acting as an AP.

10.3.7.1.4 Effect of receipt

This primitive initiates a reassociation procedure. The MLME subsequently issues a MLME-REASSOCIATE.confirm that reflects the results.

10.3.7.2 MLME-REASSOCIATE.confirm

10.3.7.2.1 Function

This primitive reports the results of a reassociation attempt with a specified peer MAC entity that is acting as an AP.

10.3.7.2.2 Semantics of the service primitive

The primitive parameters are as follows:

MLME-REASSOCIATE.confirm (
ResultCode
)

Name	Type	Valid range	Description
ResultCode	Enumeration	SUCCESS, INVALID_ PARAMETERS, TIMEOUT, REFUSED	Indicates the result of the MLME-REASSOCI-ATE.request

10.3.7.2.3 When generated

This primitive is generated by the MLME as a result of an MLME-REASSOCIATE.request to reassociate with a specified peer MAC entity that is acting as an AP.

10.3.7.2.4 Effect of receipt

The SME is notified of the results of the reassociation procedure.

10.3.7.3 MLME-REASSOCIATE.indication

10.3.7.3.1 Function

This primitive reports the establishment of a reassociation with a specified peer MAC entity.

10.3.7.3.2 Semantics of the service primitive

The primitive parameters are as follows:

MLME-REASSOCIATE.indication (
PeerSTAAddress
)

Name	Type	Valid range	Description
PeerSTAAddress	MACAddress	Any valid individual MAC address	Specifies the address of the peer MAC entity with which the reassociation was established

10.3.7.3.3 When generated

This primitive is generated by the MLME as a result of the establishment of a reassociation with a specific peer MAC entity that resulted from a reassociation procedure that was initiated by that specific peer MAC entity.

10.3.7.3.4 Effect of receipt

The SME is notified of the establishment of the reassociation.

10.3.8 Disassociate

10.3.8.1 MLME-DISASSOCIATE.request

10.3.8.1.1 Function

This primitive requests disassociation with a specified peer MAC entity that is acting as an AP.

10.3.8.1.2 Semantics of the service primitive

The primitive parameters are as follows:

```
MLME-DISASSOCIATE.request    (
                             PeerSTAAddress,
                             ReasonCode
                             )
```

Name	Type	Valid range	Description
PeerSTAAddress	MACAddress	Any valid individual MAC address	Specifies the address of the peer MAC entity with which to perform the disassociation process
ReasonCode	As defined in frame format.	As defined in frame format.	Specifies the reason for initiating the disassociation procedure

10.3.8.1.3 When generated

This primitive is generated by the SME for a STA to to establish disassociation with an AP.

10.3.8.1.4 Effect of receipt

This primitive initiates a disassociation procedure. The MLME subsequently issues a MLME-DISASSOCIATE.confirm that reflects the results.

10.3.8.2 MLME-DISASSOCIATE.confirm

10.3.8.2.1 Function

This primitive reports the results of a disassociation procedure with a specific peer MAC entity that is acting as an AP.

10.3.8.2.2 Semantics of the service primitive

The primitive parameters are as follows:

```
MLME-DISASSOCIATE.confirm    (
                             ResultCode
                             )
```

Name	Type	Valid range	Description
ResultCode	Enumeration	SUCCESS, INVALID_ PARAMETERS, TIMEOUT, REFUSED	Indicates the result of the MLME-DISASSOCIATE.request

10.3.8.2.3 When generated

This primitive is generated by the MLME as a result of an MLME-DISASSOCIATE.request to disassociate with a specified peer MAC entity that is acting as an AP.

10.3.8.2.4 Effect of receipt

The SME is notified of the results of the disassociation procedure.

10.3.8.3 MLME-DISASSOCIATE.indication

10.3.8.3.1 Function

This primitive reports disassociation with a specific peer MAC entity.

10.3.8.3.2 Semantics of the service primitive

The primitive parameters are as follows:

```
MLME-DISASSOCIATE.indication (
                              PeerSTAAddress,
                              ReasonCode
                              )
```

Name	Type	Valid range	Description
PeerSTAAddress	MACAddress	Any valid individual MAC address	Specifies the address of the peer MAC entity with which the association relationship was invalidated
ReasonCode	As defined in frame format.	As defined in frame format.	Specifies the reason the disassociation procedure was initiated

10.3.8.3.3 When generated

This primitive is generated by the MLME as a result of the invalidation of an association relationship with a specific peer MAC entity.

10.3.8.3.4 Effect of receipt

The SME is notified of the invalidation of the specific association relationship.

10.3.9 Reset

This mechanism supports the process of resetting the MAC.

10.3.9.1 MLME-RESET.request

10.3.9.1.1 Function

This primitive requests that the MAC entity be reset.

10.3.9.1.2 Semantics of the service primitive

The primitive parameters are as follows:

```
MLME-RESET.request          (
                            STAAddress,
                            SetDefaultMIB
                            )
```

Name	Type	Valid range	Description
STAAddress	MACAddress	Any valid MAC address	Specifies the MAC address that is to be used by the MAC entity that is being reset. This value may be used to provide a locally administered STA address.
SetDefaultMIB	Boolean	True, false	If true, all MIB attributes are set to their default values. The default values are implementation dependent. If false, the MAC is reset, but all MIB attributes retain the values that were in place prior to the generation of the MLME-RESET.request primitive.

10.3.9.1.3 When generated

This primitive is generated by the SME to reset the MAC to initial conditions. The MLME-RESET.request primitive must be used prior to use of MLME-START.request primitive.

10.3.9.1.4 Effect of receipt

This primitive sets the MAC to initial conditions, clearing all internal variables to the default values. MIB attributes may be reset to their implementation-dependent default values by setting the SetDefaultMIB flag to true. The MLME subsequently issues a MLME-RESET.confirm that reflects the results.

10.3.9.2 MLME-RESET.confirm

10.3.9.2.1 Function

This primitive reports the results of a reset procedure.

10.3.9.2.2 Semantics of the service primitive

The primitive parameters are as follows:

```
MLME-RESET.confirm          (
                            ResultCode
                            )
```

Name	Type	Valid range	Description
ResultCode	Enumeration	SUCCESS	Indicates the result of the MLME-RESET.request

10.3.9.2.3 When generated

This primitive is generated by the MLME as a result of an MLME-RESET.request to reset the MAC entity.

10.3.9.2.4 Effect of receipt

The SME is notified of the results of the reset procedure.

10.3.10 Start

This mechanism supports the process of creating a new BSS.

10.3.10.1 MLME-START.request

10.3.10.1.1 Function

This primitive requests that the MAC entity start a new BSS.

10.3.10.1.2 Semantics of the service primitive

The primitive parameters are as follows:

```
MLME-START.request     (
                       SSID,
                       BSSType,
                       BeaconPeriod,
                       DTIMPeriod,
                       CF parameter set,
                       PHY parameter set,
                       IBSS parameter set,
                       ProbeDelay.
                       CapabilityInformation,
                       BBSBasicRateSet,
                       OperationalRateSet
                       )
```

Name	Type	Valid range	Description
SSID	Octet string	1–32 octets	The SSID of the BSS
BSSType	Enumeration	INFRA-STRUCTURE, INDEPEN-DENT	The type of the BSS
Beacon Period	Integer	≥1	The Beacon period of the BSS (in TU)
DTIM Period	Integer	As defined in frame format	The DTIM Period of the BSS (in beacon periods)
CF parameter set	As defined in frame format	As defined in frame format	The parameter set for CF periods, if the BSS supports CF mode. aCFPPeriod is modified as a side effect of the issuance of a MLME-START.request primitive.
PHY parameter set	As defined in frame format	As defined in frame format	The parameter set relevant to the PHY
IBSS parameter set	As defined in frame format	As defined in frame format	The parameter set for the IBSS, if BSS is an IBSS

Name	Type	Valid range	Description
ProbeDelay	Integer	N/A	Delay (in μs) to be used prior to transmitting a Probe frame during active scanning
CapabilityInformation	As defined in frame format	As defined in frame format	The capabilities to be advertised for the BSS
BSSBasicRateSet	Set of integers	1 through 127 inclusive (for each integer in the set)	The set of data rates (in units of 500 kbit/s) that must be supported by all STAs to join this BSS. The STA that is creating the BSS must be able to receive at each of the data rates listed in the set.
OperationalRateSet	Set of integers	1 through 127 inclusive (for each integer in the set)	The set of data rates (in units of 500 kbit/s) that the STA may use for communication within the BSS. The STA must be able to receive at each of the data rates listed in the set. The OperationalRateSet is a superset of the BSSBasicRateSet advertised by the BSS.

10.3.10.1.3 When generated

This primitive is generated by the SME to start either an infrastructure BSS (with the MAC entity acting as an AP), or start an independent BSS (with the MAC entity acting as the first STA in the IBSS).

The MLME-START.request primitive must be generated after a MLME-RESET.request primitive has been used to reset the MAC entity and before an MLME-JOIN.request primitive has been used to successfully join an existing infrastructure BSS or independent BSS.

The MLME-START.request primitive must not be used after successful use of the MLME-START.request primitive or successful use of the MLME-JOIN.request without generating an intervening MLME-RESET.request primitive.

10.3.10.1.4 Effect of receipt

This primitive initiates the BSS initialization procedure once the current frame exchange sequence is complete. The MLME subsequently issues a MLME-START.confirm that reflects the results of the creation procedure.

10.3.10.2 MLME-START.confirm

10.3.10.2.1 Function

This primitive reports the results of a BSS creation procedure.

10.3.10.2.2 Semantics of the service primitive

The primitive parameters are as follows:

```
MLME-START.confirm          (
                            ResultCode
                            )
```

Name	Type	Valid range	resetDescription
ResultCode	Enumeration	SUCCESS, INVALID_PARAMETERS, BSS_ALREADY_STARTED_OR_JOINED	Indicates the result of the MLME-START.request

10.3.10.2.3 When generated

This primitive is generated by the MLME as a result of an MLME-START.request to create a new BSS.

10.3.10.2.4 Effect of receipt

The SME is notified of the results of the BSS creation procedure.

10.4 PLME SAP interface

The PHY management service interface consists of the generic PLMEGET and PLMESET primitives on PHY MIB attributes, as described previously, together with the PLME-RESET primitive and the following specific primitives.

10.4.1 PLME-RESET.request

10.4.1.1 Function

This primitive shall be a request by the LME to reset the PHY. The PHY shall be always reset to the receive state to avoid accidental data transmission.

10.4.1.2 Semantics of the service primitive

The primitive shall provide the following parameters:

RESET.request ( )

There are no parameters associated with this primitive.

10.4.1.3 When generated

This primitive shall be generated at any time to reset the PHY.

10.4.1.4 Effect of receipt

Receipt of this primitive by the PHY sublayer shall cause the PHY entity to reset both the transmit and the receive state machines and place the PHY into the receive state.

10.4.2 PLME-DSSSTESTMODE.request

10.4.2.1 Function

This primitive requests that the DSSS PHY entity enter a test mode operation. The parameters associated with this primitive are considered as recommendations and are optional in any particular implementation.

10.4.2.2 Semantics of the service primitive

The primitive parameters are as follows:

```
PLME-DSSSTESTMODE.request     (
                              TEST_ENABLE,
                              TEST_MODE,
                              SCRAMBLE_STATE,
```


SPREADING_STATE,
DATA_TYPE,
DATA_RATE;
)

Name	Type	Valid range	Description
TEST_ENABLE	Boolean	True, false	If true, enables the PHY test mode according to the remaining parameters
TEST_MODE	Integer	1, 2, 3	TEST_MODE selects one of three operational states: 01 = transparent receive 02 = continuous transmit 03 = 50% duty cycle
SCRAMBLE_STATE	Boolean	True, false	If true, sets the operational state of the scrambler to ON
SPREADING_STATE	Boolean	True, false	If true, selects the operational state of the chipping
DATA_TYPE	Integer	1, 2, 3	Selects one of three data patterns to be used for the transmit portions of the tests
DATA_RATE	Integer	2, 4	Selects between 1 and 2 Mbit/s operation 02 = 1 Mbit/s 04 = 2 Mbit/s

10.4.2.3 When generated

This primitive shall be generated at any time to enter the DSSS PHY test mode.

10.4.2.4 Effect of receipt

Receipt of this primitive by the PHY sublayer shall cause the DSSS PHY entity to enter the test mode of operation.

10.4.3 PLME-DSSSTESTOUTPUT.request

10.4.3.1 Function

This optional primitive shall be a request by the LME to enable selected test signals from the PHY. The parameters associated with this primitive are considered as recommendations and are optional in any particular implementation.

10.4.3.2 Semantics of the service primitive

The primitive parameters are as follows:

PLME-DSSSTESTOUTPUT.request (
TEST_OUTPUT,
)

Name	Type	Valid range	Description
TEST_OUTPUT	Boolean	True, false	If true, enable the selected test signals for testing DS PHY

TEST_OUTPUT enables and disables selected signals for debugging and testing the PHY. Some signals that may be available for output are PHY-TXSTART.request, PHY-RXSTART.indicate(RXVECTOR), PHY-CCA.indicate, the chipping clock, the data clock, the symbol clock, TX data, and RX data.

10.4.3.3 When generated

This primitive shall be generated at any time to enable the test outputs when in the DSSS PHY test mode.

10.4.3.4 Effect of receipt

Receipt of this primitive by the DSSS PHY sublayer shall cause the DSSS PHY entity to enabled the test outputs using the modes set by the most recent PLME-DSSSTESTMODE.request primitive.

11. MAC sublayer management entity

11.1 Synchronization

All STAs within a single BSS shall be synchronized to a common clock using the mechanisms defined herein.

11.1.1 Basic approach

A timing synchronization function (TSF) keeps the timers for all STAs in the same BSS synchronized. All STAs shall maintain a local TSF timer.

11.1.1.1 TSF for infrastructure networks

In an infrastructure network, the AP shall be the timing master and shall perform the TSF. The AP shall initialize its TSF timer independently of any simultaneously started APs in an effort to minimize the synchronization of the TSF timers of multiple APs. The AP shall periodically transmit special frames called *beacons* that contain a copy of its TSF timer to synchronize the other STAs in a BSS. A receiving STA shall always accept the timing information in beacons sent from the AP servicing its BSS. If a STA's TSF timer is different from the timestamp in the received beacon, the receiving STA shall set its local timer to the received timestamp value.

Beacons shall be generated for transmission by the AP once every BeaconPeriod time units.

11.1.1.2 TSF for an independent BSS (IBSS)

The TSF in an IBSS shall be implemented via a distributed algorithm that shall be performed by all of the members of the BSS. Each STA in the BSS shall transmit beacons according to the algorithm described in this clause. Each STA in an IBSS shall adopt the timing received from any beacon or probe response that has a TSF value later than its own TSF timer.

11.1.2 Maintaining synchronization

Each STA shall maintain a TSF timer with modulus 2^{64} counting in increments of microseconds. STAs expect to receive beacons at a nominal rate. The interval between beacons is defined by the aBeaconPeriod parameter of the STA. A STA sending a beacon shall set the value of the beacon's timestamp so that it equals the value of the STA's TSF timer at the time that the first bit of the timestamp is transmitted to the PHY plus the transmitting STA's delays through its local PHY from the MAC-PHY interface to its interface with the wireless medium (antenna, LED emission surface, etc.). The algorithms in this clause define a mechanism that maintains the synchronization of the TSF timers in a BSS to within 4 µs plus the maximum propagation delay of the PHY for PHYs of 1 Mb/s, or greater.

11.1.2.1 Beacon generation in infrastructure networks

The AP shall define the timing for the entire BSS by transmitting beacons according to the aBeaconPeriod attribute within the AP. This defines a series of TBTTs exactly aBeaconPeriod time units apart. Time zero is defined to be a TBTT with the beacon being a DTIM and transmitted at the beginning of a CFP. At each TBTT, the AP shall schedule a beacon as the next frame for transmission. If the medium is determined by the carrier-sense mechanism (see 9.2.1) to be unavailable, the AP shall delay the actual transmission of a beacon according to the basic medium access rules specified in Clause 9. The beacon period is included in Beacon and Probe Response frames, and STAs shall adopt that beacon period when joining the BSS.

NOTE—Though the transmission of a beacon may be delayed because of CSMA deferrals, subsequent beacons shall be scheduled at the nominal beacon interval. This is shown in Figure 64.

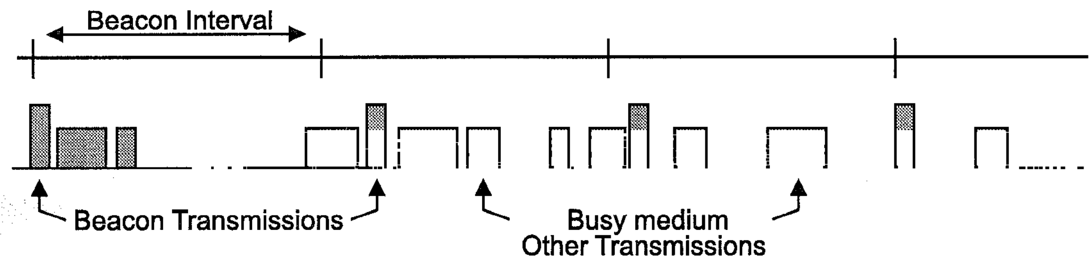


Figure 64—Beacon transmission on a busy network

11.1.2.2 Beacon generation in an IBSS

Beacon generation in an IBSS is distributed. The beacon period is included in Beacon and Probe Response frames, and STAs shall adopt that beacon period when joining the IBSS. All members of the IBSS participate in beacon generation. Each STA shall maintain its own TSF timer that is used for aBeaconPeriod timing. The beacon interval within an IBSS is established by the STA that instantiates the IBSS. This defines a series of TBTTs exactly aBeaconPeriod time units apart. Time zero is defined to be a TBTT. At each TBTT the STA shall

a) Suspend the decrementing of the backoff timer for any pending non-beacon or non-ad hoc traffic indication (ATIM) transmission,
b) Calculate a random delay uniformly distributed in the range between zero and twice aCWmin × aSlotTime,
c) Wait for the period of the random delay, decrementing the random delay timer using the same algorithm as for backoff,
d) If a beacon arrives before the random delay timer has expired, then the remaining random delay is canceled, the pending beacon transmission is canceled, and the ATIM backoff timer shall resume decrementing,
e) If the random delay has expired and no beacon has arrived during the delay period, send a beacon.

(See Figure 65.)

The beacon transmission shall always occur during the Awake Period of STAs that are operating in a low-power mode. This is described in more detail in 11.2.

11.1.2.3 Beacon reception

STAs shall use information from the CF Parameter Set element of all received Beacon frames to update their NAV as specified in 9.3.2.2.

STAs in an infrastructure network shall only use other information in received Beacon frames, if the BSSID field are equal to the MAC address currently in use by the STA contained in the AP of the BSS.

STAs in an IBSS shall use other information in any received Beacon frame for which the IBSS subfield of the Capability field is set to 1 and the content of the SSID element is equal to the SSID of the IBSS. Use of this information is specified in 11.1.4.

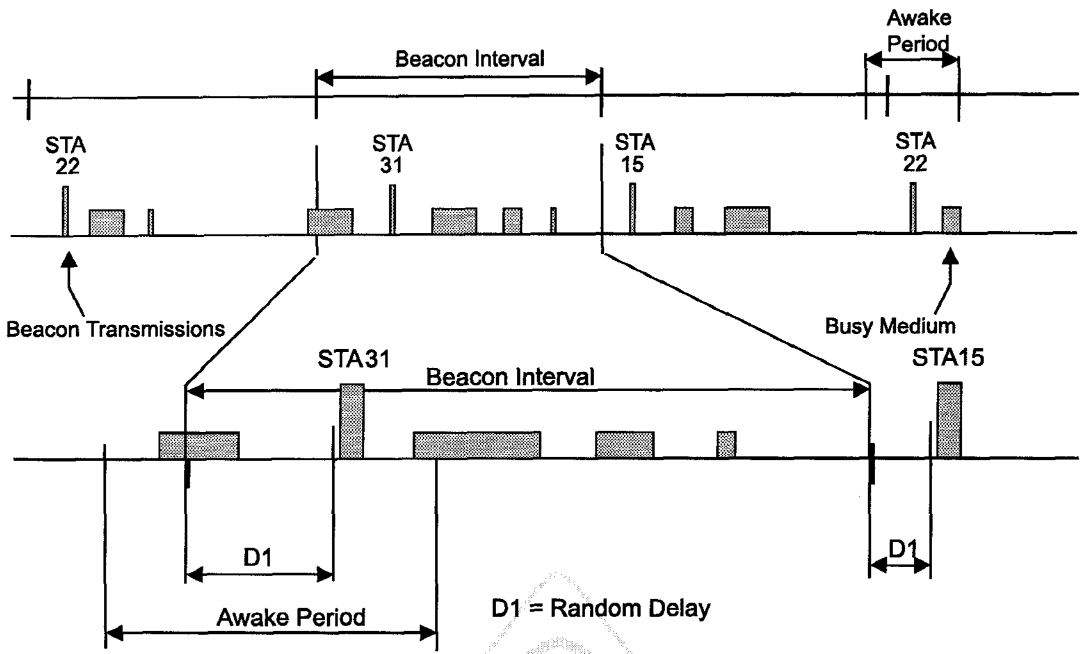


Figure 65—Beacon transmission in an IBSS

11.1.2.4 TSF timer accuracy

Upon receiving a Beacon frame with a valid FCS and BSSID or SSID, as described in 11.1.2.3, a STA shall update its TSF timer according to the following algorithm: The received timestamp value shall be adjusted by adding an amount equal to the receiving STA's delay through its local PHY components plus the time since the first bit of the timestamp was received at the MAC/PHY interface. In the case of an infrastructure BSS, the STA's TSF timer shall then be set to the adjusted value of the timestamp. In the case of an IBSS, the STA's TSF timer shall be set to the adjusted value of the received timestamp, if the adjusted value of the timestamp is later than the value of the STA's TSF timer. The accuracy of the TSF timer shall be ±0.01%.

11.1.3 Acquiring synchronization, scanning

A STA shall operate in either a Passive Scanning mode or an Active Scanning mode depending on the current value of the ScanMode parameter of the MLME-SCAN.request primitive.

Upon receipt of the MLME-SCAN.request primitive, a STA shall perform scanning. The SSID parameter indicates the SSID for which to scan. To become a member of a particular ESS using passive scanning, a STA shall scan for Beacon frames containing that ESS's SSID, returning all Beacon frames matching the desired SSID in the BSSDescriptionSet parameter of the corresponding MLME-SCAN.confirm primitive with the appropriate bits in the Capabilities Information field indicating whether the beacon came from an Infrastructure BSS or IBSS. To actively scan, the STA shall transmit Probe frames containing the desired SSID. Upon completion of scanning, an MLME-SCAN.confirm is issued by the MLME indicating all of the BSS information received.

Upon receipt of an MLME-JOIN.request, the STA will join a BSS by adopting the BSSID, TSF timer value, PHY parameters, and the beacon period specified in the request.

Upon receipt of an MLME-SCAN.request with the broadcast SSID, the STA shall passively scan for any Beacon frames, or actively transmit Probe frames containing the broadcast SSID, as appropriate depending upon the value of ScanMode. Upon completion of scanning, an MLME-SCAN.confirm is issued by the MLME indicating all of the BSS information received.

If a STA's scanning does not result in finding a BSS with the desired SSID and of the desired type, or does not result in finding any BSS, the STA may start an IBSS upon receipt of the MLME-START.request.

A STA may start its own BSS without first scanning for a BSS to join.

When a STA starts a BSS, that STA shall determine the BSSID of the BSS. If the BSSType indicates an infrastructure BSS, then the STA shall start an infrastructure BSS and the BSSID shall be equal to the STA's aStationID. The value of the BSSID shall remain unchanged, even if the value of aStationID is changed after the completion of the MLME-Start.request. If the BSSType indicates an IBSS, the STA shall start an IBSS, and the BSSID shall be an individual locally administered IEEE MAC address as defined in 5.2 of IEEE Std 802-1990. The remaining 46 bits of that MAC address shall be a number selected in a manner that minimizes the probability of STAs generating the same number, even when those STAs are subjected to the same initial conditions. The value SSID parameter shall be used as the SSID of the new BSS. It is important that designers recognize the need for statistical independence among the random number streams among STAs.

11.1.3.1 Passive scanning

If a ScanType is passive, the STA shall listen to each channel scanned for no longer than a maximum duration defined by the ChannelTime parameter.

11.1.3.2 Active scanning

Active scanning involves the generation of Probe frames and the subsequent processing of received Probe Response frames. The details of the active scanning procedures are as specified in the following subclauses.

11.1.3.2.1 Sending a probe response

STAs, subject to criteria below, receiving Probe Request frames shall respond with a probe response only if the SSID in the probe request is the broadcast SSID or matches the specific SSID of the STA. Probe Response frames shall be sent as directed frames to the address of the STA that generated the probe request. The probe response shall be sent using normal frame transmission rules. An AP shall respond to all probe requests meeting the above criteria. In an IBSS, the STA that generated the last beacon shall be the STA that responds to a probe request.

In each BSS there shall be at least one STA that is awake at any given time to respond to probe requests. A STA that sent a beacon shall remain in the Awake state and shall respond to probe requests until a Beacon frame with the current BSS ID is received. If the STA is an AP, it shall always remain in the Awake state and always respond to probe requests. There may be more than one STA in an IBSS that responds to any given probe request, particularly in cases where more than one STA transmitted a Beacon frame following the most recent TBTT, either due to not receiving successfully a previous beacon or due to collisions between beacon transmissions.

11.1.3.2.2 Active scanning procedure

Upon receipt of the MLME-SCAN.request with ScanType indicating an active scan, a STA shall use the following procedure:

For each channel to be scanned,

a) Wait until the ProbeDelay time has expired or a PHYRxStart.indication has been received;
b) Perform the Basic Access procedure as defined in 9.2.5.1;

c) Send a probe with the broadcast destination, SSID, and broadcast BSSID;
d) Clear and start a ProbeTimer;
e) If PHYCCA.indication (busy) has not been detected before the ProbeTimer reaches MinChannelTime, then clear NAV and scan the next channel, else when ProbeTimer reaches MaxChannelTime, process all received probe responses;
f) Clear NAV and scan the next channel.

See Figure 66.

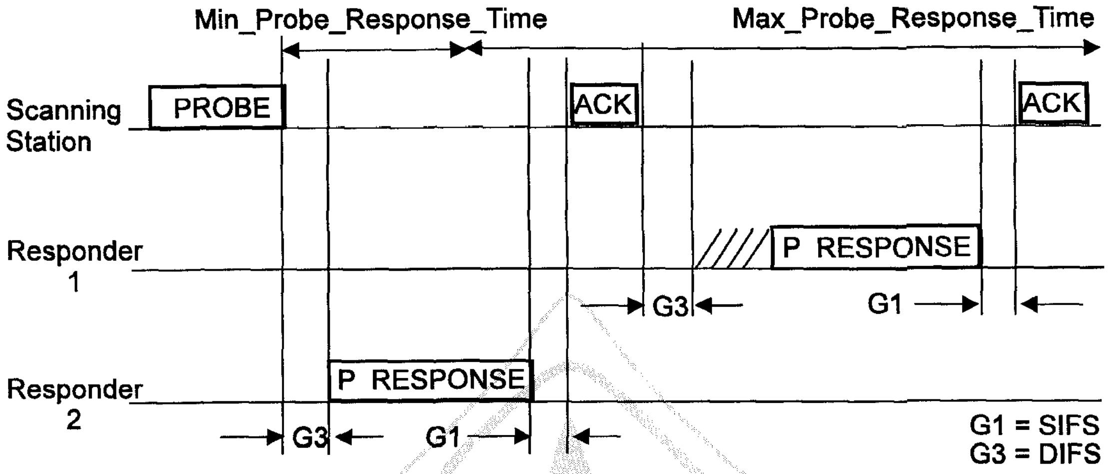


Figure 66—Probe response

When all channels in the ChannelList have been scanned, the MLME shall issue an MLME-Scan.confirm with the BSSDescriptionSet containing all of the information gathered during the scan.

11.1.3.3 Initializing a BSS

Upon receipt of an MLME-Start.request, a STA shall determine the BSS's BSSID (as described in 11.1.3), select channel synchronization information, select a beacon period, initialize and start its TSF timer, and begin transmitting beacons.

11.1.3.4 Synchronizing with a BSS

Upon receipt of an MLME-Join.request, a STA shall adopt the BSSID, channel synchronization information, and TSF timer value of the parameters in the request. Upon receipt of a Beacon frame from the BSS, the MLME shall issue an MLME-Join.confirm indicating the operation was successful. If the JoinFailureTimeout expires prior to the receipt of a Beacon frame from the BSS, the MLME shall issue an MLME-Join.confirm indicating the operation was unsuccessful.

11.1.4 Adjusting STA timers

In the infrastructure network, STAs shall always adopt the timer in a beacon or probe response coming from the AP in their BSS.

In an IBSS, a STA shall always adopt the information in the contents of a Beacon or Probe Response frame when that frame contains a matching SSID and the value of the time stamp is later than the STA's TSF timer. In response to an MLME-Join.request, a STA shall initialize its TSF timer to 0 and shall not transmit a beacon or probe response until it hears a beacon or probe response from a member of the IBSS with a matching SSID.

All Beacon and Probe Response frames carry a Timestamp field. A STA receiving such a frame from another STA in an IBSS with the same SSID shall compare the Timestamp field with its own TSF time. If the Timestamp field of the received frame is later than its own TSF time, the STA shall adopt all parameters contained in the Beacon frame.

11.1.5 Timing synchronization for frequency-hopping (FH) PHYs

NOTE—This subclause pertains only to STAs using an FH PHY.

The TSF described here provides a mechanism for STAs in an FH system to synchronize their transitions from one channel to another (their "hops"). Every STA shall maintain a table of all of the hopping sequences that are used in the system. All of the STAs in a BSS shall use the same hopping sequence. Each beacon and probe response includes the channel synchronization information necessary to determine the hop pattern and timing for the BSS.

STAs shall use their TSF timer to time the aCurrentDwellTime. The aCurrentDwellTime is the length of time that STAs shall stay on each frequency in their hopping sequence. Once STAs are synchronized, they have the same TSF timer value.

STAs in the BSS shall issue an appropriate PLME service primitive for the PHY in use to tune to the next frequency in the hopping sequence whenever

TSF timer MOD aCurrentDwellTime = 0

11.2 Power management

11.2.1 Power management in an infrastructure network

STAs changing Power Management mode shall inform the AP of this fact using the Power Management bits within the Frame Control field of transmitted frames. The AP shall not arbitrarily transmit MSDUs to STAs operating in a power-saving mode, but shall buffer MSDUs and only transmit them at designated times.

The STAs that currently have buffered MSDUs within the AP are identified in a *traffic indication map* (TIM), which shall be included as an element within all beacons generated by the AP. A STA shall determine that an MSDU is buffered for it by receiving and interpreting a TIM.

STAs operating in Power-Save (PS) modes shall periodically listen for beacons, as determined by the STA's ListenInterval and ReceiveDTIMs parameters of the MLME-Power-Mgt.request primitive.

In a BSS operating under the DCF, or during the contention period of a BSS using the PCF, upon determining that an MSDU is currently buffered in the AP, a STA operating in the *PS mode* shall transmit a short PS-Poll frame to the AP, which shall respond with the corresponding buffered MSDU immediately, or acknowledge the PS-Poll and respond with the corresponding MSDU at a later time. If the TIM indicating the buffered MSDU is sent during a contention-free period (CFP), a CF-Pollable STA operating in the PS mode does not send a PS-Poll frame, but remains active until the buffered MSDU is received (or the CFP ends). If any STA in its BSS is in PS mode, the AP shall buffer all broadcast and multicast MSDUs and deliver them to all STAs immediately following the next Beacon frame containing a *delivery TIM* (DTIM) transmission.

A STA shall remain in its current Power Management mode until it informs the AP of a Power Management mode change via a successful frame exchange. Power Management mode shall not change during any single frame exchange sequence, as described in 9.7.

11.2.1.1 STA Power Management modes

A STA may be in one of two different power states:

— *Awake:* STA is fully powered.
— *Doze:* STA is not able to transmit or receive and consumes very low power.

The manner in which a STA transitions between these two power states shall be determined by the STA's Power Management mode. These modes are summarized in Table 23.

The Power Management mode of a STA is selected by the PowerManagementMode parameter of the MLME-POWERMGT.request. Once the STA updates its Power Management mode, the MLME shall issue an MLME-POWERMGT.confirm indicating the success of the operation.

Table 23—Power Management modes

Active mode or AM	STA may receive frames at any time. In Active mode, a STA shall be in the Awake state. A STA on the polling list of a PCF shall be in Active mode for the duration of the CFP.
Power Save or PS	STA listens to selected beacons (based upon its aListenInterval) and sends PS-Poll frames to the AP if the TIM element in the most recent beacon indicates a directed MSDU buffered for that STA. The AP shall transmit buffered directed MSDUs to a PS STA only in response to a PS-Poll from that STA, or during the CFP in the case of a CF-Pollable PS STA. In PS mode, a STA shall be in the Doze state and shall enter the Awake state to receive selected beacons, to receive broadcast and multicast transmissions following certain received beacons, to transmit, and to await responses to transmitted PS-Poll frames or (for CF-Pollable STAs) to receive contention-free transmissions of buffered MSDUs.

To change Power Management modes, a STA shall inform the AP through a successful frame exchange initiated by the STA. The Power Management bit in the Frame Control field of the frame sent by the STA in this exchange indicates the Power Management mode that the STA shall adopt upon successful completion of the entire frame exchange.

A STA that is changing from Doze to Awake in order to transmit shall perform clear channel assessment (CCA) until a frame sequence is detected by which it can correctly set its NAV, or until a period of time equal to the ProbeDelay has transpired.

11.2.1.2 AP TIM transmissions

The TIM shall identify the STAs for which traffic is pending and buffered in the AP. This information is coded in a *partial virtual bitmap*, as described in 7.3.2.6. In addition, the TIM contains an indication whether broadcast/multicast traffic is pending. Every STA is assigned a Association ID code (AID) by the AP as part of the association process. AID 0 (zero) is reserved to indicate the presence of buffered broadcast/multicast MSDUs. The AP shall identify those STAs for which it is prepared to deliver buffered MSDUs by setting bits in the TIM's partial virtual bitmap that correspond to the appropriate SIDs.

11.2.1.3 TIM types

Two different TIM types are distinguished: TIM and DTIM. After a DTIM, the AP shall send out the buffered broadcast/multicast MSDUs using normal frame transmission rules, before transmitting any unicast frames.

The AP shall transmit a TIM with every beacon. Every DTIMPeriod, a TIM of type "DTIM" is transmitted within a beacon, rather than an ordinary TIM.

Figure 67 illustrates the AP and STA activity under the assumption that a DTIM is transmitted once every three TIMs. The top line in Figure 67 represents the time axis, with the beacon interval shown together with a DTIM Interval of three beacon intervals. The second line depicts AP activity. The AP schedules beacons for transmission every beacon interval, but the beacons may be delayed if there is traffic at the TBTT. This is indicated as "busy medium" on the second line. For the purposes of this figure, the important fact about beacons is that they contain TIMs, some of which may be DTIMs. Note that the second STA with ReceiveDTIMs set to false does not power up its receiver for all DTIMs.

The third and fourth lines in Figure 67 depict the activity of two STAs operating with different power management requirements. Both STAs power-on their receivers whenever they need to listen for a TIM. This is indicated in as a ramp-up of the receiver power prior to the TBTT. The first STA, for example, powers up its receiver and receives a TIM in the first beacon; that TIM indicates the presence of a buffered MSDU for the receiving STA. The receiving STA then generates a PS-Poll frame, which elicits the transmission of the buffered data MSDU from the AP. Broadcast and multicast MSDUs are sent by the AP subsequent to the transmission of a beacon containing a DTIM. The DTIM is indicated by the DTIM count field of the TIM element having a value of 0.

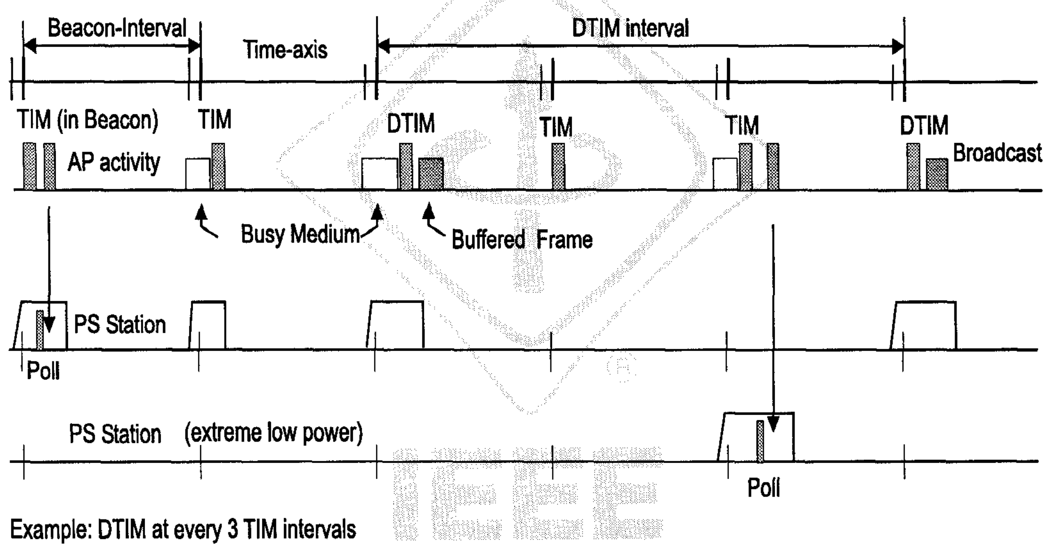


Figure 67—Infrastructure power management operation (no PCF operating)

11.2.1.4 AP operation during the contention period

APs shall maintain a Power Management status for each currently associated STA that indicates in which Power Management mode the STA is currently operating. An AP shall, depending on the Power Management mode of the STA, temporarily buffer the MSDU or management frame destined to the STA. No MSDUs or management frames received for STAs operating in the Active mode shall be buffered for power management reasons.

a) MSDUs, or management frames destined for PS STAs, shall be temporarily buffered in the AP. The algorithm to manage this buffering is beyond the scope of this standard.

b) MSDUs, or management frames destined for STAs in the Active mode, shall be directly transmitted.

c) At every beacon interval, the AP shall assemble the partial virtual bitmap containing the buffer status per destination for STAs in the PS mode, and shall send this out in the TIM field of the beacon. The bit for AID 0 (zero) shall be set whenever broadcast or multicast traffic is buffered.
d) All broadcast/multicast MSDUs, with the Order bit in the Frame Control field clear, shall be buffered if any associated STAs are in PS mode.
e) Immediately after every DTIM, the AP shall transmit all buffered broadcast/multicast MSDUs. The More Data field of each broadcast/multicast frame shall be set to indicate the presence of further buffered broadcast/multicast MSDUs. If the AP is unable to transmit all of the buffered broadcast/multicast MSDUs before the TBTT following the DTIM, the AP shall indicate that it will continue to deliver the broadcast/multicast MSDUs by setting the broadcast/multicast bit in the partial virtual bitmap of the TIM element of every Beacon frame, until all buffered broadcast/multicast frames have been transmitted.
f) A single buffered MSDU or management frame for a STA in the PS mode shall be forwarded to the STA after a PS-Poll has been received from that STA. The More Data field shall be set to indicate the presence of further buffered MSDUs or management frames for the polling STA. Further PS-Poll frames from the same STA shall be acknowledged and ignored until the MSDU or management frame has either been successfully delivered, or presumed failed due to maximum retries being exceeded. This prevents a retried PS-Poll from being treated as a new request to deliver a buffered frame.
g) An AP shall have an aging function to delete pending traffic when it is buffered for an excessive time period.
h) Whenever an AP is informed that a STA changes to the Active mode, then the AP shall send buffered MSDUs and management frames (if any exist) to that STA without waiting for a PS-Poll.

11.2.1.5 AP operation during the CFP

APs shall maintain a Power Management status for each currently associated CF-Pollable STA that indicates in which Power Management mode the STA is currently operating. An AP shall, for STAs in PS mode, temporarily buffer the MSDU destined to the STA.

a) MSDUs destined for PS STAs shall be temporarily buffered in the AP. The algorithm to manage this buffering is beyond the scope of this standard.
b) MSDUs destined to STAs in the Active mode shall be transmitted as defined in Clause 9.
c) Prior to every CFP, and at each beacon interval within the CFP, the AP shall assemble the partial virtual bitmap containing the buffer status per destination for STAs in the PS mode, set the bits in the partial virtual bitmap for STAs the point coordinator (PC) is intending to poll during this CFP, and shall send this out in the TIM field of the DTIM. The bit for AID 0 (zero) shall be set whenever broadcast or multicast traffic is buffered.
d) All broadcast and multicast MSDUs, with the Order bit in the Frame Control field clear, shall be buffered if any associated STAs are in the PS mode, whether or not those STAs are CF-Pollable.
e) Immediately after every DTIM (Beacon frame with DTIM Count field of the TIM element equal to zero), the AP shall transmit all buffered broadcast and multicast frames. The More Data field shall be set to indicate the presence of further buffered broadcast/multicast MSDUs. If the AP is unable to transmit all of the buffered broadcast/multicast MSDUs before the TBTT following the DTIM, the AP shall indicate that it will continue to deliver the broadcast/multicast MSDUs by setting the broadcast/multicast bit in the partial virtual bitmap of the TIM element of every Beacon frame, until all buffered broadcast/multicast frames have been transmitted.
f) Buffered MSDUs or management frames for STAs in the PS mode shall be forwarded to the CF-Pollable STAs under control of the PC. Transmission of these buffered MSDUs or management frames shall begin immediately after transmission of buffered broadcast and multicast frames (if any), and shall occur in order by increasing AID of CF-Pollable STAs. A CF-Pollable STA for which the TIM element of the most recent beacon indicated buffered MSDUs or management frames shall be in the Awake state at least until the receipt of a directed frame from the AP in which the Frame Control field does not indicate the existence of more buffered MSDUs or management frames. After

acknowledging the last of the buffered MSDUs or management frames, the CF-Pollable STA operating in the PS mode may enter the Doze state until the next DTIM is expected.

g) An AP shall have an aging function to delete pending traffic buffered for an excessive time period. The exact specification of the aging function is beyond the scope of this standard.

h) Whenever an AP detects that a CF-Pollable STA has changed from the PS mode to the Active mode, then the AP shall queue any buffered frames addressed to that STA for transmission to that CF-Pollable STA as directed by the AP's PC function (PCF).

11.2.1.6 Receive operation for STAs in PS mode during the contention period

STAs in PS mode shall operate as follows to receive an MSDU or management frame from the AP when no PC is operating and during the contention period when a PC is operating.

a) STAs shall wake up early enough to be able to receive the next scheduled beacon after ListenInterval from the last TBTT.

b) When a STA detects that the bit corresponding to its AID is set in the TIM, the STA shall issue a PS-Poll to retrieve the buffered MSDU or management frame. If more than one bit is set in the TIM, the PS-Poll shall be transmitted after a random delay uniformly distributed between zero and aCWmin.

c) The STA shall remain in the Awake state until it receives the response to its poll, or it receives another beacon whose TIM indicates that the AP does not have any MSDUs or management frames buffered for this STA. If the bit corresponding to the STA's AID is set in the subsequent TIM, the STA shall issue another PS-Poll to retrieve the buffered MSDU or management frame(s).

d) If the More Data field in the received MSDU or management frame indicates that more traffic for that STA is buffered, the STA, at its convenience, shall Poll until no more MSDUs or management frames are buffered for that STA.

e) When ReceiveDTIMs is true, the STA shall wake up early enough to be able to receive every DTIM. A STA receiving broadcast/multicast MSDUs shall remain awake until the More Data field of the broadcast/multicast MSDUs indicate there are no further buffered broadcast/multicast MSDUs, or until a TIM is received indicating there are no more buffered broadcast/multicast MSDUs.

11.2.1.7 Receive operation for STAs in PS mode during the CFP

STAs in PS mode that are associated as CF-Pollable shall operate as follows in a BSS with an active PC to receive MSDUs or management frames from the AP during the CFP:

a) STAs shall enter the Awake state so as to receive the Beacon frame (which contains a DTIM) at the start of each CFP.

b) To receive broadcast/multicast MSDUs, the STA shall wake up early enough to be able to receive every DTIM that may be sent during the CFP. A STA receiving broadcast/multicast MSDUs shall remain awake until the More Data field of the broadcast/multicast MSDUs indicate there are no further buffered broadcast/multicast MSDUs, or until a TIM is received indicating there are no more buffered broadcast/multicast MSDUs buffered.

c) When a STA detects that the bit corresponding to its AID is set in the DTIM at the start of the CFP (or in a subsequent TIM during the CFP), the STA shall remain in the Awake state for at least that portion of the CFP through the time that the STA receives a directed MSDU or management frame from the AP with the More Data field in the Frame Control field indicating that no further traffic is buffered.

d) If the More Data field in the Frame Control field of the last MSDU or management frame received from the AP indicates that more traffic for the STA is buffered, then, when the CFP ends, the STA may remain in the Awake state and transmit PS-Poll frames during the contention period to request the delivery of additional buffered MSDU or management frames, or may enter the Doze state during the contention period (except at TBTTs for DTIMs expected during the contention period), awaiting the start of the next CFP.

11.2.1.8 STAs operating in the Active mode

A STA operating in this mode shall have its receiver activated continuously; it does not need to interpret the traffic announcement part of the beacons.

11.2.1.9 AP aging function

The AP shall have an aging function to delete buffered traffic when it has been buffered for an excessive period of time. That function shall be based on the aListenInterval of the STA for which the traffic is buffered. The AP aging function shall not cause the buffered traffic to be discarded after any period that is shorter than the aListenInterval of the STA for which the traffic is buffered. The exact specification of the aging function is beyond the scope of this standard.

11.2.2 Power management in an IBSS

This subclause specifies the power management mechanism for use within an IBSS.

11.2.2.1 Basic approach

The basic approach is similar to the infrastructure case in that the STAs are synchronized, and multicast MSDUs and those MSDUs that are to be transmitted to a power-conserving STA are first announced during a period when all STAs are awake. The announcement is done via an ad hoc traffic indication message (ATIM). A STA in the PS mode shall listen for these announcements to determine if it needs to remain in the awake state.

When an MSDU is to be transmitted to a destination STA that is in a PS mode, the transmitting STA first transmits an ATIM frame during the ATIM Window, in which all the STAs including those operating in a PS mode are awake. The ATIM Window is defined as a specific period of time, defined by aATIMWindow, following a TBTT, during which only beacon or ATIM frames shall be transmitted. ATIM transmission times are randomized, after a Beacon frame is either transmitted or received by the STA, using the backoff procedure with the contention window equal to aCWminx. Directed ATIMs shall be acknowledged. If a STA transmitting a directed ATIM does not receive an acknowledgment, the STA shall execute the backoff procedure for retransmission of the ATIM. Multicast ATIMs shall not be acknowledged.

If a STA receives a directed ATIM frame during the ATIM Window, it shall acknowledge the directed ATIM and stay awake for the entire beacon interval waiting for the announced MSDU(s) to be received. If a STA does not receive an ATIM, it may enter the Doze state at the end of the ATIM Window. Transmissions of MSDUs announced by ATIMs are randomized after the ATIM Window, using the backoff procedure described in Clause 9.

It is possible that an ATIM may be received from more than one STA, and that a STA that receives an ATIM may receive more than a single MSDU from the transmitting STA. ATIM frames are only addressed to the destination STA of the MSDU.

An ATIM for a broadcast or multicast MSDU shall have a destination address identical to that of the MSDU.

After the ATIM interval, only those directed MSDUs that have been successfully announced with an acknowledged ATIM, and broadcast/multicast MSDUs that have been announced with an ATIM, shall be transmitted to STAs in the PS mode. Transmission of these frames shall be done using the normal DCF access procedure.

Figure 68 illustrates the basic power-save operation.

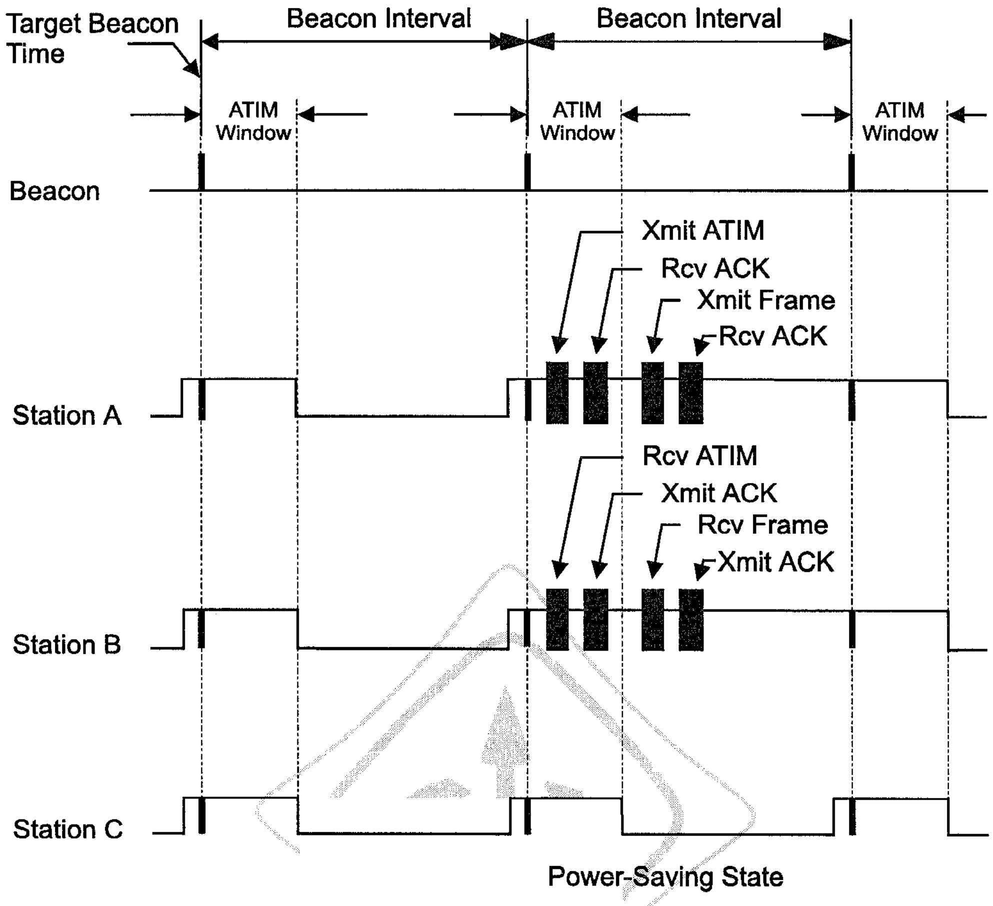


Figure 68—Power management in an IBSS—Basic operation

The estimated power-saving state of another STA may be based on the power management information transmitted by that STA and on additional information available locally, such as a history of failed transmission attempts. The use of RTS/CTS in an IBSS may reduce the number of transmissions to a STA that is in PS mode. If an RTS is sent and a CTS is not received, the transmitting STA may assume that the destination STA is in PS mode. The method of estimating the power management state of other STAs in the IBSS is outside the scope of this standard.

11.2.2.2 Initialization of power management within an IBSS

The following procedure shall be used to initialize power management within a new IBSS, or to learn about the power management being used within an existing IBSS.

a) A STA joining an existing IBSS by the procedure in 11.1.3.3 shall update its ATIM Window with the value contained in the ATIM Window field of the IBSS Parameter Set element within the Beacon or Probe Response management frame received during the scan procedure.

b) A STA creating a new IBSS by the procedure in 11.1.3.3 shall set the value of the ATIM Window field of the IBSS Parameter Set element within the Beacon management frames transmitted to the value of its ATIM Window.

c) The start of the ATIM Window shall be the TBTT, defined in 11.1.2.2. The end of the ATIM Window shall be defined as

TSF timer MOD BeaconInterval = ATIMWindow.

d) The ATIM Window period shall be static during the lifetime of the IBSS.

e) An ATIM Window value of zero shall indicate that power management is not in use within the IBSS.

11.2.2.3 STA power state transitions

A STA may enter PS mode if and only if the value of the ATIM Window in use within the IBSS is greater than zero. A STA shall set the Power Management subfield in the Frame Control field of MSDUs that it transmits according to the procedure in 7.1.3.1.7.

A STA in PS mode shall transition between Awake and Doze states according to the following rules:

a) If a STA is operating in PS mode, it shall enter the Awake state prior to each TBTT.
b) If a STA receives a directed ATIM management frame containing its individual address, or a multicast ATIM management frame during the ATIM Window it shall remain in the Awake state until the end of the next ATIM Window.
c) If a STA transmits a Beacon or an ATIM management frame, it shall remain in the Awake state until the end of the next ATIM Window regardless of whether an acknowledgment is received for the ATIM.
d) If the STA has not transmitted an ATIM and does not receive either a directed ATIM management frame containing its individual address, or a multicast ATIM management frame during the ATIM Window, it may return to the Doze state following the end of the current ATIM Window.

11.2.2.4 ATIM and frame transmission

If power management is in use within an IBSS, all STAs shall buffer MSDUs for STAs that are known to be in PS mode. The algorithm used for the estimation of the power management state of STAs within the IBSS is outside the scope of this standard. MSDUs to STAs in Active mode may be sent at any valid time.

a) Following the reception or transmission of the beacon, during the ATIM Window, the STA shall transmit a directed ATIM management frame to each STA for which it has one or more buffered unicast MSDUs. If the STA has one or more buffered multicast MSDUs, with the Strictly Ordered bit clear, it shall transmit an appropriately addressed multicast ATIM frame. A STA transmitting an ATIM management frame shall remain awake for the entire current beacon interval.
b) All STAs shall use the backoff procedure defined in 9.2.5.2 for transmission of the first ATIM following the beacon. All remaining ATIMs shall be transmitted using the conventional DCF access procedure.
c) ATIM management frames shall only be transmitted during the ATIM Window.
d) A STA shall transmit no frame types other than RTS, CTS, and ACK Control frames and Beacon and ATIM management frames during the ATIM Window.
e) Directed ATIM management frames shall be acknowledged. If no acknowledgment is received, the ATIM shall be retransmitted using the conventional DCF access procedure. Multicast ATIM management frames shall not be acknowledged.
f) If a STA is unable to transmit an ATIM during the ATIM Window, for example due to contention with other STAs, the STA shall retain the buffered MSDU(s) and attempt to transmit the ATIM during the next ATIM Window.
g) Immediately following the ATIM Window, a STA shall begin transmission of buffered broadcast/multicast frames for which an ATIM was previously transmitted. Following the transmission of any broadcast/multicast frames, any MSDUs and management frames addressed to STAs for which an acknowledgment for a previously transmitted ATIM frame was received shall be transmitted. All STAs shall use the backoff procedure defined in 9.2.5.2 for transmission of the first frame following the ATIM Window. All remaining frames shall be transmitted using the conventional DCF access procedure.

h) A buffered MSDU may be transmitted using fragmentation. If an MSDU has been partially transmitted when the next beacon frame is sent, the STA shall retain the buffered MSDU and announce the remaining fragments by transmitting an ATIM during the next ATIM Window.
i) If an STA is unable to transmit a buffered MSDU during the beacon interval in which it was announced, for example due to contention with other STAs, the STA shall retain the buffered MSDU and announce the MSDU again by transmitting an ATIM during the next ATIM Window.
j) Following the transmission of all buffered MSDUs, a STA may transmit MSDUs without announcement to STAs that are known to be in the Awake state for the current beacon interval due to an appropriate ATIM management or Beacon frame having been transmitted or received.
k) A STA may discard frames buffered for later transmission to power-saving STAs if the STA determines that the frame has been buffered for an excessive amount of time or if other conditions internal to the STA implementation make it desirable to discard buffered frames (for example, buffer starvation). In no case shall a frame be discarded that has been buffered for less than aBeaconPeriod. The algorithm to manage this buffering is beyond the scope of this standard.

11.3 Association and reassociation

This subclause defines how a STA associates and reassociates with an AP.

11.3.1 STA association procedures

Upon the receipt of an MLME-ASSOCIATE.request, a STA shall associate with an AP via the following procedure:

a) The STA shall transmit an association request to an AP with which that STA is authenticated.
b) If an Association Response frame is received with a status value of "successful," the STA is now associated with the AP and the MLME shall issue an MLME-ASSOCIATE.confirm indicating the successful completion of the operation.
c) If an Association Response frame is received with a status value other than "successful" or the AssociateFailureTimeout expires, the STA is not associated with the AP and the MLME shall issue an MLME-ASSOCIATE.confirm indicating the failure of the operation.

11.3.2 AP association procedures

An AP shall operate as follows in order to support the association of STAs.

a) Whenever an Association Request frame is received from a STA and the STA is authenticated, the AP shall transmit an association response with a status code as defined in 7.3.1.9. If the status value is "successful," the Association ID assigned to the STA shall be included in the response. If the STA is not authenticated, the AP shall transmit a Deauthentication frame to the STA.
b) When the association response with a status value of "successful" is acknowledged by the STA, the STA is considered to be associated with this AP.
c) The AP shall inform the distribution system (DS) of the association and the MLME shall issue an MLME-ASSOCIATE.indication.

11.3.3 STA reassociation procedures

Upon receipt of an MLME-REASSOCIATE.request, a STA shall reassociate with an AP via the following procedure:

a) The STA shall transmit a Reassociation Request frame to an AP.

b) If a Reassociation Response frame is received with a status value of "successful," the STA is now associated with the AP and the MLME shall issue an MLME-REASSOCIATE.confirm indicating the successful completion of the operation.
c) If a Reassociation Response frame is received with a status value other than "successful" or the ReassociateFailureTimeout expires, the STA is not associated with the AP and the MLME shall issue an MLME-REASSOCIATE.confirm indicating the failure of the operation.

11.3.4 AP reassociation procedures

An AP shall operate as follows in order to support the reassociation of STAs.

a) Whenever a Reassociation Request frame is received from a STA and the STA is authenticated, the AP shall transmit a reassociation response with a status value as defined in 7.3.1.9. If the status value is "successful," the Association ID assigned to the STA shall be included in the response. If the STA is not authenticated, the AP shall transmit a Deauthentication frame to the STA.
b) When the reassociation response with a status value of "successful" is acknowledged by the STA, the STA is considered to be associated with this AP.
c) The AP shall inform the DS of the reassociation and the MLME shall issue an MLME-REASSOCIATE.indication.

11.4 Management information base (MIB) definitions

The description of the MIB in this subclause defines the function of the various managed objects, attributes, actions, and notifications. The ASN.1 encoding of the MIB is presented in Annex D. In case of discrepancy between the definition in this subclause and that in Annex D, the definition in the annex shall take precedence.

11.4.1 MIB summary

The following summarizes the IEEE 802.11 MIB. Each group, attribute, action and notification is listed. This summary is for information purposes only. If any errors exist, the formal definitions have precedence.

11.4.1.1 STA management attributes

11.4.1.1.1 agStationConfiggrp

aStationID,
aMediumOccupancyLimit,
aCFPollable,
aAuthenticationType,
aAuthenticationAlgorithms
aCFPPeriod,
aCFPMaxDuration,
aAuthenticationResponseTimeout,
aWEPUndecryptableCount,
aReceiveDTIMs;

11.4.1.1.2 agPrivacygrp

aPrivacyOptionImplemented,
aPrivacyInvoked,
aWEPDefaultKeys,
aWEPDefaultKeyID,
aWEPKeyMappings

aWEPKeyMappingLength,
aExcludeUnencrypted,
aWEPICVErrorCount,
aWEPExcludedCount;

11.4.1.2 MAC attributes

11.4.1.2.1 agOperationgrp

aMACAddress,
aGroupAddresses,
aRTSThreshold,
aShortRetryLimit,
aLongRetryLimit,
aFragmentationThreshold,
aMaxTransmitMSDULifetime,
aMaxReceiveLifetime,
aManufacturerID,
aProductID;

11.4.1.2.2 agCountersgrp

aTransmittedFragmentCount,
aMulticastTransmittedFrameCount,
aFailedCount,
aRetryCount,
aMultipleRetryCount,
aFrameDuplicateCount,
aRTSSuccessCount,
aRTSFailureCount,
aACKFailureCount,
aReceivedFragmentCount,
aMulticastReceivedFrameCount,
aFCSErrorCount;

11.4.1.3 ResourceTypeID Attributes

11.4.1.3.1 Not grouped

aResourceTypeIDName,
aResourceInfo;

11.4.1.4 Notifications

11.4.1.4.1 SMT notifications

nDisassociate

11.4.2 Managed object class templates

11.4.2.1 SMT object class

11.4.2.1.1 oSMT

```
SMT MANAGED OBJECT CLASS
DERIVED FROM "ISO/IEC 10165-2":top;
CHARACTERIZED BY
    pSMTbase                                PACKAGE
        BEHAVIOUR
            bSMTbase BEHAVIOUR
                DEFINED AS "The SMT object class provides the necessary support at the STA to manage
                the processes in the STA such that the STA may work cooperatively as a part of an IEEE
                802.11 network.";
            ATTRIBUTES
                aStationID                          GET,
                aAuthenticationAlgorithms           GET,
                aAuthenticationType                 GET-REPLACE,
                aPrivacyOptionImplemented           GET,
                aMediumOccupancyLimit               GET-REPLACE,
                aCFPollable                         GET,
                aCFPPeriod                          GET-REPLACE,
                aCFPMaxDuration                     GET-REPLACE,
                aAuthenticationResponseTimeout      GET-REPLACE
                aReceiveDTIMs                       GET-REPLACE;
            ATTRIBUTE GROUPS
                agStationConfiggrp,
                agPrivacygrp;
            NOTIFICATIONS
                nDisassociate;
CONDITIONAL PACKAGES
    pSMTPrivacy PRESENT IF WEP Supported
        BEHAVIOUR
            bSMTPrivacy BEHAVIOUR
                DEFINED AS "The SMTPrivacy package is a set of attributes that shall be present if WEP
                is implemented in the STA."
            ATTRIBUTES
                aPrivacyInvoked                     GET-REPLACE,
                aWEPDefaultKeys                     REPLACE,
                aWEPDefaultKeyID                    GET-REPLACE,
                aWEPKeyMappings                     REPLACE,
                aWEPKeyMappingLength                GET,
                aExcludeUnencrypted                 GET-REPLACE,
                aWEPICVErrorCount                   GET-REPLACE,
                aWEPExcludedCount                   GET,
                aWEPUndecryptableCount              GET;
REGISTERED AS { iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) };
```

11.4.2.2 MAC object class

11.4.2.2.1 oMAC

```
MAC MANAGED OBJECT CLASS
DERIVED FROM "ISO/IEC 10165-2":top;
CHARACTERIZED BY
    pMACbase                                PACKAGE
        BEHAVIOUR
```

```
        bMACbase BEHAVIOUR
            DEFINED AS "The MAC object class provides the necessary support for the access con-
            trol, generation, and verification of frame check sequences, and proper delivery of valid
            data to upper layers.";
        ATTRIBUTES
            aMACAddress                          GET,
            aGroupAddresses                      GET-REPLACE,
            aTransmittedFragmentCount            GET-REPLACE,
            aMulticastTransmittedFrameCount      GET-REPLACE,
            aFailedCount                         GET-REPLACE,
            aReceivedFragmentCount               GET-REPLACE,
            aMulticastReceivedFrameCount         GET-REPLACE,
            aFCSErrorCount                       GET-REPLACE,
            aRTSThreshold                        GET-REPLACE,
            aShortRetryLimit                     GET-REPLACE,
            aLongRetryLimit                      GET-REPLACE
            aFragmentationThreshold              GET-REPLACE,
            aMaxTransmitMSDULifetime             GET-REPLACE,
            aMaxReceiveLifetime                  GET-REPLACE,
            aManufacturerID                      GET,
            aProductID                           GET;
        ATTRIBUTE GROUPS
            agOperationgrp,
agCountersgrp;
CONDITIONAL PACKAGES
    pMACStatistics PRESENT IF Supported
        BEHAVIOUR
        bMACStatistics BEHAVIOUR
            DEFINED AS "The MACStatistics package provides extended statistical information on
            the operation of the MAC."
        ATTRIBUTES
            aRetryCount                          GET-REPLACE,
            aMultipleRetryCount                  GET-REPLACE,
            aRTSSuccessCount                     GET-REPLACE,
            aRTSFailureCount                     GET-REPLACE,
            aACKFailureCount                     GET-REPLACE,
            aFrameDuplicateCount                 GET-REPLACE;
REGISTERED AS { iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) };
```

11.4.2.3 Resource type object class

11.4.2.3.1 oResourceTypeID

```
ResourceTypeID MANAGED OBJECT CLASS
DERIVED FROM IEEE802CommonDefinitions.oResourceTypeID;
CHARACTERIZED BY
    pResourceTypeID                      PACKAGE
        ATTRIBUTES
            aResourceTypeIDName                  GET,
            aResourceInfo                        GET;
REGISTERED AS { iso(1) member-body(2) us(840) ieee802dot11(10036) ResourceTypeID(3) };
```

11.4.3 Attribute group templates

11.4.3.1 STA management attribute group templates

11.4.3.1.1 agStationConfiggrp

```
StationConfiggrp ATTRIBUTE GROUP
    GROUP ELEMENTS
        aStationID,
        aMediumOccupancyLimit,
        aCFPollable,
        aAuthenticationType,
        aAuthenticationAlgorithms,
        aCFPPeriod,
        aCFPMaxDuration,
        aAuthenticationResponseTimeout,
        aWEPUndecryptableCount
        aReceiveDTIMs;
REGISTERED AS { iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) attributeGroup(8)
StationConfiggrp(1) };
```

11.4.3.1.2 agPrivacygrp

```
Privacygrp ATTRIBUTE GROUP
    GROUP ELEMENTS
        aPrivacyOptionImplemented,
        aPrivacyInvoked,
        aWEPDefaultKeys,
        aWEPDefaultKeyID,
        aExcludeUnencrypted,
        aWEPICVErrorCount,
        aWEPExcludedCount;
REGISTERED AS { iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) attributeGroup(8)
Privacygrp(2) };
```

11.4.3.2 MAC attribute group templates

11.4.3.2.1 agOperationgrp

```
Operationgrp ATTRIBUTE GROUP
    GROUP ELEMENTS
        aMACAddress,
        aGroupAddresses,
        aShortRetryLimit,
        aLongRetryLimit,
        aFragmentationThreshold,
        aMaxTransmitMSDULifetime,
        aMaxReceiveLifetime,
        aManufacturerID,
        aProductID;
REGISTERED AS { iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(0) attributeGroup(8)
Operationgrp(1) };
```

11.4.3.2.2 agCountersgrp

```
Countersgrp ATTRIBUTE GROUP
    GROUP ELEMENTS
```

aTransmittedFragmentCount,
aMulticastTransmittedFrameCount,
aFailedCount,
aRetryCount,
aMultipleRetryCount,
aRTSSuccessCount,
aRTSFailureCount,
aACKFailureCount,
aFrameDuplicateCount,
aReceivedFragmentCount,
aMulticastReceivedFrameCount,
aFCSErrorCount;
REGISTERED AS { iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(0) attributeGroup(8) Countersgrp(2) };

11.4.4 Attribute templates

11.4.4.1 SMT attribute templates

11.4.4.1.1 aStationID

StationID ATTRIBUTE
DERIVED FROM
IEEE802CommonDefinitions.MACAddress;
BEHAVIOUR DEFINES AS
"This attribute is a value that has the form of a MAC address. It is used for management purposes only to allow an external management entity to uniquely identify a STA."
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) attribute(7) StationID(1) };

11.4.4.1.2 aAuthenticationAlgorithms

AuthenticationAlgorithms ATTRIBUTE
WITH APPROPRIATE SYNTAX
set-of integer;
BEHAVIOUR DEFINED AS
"This attribute shall be a set of all the authentication algorithms supported by the STAs. The following are the default values and the associated algorithm.
Value = 1: Open System
Value = 2: Shared Key";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) attribute(7) AuthenticationAlgorithms(2) };

11.4.4.1.3 aPrivacyOptionImplemented

PrivacyOptionImplemented ATTRIBUTE
WITH APPROPRIATE SYNTAX
boolean;
BEHAVIOUR DEFINED AS
"When this attribute is true, it shall indicate that the IEEE 802.11 WEP option is implemented. The default value of this attribute shall be false.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) attribute(7) PrivacyOptionImplemented(3) };

11.4.4.1.4 aAuthenticationType

AuthenticationType ATTRIBUTE
WITH APPROPRIATE SYNTAX
set of integer;
BEHAVIOUR DEFINED AS
"This attribute shall indicate the authentication algorithms acceptable to the STA during the authentication sequence. The value of this attribute shall be selected from the set in the aAuthenticationAlgorithms attribute. The default value of this attribute shall be {1}.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) attribute(7) AuthenticationType(4) };

11.4.4.1.5 aPrivacyInvoked

PrivacyInvoke ATTRIBUTE
WITH APPROPRIATE SYNTAX
boolean;
BEHAVIOUR DEFINED AS
"When this attribute is true, it shall indicate that the IEEE 802.11 WEP mechanism is used for transmitting frames of type Data. The default value of this attribute shall be false.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) attribute(7) PrivacyInvoked(5) };

11.4.4.1.6 aWEPDefaultKeys

DefaultWEPKey ATTRIBUTE
WITH APPROPRIATE SYNTAX
set of DefaultWEPKey.type;
BEHAVIOUR DEFINED AS
"This attribute shall contain the four default WEP secret key values corresponding to the four possible aWEPDefaultKeyID values. The default value of each of the keys in this attribute shall be null.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) attribute(7) WEPDefaultKeys(6) };

11.4.4.1.7 aWEPDefaultKeyID

WEPDefault ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"This attribute shall indicate the use of the first, second, third, or fourth element of the aDefaultWEPKey array when set to values of zero, one, two, or three. The default value of this attribute shall be zero.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) attribute(7) WEPDefaultKeyID(7) };

11.4.4.1.8 aWEPKeyMappings

WEPKeyMapping ATTRIBUTE
WITH APPROPRIATE SYNTAX
set of ordered triples of type { MAC address, boolean, defaultWEPKey.type };
BEHAVIOUR DEFINED AS
"See 8.3.2 for detailed behaviour.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) attribute(7) WEPKeyMappings(8) };

11.4.4.1.9 aWEPKeyMappingLength

WEPKeyMapping ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The maximum number of tuples that aWEPKeyMappings can hold.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) attribute(7)
WEPKeyMappingLength(9) };

11.4.4.1.10 aExcludeUnencrypted

ExcludeUnencrypted ATTRIBUTE
WITH APPROPRIATE SYNTAX
boolean;
BEHAVIOUR DEFINED AS
"When this attribute is true, the STA shall not indicate at the MAC service interface received MSDUs that have the WEP subfield of the Frame Control field equal to zero. When this attribute is false, the STA may accept MSDUs that have the WEP subfield of the Frame Control field equal to zero. The default value of this attribute shall be false.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) attribute(7) ExcludeUnencrypted
(10) };

11.4.4.1.11 aMediumOccupancyLimit

MediumOccupancyLimit ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"This attribute shall indicate the maximum amount of time, in TU, that a PC may control the usage of the wireless medium without relinquishing control for long enough to allow at least one instance of DCF access to the medium. The default value of this attribute shall be 100. The maximum value of this attribute shall be 1000.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(0) attribute(7)
MediumOccupancyLimit(11) };

11.4.4.1.12 aCFPollable

CFPollable ATTRIBUTE
WITH APPROPRIATE SYNTAX
boolean;
BEHAVIOUR DEFINED AS
"When this attribute is true, it shall indicate that the STA is able to respond to a CF-Poll with a data frame within a SIFS time. This attribute shall be false if the STA is not able to respond to a CF-Poll with a data frame within a SIFS time.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(0) attribute(7) CFPollable(12) };

11.4.4.1.13 aWEPICVErrorCount

ICVErrorCount ATTRIBUTE
DERIVED FROM
"ISO/IEC 10165-2":counter;
BEHAVIOUR DEFINED AS

"This counter shall increment when a frame is received with the WEP subfield of the Frame Control field set to one and the value of the ICV as received in the frame does not match the ICV value that is calculated for the contents of that frame.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) attribute(7) WEPICVErrorCount(13) };

11.4.4.1.14 aWEPExcludedCount

ICVErrorCount ATTRIBUTE
DERIVED FROM
"ISO/IEC 10165-2":counter;
BEHAVIOUR DEFINED AS
"This counter shall increment when a frame is received with the WEP subfield of the Frame Control field set to zero and the value of aExcludedUnencrypted causes that frame to be discarded.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) attribute(7) WEPExcludedCount(14) };

11.4.4.1.15 aCFPPeriod

CFPPeriod ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The attribute shall describe the number of DTIM intervals between the start of CFPs. It is modified by MLME-START.request primitive.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) attribute(7) CFPPeriod(15) };

11.4.4.1.16 aCFPMaxDuration

CFPMaxDuration ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The attribute shall describe the maximum duration of the CFP in TU that may be generated by the PCF. It is modified by MLME-START.request primitive.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) attribute(7) CFPMaxDuration(16) };

11.4.4.1.17 aAuthenticationResponseTimeout

AuthenticationResponseTimeout ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The attribute shall describe the number of TU that a responding STA should wait for the next frame in the authentication sequence.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) attribute(7) AuthenticationResponseTimeout(17) };

11.4.4.2 MAC attribute templates

11.4.4.2.1 aMACAddress

MACAddress ATTRIBUTE
DERVIVED FROM

IEEE802CommonDefinitions.MACAddress;
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) MACAddress(1) };

11.4.4.2.2 aGroupAddresses

GroupAddresses ATTRIBUTE
WITH APPROPRIATE SYNTAX
set of IEEE802CommonDefinitions.MACAddress;
BEHAVIOUR DEFINED AS
"A set of MACAddresses identifying the multicast addresses, excluding the broadcast address, for which this STA shall receive frames. The default value of this attribute shall be null."
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) GroupAddresses(2) };

11.4.4.2.3 aTransmittedFragmentCount

TransmittedFragmentCount ATTRIBUTE
DERIVED FROM
"ISO/IEC 10165-2":pdusSentCounter;
BEHAVIOUR DEFINED AS
"This counter shall be incremented for each successfully delivered fragment of type Data or Management."
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) TransmittedFragmentCount(3) };

11.4.4.2.4 aMulticastTransmittedFrameCount

MulticastTransmittedFrameCount ATTRIBUTE
DERIVED FROM
"ISO/IEC 10165-2":pdusSentCounter;
BEHAVIOUR DEFINED AS
"This counter shall increment only when the multicast/broadcast bit is set in the destination MAC address of a transmitted frame.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) MulticastTransmittedFrameCount(4) };

11.4.4.2.5 aFailedCount

FailedCount ATTRIBUTE
DERIVED FROM
"ISO/IEC 10165-2":counter;
BEHAVIOUR DEFINED AS
"This counter shall increment when a frame is not transmitted due to the number of transmit attempts exceeding either the aShortRetryLimit or aLongRetryLimit.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) FailedCount(5) };

11.4.4.2.6 aRetryCount

RetryCount ATTRIBUTE
DERIVED FROM
"ISO/IEC 10165-2":counter;
BEHAVIOUR DEFINED AS
"This counter shall increment when a frame is successfully transmitted after one or more retransmissions.";

REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) RetryCount(6) };

11.4.4.2.7 aMultipleRetryCount

MultipleRetryCount ATTRIBUTE
DERIVED FROM
"ISO/IEC 10165-2":counter;
BEHAVIOUR DEFINED AS
"This counter shall increment when a frame is successfully transmitted after more than one retransmission.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) MultipleRetryCount(7) };

11.4.4.2.8 aRTSSuccessCount

RTSSuccessCount ATTRIBUTE
DERIVED FROM;
"ISO/IEC 10165-2":counter
BEHAVIOUR DEFINED AS
"This counter shall increment when a CTS is received in response to an RTS.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) RTSSuccessCount(8) };

11.4.4.2.9 aRTSFailureCount

RTSFailureCount ATTRIBUTE
DERIVED FROM
"ISO/IEC 10165-2":counter;
BEHAVIOUR DEFINED AS
"This counter shall increment when a CTS is not received in response to an RTS.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) RTSFailureCount(9) };

11.4.4.2.10 aACKFailureCount

ACKFailureCount ATTRIBUTE
DERIVED FROM
"ISO/IEC 10165-2":counter;
BEHAVIOUR DEFINED AS
"This counter shall increment when an ACK is not received when expected.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) ACKFailureCount(10) };

11.4.4.2.11 aFrameDuplicateCount

FrameDuplicateCount ATTRIBUTE
DERIVED FROM
"ISO/IEC 10165-2":counter;
BEHAVIOUR DEFINED AS
"This counter shall increment when a frame is received that the Sequence Control field indicates is a duplicate.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) attribute(7) FrameDuplicateCount(11) };

11.4.4.2.12 aReceivedFragmentCount

ReceivedFragmentCount ATTRIBUTE
DERIVED FROM
    "ISO/IEC 10165-2":pdusReceivedCounter;
BEHAVIOUR DEFINED AS
    "This counter shall be incremented for each successfully received fragment of type Data or Management."
REGISTERED AS
    { iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) ReceivedFragmentCount(12) };

11.4.4.2.13 aMulticastReceivedFrameCount

MulticastReceivedFrameCount ATTRIBUTE
DERIVED FROM
    "ISO/IEC 10165-2":pdusReceivedCounter;
BEHAVIOUR DEFINED AS
    "This counter shall increment when a frame is received with the multicast/broadcast bit set in the destination MAC address.";
REGISTERED AS
    { iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) MulticastReceivedFrameCount(13) };

11.4.4.2.14 aFCSErrorCount

FCSErrorCount ATTRIBUTE
DERIVED FROM
    "ISO/IEC 10165-2": CorruptedPDUsReceivedCounter;
BEHAVIOUR DEFINED AS
    "This counter shall increment when an FCS error is detected in a received frame.";
REGISTERED AS
    { iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) FCSErrorCount(14) };

11.4.4.2.15 aRTSThreshold

RTSThreshold ATTRIBUTE
WITH APPROPRIATE SYNTAX
    integer;
BEHAVIOUR DEFINED AS
    "This attribute shall indicate the number of bytes in an MPDU, below which an RTS/CTS handshake shall not be performed. An RTS/CTS handshake shall be performed at the beginning of any frame exchange sequence where the MPDU is of type Data or Management, the MPDU has an individual address in the Address1 field, and the length of the MPDU is equal to or larger than this threshold. (For additional details, refer to Table 21 in 9.7.) Setting this attribute to be larger than the maximum MSDU size shall have the effect of turning off the RTS/CTS handshake for frames of Data or Management type transmitted by this STA. Setting this attribute to zero shall have the effect of turning on the RTS/CTS handshake for all frames of Data or Management type transmitted by this STA. The default value of this attribute shall be 3000.";
REGISTERED AS
    { iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) RTSThreshold(15) };

11.4.4.2.16 aShortRetryLimit

ShortRetryLimit ATTRIBUTE
WITH APPROPRIATE SYNTAX
    integer;

BEHAVIOUR DEFINED AS
"This attribute shall indicate the maximum number of transmission attempts of a frame, the length of which is less than or equal to aRTSThreshold, that shall be made before a failure condition is indicated. The default value of this attribute shall be 7.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) ShortRetryLimit(16) };

11.4.4.2.17 aLongRetryLimit

LongRetryLimit ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"This attribute shall indicate the maximum number of transmission attempts of a frame, the length of which is greater than aRTSThreshold, that shall be made before a failure condition is indicated. The default value of this attribute shall be 4.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) LongRetryLimit(17) };

11.4.4.2.18 aFragmentationThreshold

FragmentationThreshold ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR
"This attribute shall specify the current maximum size, in octets, of the MPDU that may be delivered to the PHY. An MSDU shall be broken into fragments if its size exceeds the value of this attribute after adding MAC headers and trailers. The default value for this attribute shall be equal to aMPDUMaxLength of the attached PHY and shall never exceed aMPDUMaxLength of the attached PHY. The value of this attribute shall never be less than 256. The default value of this attribute is 2346.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) FragmentationThreshold(18) };

11.4.4.2.19 aMaxTransmitMSDULifetime

MaxTransmitMSDULifetime ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The MaxTransmitMSDULifetime shall be the elapsed time in TU, after the initial transmission of an MSDU, after which further attempts to transmit the MSDU shall be terminated. The default value of this attribute shall be 512.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) MaxTransmitMSDULifetime(19) };

11.4.4.2.20 aMaxReceiveLifetime

MaxReceiveLifetime ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The MaxReceiveLifetime shall be the elapsed time in TU, after the initial reception of a fragmented MMPDU or MSDU, after which further attempts to reassemble the MMPDU or MSDU shall be terminated. The default value shall be 512.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) MaxReceiveLifetime(20) };

11.4.4.2.21 aWEPUndecryptableCount

ICVErrorCount ATTRIBUTE
DERIVED FROM
"ISO/IEC 10165-2":counter;
BEHAVIOUR DEFINED AS
"This counter shall increment when a frame is received with the WEP subfield of the Frame Control field set to one and the WEPOn value for the key mapped to the TA's MAC address indicates that the frame should not have been encrypted or that frame is discarded due to the receiving STA not implementing the privacy option.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) WEPUndecryptableCount(21) };

11.4.4.2.22 aManufacturerID

ManufacturerID ATTRIBUTE
WITH APPROPRIATE SYNTAX
octet string;
BEHAVIOUR DEFINED AS
"The ManufacturerID shall include, at a minimum, the name of the manufacturer. It may include additional information at the manufacturer's discretion. The default value of this attribute shall be null.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) ManufacturerID(22) };

11.4.4.2.23 aProductID

ProductID ATTRIBUTE
WITH APPROPRIATE SYNTAX
octet string;
BEHAVIOUR DEFINED AS
"The ProductID shall include, at a minimum, an identifier that is unique to the manufacturer. It may include additional information at the manufacturer's discretion. The default value of this attribute shall be null.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) MAC(2) attribute(7) ProductID(23) };

11.4.4.3 Resource type attribute templates

11.4.4.3.1 aResourceTypeIDName

ResourceTypeIDName ATTRIBUTE
DERIVED FROM
IEEE802CommonDefinitions.ResourceTypeIDName;
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) ResourceTypeID(3) attribute(7) ResourceTypeIDName(1) };

11.4.4.3.2 aResourceInfo

ResourceInfo ATTRIBUTE
DERIVED FROM
IEEE802CommonDefinitions.ResourceInfo;
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) ResourceTypeID(3) attribute(7) ResourceInfo(2) };

11.4.5 Notification templates

11.4.5.1 SMT notification templates

11.4.5.1.1 nDisassociate

Disassociate NOTIFICATION
WITH APPROPRIATE SYNTAX
    IEEE802CommonDefinitions.MACAddress;
BEHAVIOUR DEFINED AS
    "The disassociate notification shall be sent when the STA receives a Disassociate frame. The value of the notification shall be the BSSID of the BSS from which the Disassociate frame was received.";
REGISTERED AS
    { iso(1) member-body(2) us(840) ieee802dot11(10036) SMT(1) notification(10) Disassociate(1) };

12. Physical layer (PHY) service specification

12.1 Scope

The PHY services provided to the IEEE 802.11 wireless LAN MAC are described in this clause. Different PHYs are defined as part of the IEEE 802.11 standard. Each PHY can consist of two protocol functions as follows:

a) A physical layer convergence function, which adapts the capabilities of the physical medium dependent (PMD) system to the PHY service. This function is supported by the physical layer convergence procedure (PLCP), which defines a method of mapping the IEEE 802.11 MAC sublayer protocol data units (MPDU) into a framing format suitable for sending and receiving user data and management information between two or more STAs using the associated PMD system.
b) A PMD system, whose function defines the characteristics of, and method of transmitting and receiving data through, a wireless medium (WM) between two or more STAs.

Each PMD sublayer may require the definition of a unique PLCP. If the PMD sublayer already provides the defined PHY services, the physical layer convergence function might be null.

12.2 PHY functions

The protocol reference model for the IEEE 802.11 architecture is shown in Figure 11. Most PHY definitions contain three functional entities: the PMD function, the physical layer convergence function, and the layer management function.

The PHY service is provided to the MAC entity at the STA through a service access point (SAP), called the PHY-SAP, as shown in Figure 11. A set of primitives might also be defined to describe the interface between the physical layer convergence protocol sublayer and the PMD sublayer, called the PMD-SAP.

12.3 Detailed PHY service specifications

12.3.1 Scope and field of application

The services provided by the PHY to the IEEE 802.11 MAC are specified in this subclause. These services are describe in an abstract way and do not imply any particular implementation or exposed interface.

12.3.2 Overview of the service

The PHY function as shown in is separated into two sublayers: the PLCP sublayer and the PMD sublayer. The function of the PLCP sublayer is to provide a mechanism for transferring MAC protocol data units (MPDU) between two or more STAs over the PMD sublayer.

12.3.3 Overview of interactions

The primitives associated with communication between the IEEE 802.11 MAC sublayer and the IEEE 802.11 PHY fall into two basic categories:

a) Service primitives that support MAC peer-to-peer interactions
b) Service primitives that have local significance and support sublayer-to-sublayer interactions.

12.3.4 Basic service and options

All of the service primitives described here are considered mandatory unless otherwise specified.

12.3.4.1 PHY-SAP peer-to-peer service primitives

Table 24 indicates the primitives for peer-to-peer interactions.

Table 24—PHY-SAP peer-to-peer service primitives

Primitive	Request	Indicate	Confirm	Response
PHY-DATA	X	X	X	

12.3.4.2 PHY-SAP sublayer-to-sublayer service primitives

Table 25 indicates the primitives for sublayer-to-sublayer interactions.

Table 25—PHY-SAP sublayer-to-sublayer service primitives

Primitive	Request	Indicate	Confirm	Response
PHY-TXSTART	X		X	
PHY-TXEND	X		X	
PHY-CCARESET	X		X	
PHY-CCA		X		
PHY-RXSTART		X		
PHY-RXEND		X		

12.3.4.3 PHY-SAP service primitives parameters

Table 26 shows the parameters used by one or more of the PMD-SAP service primitives.

Table 26—PHY-SAP service primitive parameters

Parameter	Associated primitive	Value
DATA	PHY-DATA.request PHY-DATA.indication	Octet value X'00'–X'FF'
TXVECTOR	PHY-TXSTART.request	A set of parameters
STATUS	PHY-CCA.indication	BUSY, IDLE
RXVECTOR	PHY-RXSTART.indication	A set of parameters
RXERROR	PHY-RXEND.indication	NoError, FormatViolation, Carrier-Lost, UnsupportedRate

12.3.4.4 Vector descriptions

Several service primitives include a parameter vector. This vector is a list of parameters that may vary depending on the PHY type. Table 27 lists the parameter values required by the MAC or PHY in each of the parameter vectors. Parameters in the vectors that are management rather than MAC may be specific to the PHY and are listed in the clause covering that PHY.

Table 27—Vector descriptions

Parameter	Associate vector	Value
DATARATE	TXVECTOR, RXVECTOR	PHY dependent. The name of the field used to specify the Tx data rate and report the Rx data rate may vary for different PHYs.
LENGTH	TXVECTOR, RXVECTOR	PHY dependent

12.3.5 PHY-SAP detailed service specification

The following subclause describes the services provided by each PHY sublayer primitive.

12.3.5.1 PHY-DATA.request

12.3.5.1.1 Function

This primitive defines the transfer of an octet of data from the MAC sublayer to the local PHY entity.

12.3.5.1.2 Semantics of the service primitive

The primitive provides the following parameters:

PHY-DATA.request (DATA)

The DATA parameter is an octet of value X'00' through X'FF'.

12.3.5.1.3 When generated

This primitive is generated by the MAC sublayer to transfer an octet of data to the PHY entity. This primitive can only be issued following a transmit initialization response (PHY-TXSTART.confirm) from the PHY layer.

12.3.5.1.4 Effect of receipt

The receipt of this primitive by the PHY entity causes the PLCP transmit state machine to transmit an octet of data. When the PHY entity receives the octet, it will issue a PHY-DATA.confirm to the MAC sublayer.

12.3.5.2 PHY-DATA.indication

12.3.5.2.1 Function

This primitive indicates the transfer of data from the PHY sublayer to the local MAC entity.

12.3.5.2.2 Semantics of the service primitive

The primitive provides the following parameters:

PHY-DATA.indication (DATA)

The DATA parameter is an octet of value X'00' through X'FF'.

12.3.5.2.3 When generated

The PHY-DATA.indication is generated by a receiving PHY entity to transfer the received octet of data to the local MAC entity. The time between receipt of the last bit of the provided octet from the wireless medium and the receipt of this primitive by the MAC entity will be the sum of aRXRFDelay + aRxPLCPDelay.

12.3.5.2.4 Effect of receipt

The effect of receipt of this primitive by the MAC is unspecified.

12.3.5.3 PHY-DATA.confirm

12.3.5.3.1 Function

This primitive issued by the PHY sublayer to the local MAC entity to confirm the transfer of data from the MAC entity to the PHY sublayer.

12.3.5.3.2 Semantics of the service primitive

The primitive provides the following parameters:

PHY-DATA.confirm

This primitive has no parameters.

12.3.5.3.3 When generated

This primitive will be issued by the PHY sublayer to the MAC entity whenever the PLCP has completed the transfer of data from the MAC entity to the PHY sublayer. The PHY sublayer will issue this primitive in response to every PHY-DATA.request primitive issued by the MAC sublayer.

12.3.5.3.4 Effect of receipt

The receipt of this primitive by the MAC will cause the MAC to start the next MAC entity request.

12.3.5.4 PHY-TXSTART.request

12.3.5.4.1 Function

This primitive is a request by the MAC sublayer to the local PHY entity to start the transmission of an MPDU.

12.3.5.4.2 Semantics of the service primitive

The primitive provides the following parameters:

PHY-TXSTART.request (TXVECTOR)

The TXVECTOR represents a list of parameters that the MAC sublayer provides the local PHY entity in order to transmit an MPDU. This vector contains both PLCP and PHY management parameters. The required PHY parameters are listed in 12.3.4.4.

12.3.5.4.3 When generated

This primitive will be issued by the MAC sublayer to the PHY entity whenever the MAC sublayer needs to begin the transmission of an MPDU.

12.3.5.4.4 Effect of receipt

The effect of receipt of this primitive by the PHY entity will be to start the local transmit state machine.

12.3.5.5 PHY-TXSTART.confirm

12.3.5.5.1 Function

This primitive issued by the PHY sublayer to the local MAC entity to confirm the start of a transmission. The PHY sublayer will issue this primitive in response to every PHY-TXSTART.request primitive issued by the MAC sublayer.

12.3.5.5.2 Semantics of the service primitive

The primitive provides the following parameters:

PHY-TXSTART.confirm

There are no parameters associated with this primitive.

12.3.5.5.3 When generated

This primitive will be issued by the PHY sublayer to the MAC entity whenever the PHY has received a PHY-TXSTART.request from the MAC entity and is ready to begin receiving data octets.

12.3.5.5.4 Effect of receipt

The receipt of this primitive by the MAC entity will cause the MAC to start the transfer of data octets.

12.3.5.6 PHY-TXEND.request

12.3.5.6.1 Function

This primitive is a request by the MAC sublayer to the local PHY entity that the current transmission of the MPDU is completed.

12.3.5.6.2 Semantics of the service primitive

The primitive provides the following parameters:

PHY-TXEND.request

There are no parameters associated with this primitive.

12.3.5.6.3 When generated

This primitive will be generated whenever the MAC sublayer has received the last PHY-DATA.confirm from the local PHY entity for the MPDU currently being transferred.

12.3.5.6.4 Effect of receipt

The effect of receipt of this primitive by the local PHY entity will be to stop the transmit state machine.

12.3.5.7 PHY-TXEND.confirm

12.3.5.7.1 Function

This primitive issued by the PHY sublayer to the local MAC entity to confirm the completion of a transmission. The PHY sublayer issues this primitive in response to every PHY-TXEND.request primitive issued by the MAC sublayer.

12.3.5.7.2 Semantics of the service primitive

The primitive provides the following parameters:

PHY-TXEND.confirm

There are no parameters associated with this primitive.

12.3.5.7.3 When generated

This primitive will be issued by the PHY sublayer to the MAC entity whenever the PHY has received a PHY-TXEND.request immediately after transmitting the end of the last bit of the last data octet indicating the last data octet has been transferred.

12.3.5.7.4 Effect of receipt

The receipt of this primitive by the MAC entity provides the time reference for the contention backoff protocol.

12.3.5.8 PHY-CCARESET.request

12.3.5.8.1 Function

This primitive is a request by the MAC sublayer to the local PHY entity to reset the clear channel assessment (CCA) state machine.

12.3.5.8.2 Semantics of the service primitive

The primitive provides the following parameters:

PHY-CCARESET.request

There are no parameters associated with this primitive.

12.3.5.8.3 When generated

This primitive is generated by the MAC sublayer for the local PHY entity at the end of a NAV timer. This request can be used by some PHY implementations that may synchronize antenna diversity with slot timings.

12.3.5.8.4 Effect of receipt

The effect of receipt of this primitive by the PHY entity is to reset the PLCP CS/CCA assessment timers to the state appropriate for the end of a received frame.

12.3.5.9 PHY-CCARESET.confirm

12.3.5.9.1 Function

This primitive issued by the PHY sublayer to the local MAC entity to confirm that the PHY has reset the CCA state machine.

12.3.5.9.2 Semantics of the service primitive

The primitive provides the following parameters:

PHY-CCARESET.request

There are no parameters associated with this primitive.

12.3.5.9.3 When generated

This primitive is issued by the PHY sublayer to the MAC entity whenever the PHY has received a PHY-CCARESET.request

12.3.5.9.4 Effect of receipt

The effect of receipt of this primitive by the MAC is unspecified.

12.3.5.10 PHY-CCA.indication

12.3.5.10.1 Function

This primitive is an indication by the PHY sublayer to the local MAC entity of the current state of the medium.

12.3.5.10.2 Semantics of the service primitive

The primitive provides the following parameters:

PHY-CCA.indication (STATE)

The STATE parameter can be one of two values: BUSY or IDLE. The parameter value is BUSY if the channel assessment by the PHY sublayer determines that the channel is not available. Otherwise, the value of the parameter is IDLE.

12.3.5.10.3 When generated

This primitive is generated every time the status of the channel changes from channel idle to channel busy or from channel busy to channel idle. This includes the period of time when the PHY sublayer is receiving data. The PHY sublayer maintains the channel busy indication until the period indicated by the length field in a valid PLCP Header has expired.

12.3.5.10.4 Effect of receipt

The effect of receipt of this primitive by the MAC is unspecified.

12.3.5.11 PHY-RXSTART.indication

12.3.5.11.1 Function

This primitive is an indication by the PHY sublayer to the local MAC entity that the PLCP has received a valid start frame delimiter (SFD) and PLCP Header.

12.3.5.11.2 Semantics of the service primitive

The primitive provides the following parameters:

PHY-RXSTART.indication (RXVECTOR)

The RXVECTOR represents a list of parameters that the PHY sublayer provides the local MAC entity upon receipt of a valid PLCP Header. This vector may contain both MAC and MAC management parameters. The required parameters are listed in 12.3.4.4.

12.3.5.11.3 When generated

This primitive is generated by the local PHY entity to the MAC sublayer whenever the PHY has successfully validated the PLCP Header error check CRC at the start of a new PLCP PDU.

12.3.5.11.4 Effect of receipt

The effect of receipt of this primitive by the MAC is unspecified.

12.3.5.12 PHY-RXEND.indication

12.3.5.12.1 Function

This primitive is an indication by the PHY sublayer to the local MAC entity that the MPDU currently being received is completed.

12.3.5.12.2 Semantics of the service primitive

The primitive provides the following parameters:

PHY-RXEND.indication (RXERROR)

The RXERROR parameter can convey one or more of the following values: NoError, FormatViolation, CarrierLost, or UnsupportedRate. A number of error conditions may occur after the PLCP's receive state machine has detected what appeared to be a valid preamble and SFD. The following describes the parameter returned for each of those error conditions.

- *NoError.* This value is used to indicate that no error occurred during the receive process in the PLCP.
- *FormatViolation.* This value is used to indicate that the format of the received PLCPPDU was in error.
- *CarrierLost.* This value is used to indicate that during the reception of the incoming MPDU, carrier was lost and no further processing of the MPDU could be accomplished.

— *UnsupportedRate.* This value is used to indicate that during the reception of the incoming PLCP-PDU, a nonsupported date rate was detected.

12.3.5.12.3 When generated

This primitive is generated by the PHY sublayer for the local MAC entity to indicate that the receive state machine has completed a reception with or without errors.

12.3.5.12.4 Effect of receipt

The effect of receipt of this primitive by the MAC is unspecified.

13. PHY management

The attribute definitions and templates of PHY management are described in this clause. Both the PHY-dependent and the PHY-independent portions of the physical layer management information base (MIB) are included. Not all attributes in the following subclause are supported by every PHY. Each PHY contains a Managed Object list, which defines the PHY-specific values required for each PHY implementation. The ASN.1 encoding of the MIB is presented in Annex D. In any discrepancy between the definition in this clause and that in Annex D, the definition in the annex shall take precedence.

13.1 PHY MIB

13.1.1 PHY attributes

13.1.1.1 agPhyOperationGroup

aPHYType,
aRegDomainsSupported,
aCurrentRegDomain,
aSlotTime,
aCCATime,
aRxTxTurnaroundTime,
aTxPLCPDelay,
aRxTxSwitchTime,
aTxRampOnTime,
aTxRFDelay,
aSIFSTime,
aRxRFDelay,
aRxPLCPDelay,
aMACProcessingDelay,
aTxRampOffTime,
aPreambleLength,
aPLCPHeaderLength,
aMPDUDurationFactor,
aAirPropagationTime,
aTempType,
aCWmin,
aCWmax;

13.1.1.2 agPhyRateGroup

aSupportedDataRatesTx,
aSupportedDataRatesRx;
aMPDUMaxLength;

13.1.1.3 agPhyAntennaGroup

aCurrentTxAntenna,
aDiversitySupport;

13.1.1.4 agPhyTxPowerGroup

aNumberSupportedPowerLevels,
aTxPowerLevel1,

aTxPowerLevel2,
aTxPowerLevel3,
aTxPowerLevel4,
aTxPowerLevel5,
aTxPowerLevel6,
aTxPowerLevel7,
aTxPowerLevel8,
aCurrentTxPowerLevel;

13.1.1.5 agPhyFHSSGroup

aHopTime,
aCurrentChannelNumber,
aMaxDwellTime,
aCurrentSet,
aCurrentPattern,
aCurrentIndex;

13.1.1.6 agPhyDSSSGroup

aCurrentChannel,
aCCAModeSupported,
aCurrentCCAMode,
aEDThreshold;

13.1.1.7 agPhyIRGroup

aCCAWatchdogTimerMax,

aCCAWatchdogCountMax,

aCCAWatchdogTimerMin,

aCCAWatchdogCountMin;

13.1.1.8 agPhyStatusGroup

aSynthesizerLocked;

13.1.1.9 agPhyPowerSavingGroup

aCurrentPowerState,
aDozeTurnonTime;

13.1.1.10 agAntennasList

aSupportedTxAntennas,
aSupportedRxAntennas,
aDiversitySelectionRx;

13.1.2 PHY object class

PHY MANAGED OBJECT CLASS
DERIVED FROM "ISO/IEC 10165-2":top;
CHARACTERIZED BY
pPHYbase PACKAGE
BEHAVIOUR
bPHYbase BEHAVIOUR
DEFINED AS "The PHY object class provides the necessary support for all the required PHY operational information, which may vary from PHY to PHY and from STA to STA, to be communicated to upper layers."
ATTRIBUTES

aPHYType	GET,
aRegDomainsSupported	GET,
aCurrentRegDomain	GET-REPLACE,
aSlotTime	GET,
aCCATime	GET,
aRxTxTurnaroundTime	GET,
aTxPLCPDelay	GET,
aRxTxSwitchTime	GET,
aTxRampOnTime	GET,
aTxRFDelay	GET,
aSIFSTime	GET,
aRxRFDelay	GET,
aRxPLCPDelay	GET,
aMACProcessingDelay	GET,
aTxRampOffTime	GET,
aPreambleLength	GET,
aPLCPHeaderLength	GET,
aMPDUDurationFactor	GET,
aAirPropagationTime	GET,
aTempType	GET,
aCWmin	GET,
aCWmax	GET,
aSupportedDataRatesTx	GET,
aSupportedDataRatesRx	GET,
aMPDUMaxLength	GET,
aSupportedTxAntennas	GET,
aCurrentTxAntenna	GET-REPLACE,
aSupportedRxAntennas	GET,
aDiversitySupport	GET,
aDiversitySelectionRx	GET-REPLACE,
aNumberSupportedPowerLevels	GET,
aTxPowerLevel1	GET,
aTxPowerLevel2	GET,
aTxPowerLevel3	GET,
aTxPowerLevel4	GET,
aTxPowerLevel5	GET,
aTxPowerLevel6	GET,
aTxPowerLevel7	GET,
aTxPowerLevel8	GET,
aCurrentTxPowerLevel	GET-REPLACE,
aHopTime	GET,
aCurrentChannelNumber	GET-REPLACE,
aMaxDwellTime	GET,
aCurrentSet	GET-REPLACE,
aCurrentPattern	GET-REPLACE,

```
        aCurrentIndex                              GET-REPLACE,
        aCurrentChannel                            GET-REPLACE,
        aCCAModeSupported                          GET,
        aCurrentCCAMode                            GET-REPLACE,
        aEDThreshold                               GET-REPLACE,
        aSynthesizerLocked                         GET,
        aCurrentPowerState                         GET-REPLACE,
        aDozeTurnonTime                            GET;

    ATTRIBUTE GROUPS
        agPhyOperationGroup,
        agPhyRateGroup,
        agPhyAntennaGroup,
        agPhyTxPowerGroup,
        agPhyFHSSGroup,
        agPhyDSSSGroup,
        agPhyIRGroup,
        agPhyStatusGroup,
        agPhyPowerSavingGroup,
        agAntennaListGroup;
    ACTIONS
        acPHYreset;
    NOTIFICATIONS
REGISTERED AS { iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) };
```

13.1.3 PHY attribute group templates

13.1.3.1 agPhyOperationGroup

```
PhyOperationGroup ATTRIBUTE GROUP
    GROUP ELEMENTS
        aPHYType,
        aRegDomainsSupported,
        aCurrentRegDomain,
        aSlotTime,
        aCCATime,
        aRxTxTurnaroundTime,
        aTxPLCPDelay,
        aRxTxSwitchTime,
        aTxRampOntime,
        aTxRFDelay,
        aSIFSTime,
        aRxRFDelay,
        aRxPLCPDelay,
        aMACProcessingDelay,
        aTxRampOffTime,
        aPreambleLength,
        aPLCPHeaderLength,
        aMPDUDurationFactor,
        aAirPropagationTime,
        aTempType,
        aCWmin,
        aCWmax;
REGISTERED AS { iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attributeGroup(8)
    PhyOperationGroup(0) };
```


13.1.3.2 agPhyRateGroup

```
PhyRateGroup ATTRIBUTE GROUP
    GROUP ELEMENTS
        aSupportedDataRatesTx,
        aSupportedDataRatesRx,
        aMPDUMaxLength,
REGISTERED AS { iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attributeGroup(8)
    PhyRateGroup(1) };
```

13.1.3.3 agPhyAntennaGroup

```
PhyAntennaGroup ATTRIBUTE GROUP
    GROUP ELEMENTS
        aCurrentTxAntenna,
        aDiversitySupport;
REGISTERED AS { iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attributeGroup(8)
    PhyAntennaGroup(2) };
```

13.1.3.4 agPhyTxPowerGroup

```
PhyTxPowerGroup ATTRIBUTE GROUP
    GROUP ELEMENTS
        aNumberSupportedPowerLevels,
        aTxPowerLevel1,
        aTxPowerLevel2,
        aTxPowerLevel3,
        aTxPowerLevel4,
        aTxPowerLevel5,
        aTxPowerLevel6,
        aTxPowerLevel7,
        aTxPowerLevel8,
        aCurrentTxPowerLevel;
REGISTERED AS { iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attributeGroup(8)
    PhyTxPowerGroup(3) };
```

13.1.3.5 agPhyFHSSGroup

```
PhyFHSSGroup ATTRIBUTE GROUP
    GROUP ELEMENTS
        aHopTime,
        aCurrentChannelNumber,
        aMaxDwellTime,
        aCurrentSet,
        aCurrentPattern,
        aCurrentIndex;
REGISTERED AS { iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attributeGroup(8)
    PhyFHSSGroup(4) };
```

13.1.3.6 agPhyDSSSGroup

```
PhyStatusGroup ATTRIBUTE GROUP
    GROUP ELEMENTS
        aCurrentChannel,
        aCCAModeSupported,
        aCurrentCCAMode,
```

```
        aEDThreshold;
REGISTERED AS { iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attributeGroup(8)
    PhyDSSSGroup(5) };
```

13.1.3.7 agPhyIRGroup

```
PhyStatusGroup ATTRIBUTE GROUP
    GROUP ELEMENTS
        aCCAWatchdogTimerMax,
        aCCAWatchdogCountMax,
        aCCAWatchdogTimerMin,
        aCCAWatchdogCountMin;
REGISTERED AS { iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attributeGroup(8)
    PhyIRGroup(6) };
```

13.1.3.8 agPhyStatusGroup

```
PhyStatusGroup ATTRIBUTE GROUP
    GROUP ELEMENTS
        aSynthesizerLocked;
REGISTERED AS { iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attributeGroup(8)
    PhyStatusGroup(7) };
```

13.1.3.9 agPhyPowerSavingGroup

```
PhyStatusGroup ATTRIBUTE GROUP
    GROUP ELEMENTS
        aCurrentPowerState,
        aDozeTurnonTime;
REGISTERED AS { iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attributeGroup(8)
    PhyPowerSavingGroup(8) };
```

13.1.3.10 agAntennaListGroup

```
agAntennasList ATTRIBUTE GROUP
    GROUP ELEMENTS
        aSupportedTxAntennas,
        aSupportedRxAntennas,
        aDiversitySelectionRx;
REGISTERED AS { iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attributeGroup(8)
    AntennaListGroup(9) };
```

13.1.4 PHY attribute templates

13.1.4.1 aPHYType

```
PHYType ATTRIBUTE
WITH APPROPRIATE SYNTAX
    integer;
BEHAVIOUR DEFINED AS
        "This is an 8-bit integer value that identifies the PHY type supported by the attached PLCP and
        PMD. Currently defined values and their corresponding PHY types are:
        FHSS 2.4 GHz = 01 , DSSS 2.4 GHz = 02, IR Baseband = 03";
REGISTERED AS
    { iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) PHYType(1) };
```

13.1.4.2 aRegDomainsSupported

RegDomainsSupported ATTRIBUTE
WITH APPROPRIATE SYNTAX
Null Terminated list of byte integers;
BEHAVIOUR DEFINED AS
"There are different operational requirements dependent on the regulatory domain. This attribute list describes the regulatory domains the PLCP and PMD support in this implementation. Currently defined values and their corresponding Regulatory Domains are:
FCC (USA) = X'10', IC (Canada) = X'20', ETSI (most of Europe) = X'30', Spain = X'31', France = X'32', MKK (Japan) = X'40', list terminator = X'00'";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) RegDomainsSupported(2) };

13.1.4.3 aCurrentRegDomain

CurrentRegDomain ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The current regulatory domain that this instance of the PMD is supporting. This octet corresponds to one of the RegDomains listed in aRegDomainsSupported";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) CurrentRegDomain(3) };

13.1.4.4 aSlotTime

SlotTime ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The time in microseconds that the MAC will use for defining the PIFS and DIFS periods. The SlotTime is defined as a function of the following the equation:
CCATime + RxTxTurnaroundTime + AirPropagationTime + aMACProcessingDelay.
AirPropagationTime is defined as 1 μs.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) SlotTime(4) };

13.1.4.5 aCCATime

CCATime ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The minimum time in microseconds that the CCA mechanism has available to assess the medium within every time slot to determine whether the medium is busy or idle";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) CCATime(5) };

13.1.4.6 aRxTxTurnaroundTime

RxTxTurnaroundTime ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The maximum time in microseconds that the PHY requires to change from receiving to transmitting the start of the first symbol. The following equation is used to derive the RxTxTurnaroundTime:

aTxPLCPDelay + aRxTxSwitchTime + aTxRampOnTime + aTxRFDelay.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) RxTxTurnaroundTime(6) };

13.1.4.7 aTxPLCPDelay

TxPLCPDelay ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The nominal time in microseconds that the PLCP uses to deliver a symbol from the MAC interface to the transmit data path of the PMD";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) TxPLCPDelay(7) };

13.1.4.8 aRxTxSwitchTime

RxTxSwitchTime ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The nominal time in microseconds that the PMD takes to switch from Receive to Transmit";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) RxTxSwitchTime(8) };

13.1.4.9 aTxRampOnTime

TxRampOnTime ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The maximum time in microseconds that the PMD takes to turn the Transmitter on";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) TxRampOnTime(9) };

13.1.4.10 aTxRFDelay

TxRFDelayWITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The nominal time in microseconds between the issuance of a PMD-DATA.request to the PMD and the start of the corresponding symbol at the air interface. The start of a symbol is defined to be 1/2 symbol period prior to the center of the symbol for FH, or 1/2 chip period prior to the center of the first chip of the symbol for DS, or 1/2 slot time prior to the center of the corresponding slot for IR.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) TxRFDelay(10) };

13.1.4.11 aSIFSTime

SIFSTime ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The nominal time in microseconds that the MAC and PHY will require to receive the last symbol of a frame at the air interface, process the frame, and respond with the first symbol on the air interface of the earliest possible response frame. The following equation is used to determine the SIFSTime:
aRxRFDelay + aRxPLCPDelay + aMACProcessingDelay +

aRxTxTurnaroundTime";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) SIFSTime(11) };

13.1.4.12 aRxRFDelay

RxRFDelay ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The nominal time in microseconds between the end of a symbol at the air interface to the issuance of a PMD-DATA.indicate to the PLCP. The end of a symbol is defined to be 1/2 symbol period after the center of the symbol for FH, or 1/2 chip period after the center of the last chip of the symbol for DS, or 1/2 slot time after the center of the corresponding slot for IR.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) RxRFDelay(12) };

13.1.4.13 aRxPLCPDelay

RxPLCPDelay ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The nominal time in microseconds that the PLCP uses to deliver a bit from the PMD receive path to the MAC";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) RxPLCPDelay(13) };

13.1.4.14 aMACProcessingDelay

MACProcessingDelay ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The nominal time in microseconds that the MAC uses to process a frame and prepare a response to the frame";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) MACProcessingDelay(14) };

13.1.4.15 aTxRampOffTime

TxRampOffTime ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The nominal time in microseconds that the PMD takes to turn the Transmit Power Amplifier off";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) TxRampOffTime(15) };

13.1.4.16 aPreambleLength

PreambleLength ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The current PHY's Preamble Length in microseconds. If the actual value of the length of the modulated pre-

amble is not an integral number of microseconds, the value shall be rounded up to the next higher value.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) PreambleLength (16) };

13.1.4.17 aPLCPHeaderLength

PLCPHeaderLength ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The current PHY's PLCP Header Length in microseconds. If the actual value of the length of the modulated header is not an integral number of microseconds, the value shall be rounded up to the next higher value.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) PLCPHdrLength (17) };

13.1.4.18 aMPDUDurationFactor

MPDUDurationFactor ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The overhead added by the PHY to the MPDU as it is transmitted through the wireless medium.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) MPDUDurationFactor(18) };

13.1.4.19 aAirPropagationTime

AirPropagationTime ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The anticipated time it takes a transmitted signal to go from the transmitting station to the receiving station.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) AirPropagationTime(19) };

13.1.4.20 aTempType

TempType ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"There are different operating temperature requirements dependent on the anticipated environmental conditions. This attribute describes the current PHY's operating temperature range capability. Currently defined values and their corresponding temperature ranges are:
Type 1 = X'01'—Commercial range of 0 to 40 °C,
Type 2 = X'02'—Industrial range of –20 to 55 °C,
Type 3 = X'03'—Industrial range of –30 to 70 °C.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) TempType (20) };

13.1.4.21 aCWmin

CWmin ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The minimum size of the contention window, in units of aSlotTime.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) CWmin (21) };

13.1.4.22 aCWmax

CWmax ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The maximum size of the contention window, in units of aSlotTime.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) CWmax(22) };

13.1.4.23 aSupportedDataRatesTX

SupportedDataRates ATTRIBUTE
WITH APPROPRIATE SYNTAX
Null Terminated list of byte integers;
BEHAVIOUR DEFINED AS
"The transmit bit rates supported by the PLCP and PMD, represented by a count from X'00'–X'7F', corresponding to data rates in increments of 500 kbit/s from 0 to 63.5 Mbit/s subject to limitations of each individual PHY.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) SupportedDataRatesTX (23) };

13.1.4.24 aSupportedDataRatesRX

SupportedDataRXRates ATTRIBUTE
WITH APPROPRIATE SYNTAX
Null Terminated list of byte integers;
BEHAVIOUR DEFINED AS
"The receive bit rates supported by the PLCP and PMD, represented by a count from X'00'–X'7F', corresponding to data rates in increments of 500 kbit/s from 0 to 63.5 Mbit/s.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) SupportedDataRatesRX(24) };

13.1.4.25 aMPDUMaxLength

MPDUMaxLength ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The maximum number of octets of an MPDU that can be conveyed by a PLCPPDU";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) MPDUMaxLength (25) };

13.1.4.26 aSupportedTxAntennas

SupportedTxAntennas ATTRIBUTE
WITH APPROPRIATE SYNTAX
Null terminated list of integers;
BEHAVIOUR DEFINED AS
"A list of one or more antennas that can be used as the transmit antenna. Each antenna is represented by an integer, starting with antenna 1, and through antenna N, where N ≤ 255.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) SupportedTxAntennas (26) };

13.1.4.27 aCurrentTxAntenna

CurrentTxAntenna ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The current antenna being used to transmit. This value is one of the values appearing in aSupportedTx-Antennas.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) MPDUMaxLength1M (27) };

13.1.4.28 aSupportedRxAntennas

SupportedRxAntennas ATTRIBUTE
WITH APPROPRIATE SYNTAX
Null terminated list of integers;
BEHAVIOUR DEFINED AS
"A list of one or more antennas that can be used as the receive antenna. Each antenna is represented by an integer, starting with antenna 1, and through antenna N, where N ≤ 255.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) SupportedRxAntennas (28) };

13.1.4.29 aDiversitySupport

DiversitySupport ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"This implementation's support for diversity, encoded as:
X'01'—diversity is available and is performed over the fixed list of antennas defined in aDiversitySelectionRx.
X'02'—diversity is not supported.
X'03'—diversity is supported and control of diversity is also available, in which case the attribute aDiversitySelectionRx can be dynamically modified by the LME.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) MPDUMaxLength1M (29) };

13.1.4.30 aDiversitySelectionRx

DiversitySelectionRx ATTRIBUTE
WITH APPROPRIATE SYNTAX
Null terminated list of integers;
BEHAVIOUR DEFINED AS
"A list of one or more antennas that can be used as receive antennas. Each antenna is represented by an integer, starting with antenna 1, and through antenna N, where N ≤ 255.";
REGISTERED AS

{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) DiversitySelectionRx (30) };

13.1.4.31 aNumberSupportedPowerLevels

NumberSupportedPowerLevels ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The number of power levels supported by the PMD. This attribute can have a value of 1 to 8.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7)
NumberSupportedPowerLevels (31) };

13.1.4.32 aTxPowerLevel1

TxPowerLevel1 ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The transmit output power for LEVEL1 in mW. This is also the default power level.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) TxPowerLevel1 (32) };

13.1.4.33 aTxPowerLevel2

TxPowerLevel2 ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The transmit output power for LEVEL2 in mW.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) TxPowerLevel2 (33) };

13.1.4.34 aTxPowerLevel3

TxPowerLevel3 ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The transmit output power for LEVEL3 in mW.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) TxPowerLevel3 (34) };

13.1.4.35 aTxPowerLevel4

TxPowerLevel4 ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The transmit output power for LEVEL4 in mW.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) TxPowerLevel4 (35) };

13.1.4.36 aTxPowerLevel5

TxPowerLevel5 ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The transmit output power for LEVEL5 in mW.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) TxPowerLevel5 (36) };

13.1.4.37 aTxPowerLevel6

TxPowerLevel6 ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The transmit output power for LEVEL6 in mW.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) TxPowerLevel6 (37) };

13.1.4.38 aTxPowerLevel7

TxPowerLevel7 ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The transmit output power for LEVEL7 in mW.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) TxPowerLevel7 (38) };

13.1.4.39 aTxPowerLevel8

TxPowerLevel8 ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The transmit output power for LEVEL8 in mW.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) TxPowerLevel8 (39) };

13.1.4.40 aCurrentTxPowerLevel

CurrentTxPowerLevel ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The TxPowerLevelN currently being used to transmit data. Some PHYs also use this value to determine the receiver sensitivity requirements for CCA.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) CurrentTxPowerLevel (40) };

13.1.4.41 aHopTime

HopTime ATTRIBUTE
WITH APPROPRIATE SYNTAX
  integer;
BEHAVIOUR DEFINED AS
  "The time in microseconds for the PMD to change from channel 2 to channel 80.";
REGISTERED AS
  { iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) HopTime (41) };

13.1.4.42 aCurrentChannelNumber

CurrentChannelNumber ATTRIBUTE
WITH APPROPRIATE SYNTAX
  integer;
BEHAVIOUR DEFINED AS
  "The current channel number of the frequency output by the RF synthesizer.";
REGISTERED AS
  { iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) CurrentChannelNumber (42) };

13.1.4.43 aMaxDwellTime

MaxDwellTime ATTRIBUTE
WITH APPROPRIATE SYNTAX
  integer;
BEHAVIOUR DEFINED AS
  "The maximum time in TU that the transmitter is permitted to operate on a single channel.";
REGISTERED AS
  { iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) MaxDwellTime (43) };

13.1.4.44 aCurrentSet

CurrentSet ATTRIBUTE
WITH APPROPRIATE SYNTAX
  integer;
BEHAVIOUR DEFINED AS
  "The current set of patterns that the PHY LME is using to determine the hop sequence.";
REGISTERED AS
  { iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) CurrentSet (44) };

13.1.4.45 aCurrentPattern

CurrentPattern ATTRIBUTE
WITH APPROPRIATE SYNTAX
  integer;
BEHAVIOUR DEFINED AS
  "The current pattern that the PHY LME is using to determine the hop sequence.";
REGISTERED AS
  { iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) CurrentPattern (45) };

13.1.4.46 aCurrentIndex

CurrentIndex ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The current index value the PHY LME is using to determine the CurrentChannelNumber.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) CurrentIndex (46) };

13.1.4.47 aCurrentChannel

CurrentChannel ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The current operating frequency channel of the DSSS PHY. Valid channel numbers are as defined in 15.4.6.2.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) CurrentChannel (47) };

13.1.4.48 aCCAModeSupported

CCAModeSupported ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"A list of the DSSS PHY CCA modes that are supported. Valid values are:
energy detect only (ED_ONLY) = 01,
carrier sense only (CS_ONLY) = 02,
carrier sense and energy detect (ED_and_CS)= 03.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) CCAModeSupported (48) };

13.1.4.49 aCurrentCCAMode

CurrentCCAMode ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The current CCA method in operation. Valid values are:
energy detect only (ED_ONLY) = 01,
carrier sense only (CS_ONLY) = 02,
carrier sense and energy detect (ED_and_CS)= 03.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) CurrentCCAMode (49) };

13.1.4.50 aEDThreshold

EDThreshold ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The current Energy Detect Threshold being used by the DSSS PHY.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) EDThreshold (50) };

13.1.4.51 aSynthesizerLocked

SynthesizerLocked ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"This is an indication that the PMD's synthesizer is locked to the current channel specified by aCurrentChannelNumber. X'00' represents unlocked while X'FF' represents locked.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) SynthesizerLocked (51) };

13.1.4.52 aCurrentPowerState

CurrentPowerState ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The current power state of the PHY. Valid values are ACTIVE=01 and DOZE=02.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) CurrentPowerState (52) };

13.1.4.53 aDozeTurnonTime

DozeTurnonTime ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The time in microseconds required by the PHY to progress from the Doze power down state to the ACTIVE operating state.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) DozeTurnonTime (53) };

13.1.4.54 aCCAWatchdogTimerMax

CCAWatchdogTimerMax ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"This parameter, together with CCAWatchdogCountMax, determines when energy detected in the channel can be ignored.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) CCAWatchdogTimerMax (54) };

13.1.4.55 aCCAWatchdogCountMax

CCAWatchdogCountMax ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"This parameter, together with CCAWatchdogTimerMax, determines when energy detected in the channel can be ignored.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) CCAWatchdogCountMax (55) };

13.1.4.56 aCCAWatchdogTimerMin

CCAWatchdogTimerMin ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The minimum value to which CCAWatchdogTimerMax can be set.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) CCAWatchdogTimerMin (56) };

13.1.4.57 aCCAWatchdogCountMin

CCAWatchdogCountMin ATTRIBUTE
WITH APPROPRIATE SYNTAX
integer;
BEHAVIOUR DEFINED AS
"The minimum value to which CCAWatchdogCountMin can be set.";
REGISTERED AS
{ iso(1) member-body(2) us(840) ieee802dot11(10036) phy(3) attribute(7) CCAWatchdogCountMin (57) };

14. Frequency-Hopping spread spectrum (FHSS) PHY specification for the 2.4 GHz Industrial, Scientific, and Medical (ISM) band

14.1 Overview

14.1.1 Overview of FHSS PHY

The PHY services provided to the IEEE 802.11 wireless LAN MAC for the 2.4 GHz frequency-hopping spread spectrum (FHSS) system are described in this clause. The FHSS PHY consists of the following two protocol functions:

a) A physical layer convergence function, which adapts the capabilities of the physical medium dependent (PMD) system to the PHY service. This function is supported by the physical layer convergence procedure (PLCP), which defines a method of mapping the IEEE 802.11 MAC sublayer protocol data units (MPDUs) into a framing format suitable for sending and receiving user data and management information between two or more STAs using the associated PMD system.
b) A PMD system, whose function defines the characteristics of, and method of transmitting and receiving data through, a wireless medium (WM) between two or more STAs.

14.1.2 FHSS PHY functions

The 2.4 GHz FHSS PHY architecture is shown in Figure 11. The FHSS PHY contains three functional entities: the PMD function, the physical layer convergence function, and the physical layer management function. Each of these functions is described in detail in the following subclauses.

The FHSS PHY service is provided to the MAC entity at the STA through a PHY service access point (SAP) as shown in called the PHY-SAP, as shown in Figure 11. A set of primitives might also be defined that describe the interface between the physical layer convergence protocol sublayer and the PMD sublayer, called the PMD-SAP.

14.1.2.1 PLCP sublayer

To allow the IEEE 802.11 MAC to operate with minimum dependence on the PMD sublayer, a PHY convergence sublayer is defined. This function simplifies provision of a PHY service interface to the IEEE 802.11 MAC services.

14.1.2.2 Physical layer management entity (PLME)

The PLME performs management of the local PHY functions in conjunction with the MAC management entity.

14.1.2.3 PMD sublayer

The PMD sublayer provides a transmission interface used to send and receive data between two or more STAs.

14.1.3 Service specification method and notation

The models represented by state diagrams in the following subclauses are intended as the primary specifications of the functions provided. It is important to distinguish, however, between a model and a real implementation. The models are optimized for simplicity and clarity of presentation, while any realistic implementation may place heavier emphasis on efficiency and suitability to a particular implementation technology.

The service of a layer or sublayer is the set of capabilities that it offers to a user in the next higher layer (or sublayer). Abstract services are specified here by describing the service primitives and parameters that characterize each service. This definition of service is independent of any particular implementation.

14.2 FHSS PHY specific service parameter lists

14.2.1 Overview

The architecture of the IEEE 802.11 MAC is intended to be PHY independent. Some PHY implementations require medium management state machines running in the MAC sublayer in order to meet certain PMD requirements. These PHY dependent MAC state machines reside in a sublayer defined as the MAC sublayer management entity (MLME). The MLME in certain PMD implementations may need to interact with the physical layer management entity (PLME) as part of the normal PHY-SAP primitives. These interactions are defined by the PLME parameter list currently defined in the PHY Service Primitives as TXVECTOR and RXVECTOR. The list of these parameters and the values they may represent are defined in the specific PHY specifications for each PMD. This subclause addresses the TXVECTOR and RXVECTOR for the FHSS PHY.

All of the values included in the TXVECTOR or RXVECTOR described in this subclause are considered mandatory unless otherwise specified. The 1 Mbit/s and 2 Mbit/s are the only rates currently supported. Other indicated data rates are for possible future use.

14.2.2 TXVECTOR parameters

The parameters in Table 28 are defined as part of the TXVECTOR parameter list in the PHY-TXSTART.request service primitive.

Table 28—TXVECTOR parameters

Parameter	Associate primitive	Value
LENGTH	PHY-TXSTART.request (TXVECTOR)	1–4095
DATARATE	PHY-TXSTART.request (TXVECTOR)	1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5

14.2.2.1 TXVECTOR LENGTH

The LENGTH parameter has the value of 1 to 4095. This parameter is used to indicate the number of octets in the MPDU that the MAC is currently requesting the PHY to transmit. This value is used by the PHY to determine the number of octet transfers that will occur between the MAC and the PHY after receiving a request to start a transmission.

14.2.2.2 TXVECTOR DATARATE

The DATARATE parameter describes the bit rate at which the PLCP should transmit the PSDU. Its value can be any of the rates as defined in Table 28, and supported by the conformant FH PHY.

14.2.3 RXVECTOR parameters

The parameters in Table 29 are defined as part of the RXVECTOR parameter list in the PHY-RXSTART.indicate service primitive.

Table 29—RXVECTOR parameters

Parameter	Associate primitive	Value
LENGTH	PHY-RXSTART.indicate (RXVECTOR)	1–4095
RSSI	PHY-RXSTART.indicate (RXVECTOR)	0–RSSI Max
DATARATE	PHY-RXSTART.request (RXVECTOR)	1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5

14.2.3.1 TRXVECTOR LENGTH

The LENGTH parameter has the value of 1 to 4095. This parameter is used to indicate the value contained in the LENGTH field that the PLCP has received in the PLCP Header. The MAC and PLCP will use this value to determine the number of octet transfers that will occur between the two sublayers during the transfer of the received PSDU.

14.2.3.2 RXVECTOR RSSI

The receive signal strength indicator (RSSI) is an optional parameter that has a value of 0 through RSSI Max. This parameter is a measure by the PHY sublayer of the energy observed at the antenna used to receive the current PPDU. RSSI shall be measured between the beginning of the start frame delimiter (SFD) and the end of the PLCP header error check (HEC). RSSI is intended to be used in a relative manner. Absolute accuracy of the RSSI reading is not specified.

14.3 FHSS PLCP sublayer

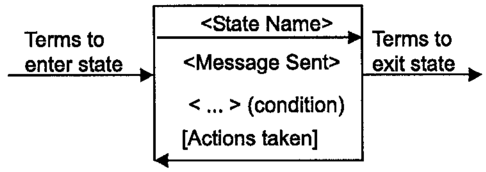
14.3.1 Overview

This subclause provides a convergence procedure to map MPDUs into a frame format designed for FHSS radio transceivers. The procedures for transmission, carrier sense, and reception are defined for single and multiple antenna diversity radios.

14.3.1.1 State diagram notation

The operation of the procedures can be described by state diagrams. Each diagram represents the domain and consists of a group of connected, mutually exclusive states. Only one state is active at any given time. Each state is represented by a rectangle as shown in Figure 69. These are divided into two parts by a horizontal line. In the upper part the state is identified by a name. The lower part contains the name of any signal that is generated. Actions are described by short phrases and enclosed in brackets.

Each permissible transition between the states is represented graphically by an arrow from the initial to the terminal state. A transition that is global in nature (for example, an exit condition from all states to the IDLE or RESET state) is indicated by an open arrow. Labels on transitions are qualifiers that must be fulfilled before the transition will be taken. The label UCT designates an unconditional transition. Qualifiers described by short phrases are enclosed in parentheses.



Key:
( ) = condition, for example, (if no_collision)
[ ] = action, for example, [resetPLSfunctions]
* = logical AND
+ = logical OR
UCT = unconditional transition

Figure 69—State diagram notation example

State transitions and sending and receiving of messages occur instantaneously. When a state is entered and the condition to leave that state is not immediately fulfilled, the state executes continuously, sending the messages and executing the actions contained in the state in a continuous manner.

Some devices described in this standard are allowed to have two or more ports. State diagrams capable of describing the operation of devices with an unspecified number of ports require qualifier notation that allows testing for conditions at multiple ports. The notation used is a term that includes a description in parentheses of which ports must meet the term for the qualifier to be satisfied (e.g., ANY and ALL). It is also necessary to provide for term-assignment statements that assign a name to a port that satisfies a qualifier. The following convention is used to describe a term-assignment statement that is associated with a transition:

a) The character “:” (colon) is a delimiter used to denote that a term assignment statement follows.
b) The character “<” (left arrow) denotes assignment of the value following the arrow to the term preceding the arrow.

The state diagrams contain the authoritative statement of the procedures they depict; when apparent conflicts between descriptive text and state diagrams arise, the state diagrams are to take precedence. This does not, however, override any explicit description in the text that has no parallel in the state diagrams.

The models presented by state diagrams are intended as the primary specifications to be provided. It is important to distinguish, however, between a model and a real implementation. The models are optimized for simplicity and clarity of presentation, while any realistic implementation may place heavier emphasis on efficiency and suitability to a particular implementation technology. It is the functional behavior of any unit that must match the standard, not its internal structure. The internal details of the model are useful only to the extent that they specify the external behavior clearly and precisely.

14.3.2 PLCP frame format

The PLCP protocol data unit (PPDU) frame format provides for the asynchronous transfer of MAC sublayer MPDUs from any transmitting STA to all receiving STAs within the wireless LAN’s BSS. The PPDU illustrated in Figure 70 consists of three parts: a PLCP Preamble, a PLCP Header, and a PSDU. The PLCP Preamble provides a period of time for several receiver functions. These functions include antenna diversity, clock and data recovery, and field delineation of the PLCP Header and the PSDU. The PLCP Header is used to specify the length of the whitened PSDU field and support any PLCP management information. The PPDU contains the PLCP Preamble, the PLCP Header, and the PSDU modified by the PPDU data whitener.

PLCP Preamble		PLCP Header			
Sync	Start Frame Delimiter	PLW	PSF	Header Error Check	Whitened PSDU
80 bits	16 bits	12 bits	4 bits	16 bits	Variable number of octets

Figure 70—PLCP frame format

14.3.2.1 PLCP Preamble

The PLCP Preamble contains two separate subfields; the Preamble Synchronization (SYNC) field and the Start Frame Delimiter (SFD), to allow the PHY circuitry to reach steady-state demodulation and synchronization of bit clock and frame start.

14.3.2.1.1 Preamble SYNC field

The Preamble SYNC field is an 80-bit field containing an alternating zero-one pattern, transmitted starting with zero and ending with one, to be used by the PHY sublayer to detect a potentially receivable signal, select an antenna if diversity is utilized, and reach steady-state frequency offset correction and synchronization with the received packet timing.

14.3.2.1.2 Start Frame Delimiter (SFD)

The SFD consists of the 16-bit binary pattern 0000 1100 1011 1101 (transmitted leftmost bit first). The first bit of the SFD follows the last bit of the sync pattern. The SFD defines the frame timing.

14.3.2.2 PLCP Header field

The PLCP Header field contains three separate subfields: a 12-bit PSDU Length Word (PLW), a 4-bit PLCP Signaling field (PSF), and a 16-bit PLCP HEC field.

14.3.2.2.1 PSDU length word

The PSDU length word (PLW) is passed from the MAC as a parameter within the PHY-TXSTART.request primitive. The PLW specifies the number of octets contained in the PSDU. Its valid values are X'001'–X'FFF', representing counts of one to 4095 octets. The PLW is transmitted lsb first and msb last. The PLW is used by the receiving STA, in combination with the 32/33 coding algorithm specified in this clause, to determine the last bit in the packet.

14.3.2.2.2 PLCP Signaling field (PSF)

The 4-bit PSF is defined in Table 30. The PSF is transmitted bit 0 first and bit 3 last.

Table 30—PSF bit descriptions

Bit	Parameter name	Parameter values	Description
0	Reserved	Default = 0	Reserved
1:3	PLCP_BITRATE	b1 b2 b3 = Data Rate 0 0 0 = 1.0 Mbit/s, 0 0 1 = 1.5 Mbit/s, 0 1 0 = 2.0 Mbit/s, 0 1 1 = 2.5 Mbit/s, 1 0 0 = 3.0 Mbit/s, 1 0 1 = 3.5 Mbit/s, 1 1 0 = 4.0 Mbit/s, 1 1 1 = 4.5 Mbit/s	This field indicates the data rate of the whitened PSDU from 1 Mbit/s to 4.5 Mbit/s in 0.5 Mbit/s increments.

14.3.2.2.3 Header Error Check (HEC) field

The HEC field is a 16-bit CCITT CRC-16 error detection field. The HEC uses the CCITT CRC-16 generator polynomial G(x) as follows:

$$G(x) = x^{16}+x^{12}+x^5+1$$

The HEC shall be the one's complement of the sum (modulo 2) of the following:

a) The remainder of $x^k \times (x^{15} + x^{14} + \ldots + x^2 + x^1 + 1)$ divided (modulo 2) by $G(x)$, where k is the number of bits in the PSF and PLW fields of the PLCP Header;
b) The remainder after multiplication by x^{16} and then division (modulo 2) by $G(x)$ of the content (treated as a polynomial) of the PSF and PLW fields.

The HEC shall be transmitted with the coefficient of the highest term first.

As a typical implementation, at the transmitter, the initial remainder of the division is preset to all ones and is then modified by division of the PSF and PLW fields by the generator polynomial, $G(x)$. The one's complement of this remainder is inserted in the HEC field with the most significant bit transmitted first.

At the receiver, the initial remainder of the division is again preset to all ones. The division of the received PSF, PLW, and HEC fields by the generator polynomial, $G(x)$, results, in the absence of transmission errors, in a unique nonzero value, which is the following polynomial $R(x)$:

$$R(x) = x^{12} + x^{11} + x^{10} + x^8 + x^3 + x^2 + x^1 + 1$$

14.3.2.3 PLCP data whitener

The PLCP data whitener uses a length-127 frame-synchronous scrambler followed by a 32/33 bias-suppression encoding to randomize the data and to minimize the data dc bias and maximum run lengths. Data octets are placed in the transmit serial bit stream lsb first and msb last. The frame synchronous scrambler uses the generator polynomial $S(x)$ as follows:

$S(x) = x^7 + x^4 + 1$

and is illustrated in Figure 71. The 127-bit sequence generated repeatedly by the scrambler is (leftmost bit used first) 00001110 11110010 11001001 00000010 00100110 00101110 10110110 00001100 11010100 11100111 10110100 00101010 11111010 01010001 10111000 1111111. The same scrambler is used to scramble transmit data and to descramble receive data. The data whitening starts with the first bit of the PSDU, which follows the last bit of the PLCP Header. The specific bias suppression encoding and decoding method used is defined in Figure 75 and Figure 80. The format of the packet after data whitening is as shown in Figure 72.

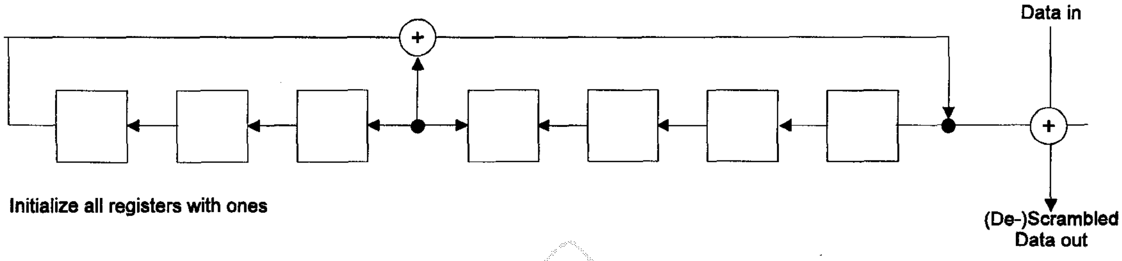


Figure 71—Frame synchronous scrambler/descrambler

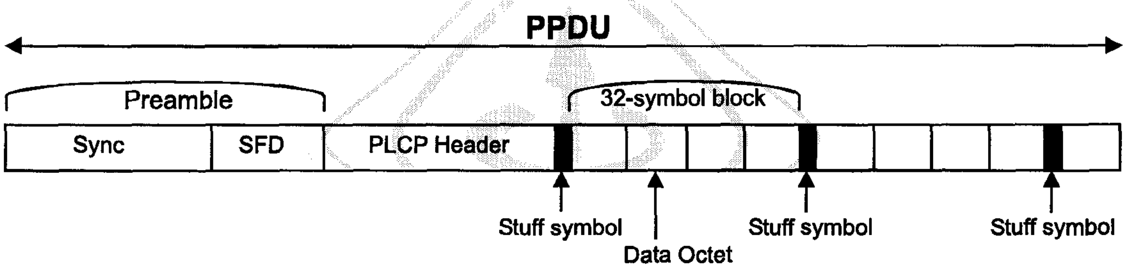


Figure 72—PLCP data whitener format

14.3.3 PLCP state machines

The PLCP consists of three state machines, as illustrated in the overview diagram of Figure 73: the transmit (TX), carrier sense/clear channel assessment (CS/CCA), and receive (RX) state machines. The three PLCP state machines are defined in the subclauses below; Figure 73 is not a state diagram itself. Execution of the PLCP state machines normally is initiated by the FH PLME state machine and begins at the CS/CCA state machine. The PLCP returns to the FH PLME state machine upon interrupt to service a PLME service request, such as PLME-SET, PLME-RESET, etc.

14.3.3.1 PLCP transmit procedure

The PLCP transmit procedure is invoked by the CS/CCA procedure immediately upon receiving a *PHY-TXSTART.request(TXVECTOR)* from the MAC sublayer. The CSMA/CA protocol is performed by the MAC with the PHY PLCP in the CS/CCA procedure prior to executing the transmit procedure.

14.3.3.1.1 Transmit state machine

The PLCP transmit state machine illustrated in Figure 74 includes functions that must be performed prior to, during, and after PPDU data transmission. Upon entering the transmit procedure in response to a *PHY-TXSTART.request (TXVECTOR)* from the MAC, the PLCP shall switch the PHY PMD circuitry from

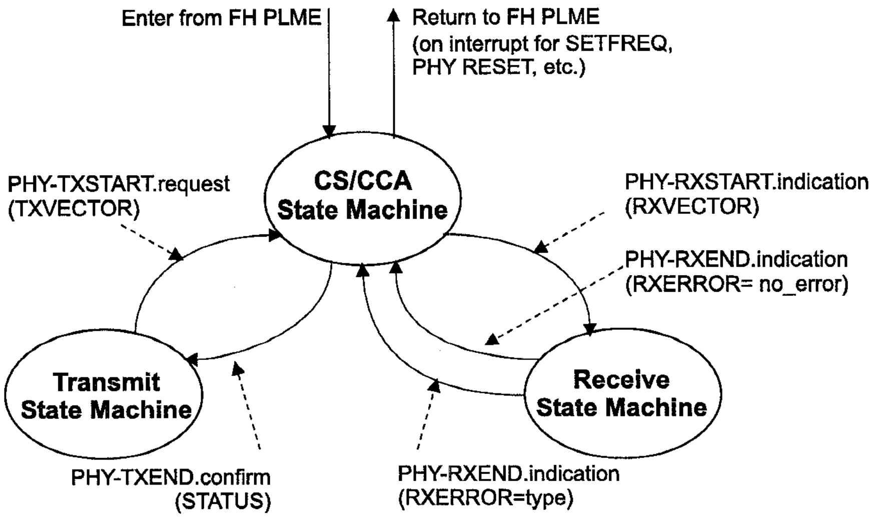


Figure 73—PLCP top-level state diagram

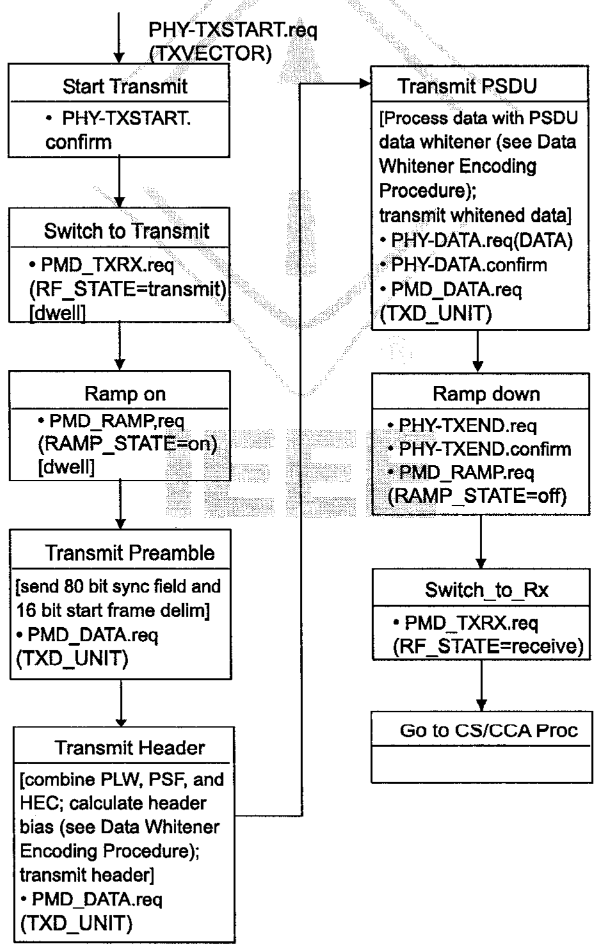


Figure 74—Transmit state machine

receive to transmit state; ramp on the transmit power amplifier in the manner prescribed in 14.6; and transmit the preamble sync pattern and SFD. The PLCP shall generate the PLCP Header as defined in 14.3.2.2 in sufficient time to send the bits at their designated bit slot time. The PLCP shall add the PLCP Header to the start of the PSDU data.

Prior to transmitting the first PSDU data bit, the PLCP shall send a *PHY-TXSTART.confirm* message to the MAC indicating that the PLCP is ready to receive an MPDU data octet. The MAC will pass an MPDU data octet to the PHY with a *PHY-DATA.request(DATA)*, which the PHY will respond to with a *PHY-DATA.confirm*. This sequence of *PHY-DATA.request(DATA)* and *PHY-DATA.confirm* shall be executed until the last data octet is passed to the PLCP. During transmission of the PSDU data, each bit of the PSDU shall be processed by the data whitener algorithm defined in Figure 75 and described in 14.3.2.3. Each PSDU data octet is processed and transmitted lsb first and msb last.

Data whitener encoding algorithm:

```
/*        If msb of stuff symbol = 1 then the next block is inverted; 0 = not inverted */
/*        Accumulate PLCP Header; begin stuffing on first bit of the PSDU */

/********* Calculate number of 32-symbol BSE blocks required to send PSDU;
                no padding is necessary when the number of symbols is not a multiple of 32 *********/
Input parameter: number_of_PSDU_octets, rate;              /* rate is 1 or 2*/
number_of_symbols= (number_of_PSDU_octets *8) /rate;
number_of_blocks_in_packet = truncate{(number_of_symbols + 31) / 32)};

/********* Accumulate the bias in the header to use in calculating the inversion state of the first
                block of PSDU data *********/
Read in header {b(1),...,b(32)};                 /* b(1) is first bit in */
header_bias = Sum{weight(b(1)),...,weight(b(32))};
                        /* calculate bias in header; weights are defined in Table 31*/
Transmit {b(1),...,b(32)};                       /* no stuffing on header */
accum=header_bias;                               /* initialize accum */
Initialize scrambler to all ones;

/********* Whiten the PSDU data with scrambler and BSE encoder *********/
For n = 1 to number_of_blocks_in_packet
{
        b(0) = 0 for 1 Mbit/s; b(0)=00 for 2 Mbit/s;            /* b(0) is the stuff symbol */
        N = min(32, number_of_symbols);                          /* N= block size in symbols */
        Read in next symbol block {b(1),...,b(N)};               /* b(n) = {0,1} or {0,1,2,3};
        1 - 8 octets, use PHY-DATA.req(DATA), PHY-DATA.confirm for each octet*/
        Scramble {b(1),...,b(N)};                                    /* see 14.3.2.3*/
        bias_next_block = Sum{weight(b(0)),...,weight(b(N))};        /* calculate bias with b(0)=0 */

        /***** if accum and bias of next block has the same sign, then invert block;
                if accum=0 or bias_next_block=0, don't invert *****/
        If {[accum * bias_next_block > 0] then
         {
                Invert {b(0),...,b(N)};        /* Invert deviation, or, negate msb of symbol */
                bias_next_block = - bias_next_block;
         }

        accum = accum + bias_next_block;
        transmit {b(0),...,b(N)};                   /* b(0) is first symbol out */
        number_of_symbols = number_of_symbols - N
}
```

Figure 75—Data whitener encoding procedure

After the last MPDU octet is passed to the PLCP, the MAC will indicate the end of the frame with a *PHY-TXEND.request*. After the last bit of the PSDU data has completed propagation through the radio and been transmitted into the air, the PLCP shall complete the transmit procedure by sending a *PHY-TXEND.confirm* to the MAC sublayer, ramp off the power amplifier in the manner prescribed in 14.6, and switch the PHY PMD circuitry from transmit to receive state. The execution shall then return to the CS/CCA procedure.

The weights assigned to each value of the symbols are defined in Table 31 for the 1 Mbit/s (2GFSK) and 2 Mbit/s (4GFSK) symbols.

Table 31—PLCP field bit descriptions

2GFSK	4GFSK	Weight
	10	3
1		2
	11	1
Center	Center	0
	01	–1
0		–2
	00	–3

14.3.3.1.2 Transmit state timing

The transmit timing illustrated in Figure 76 is defined from the instant that the *PHY-TXSTART.request(TXVECTOR)* is received from the MAC sublayer. The PLCP shall switch the PMD circuitry from receive to transmit, turn on and settle the transmitter, and begin transmitting the first bit of the preamble at the antenna within a maximum of 20 µs of receiving the *PHY-TXSTART.request(TXVECTOR)*. The PLCP Preamble shall be transmitted at 1 Mbit/s and be completed in 96 µs. The PLCP Header shall be transmitted at 1 Mbit/s and be completed in 32 µs. The variable length PSDU shall be transmitted at the selected data rate. After the last bit of the PSDU data has completed propagation through the radio and been transmitted onto the air, the PLCP shall send the *PHY-TXEND.confirm* to the MAC sublayer. The PLCP shall turn off the transmitter, reducing the output energy to less than the specified off-mode transmit power within the time specified in 14.6. At the end of the power amplifier ramp down period, the PLCP shall switch the PMD circuitry from transmit to receive.

14.3.3.2 Carrier sense/clear channel assessment (CS/CCA) procedure

The PLCP CS/CCA procedure is executed while the receiver is turned on and the STA is not currently receiving or transmitting a packet. The CS/CCA procedure is used for two purposes: to detect the start of a network signal that can be received (CS) and to determine whether the channel is clear prior to transmitting a packet (CCA).

14.3.3.2.1 CS/CCA state machine

Timing for priority (PIFS, DIFS), contention backoff (slot times), and CS/CCA assessment windows are defined relative to the end of the last bit of the last packet on the air. The CS/CCA state machine is shown in Figure 77. The PLCP shall perform a CS/CCA assessment on a minimum of one antenna within a MAC contention backoff slot time of 50 µs. The PLCP shall be capable of detecting within the slot time an FH PHY conformant signal that is received at the selected antenna up to 22 µs after the start of the slot time with the synchronous detection performance specified in 14.6.15.3. Subclause 14.6.15.3 specifies detection performance with zero-one sync patterns and with random data patterns. If a start of a transmission is asynchronous with the BSS and arrives after the start of the slot but at least 16 µs prior to the end of the slot, the PLCP shall indicate a busy channel prior to the end of the slot time with the asynchronous detection performance specified

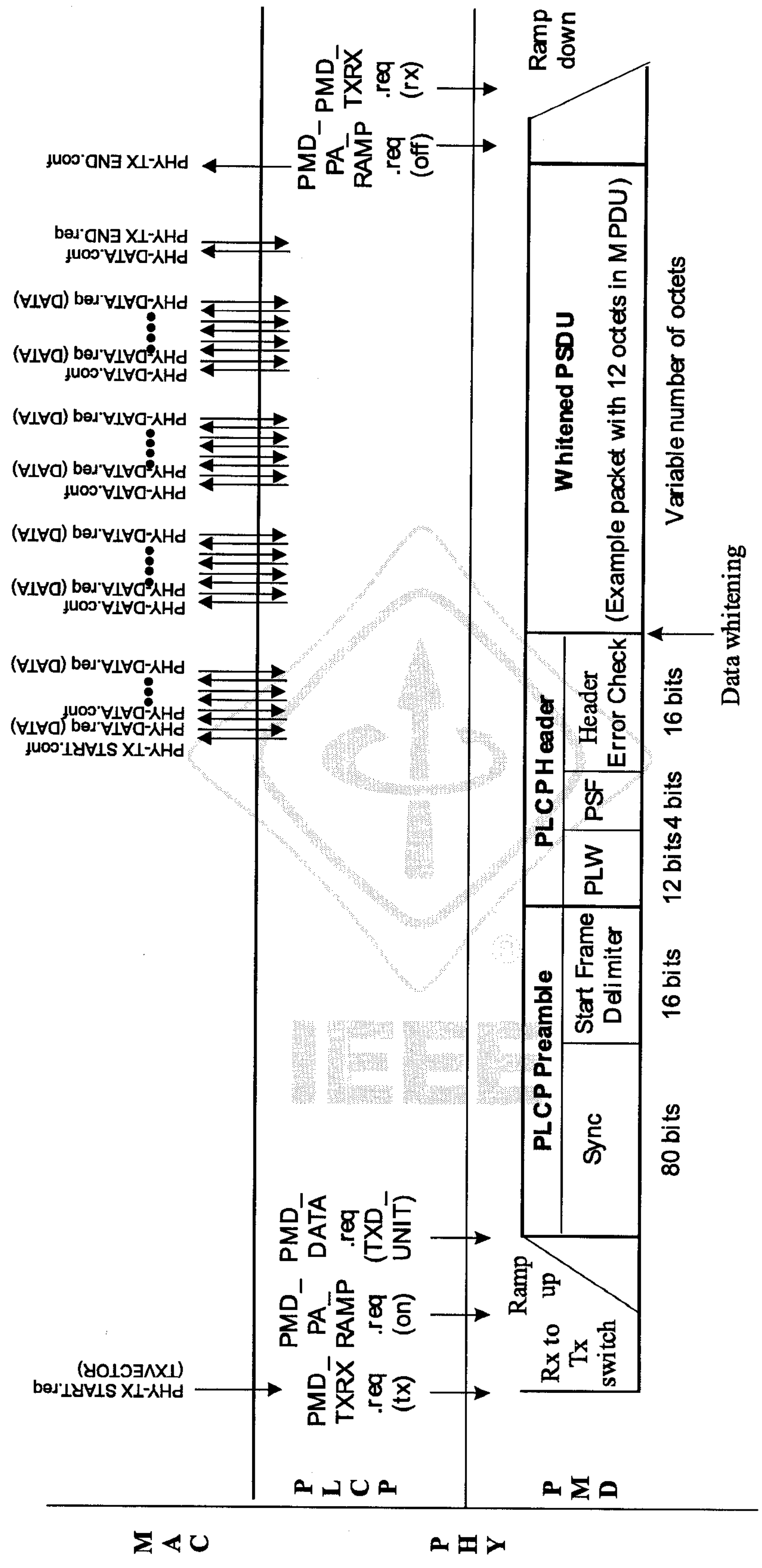


Figure 76—Transmit state timing

in 14.6.15.3. The CCA indication immediately prior to transmission shall be performed on an antenna with essentially the same free space gain and gain pattern as the antenna to be used for transmission. The method of determining CS/CCA is unspecified except for the detection performance of a conformant method as specified in 14.6.15.3.

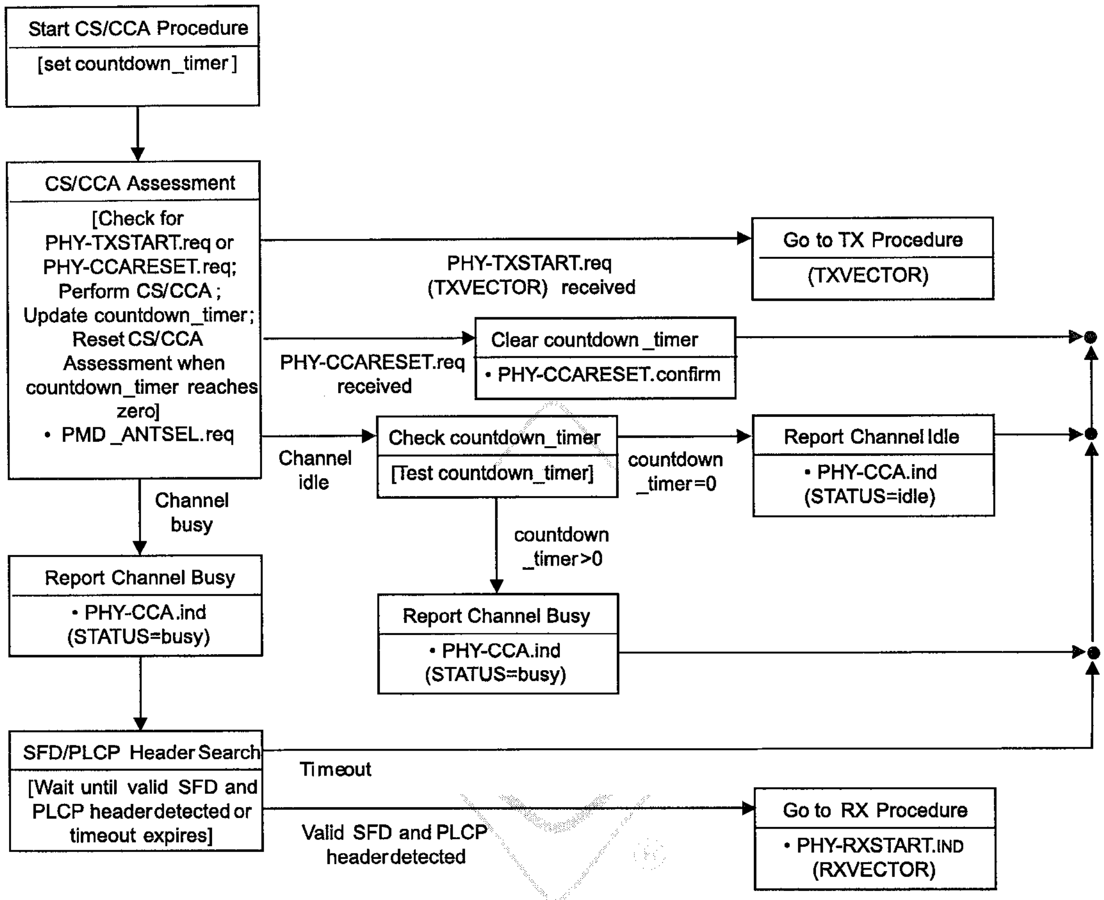


Figure 77—CS/CCA state machine

If a *PHY-TXSTART.request (TXVECTOR)* is received, the CS/CCA procedure shall exit to the transmit procedure within 1 μs. If a *PHY-CCARESET.request* is received, the PLCP shall reset the CS/CCA state machine to the state appropriate for the end of a complete received frame. This service primitive is generated by the MAC at the end of a NAV period. The PHY shall indicate completion of the request by sending a *PHY-CCARESET.confirm* to the MAC.

If a CS/CCA assessment returns a channel idle result, the PHY shall send a *PHY-CCA.indicate(STATUS=idle)* to the MAC.

If a CS/CCA assessment returns a channel busy result, the PHY shall send a *PHY-CCA.indicate(STATUS=busy)* to the MAC. Upon a channel busy assessment, the PLCP shall stop any antenna switching prior to the earliest possible arrival time of the SFD and detect a valid SFD and PLCP Header if received. A valid PLCP Header is defined as containing valid PLCP Length Word and PHY Signaling field values and a valid HEC field. If a valid SFD/PLCP Header is detected, the CS/CCA procedure shall send a PHY-RXSTART.indicate(RXVECTOR) message to the MAC sublayer and exit to the receive procedure. The PLCP shall dwell and search for the SFD/PLCP Header for a minimum period longer than the latest possible arrival time of the SFD/PLCP Header. Indication of a busy channel does not necessarily lead to the successful reception of a frame.

The octet/bit count remaining may be a nonzero value when returning from the receive procedure if a signal in the process of being received was lost prior to the end as determined from the Length field of a valid PLCP Header. The countdown timer shall be set to the octet/bit count and used to force the CS/CCA indication to remain in the BUSY state until the predicted end of the frame regardless of actual CS/CCA indications.

However, if the CS/CCA procedure indicates the start of a new frame within the countdown timer period, it is possible to transition to the receive procedure prior to the end of the countdown timer period. If the PHY transitions to receive under these conditions, the countdown timer shall be reset to the longer of (1) the remaining time of the current frame and (2) the length of the new frame.

When a nonzero countdown timer reaches zero, the PLCP shall reset the CS/CCA state machine to the state appropriate for the end of a complete received frame and the CS/CCA indication shall reflect the state of the channel.

If the receive procedure encountered an unsupported rate error, the PLCP shall keep the CS/CCA state at Busy for the duration of the frame by setting the countdown timer to the value corresponding to the calculated time based on the information in the PLCP Header and the 33/32 expansion factor.

14.3.3.2.2 CS/CCA state timing

Timing for priority (PIFS, DIFS), contention backoff (slot times), and CS/CCA assessment windows is defined relative to the end of the last bit of the last packet on the air. The PLCP shall perform a CS/CCA assessment on a minimum of one antenna within a slot time. The appropriate CS/CCA indication shall be available prior to the end of each 50 μs slot time with the performance specified in 14.6. See Figure 78.

If a STA has not successfully received the previous packet, the perceived packet end time and slot boundary times will have a higher uncertainty for that STA.

14.3.3.3 PLCP receive procedure

The PLCP receive procedure is invoked by the PLCP CS/CCA procedure upon detecting a portion of the preamble sync pattern followed by a valid SFD and PLCP Header.

14.3.3.3.1 Receive state machine

The PLCP receive procedure shown in Figure 79 includes functions that must be performed while the PPDU is being received. The PLCP receive procedure begins upon detection of a valid SFD and PLCP Header in the CS/CCA procedure. The PLCP shall set a PPDU octet/bit counter to indicate the last bit of the packet, receive the PPDU bits, and perform the data whitening decoding procedure shown in Figure 80 on each PPDU bit. The PLCP shall pass correctly received data octets to the MAC with a series of *PHY-DATA.indicate(DATA)*. After the last PPDU bit is received and the last octet is passed to the MAC, the PLCP shall send a *PHY-RXEND.indicate(RXERROR=no_error)* to the MAC sublayer. Upon error-free completion of a packet reception, the PLCP shall exit the receive procedure and return to the PLCP CS/CCA procedure with the octet/bit count set to 0.

If the PLCP Header was decoded without a CRC error but encountered an unsupported rate, then the PLCP shall immediately complete the receive procedure with a *PHY-RXEND.indicate (RXERROR = unsupported_rate)* to the MAC, and return to the CS/CCA procedure with the octet/bit count remaining and the data rate value contained in the PLCP Header.

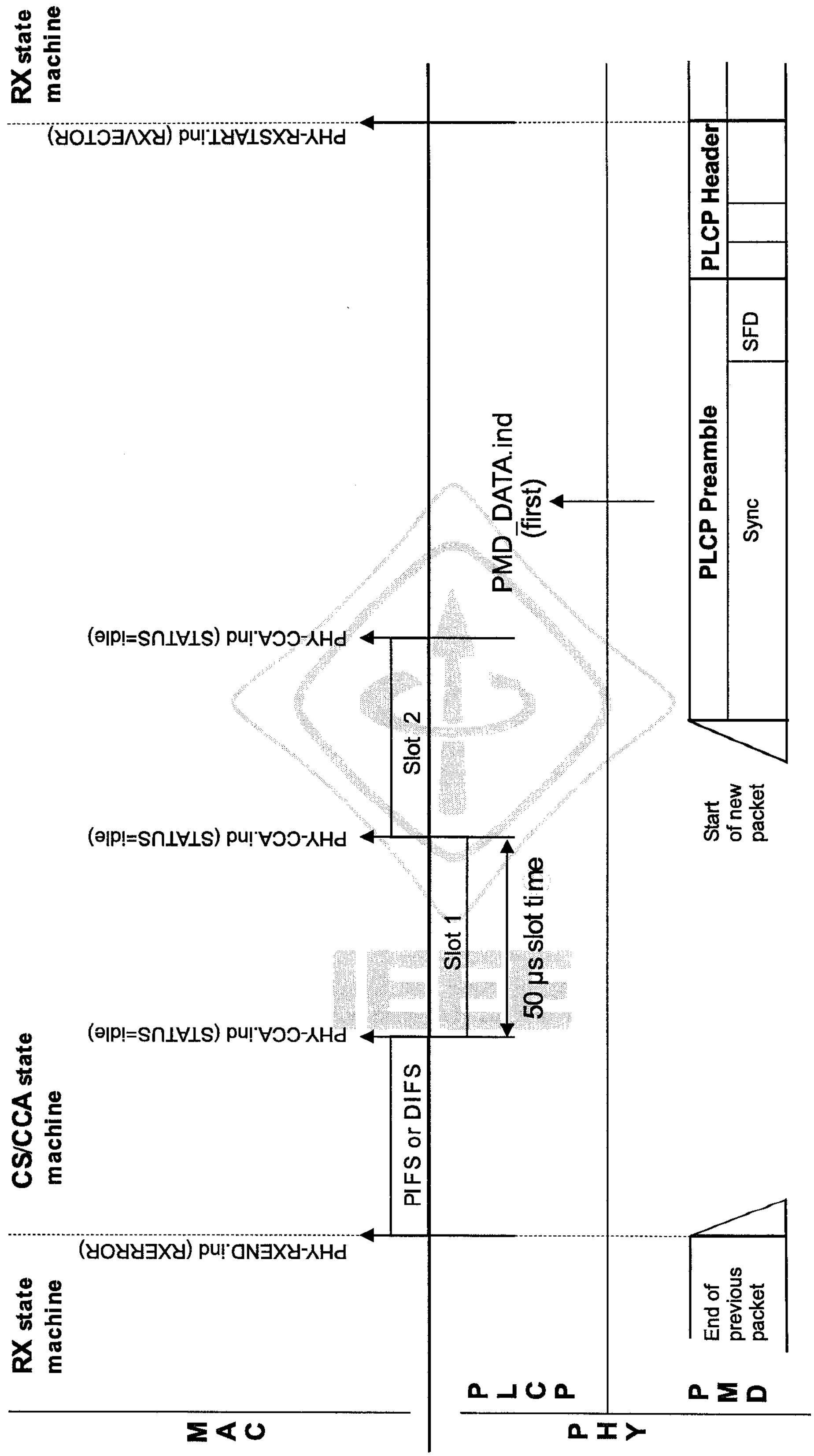


Figure 78—CS/CCA state timing

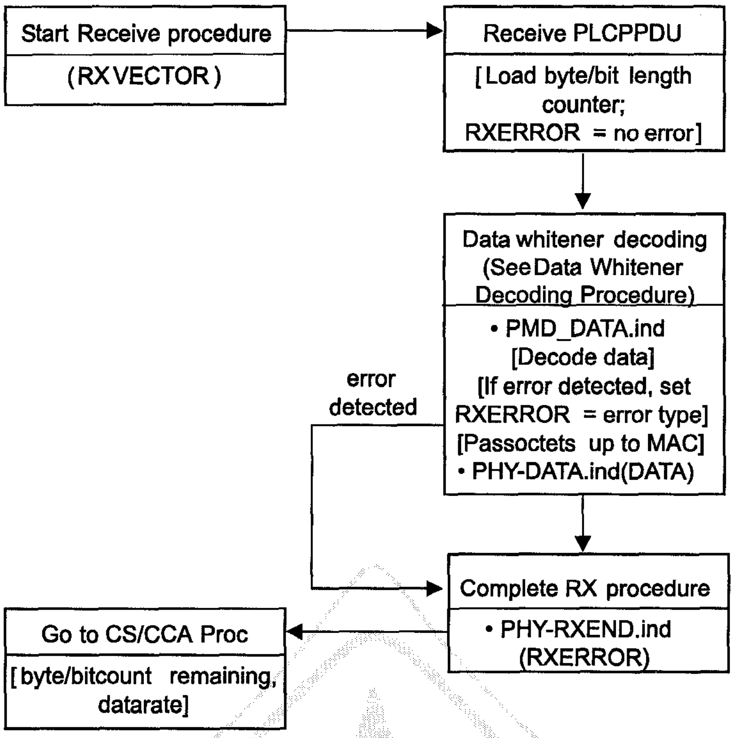


Figure 79—Receive state machine

If an error was detected during the reception of the PPDU, the PLCP shall immediately complete the receive procedure with a *PHY-RXEND.indicate(RXERROR=carrier_lost)* to the MAC, and return to the CS/CCA procedure with the octet/bit count remaining and the data rate value contained in the PLCP Header.

Data whitener decoding algorithm:

```
/*      If msb of stuff symbol = 1 then the next block is inverted; 0 = not inverted */
/*      Stuffing begins on first symbol of PLCP Header following the SFD */
/*      Algorithm begins after verifying validity of header with HEC */

/********* Read header *********/
Read in header {b(1),...,b(32)};                                /* b(1) is first bit in */

Get number_of_PSDU_octets, rate from header;                    /* rate is 1 or 2 */
number_of_symbols = (number_of_PSDU_octets*8)/rate
number_of_blocks_in_packet = truncate{(number_of_symbols + 31) / 32};
Initialize scrambler to all ones;

/********* De-whiten the PPDU data with BSE decoder and de-scrambler *********/
For n = 1 to number_of_blocks_in_packet
{
        N = min(32, # of symbols remaining);            /* N= block size in symbols */
        Read in next block {b(0),...,b(N)};/                    * b(n) = {0,1} or {0,1,2,3} */

        If {[msb of b(0)=1] then Invert {b(1),...,b(N)};        /* if invert bit=true */
        Descramble {b(1),...,b(N)};                     /* see 14.3.2.3*/
        Send {b(1),...,b(N)} to MAC
                                /* 1 • 8 octets; use PHY-DATA.ind(DATA) for each octet. */
}
```

Figure 80—Data whitener decoding procedure

14.3.3.3.2 Receive state timing

The receive state timing shown in Figure 81 is defined to begin upon detection of a valid SFD and PLCP Header in the CS/CCA procedure. The PLCP shall begin receiving the variable length whitened PSDU immediately after the end of the last bit of the PLCP Header. The PLCP shall send a *PHY-RXEND.indicate(RXERROR)* after receiving the last PPDU data bit.

If any error was detected during the reception of the PPDU, the PLCP may send a PHY-RXEND.indicate(RXERROR) and terminate the receive procedure before the last bit arrives.

14.4 PLME SAP layer management

14.4.1 Overview

This subclause describes the services provided by the FHSS PLME to the upper layer management entities. The PLME/PMD services are defined in terms of service primitives. These primitives are abstract representations of the services and are not intended to restrict implementations.

14.4.2 FH PHY specific MAC sublayer management entity (MLME) procedures

14.4.2.1 Overview

The specific MAC sublayer management entity (MLME) procedures required for operating the FHSS PHY are specified in this portion of the subclause. The relationship between the MLME and FH PLME procedures is also described.

14.4.2.2 FH synchronization

The MLME of a compliant FH PHY STA shall perform the FH time synchronization procedure as defined in 11.1.5. This procedure provides for synchronized frequency hopping for all compliant FH PHY STAs within a single BSS or ad hoc network. The FH PLME accepts PLME-SET.request commands from the MLME to change the tune frequency at the time determined by the MLME. The tune frequency is changed by updating any combination of the Set, Pattern, and Index PHY MIB parameters.

14.4.3 FH PHY layer management entity state machines

14.4.3.1 Overview

This portion of this subclause describes the FH PHY layer management state machines to turn the PMD on/off, reset the PLCP state machine, and change the frequency hop channel.

14.4.3.2 PLME state machine

The PLME state machine in Figure 82 begins with a *PLME-SET.request (aCurrentPowerState= ON)* request, which turns on the PHY circuitry, resets the PLME and PLCP state machines, and sends a *PLME-SET.confirm*. The MAC then sends a series of three *PLME-SET.request* primitives to update the aCurrentSet, aCurrentPattern, and aCurrentIndex PHY MIB parameters, which together tune the PMD to the selected channel. The PLME then transfers execution to the PLCP state machine as defined in 14.3.3.

Upon receiving a PLME request from a higher-level LME, the PLCP shall return execution to the PLME state machine and process the request. A *PLME-RESET.request* shall cause a reset to the PLME and PLCP state machines. A *PLME-SET.request* updating the aCurrentIndex or a combination of the aCurrentSet, aCurrentPattern, and aCurrentIndex shall cause the PLCP to terminate a receive or CS/CCA process and

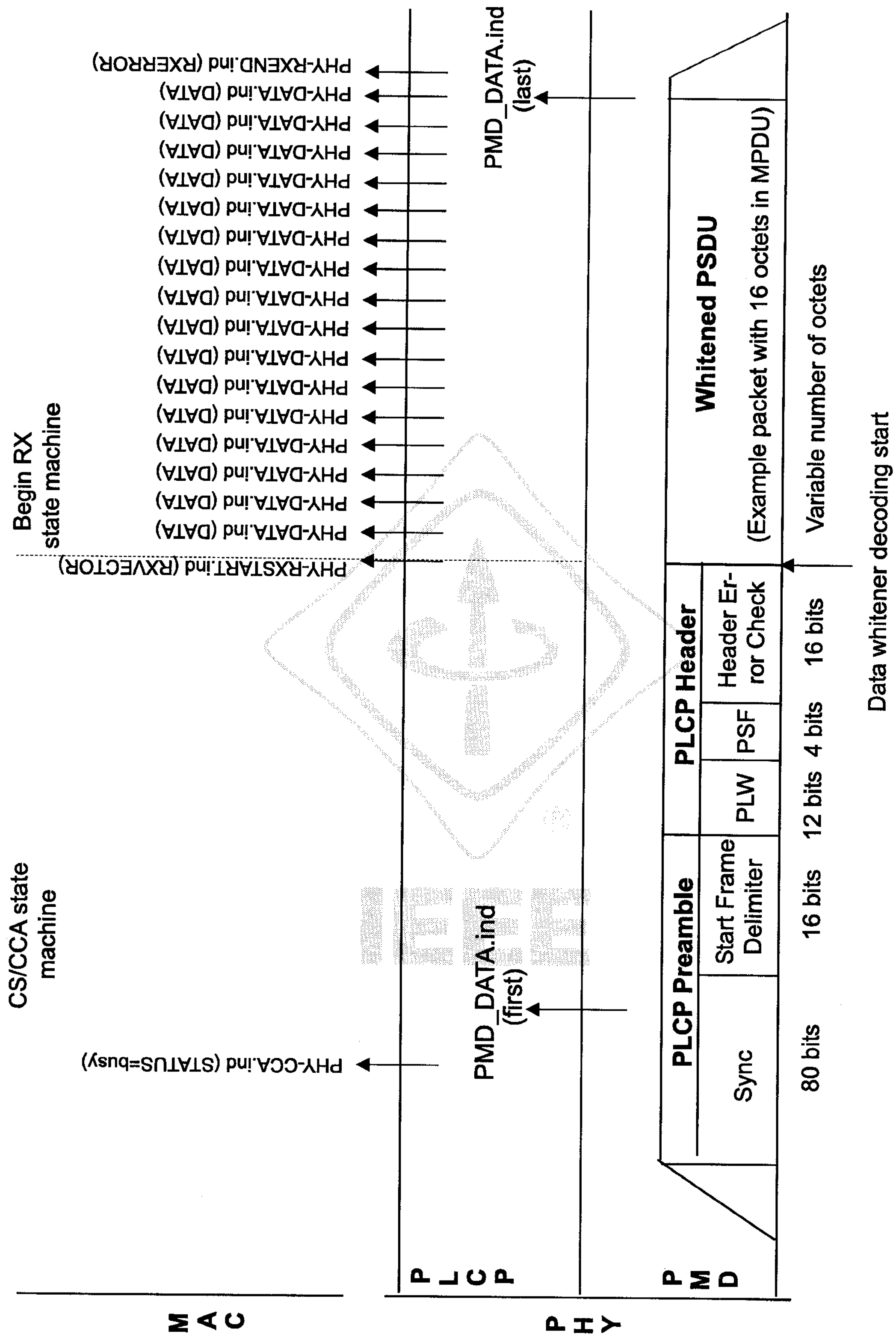


Figure 81—Receive timing

change frequency before returning to the PLCP state machine. A *PLME-SET.request(aCurrentPowerState=OFF)* request shall cause the PLCP to terminate a receive or CS/CCA process, power down the PMD circuitry, and return the PLME state machine to the idle state. *PLME-SET.requests* to any parameter other than the ones identified within this paragraph shall be executed and control returned to the PLCP state machine. The MAC should not send a PLME request while the PLCP is in the transmit state.

All *PLME-GET.requests* shall be processed in parallel and with no interruption to the execution of any state machine in process.

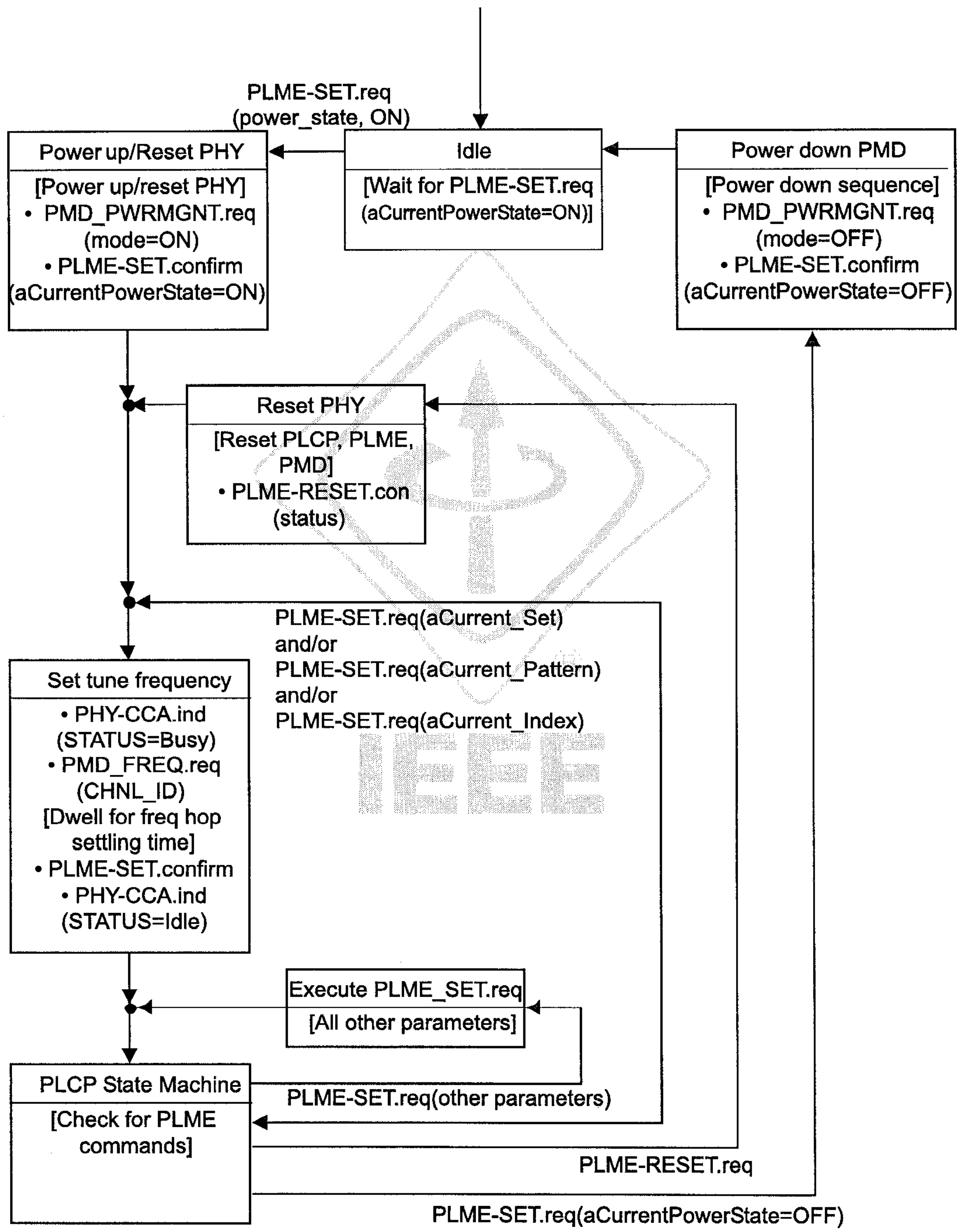


Figure 82—PLME state machine

14.4.3.3 PLME management primitives

The FH PLME uses the generic management primitives defined in 10.2 to manage all FH PHY parameters.

14.5 FHSS PMD sublayer services

14.5.1 Scope and field of application

The PMD services provided to the PLCP for the FHSS PHY are described in this subclause. Also defined in this subclause are the functional, electrical, and RF characteristics required for interoperability of implementations conforming to this specification. The relationship of this specification to the entire FHSS PHY is shown in Figure 83.

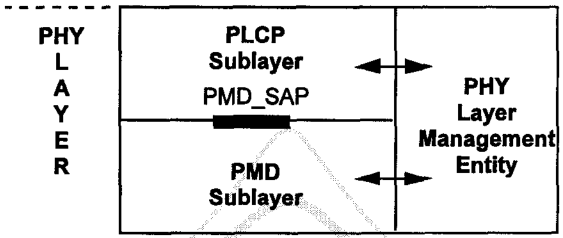


Figure 83—PMD layer reference model

14.5.2 Overview of services

In general, the FHSS PMD sublayer accepts PLCP sublayer service primitives and provides the actual means by which the signals required by these primitives are imposed onto the medium. In the FHSS PMD sublayer at the receiver the process is reversed. The combined function of the transmitting and receiving FHSS PMD sublayers results in a data stream, timing information, and receive parameter information being delivered to the receiving PLCP sublayer.

14.5.3 Overview of interactions

The primitives associated with the IEEE 802.11 PLCP sublayer to the FHSS PMD sublayer falls into two basic categories:

a) Service primitives that support PLCP peer-to-peer interactions
b) Service primitives that have local significance and support sublayer-to-sublayer interactions

14.5.4 Basic service and options

All of the service primitives described in this subclause are considered mandatory unless otherwise specified.

14.5.4.1 PMD_SAP peer-to-peer service primitives

Table 32 indicates the primitives for peer-to-peer interactions.

Table 32—PMD-SAP peer-to-peer service primitives

Primitive	Request	Indicate	Confirm	Response
PMD_DATA	X	X		

14.5.4.2 PMD_SAP sublayer-to-sublayer service primitives

Table 33 indicates the primitives for sublayer-to-sublayer interactions.

Table 33—PMD_SAP sublayer-to-sublayer service primitives

Primitive	Request	Indicate	Confirm	Response
PMD_TXRX	X			
PMD_PA_RAMP	X			
PMD_ANTSEL	X			
PMD_TXPWRLVL	X			
PMD_FREQ	X			
PMD_RSSI		X		
PMD_PWRMGMT	X			

14.5.4.3 PMD_SAP service primitives parameters

Table 34 shows the parameters used by one or more of the PMD_SAP service primitives.

Table 34—List of parameters for PMD primitives

Parameter	Associate primitive	Value
TXD_UNIT	PMD_DATA.request	1 Mbit/s: 0, 1 2 Mbit/s: 0, 1, 2, 3
RXD_UNIT	PMD_DATA.indicate	1 Mbit/s: 0, 1 2 Mbit/s: 0, 1, 2, 3
RF_STATE	PMD_TXRX.request	TRANSMIT, RECEIVE
RAMP_STATE	PMD_PA_RAMP.request	ON, OFF
ANTENNA_STATE	PMD_ANTSEL.request	1 to 255
TXPWR_LEVEL	PMD_TXPWRLVL.request	LEVEL1, LEVEL2, LEVEL3, LEVEL 4
CHNL_ID	PMD_FREQ.request	2 through 80 inclusive
STRENGTH	PMD_RSSI.indicate	0 to RSSI Max
MODE	PMD_PWRMGMT.request	ON, OFF

14.5.5 PMD_SAP detailed service specification

This subclause describes the services provided by each PMD primitive.

14.5.5.1 PMD_DATA.request

14.5.5.1.1 Function

This primitive defines the transfer of data from the PLCP sublayer to the PMD entity.

14.5.5.1.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_DATA.request (TXD_UNIT)

The TXD_UNIT parameter can take on one of two values: ONE or ZERO. This parameter represents a single data bit. The effect of this parameter is that the PMD will properly modulate the medium to represent ONEs or ZEROs as defined in the FHSS PMD modulation specifications for a given data rate.

14.5.5.1.3 When generated

This primitive is generated by the PLCP sublayer to request the transmission of a single data bit on the PMD sublayer. The bit clock is assumed to be resident or part of the PLCP and this primitive is issued at every clock edge once the PLCP has begun transmitting data.

14.5.5.1.4 Effect of receipt

The receipt of this primitive will cause the PMD entity to encode and transmit a single data bit.

14.5.5.2 PMD_DATA.indicate

14.5.5.2.1 Function

This primitive defines the transfer of data from the PMD entity to the PLCP sublayer.

14.5.5.2.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_DATA.indicate (RXD_UNIT)

The RXD_UNIT parameter can take on one of two values: ONE or ZERO. This parameter represents the current state of the medium as determined by the FHSS PMD modulation specifications for a given data rate.

14.5.5.2.3 When generated

The PMD_DATA.indicate is generated to all receiving PLCP entities in the network after a PMD_DATA.request is issued.

14.5.5.2.4 Effect of receipt

The effect of receipt of this primitive by the PLCP is unspecified.

14.5.5.3 PMD_TXRX.request

14.5.5.3.1 Function

This primitive is used to place the PMD entity into the transmit or receive function.

14.5.5.3.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_TXRX.request (RF_STATE)

The RF_STATE parameter can take on one of two values: TRANSMIT or RECEIVE. When the value of the primitive is TRANSMIT, the RF state of the radio is transmit. If the value of the primitive is RECEIVE, the RF state of the radio is receive.

14.5.5.3.3 When generated

This primitive is generated whenever the mode of the radio needs to be set or when changing from transmit to receive or receive to transmit.

14.5.5.3.4 Effect of receipt

The receipt of this primitive by the PMD entity will cause the mode of the radio to be in either transmit or receive.

14.5.5.4 PMD_PA_RAMP.request

14.5.5.4.1 Function

This primitive defines the start of the ramp-up or ramp-down of the radio transmitter's power amplifier.

14.5.5.4.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_PA_RAMP.request (RAMP_STATE)

The RAMP_STATE parameter can take on one of two values: ON or OFF. When the value of the primitive is ON, the state of the transmit power amplifier is "on." If the value of the primitive is OFF, the state of the transmit power amplifier is "off."

14.5.5.4.3 When generated

This primitive is issued only during transmit and to establish the initial state. It is generated by the PLCP at the start of the transmit function to turn the transmitter's power amplifier "on." A power amplifier ramp-up period follows the change of state from "off" to "on." After the PLCP has transferred all required data to the PMD entity, this primitive again will be issued by the PLCP to place the transmit power amplifier back into the "off" state. A power amplifier ramp-down period follows the change of state from "on" to "off."

14.5.5.4.4 Effect of receipt

The receipt of this primitive by the PMD entity will cause the transmit power amplifier to become on or off.

14.5.5.5 PMD_ANTSEL.request

14.5.5.5.1 Function

This primitive is used to select which antenna the PMD entity will use to transmit or receive data.

14.5.5.5.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_ANTSEL.request (ANTENNA_STATE)

The ANTENNA_STATE parameter can take on values from one to N (where N is the number of antennas supported). When the value of the primitive is a ONE, the PMD will switch to antenna 1 for receive or transmit; if the value of the primitive is TWO, the PMD entity will switch to antenna 2 for receive or transmit, etc.

14.5.5.5.3 When generated

This primitive is generated at various times by the PLCP entity to select an antenna. During receive, this primitive can be used to manage antenna diversity. During transmit, this primitive can be use to select a transmit antenna. This primitive will also be used during CCA.

14.5.5.5.4 Effect of receipt

The receipt of this primitive by the PMD entity will cause the radio to select the antenna specified.

14.5.5.6 PMD_TXPWRLVL.request

14.5.5.6.1 Function

This primitive defines the power level the PMD entity will use to transmit data.

14.5.5.6.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_TXPWRLVL.request (TXPOWER_LEVEL)

The TXPOWER_LEVEL parameter can be one of the values listed in Table 35.

Table 35—Transmit power levels

TXPWR_LEVEL	Level description
LEVEL1	Defined as TxPowerLevel1 in MIB
LEVEL2	Defined as TxPowerLevel2 in MIB
LEVEL3	Defined as TxPowerLevel3 in MIB
LEVEL4	Defined as TxPowerLevel4 in MIB
LEVEL5	Defined as TxPowerLevel5 in MIB
LEVEL6	Defined as TxPowerLevel6 in MIB
LEVEL7	Defined as TxPowerLevel7 in MIB
LEVEL8	Defined as TxPowerLevel8 in MIB

14.5.5.6.3 When generated

This primitive is generated as part of the transmit sequence.

14.5.5.6.4 Effect of receipt

The receipt of this primitive by the PMD entity will cause the transmit power level to be modify.

14.5.5.7 PMD_FREQ.request

14.5.5.7.1 Function

This primitive defines the frequency the PMD entity will use to receive or transmit data. Since changing the radio frequency is not an immediate function, this primitive serves also as an indication of the start of this process. The completion of this process is dictated by other PMD specifications.

14.5.5.7.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_FREQ.request (CHANNEL_ID)

The CHANNEL_ID parameter can be one of the values listed in Table 38, Table 39, Table 40, or Table 41.

14.5.5.7.3 When generated

This primitive is generated by the PLCP whenever a change to a new frequency is required.

14.5.5.7.4 Effect of receipt

The receipt of this primitive by the PMD entity will cause the radio to change to a new frequency defined by the value of the CHNL_ID.

14.5.5.8 PMD_RSSI.indicate

14.5.5.8.1 Function

This primitive transfers a receiver signal strength indication of the physical medium from the PMD sublayer to the PLCP sublayer. This value will be used by the PLCP to perform any diversity or clear channel assessment functions required by the PLCP or other sublayers.

14.5.5.8.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_RSSI.indicate (STRENGTH)

The STRENGTH parameter can be a value from 0 to 15. This parameter is an indication by the PMD sublayer of the magnitude of the energy observed at the selected antenna. This reported value is used to generate the RSSI term in the PHY-RXSTART.ind(RXVECTOR) primitive and might also be used by any diversity function. Since RSSI is only used in a relative manner by the MAC sublayer, this parameter is defined to have no more than 16 values, ranging from 0 through RSSI_Max. The value zero is the weakest signal strength, while RSSI_Max is the strongest signal strength.

14.5.5.8.3 When generated

This primitive is generated continually by the PMD entity to transfer a receive signal strength indication to the PLCP.

14.5.5.8.4 Effect of receipt

The effect of receipt of this primitive by the PLCP is unspecified.

14.5.5.9 PMD_PWRMGMT.request

14.5.5.9.1 Function

This primitive is used by the higher-layer entities to manage or control the power consumption of the PMD when not in use. This allows higher-layer entities to put the radio into a sleep or standby mode when receipt or sending of any data is not expected.

14.5.5.9.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_PWRMGMT.request (MODE)

The MODE parameter can have one of two values: ON or OFF. When the value of the parameter is ON, the PMD entity will enter into a fully functional mode that allows it to send or receive data. When the value of the parameter is OFF, the PMD entity will place itself in a standby or power-saving mode. In the low-power mode, the PMD entity is not expected to be able to perform any request by the PLCP, nor is it expected to indicate any change in PMD state or status.

14.5.5.9.3 When generated

This primitive is delivered by the PLCP but actually is generated by a higher layer management entity.

14.5.5.9.4 Effect of receipt

Upon receipt of this primitive, the PMD entity will enter a fully functional or low power consumption state depending on the value of the primitive's parameter.

14.6 FHSS PMD sublayer, 1.0 Mbit/s

14.6.1 1 Mbit/s PMD operating specifications, general

In general, the PMD accepts convergence layer service primitives and provides the actual means by which the signals required by these primitives are imposed onto the medium. In the PMD sublayer at the receiver, the process is reversed. The combined function of the transmitting and receiving PMD sublayers results in a data stream, timing information, and receive parameter information being delivered to the receiving convergence sublayer.

14.6.2 Regulatory requirements

Wireless LANs implemented in accordance with this standard are subject to equipment certification and operating requirements established by regional and national regulatory administrations. The PMD specification establishes minimum technical requirements for interoperability, based upon established regulations for

Europe, Japan, and North America at the time this standard was issued. These regulations are subject to revision, or may be superseded. Requirements that are subject to local geographic regulations are annotated within the PMD specification. Regulatory requirements that do not affect interoperability are not addressed within this standard. Implementors are referred to the following regulatory sources for further information. Operation in countries within Europe, or other regions outside Japan or North America, may be subject to additional or alternative national regulations.

The documents listed below specify the current regulatory requirements for various geographic areas at the time the standard was developed. They are provided for information only, and are subject to change or revision at any time.

Geographic area	Approval standards	Documents	Approval authority
Europe	European Telecommunications Standards Institute (ETSI)	ETS 300-328, ETS 300-339	National type approval authorities
France	Règle technique applicable aux équipements radioélectriques de transmission de données à large bande fonctionnant dans la bande de fréquences à 2,4 GHz et utilisant la technique de l'étalement de spectre (Édition février 1995)	SP/DGPT/ATAS/23, ETS 300-328, ETS 300-339	Direction Générale des Postes et Télécommunications (DGPT)
Japan	Association of Radio Industries and Businesses (ARIB)	RCR STD-33A	Ministry of Telecommunications (MKK)
North America Canada USA	 Industry Canada (IC) Federal Communications Commission (FCC)	 GL36 CFR47, Part 15, Sections 15.205, 15.209, 15.247	 IC FCC
Spain	Supplemento Del Numero 164 Del Boletin Oficial Del Estado (Published 10 July 1991, Revised 25 June 1993)	ETS 300-328, ETS 300-339	Cuadro Nacional De Atribucion De Frecuesias

14.6.3 Operating frequency range

A conformant PMD implementation shall be able to select the carrier frequency (F_c) from the full geographic-specific set of available carrier frequencies. Table 36 summarizes these frequencies for a number of geographic locations.

Table 36—Operating frequency range

Lower Limit	Upper limit	Regulatory range	Geography
2.402 GHz	2.480 GHz	2.400–2.4835 GHz	North America
2.402 GHz	2.480 GHz	2.400–2.4835 GHz	Europe[a]
2.473 GHz	2.495 GHz	2.471–2.497 GHz	Japan
2.447 GHz	2.473 GHz	2.445–2.475 GHz	Spain
2.448 GHz	2.482 GHz	2.4465–2.4835 GHz	France
NOTE—The frequency ranges in this table are subject to the geographic specific regulatory authorities.			

[a]Excluding Spain and France.

14.6.4 Number of operating channels

The number of transmit and receive frequency channels used for operating the PMD entity is 79 for the US and Europe, and 23 for Japan. Table 37 summarizes these frequencies for a number of geographic locations. This is more fully defined in Table 38 through Table 41 of 14.6.5.

Table 37—Number of operating channels

Minimum	Hopping set	Geography
75	79	North America
20	79	Europe[a]
Not applicable	23	Japan
20	27	Spain
20	35	France
NOTE—The number of required hopping channels is subject to the geographic specific regulatory authorities.		

[a]Excluding Spain and France.

14.6.5 Operating channel center frequency

The channel center frequency is defined in sequential 1.0 MHz steps beginning with the first channel, channel 2.402 GHz for the USA and Europe excluding Spain and France, as listed in Table 38. The channel centers for Japan, starting at 2.473 GHz with 1 MHz increments, are listed in Table 39. The channel centers for Spain and France are listed in Table 40 and Table 41, respectively.

Table 38—Requirements in North America and Europe (excluding Spain and France; values specified in GHz)

Channel #	Value	Channel #	Value	Channel #	Value
2	2.402	28	2.428	54	2.454
3	2.403	29	2.429	55	2.455
4	2.404	30	2.430	56	2.456
5	2.405	31	2.431	57	2.457
6	2.406	32	2.432	58	2.458
7	2.407	33	2.433	59	2.459
8	2.408	34	2.434	60	2.460
9	2.409	35	2.435	61	2.461
10	2.410	36	2.436	62	2.462
11	2.411	37	2.437	63	2.463
12	2.412	38	2.438	64	2.464

Table 38—Requirements in North America and Europe (excluding Spain and France; values specified in GHz) *(continued)*

Channel #	Value	Channel #	Value	Channel #	Value
13	2.413	39	2.439	65	2.465
14	2.414	40	2.440	66	2.466
15	2.415	41	2.441	67	2.467
16	2.416	42	2.442	68	2.468
17	2.417	43	2.443	69	2.469
18	2.418	44	2.444	70	2.470
19	2.419	45	2.445	71	2.471
20	2.420	46	2.446	72	2.472
21	2.421	47	2.447	73	2.473
22	2.422	48	2.448	74	2.474
23	2.423	49	2.449	75	2.475
24	2.424	50	2.450	76	2.476
25	2.425	51	2.451	77	2.477
26	2.426	52	2.452	78	2.478
27	2.427	53	2.453	79	2.479
—	—	—	—	80	2.480

Table 39—Requirements in Japan (values specified in GHz)

Channel #	Value	Channel #	Value	Channel #	Value
73	2.473	81	2.481	89	2.489
74	2.474	82	2.482	90	2.490
75	2.475	83	2.483	91	2.491
76	2.476	84	2.484	92	2.492
77	2.477	85	2.485	93	2.493
78	2.478	86	2.486	94	2.494
79	2.479	87	2.487	95	2.495
80	2.480	88	2.488	—	—

Table 40—Requirements in Spain (values specified in GHz)

Channel #	Value	Channel #	Value	Channel #	Value
47	2.447	56	2.456	65	2.465
48	2.448	57	2.457	66	2.466
49	2.449	58	2.458	67	2.467
50	2.450	59	2.459	68	2.468
51	2.451	60	2.460	69	2.469
52	2.452	61	2.461	70	2.470
53	2.453	62	2.462	71	2.471
54	2.454	63	2.463	72	2.472
55	2.455	64	2.464	73	2.473

Table 41—Requirements in France (values specified in GHz)

Channel #	Value	Channel #	Value	Channel #	Value
48	2.448	60	2.460	72	2.472
49	2.449	61	2.461	73	2.473
50	2.450	62	2.462	74	2.474
51	2.451	63	2.463	75	2.475
52	2.452	64	2.464	76	2.476
53	2.453	65	2.465	77	2.477
54	2.454	66	2.466	78	2.478
55	2.455	67	2.467	79	2.479
56	2.456	68	2.468	80	2.480
57	2.457	69	2.469	81	2.481
58	2.458	70	2.470	82	2.482
59	2.459	71	2.471	—	—

14.6.6 Occupied channel bandwidth

Occupied channel bandwidth shall meet all applicable local geographic regulations for 1 MHz channel spacing. The rate at which the PMD entity will hop is governed by the MAC. The hop rate is an attribute with a maximum dwell time subject to local geographic regulations.

14.6.7 Minimum hop rate

The minimum hop rate shall be governed by the regulatory authorities.

14.6.8 Hop sequences

The hopping sequence of an individual PMD entity is used to create a pseudorandom hopping pattern utilizing uniformly the designated frequency band. Sets of hopping sequences are used to co-locate multiple PMD entities in similar networks in the same geographic area and to enhance the overall efficiency and throughput capacity of each individual network.

An FH pattern, F_x, consists of a permutation of all frequency channels defined in Table 38 and Table 39. For a given pattern number, x, the hopping sequence can be written as follows:

$$F_x = \{f_x(1), f_x(2), \ldots f_x(p)\}$$

where

$f_x(i)$ is the channel number (as defined in 14.6.4) for i^{th} frequency in x^{th} hopping pattern;
p is the number of frequency channels in hopping pattern (79 for North America and most of Europe, 23 for Japan, 27 for France, 35 for Spain)

Given the hopping pattern number, x, and the index for the next frequency, i (in the range 1 to p), the channel number shall be defined to be as follows:

$f_x(I) = [b(i) + x] \bmod (79) + 2$ in North America and most of Europe, with $b(i)$ defined in Table 42.
$= [(i-1) \times x] \bmod (23) + 73$ in Japan.
$= [b(i) + x] \bmod (27) + 47$ in Spain with $b(i)$ defined in Table 43.
$= [b(i) + x] \bmod (35) + 48$ in France with $b(i)$ defined in Table 44.

Table 42—Base-Hopping sequence *b(i)* for North America and most of Europe

i	*b(i)*	*i*	*b(i)*	*i*	*b(i)*	*i*	*b(i)*	*i*	*b(i)*	*i*	*b(i)*	*i*	*b(i)*	*i*	*b(i)*
1	0	11	76	21	18	31	34	41	14	51	20	61	48	71	55
2	23	12	29	22	11	32	66	42	57	52	73	62	15	72	35
3	62	13	59	23	36	33	7	43	41	53	64	63	5	73	53
4	8	14	22	24	72	34	68	44	74	54	39	64	17	74	24
5	43	15	52	25	54	35	75	45	32	55	13	65	6	75	44
6	16	16	63	26	69	36	4	46	70	56	33	66	67	76	51
7	71	17	26	27	21	37	60	47	9	57	65	67	49	77	38
8	47	18	77	28	3	38	27	48	58	58	50	68	40	78	30
9	19	19	31	29	37	39	12	49	78	59	56	69	1	79	46
10	61	20	2	30	10	40	25	50	45	60	42	70	28	—	—

Table 43—Base-Hopping sequence *b(i)* for Spain

i	*b(i)*	*i*	*b(i)*	*i*	*b(i)*
1	13	10	19	19	14
2	4	11	8	20	1
3	24	12	23	21	20
4	18	13	15	22	7
5	5	14	22	23	16
6	12	15	9	24	2
7	3	16	21	25	11
8	10	17	0	26	17
9	25	18	6	27	26

Table 44—Base-Hopping sequence *b(i)* for France

i	*b(i)*	*i*	*b(i)*	*i*	*b(i)*
1	17	13	31	25	15
2	5	14	20	26	3
3	18	15	29	27	11
4	32	16	22	28	30
5	23	17	12	29	24
6	7	18	6	30	9
7	16	19	28	31	27
8	4	20	14	32	19
9	13	21	25	33	2
10	33	22	0	34	21
11	26	23	8	35	34
12	10	24	1	—	—

The sequences are designed to ensure some minimum distance in frequency between contiguous hops. The minimum hop size is 6 MHz for North America and Europe, including Spain and France, and 5 MHz for Japan.

The hopping pattern numbers x are divided into three sets. The sets are designed to avoid prolonged collision periods between different hopping sequences in a set. Hopping sequence sets contain 26 sequences for North America and Europe, and 4 sequences per set for Japan:

For North America and most of Europe:

$x = \{0,3,6,9,12,15,18,21,24,27,30,33,36,39,42,45,48,51,54,57,60,63,66,69,72,75\}$ Set 1
$x = \{1,4,7,10,13,16,19,22,25,28,31,34,37,40,43,46,49,52,55,58,61,64,67,70,73,76\}$ Set 2

$x = \{2,5,8,11,14,17,20,23,26,29,32,35,38,41,44,47,50,53,56,59,62,65,68,72,74,77\}$ Set 3

For Japan:

$x = \{6,9,12,15\}$ Set 1
$x = \{7,10,13,16\}$ Set 2
$x = \{8,11,14,17\}$ Set 3

For Spain:

$x = \{0,3,6,9,12,15,18,21,24\}$ Set 1
$x = \{1,4,7,10,13,16,19,22,25\}$ Set 2
$x = \{2,5,8,11,14,17,20,23,26\}$ Set 3

For France:

$x = \{0,3,6,9,12,15,18,21,24,27,30\}$ Set 1
$x = \{1,4,7,10,13,16,19,22,25,28,31\}$ Set 2
$x = \{2,5,8,11,14,17,20,23,26,29,32\}$ Set 3

The three sets of hopping sequences for North America and most of Europe, of 26 patterns each, are listed Tables B.1, B.2, and B.3 in Annex B. Similarly, there are three sets for Japan of four patterns each. The three sets for Spain have nine patterns each. The three sets for France have 11 patterns each. The channel numbers listed under each pattern refer to the actual frequency values listed in Table 38 and Table 39.

14.6.9 Unwanted emissions

Conformant PMD implementations of this FHSS standard shall limit the emissions that fall outside of the operating frequency range, defined in Table 36 of 14.6.3, to the geographically applicable limits.

14.6.10 Modulation

The minimum set of requirements for a PMD to be compliant with the IEEE 802.11 FHSS PHY shall be as follows.

The PMD shall be capable of operating using two-level Gaussian frequency shift key (GFSK) modulation with a nominal bandwidth bit-period (BT)=0.5. The PMD shall accept symbols from the set {{1},{0}}from the PLCP. The symbol {1} shall be encoded with a peak deviation of $(+f_d)$, giving a peak transmit frequency of (F_c+f_d), which is greater than the carrier center frequency (F_c). The symbol {0} shall be encoded with a peak frequency deviation of $(-f_d)$, giving a peak transmit frequency of (F_c-f_d).

An incoming bit stream at 1 Mbit/s will be converted to symbols at $Fclk = 1$ Msymbols/s, as shown in Table 45.

Table 45—Symbol encoding into carrier deviation (1 Mbit/s, 2-GFSK)

Symbol	Carrier deviation
1	1/2 × h2 × Fclk
0	–1/2 × h2 × Fclk
NOTE—These deviation values are measured using the center symbol of 7 consecutive symbols of the same value. The instantaneous deviation will vary due to Gaussian pulse shaping.	

The deviation factor h2 for 2GFSK (measured as difference between frequencies measured in the middle of 0000 and 1111 patterns encountered in the SFD, divided by 1 MHz) will nominally be 0.32.

The minimum frequency deviation, as shown in Figure 84, shall be greater than 110 kHz relative to the nominal center frequency F_c. F_d is the average center frequency of the last 8 bits of the Preamble Sync field, measured as the deviation at the midsymbol. Midsymbol is defined as the point that is midway between the zero crossings derived from a best fit to the last 8 bits of the Sync field. Maximum deviation is not specified, but modulation is subject to the occupied bandwidth limits of 14.6.5.

The zero crossing error shall be less than ±1/8 of a symbol period. The zero crossing error is the time difference between the ideal symbol periods and measured crossings of F_c. This is illustrated in Figure 84.

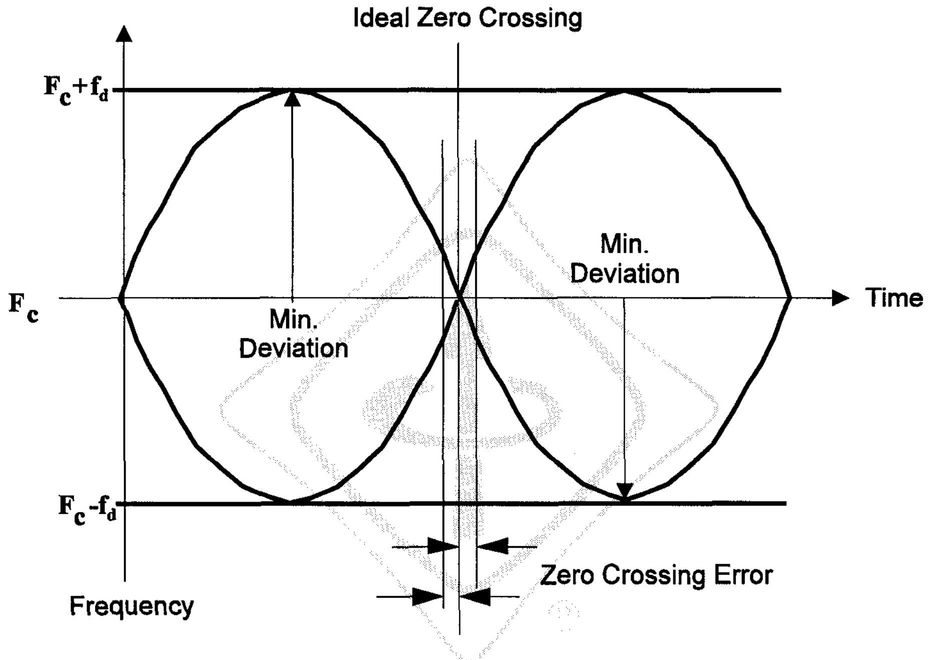


Figure 84—Transmit modulation mask

14.6.11 Channel data rate

A compliant IEEE 802.11 FHSS PMD shall be capable of transmitting and receiving at a nominal data rate of 1.0 Mbit/s ± 50 ppm.

14.6.12 Channel switching/settling time

The time to change from one operating channel frequency, as specified in 14.6.3, is defined as 224 μs. A conformant PMD meets this switching time specification when the operating channel center frequency has settled to within ±60 kHz of the nominal channel center frequency as outlined in 14.6.3.

14.6.13 Receive to transmit switch time

The maximum time for a conformant PMD to switch the radio from the receive state to the transmit state and place the start of the first bit on the air shall be 19 μs. At the end of this 19 μs, the RF carrier shall be within the nominal transmit power level range, and within the described modulation specifications.

14.6.14 PMD transmit specifications

The following portion of this subclause describes the transmit functions and parameters associated with the PMD sublayer. In general, these are specified by primitives from the PLCP, and the transmit PMD entity provides the actual means by which the signals required by the PLCP primitives are imposed onto the medium.

14.6.14.1 Nominal transmit power

The nominal transmit power of a frame is defined as the power averaged between the start of the first symbol in the PLCP Header to the end of the last symbol in the PLCP Header. When in the transmit state, the transmit power shall be within 2 dB of the nominal transmit power from the start of the Preamble SYNC field to the last symbol at the end of the frame.

14.6.14.2 Transmit power levels

Unless governed by more stringent local geographic regulations, the radiated emissions from compliant devices shall meet IEEE Std C95.1-1991 limits for controlled or uncontrolled environments, in accordance with their intended usage. In addition, all conformant PMD implementations shall support at least one power level with a minimum equivalent isotropically radiated power (EIRP) of 10 mW.

14.6.14.3 Transmit power level control

If a conformant PMD implementation has the ability to transmit in a manner that results in the EIRP of the transmit signal exceeding the level of 100 mW, at least one level of transmit power control shall be implemented. This transmit power control shall be such that the level of the emission is reduced to a level at or below 100 mW under the influence of said power control.

14.6.14.4 Transmit spectrum shape

Within the operational frequency band the transmitter shall pass a spectrum mask test. The duty cycle between Tx and Rx is nominally 50% and the transmit frame length is nominally 400 μs. The adjacent channel power is defined as the sum of the power measured in a 1 MHz band. For a pseudorandom data pattern, the adjacent channel power shall be a function of the offset between channel number N and the assigned transmitter channel M, where M is the actual transmitted center frequency and N a channel separated from it by an integer number of MHz.

Channel offset:

$|N-M|=2$ –20 dBm or –40 dBc, whichever is the lower power.

$|N-M|\geq 3$ –40 dBm or –60 dBc, whichever is the lower power.

The levels given in dBc are measured relative to the transmitter power measured in a 1 MHz channel centered on the transmitter center frequency. The adjacent channel power and the transmitter power for this subclause of the specification shall be measured with a resolution bandwidth of 100 kHz, a video bandwidth of 300 kHz, and a peak detector, and with the measurement device set to maximum hold.

For any transmit center frequency M, two exceptions to the spectrum mask requirements are permitted within the operational frequency band, provided the exceptions are less than –50 dBc, where each offset channel exceeded counts as a separate exception. An exception occurs when the total energy within a given 1 MHz channel as defined in 14.6.5 exceeds the levels specified above.

14.6.14.5 Transmit center frequency tolerance

The PMD transmit center frequency shall be within ±60 kHz of the nominal center frequency as specified in 14.6.5.

14.6.14.6 Transmitter ramp periods

The transmitter shall go from off to within 2 dB of the nominal transmit power in 8 μs or less. The transmitter shall go from within 2 dB of the nominal transmit power to off (less than –50 dBm) in 8 μs or less.

14.6.15 PMD receiver specifications

The following portion of this subclause describes the receive functions and parameters associated with the PMD sublayer. In general, these are specified by primitives from the PLCP. The Receive PMD entity provides the actual means by which the signals required by the PLCP primitives are recovered from the medium. The PMD sublayer monitors signals on the medium and will return symbols from the set {{1},{0}} to the PLCP sublayer.

14.6.15.1 Input signal range

The PMD shall be capable of recovering a conformant PMD signal from the medium, as described in related subclauses, with a frame error ratio (FER) ≤3% for PSDUs of 400 octets generated with pseudorandom data, for receiver input signal levels in the range from –20 dBm to the receiver sensitivity (as specified in 14.6.15.4), across the frequency band of operation.

14.6.15.2 Receive center frequency acceptance range

An IEEE 802.11 FHSS compliant PMD shall meet all specifications with an input signal having a center frequency range of ±60 kHz from nominal.

14.6.15.3 CCA power threshold

In the presence of any IEEE 802.11-compliant 1 Mbit/s FH PMD signal above –85 dBm that starts synchronously with respect to slot times as specified in 14.3.3.2.1, the PHY shall signal busy, with a 90% probability of detection, during the preamble within the CCA assessment window. In the presence of any IEEE 802.11-compliant 1 Mbit/s FH PMD signal above –85 dBm that starts asynchronously with respect to slot times as specified in 14.3.3.2.1, the PHY shall signal busy, with a 70% probability of detection, during the preamble within the CCA window. In the presence of any IEEE 802.11 compliant 1 Mbit/s FH PMD signal above –65 dBm, the PHY shall signal busy, with a 70% probability of detection, during random data within the CCA window. This specification applies to a PMD operating with a nominal EIRP of < 100 mW. A compliant PMD operating at a nominal output power greater than 100 mW shall use the following equation to define the CCA threshold, where P_t represents transmit power.

$$\text{CCA threshold (preamble)} = -85\ \text{dBm} - \left[5 \times \log_{10}\left(\frac{P_t}{100\ \text{mW}}\right)\right]\ \text{dBm}$$

CCA threshold (random data) = CCA threshold (preamble) + 20 dB

14.6.15.4 Receiver sensitivity

The sensitivity is defined as the minimum signal level required for an FER of 3% for PSDUs of 400 octets generated with pseudorandom data. The sensitivity shall be less than or equal to –80 dBm. The reference sensitivity is defined as –80 dBm for the 1 Mbit/s FH PHY specifications.

14.6.15.5 Intermodulation

Intermodulation protection (IMp) is defined as the ratio of the minimum amplitude of one of two equal interfering signals to the desired signal amplitude, where the interfering signals are spaced 4 and 8 MHz removed from the center frequency of the desired signal, both on the same side of center frequency. The IMp protection ratio is established at the interfering signal level that causes the FER of the receiver to be increased to 3% for PSDUs of 400 octets generated with pseudorandom data, when the desired signal is –77 dBm. Each interfering signal is modulated with the FH PMD modulation uncorrelated in time to each other or the desired signal. The PMD shall have the IMp for the interfering signal at 4 and 8 MHz be ≥30 dB.

14.6.15.6 Desensitization

Desensitization (Dp) is defined as the ratio to measured sensitivity of the minimum amplitude of an interfering signal that causes the FER at the output of the receiver to be increased to 3% for PSDUs of 400 octets generated with pseudorandom data, when the desired signal is –77 dBm. The interfering signal shall be modulated with the FHSS PMD modulation uncorrelated in time to the desired signal. The minimum Dp shall be as given in Table 46. The spectral purity of the interferer shall be sufficient to ensure that the measurement is limited by the receiver performance.

Table 46—1 Mbit/s Dp

Interferer frequency[a]	Dp minimum
$M = N \pm 2$	30 dB
$M = N \pm 3$ or more	40 dB

[a]Where *M* is the interferer frequency and *N* is the desired channel frequency.

14.6.15.7 Receiver radiation

The signal leakage when receiving shall not exceed –50 dBm EIRP in the operating frequency range.

14.6.16 Operating temperature range

Two temperature ranges for full operation compliance to the FH PHY are specified. Type 1 is defined as 0 °C to 40 °C and is designated for office environments. Type 2 is defined as –30 °C to +70 °C and is designated for industrial environments.

14.7 FHSS PMD sublayer, 2.0 Mbit/s

14.7.1 Overview

This subclause details the RF specification differences of the optional 2 Mbit/s operation from the baseline 1 Mbit/s PMD as contained in 14.6. Unless otherwise specified in this subclause, the compliant PMD shall also meet all requirements of 14.6 when transmitting at 2 Mbit/s. When implementing the 2 Mbit/s option, the preamble and PHY Header shall be transmitted at 1 Mbit/s. STAs implementing the 2 Mbit/s option shall also be capable of transmitting and receiving PPDUs at 1 Mbit/s.

14.7.2 Four-Level GFSK modulation

For an FHSS 2 Mbit/s PMD, the modulation scheme shall be four-level Gaussian frequency shift keying (4GFSK), with a nominal symbol-period bandwidth product (BT) of 0.5. The four-level deviation factor, defined as the frequency separation of adjacent symbols divided by symbol rate, h4, shall be related to the deviation factor of the 2GFSK modulation, h2, by the following equation:

$$h4/h2 = 0.45 \pm 0.01$$

An incoming bit stream at 2 Mbit/s will be converted to 2-bit words or symbols, with a rate of Fclk = 1 *M*symbol/s. The first received bit will be encoded as the LMB of the symbol in Table 47. The bits will be encoded into symbols as shown in Table 47.

Table 47—Symbol encoding into carrier deviation

1 Mbit/s, 2GFSK	
Symbol	**Carrier deviation**
1	$1/2 \times h2 \times Fclk$
0	$-1/2 \times h2 \times Fclk$
2 Mbit/s, 4GFSK	
Symbol	**Carrier deviation**
10	$3/2 \times h4 \times Fclk$
11	$1/2 \times h4 \times Fclk$
01	$-1/2 \times h4 \times Fclk$
00	$-3/2 \times h4 \times Fclk$
NOTE—These deviation values are measured using the center symbol of 7 consecutive symbols of the same value. The instantaneous deviation will vary due to Gaussian pulse shaping.	

The deviation factor h2 for 2GFSK (measured as the difference between frequencies measured in the middle of 0000 and 1111 patterns encountered in the SFD, divided by 1 MHz) will nominally be 0.32. The deviation factor h2 will be no less than 0.30 (with maximum dictated by regulatory bandwidth requirement). Accordingly, h4 (measured as a difference between the outermost frequencies, divided by 3, divided by 1 MHz) is nominally $0.45 \times 0.32 = 0.144$, and it will be no less than $0.45 \times 0.3 = 0.135$.

The modulation error shall be less than ±15 kHz at the midsymbol time for 4GFSK, from the frequency deviations specified above, for a symbol surrounded by identical symbols, and less than ±25 kHz for any symbol. The deviation is relative to the actual center frequency of the RF carrier. For definition purposes, the actual center frequency is the midfrequency between symbols 11 and 01. The actual center frequency shall be within ±60 kHz of the nominal channel center frequency defined in 14.6.5 and shall not vary by more than ±10 kHz/ms, from the start to end of the PPDU. The peak-to-peak variation of the actual center frequency over the PPDU shall not exceed 15 kHz. Symbols and terms used within this subclause are illustrated in Figure 85.

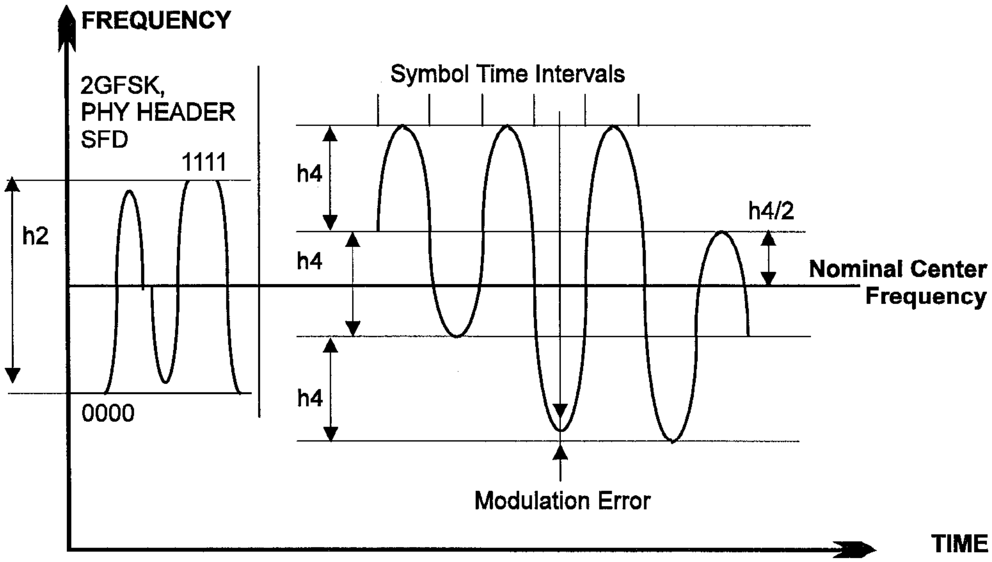


Figure 85—Four-Level GFSK transmit modulation

14.7.2.1 Frame structure for HS FHSS PHY

The high rate FHSS PPDU consists of PLCP Preamble, PLCP Header, and whitened PSDU. The PLCP Preamble and PLCP Header format are identical to 1 Mbit/s PHY, as described in 14.3.2. The whitened PSDU is transmitted in 2GFSK, 4GFSK, or potentially a higher-rate format, according to the rate chosen. The rate is indicated in a 3-bit field in a PLCP Header, having a value of 1 or 2 bits/symbol (or Mbit/s).

The PPDU is transmitted as four-level symbols, with thc amount determined by number_of_symbols = (number_of_PSDU_octets × 8)/rate.

The input bits are scrambled according to the method in 14.3.2.3.

The scrambled bit stream is divided into groups of rate (1 or 2) consecutive bits. The bits are mapped into symbols according to Table 47.

A bias suppression algorithm is applied to the resulting symbol stream. The bias suppression algorithm is defined in 14.3.2.3, Figure 72, and Figure 75. A polarity control symbol is inserted prior to each block of 32 symbols (or less for the last block). The polarity control signals are 4GFSK symbols 10 or 00. The algorithm is equivalent to the case of 2GFSK, with the polarity symbol 2GFSK "1" replaced with 4GFSK symbol "10," and the 2GFSK polarity symbol "0" replaced with a 4GFSK symbol "00."

14.7.3 Channel data rate

The data rate for the whitened PSDU at the optional rate shall be 2.0 Mbit/s ± 50 ppm.

14.7.3.1 Input dynamic range

The PMD shall be capable of recovering a conformant PMD signal from the medium, as described in related subclauses, with an FER ≤3% for PSDUs of 400 octets generated with pseudorandom data, for receiver input signal levels in the range from –20 dBm to the receiver sensitivity (as specified in 14.7.3.2), across the frequency band of operation.

14.7.3.2 Receiver sensitivity

The sensitivity is defined as the minimum signal level required for an FER of 3% for PSDUs of 400 octets generated with pseudorandom data. The sensitivity shall be less than or equal to –75 dBm. The reference sensitivity is defined as –75 dBm for the 2 Mbit/s FH PHY specifications.

14.7.3.3 IMp

IMp is defined as the ratio to –77 dBm of the minimum amplitude of one of the two equal level interfering signals at 4 and 8 MHz removed from center frequency, both on the same side of center frequency, that cause the FER of the receiver to be increased to 3% for PSDUs of 400 octets generated with pseudorandom data, when the desired signal is –72 dBm (3 dB above the specified sensitivity specified in 14.7.3.2). Each interfering signal is modulated with the FH 1 Mbit/s PMD modulation uncorrelated in time to each other or the desired signal. The FHSS optional 2 Mbit/s rate IMp shall be ≥25 dB.

14.7.3.4 Dp

Dp is defined as the ratio to measured sensitivity of the minimum amplitude of an interfering signal that causes the FER of the receiver to be increased to 3% for PSDUs of 400 octets generated with pseudorandom data, when the desired signal is –72 dB (3 dB above sensitivity specified in 14.7.3.2). The interfering signal shall be modulated with the FHSS PMD modulation uncorrelated in time to the desired signal. The minimum Dp shall be as given in Table 48.

Table 48—2 Mbit/s Dp

Interferer frequency[a]	DP minimum
$M = N \pm 2$	20 dB
$M = N \pm 3$ or more	30 dB

[a]Where M is the interferer frequency and N is the desired channel frequency.

14.8 FHSS PHY management information base (MIB)

14.8.1 Overview

The following is the MIB for the FHSS PHY.

14.8.2 FH PHY attributes

This subclause defines the attributes for the FHSS MIB. Table 49 lists these attributes and the default values. Following the table is a description of each attribute.

Table 49—FHSS PHY attributes

Attribute	Default value	Operational semantics	Operational behavior
aPHYType	FHSS = X'01'	Static	Identical for all FH PHYs
aRegDomainsSupported	FCC = X'10' IC = X'20' ETSI = X'30' Spain = X'31' France = X'32' MKK = X'40'	Static	Implementation dependent
aCurrentRegDomain	X'00'	Dynamic LME	Implementation dependent
aSlotTime	50 µs	Static	Identical for all FH PHYs
aCCATime	27 µs	Static	Identical for all FH PHYs
aRxTxTurnaroundTime	20 µs	Static	Identical for all FH PHYs
aTxPLCPDelay	1 µs	Static	Identical for all FH PHYs
aRxTxSwitchTime	10 µs	Static	Identical for all FH PHYs
aTxRampOnTime	8 µs	Static	Identical for all FH PHYs
aTxRFDelay	1 µs	Static	Identical for all FH PHYs
aSIFSTime	28 µs	Static	Identical for all FH PHYs
aRxRFDelay	4 µs	Static	Identical for all FH PHYs
aRxPLCPDelay	2 µs	Static	Identical for all FH PHYs
aMACProcessingDelay	2 µs	Static	Identical for all FH PHYs
aTxRampOffTime	8 µs	Static	Identical for all FH PHYs
aPreambleLength	96 µs	Static	Identical for all FH PHYs
aPLCPHdrLength	32 µs	Static	Identical for all FH PHYs
aMPDUDurationFactor	1.03125	Static	Identical for all FH PHYs
aAirPropagationTime	1 µs	Static	Identical for all FH PHYs
aTempType	Type 1 = X'01' Type 2 = X'02' Type 3 = X'03'	Static	Implementation dependent
aCWmin	15	Static	Identical for all FH PHYs
aCWmax	1023	Static	Identical for all FH PHYs
aSupportedDataRatesTX	1 Mbit/s = X'02' mandatory 2 Mbit/s = X'04' optional	Static	Identical for all FH PHYs
aSupportedDataRatesRX	1 Mbit/s = X'02' mandatory 2 Mbit/s = X'04' optional	Static	Identical for all FH PHYs
aMPDUMaxLength	4095 octets	Static	Identical for all FH PHYs
aSupportedTxAntennas	Ant 1 = X'01' Ant 2 = X'02' Ant 3 = X'03' Ant n = n	Static	Implementation dependent
aCurrentTxAntenna	Ant 1 = default	Dynamic LME	Implementation dependent

Table 49—FHSS PHY attributes *(continued)*

Attribute	Default value	Operational semantics	Operational behavior
aSupportedRxAntennas	Ant 1 = X'01' Ant 2 = X'02' Ant 3 = X'03' Ant n = n	Static	Implementation dependent
aDiversitySupport	Available = X'01' Not avail. = X'02' Control avail. = X'03'	Static	Implementation dependent
aDiversitySelectionRx	Ant 1 = X'01' Ant 2 = X'02' Ant 3 = X'03' Ant 4 = X'04' Ant 5 = X'05' Ant 6 = X'06' Ant 7 = X'07' Ant 8 = X'08'	Dynamic LME	Implementation dependent
aNumberSupportedPowerLevels	Lvl1 = X'01' Lvl2 = X'02' Lvl3 = X'03' Lvl4 = X'04' Lvl5 = X'05' Lvl6 = X'06' Lvl7 = X'07' Lvl8 = X'08'	Static	Implementation dependent
aTxPowerLevel1	Factory def. default	Static	Implementation dependent
aTxPowerLevel2	Factory def.	Static	Implementation dependent
aTxPowerLevel3	Factory def.	Static	Implementation dependent
aTxPowerLevel4	Factory def.	Static	Implementation dependent
aTxPowerLevel5	Factory def.	Static	Implementation dependent
aTxPowerLevel6	Factory def.	Static	Implementation dependent
aTxPowerLevel7	Factory def.	Static	Implementation dependent
aTxPowerLevel8	Factory def.	Static	Implementation dependent
aCurrentTxPowerLevel	TxPowerLevel1	Dynamic LME	Implementation dependent
aHopTime	224 µs	Static	Identical for all FH PHYs
aCurrentChannelNumber	X'00'	Dynamic PLME	
aMaxDwellTime	390 TU	Static	Regulatory domain dependent
aCurrentSet	X'00'	Dynamic PLME	
aCurrentPattern	X'00'	Dynamic PLME	
aCurrentIndex	X'00'	Dynamic PLME	
aCurrentPowerState	X'01' off X'02' on	Dynamic LME	

NOTE—The column titled "Operational semantics" contains two types: static and dynamic. Static MIB attributes are fixed and cannot be modified for a given PHY implementation. MIB attributes defined as dynamic can be modified by some management entity. Whenever an attribute is defined as dynamic, the column also shows which entity has control over the attribute. LME refers to the MAC sublayer management entity (MLME), while PHY refers to the physical layer management entity (PLME).

14.8.2.1 FH PHY attribute definitions

14.8.2.1.1 aPHYType

The aPHYType is FHSS. The LME uses this attribute to determine what PLCP and PMD is providing services to the MAC. It also is used by the MAC to determine what MAC sublayer management state machines must be invoke to support the PHY. The value of this attribute is defined as the integer 01 to indicate the FHSS PHY.

14.8.2.1.2 aRegDomainsSupported

Operational requirements for FHSS PHY are defined by agencies representing certain geographical regulatory domains. These regulatory agencies may define limits on various parameters that differ from region to region. This parameters may include aTxPowerLevels, and aMaxDwellTime, as well as the total number of frequencies in the hopping pattern. The values shown in Table 50 indicate regulatory agencies supported by this document.

Table 50—Regulatory domain codes

Code point	Regulatory agency	Region
X'10'	FCC	United States
X'20'	IC	Canada
X'30'	ETSI	Most of Europe
X'31'	Spain	Spain
X'32'	France	France
X'40'	MKK	Japan
X'00'	Null terminator	—

Since a PLCP and PMD might be designed to support operation in more than one regulatory domain, this attribute can actually represent a list of agencies. This list can be one or more of the above agencies and must be terminated using the null terminator. Upon activation of the PLCP and PMD, the information in this list must be used to set the value of the aCurrentRegDomain attribute.

14.8.2.1.3 aCurrentRegDomain

The aCurrentRegDomain attribute for the FHSS PHY is defined as the regulatory domain under which the PMD is currently operating. This value must be one of the values listed in the aRegDomainsSupported list. This MIB attribute is managed by the LME.

14.8.2.1.4 aSlotTime

The aSlotTime is a PHY dependent attribute used by the MAC sublayer to determine the PIFS and DIFS periods. It is defined using the following equation:

aCCATime + aRxTxTurnaroundTime + aAirPropagationTime + aMACProcessingDelay

For the FHSS PHY, the aCCATime is 27 μs, and the aRxTxTurnaroundTime is 20 μs. The aAirPropagationTime is fixed at 1 μs. The aMACProcessingDelay is nominally 2 μs. The value of this attribute is 50 μs.

14.8.2.1.5 aCCATime

The aCCATime for the FHSS PHY is defined as the time the receiver must use to evaluate the medium at the antenna to determine the state of the channel. This time period for the FHSS PHY is 27 μs. This period includes the aRxRFDelay and the aRxPLCPDelay.

14.8.2.1.6 aRxTxTurnaround Time

The aRxTxTurnaroundTime for the FHSS PHY is defined as the time it takes a STA to place a valid symbol on the medium after a PHY-TXSTART.request. The aRxTxTurnaroundTime is determined using the following equation.

aTxPLCPDelay + aRxTxSwitchTime + aTxRampOnTime + aTxRFDelay

For the FHSS PHY, the aTxPLCPDelay is 1 μs, the aRxTxSwitchTime is 10 μs, the aTxRampOnTime is 8 μs, and the aTxRFDelay is 1 μs, for a total of 20 μs. This is the maximum time for getting valid data on the medium. STAs can use less time but not more than 20 μs.

14.8.2.1.7 aTxPLCPDelay

The aTxPLCPDelay for the FHSS PHY is defined as the delay the PLCP introduces in getting data onto the air in the transmit direction. This value for the FHSS PHY is nominally 1 μs. Implementors may choose to increase or decrease this delay as long as the requirements of aRxTxTurnaroundTime are met.

14.8.2.1.8 aRxTxSwitchTime

The aRxTxSwitchTime for the FHSS PHY is defined as the delay the PMD requires to change from receive to transmit. This value for the FHSS PHY is nominally 10 μs. Implementors may choose to increase or decrease this delay as long as the requirements of aRxTxTurnaroundTime are met.

14.8.2.1.9 aTxRampOnTime

The aTxRampOnTime for the FHSS PHY is defined as the delay the PMD requires to turn on the transmit power amplifier. This value for the FHSS PHY is nominally 8 μs. Implementors may choose to increase or decrease this delay as long as the requirements of aRxTxTurnaroundTime are met.

14.8.2.1.10 aTxRFDelay

The aTxRFDelay for the FHSS PHY is defined as the nominal time in microseconds between the issuance of a PMDDATA.request to the PMD and the start of the corresponding symbol at the air interface. The start of a symbol is defined to be 1/2 symbol period prior to the center of the symbol. This value for the FHSS PHY is nominally 1 μs. Implementors may choose to increase or decrease this delay as long as the requirements of aRxTxTurnaroundTime are met.

14.8.2.1.11 aSIFSTime

The aSIFSTime for the FHSS PHY is defined as the time the MAC and PHY sublayers will require to receive the last symbol of a frame at the air interface, process the frame, and respond with the first symbol of a preamble on the air interface. The aSIFSTime is determined using the following equation:

aRxRFDelay + aRxPLCPDelay + aMACProcessingDelay + aRxTxTurnaroundTime

For the FHSS PHY, the aRxRFDelay is 4 μs, the aRxPLCPDelay is 2 μs, the aMACProcessingDelay is 2 μs, and the aRxTxTurnaroundTime is 20 μs, for a total of 28 μs. This is the nominal value for aSIFSTime. In order to account for variations between implementations, this value has a tolerance as specified in 9.2.3.1.

14.8.2.1.12 aRXRFDelay

The aRxRFDelay for the FHSS PHY is defined as the nominal time in microseconds between the end of a symbol at the air interface to the issuance of a PMDDATA.indicate to the PLCP. The end of a symbol is defined to be 1/2 symbol period after the center of the symbol. This value for the FHSS PHY is nominally 4 μs. Implementors may choose to increase or decrease this delay as long as the requirements of aSIFSTime and aCCATime are met.

14.8.2.1.13 aRxPLCPDelay

The aRxPLCPDelay for the FHSS PHY is defined as the delay the PLCP introduces in the data path between the PMD and the MAC sublayer. This value for the FHSS PHY is nominally 2 μs. Implementors may choose to increase or decrease this delay as long as the requirements of aSIFSTime and aCCATime are met.

14.8.2.1.14 aMACProcessingDelay

The aMACProcessingDelay for the FHSS PHY is defined as the delay between when a PHY-RXEND.indicate is issued by the PHY till a corresponding PHY-TXSTART.request is issued by the MAC. This value for the FHSS PHY is nominally 2 μs. Implementors may choose to increase or decrease this delay as long as the requirements of aSIFSTime are met.

14.8.2.1.15 aTxRampOffTime

The aTxRampOffTime for the FHSS PHY is defined as the delay the PMD requires to turn off the transmit power amplifier. This value for the FHSS PHY is a maximum of 8 μs.

14.8.2.1.16 aPreambleLength

The parameter aPreambleLength defines the time required by the FHSS PHY to transmit the PLCP Preamble. For both the 1 and 2 Mbit/s FHSS PHYs, this value is 96 μs.

14.8.2.1.17 aPLCPHdrLength

The parameter aPLCPHdrLength defines the time required by the FHSS PHY to transmit the PLCP Header. For both the 1 and 2 Mbit/s FHSS PHYs, this value is 32 μs.

14.8.2.1.18 aMPDUDurationFactor

The parameter aMPDUDurationFactor defines the overhead added by the PHY to the PSDU as it is transmitted over the air. For the FHSS PHY, this factor is 1.03125. This factor is calculated as 33/32 to account for the expansion due to the data whitener encoding algorithm. The total time to transmit a PPDU over the air is the following equation rounded up to the next integer microsecond:

$$\text{aPreambleLength} + \text{aPLCPHdrLength} + \text{aMPDUDurationFactor} \times 8 \times \text{PSDU length (octets)/data rate}$$

The total time in microseconds to the beginning of any octet in an PPDU from the first symbol of the preamble can be calculated using the duration factor in the following equation:

$$\text{Truncate}[\text{aPreambleLength} + \text{aPLCPHdrLength} + \text{aMPDUDurationFactor} \times 8 \times N/\text{data rate}] + 1$$

where *N* is the number of octets prior to the desired octet.

14.8.2.1.19 aAirPropagationTime

The parameter aAirPropagationTime is the time it takes a transmitted signal to go from the transmitting STA to the receiving STA. A nominal value of 1 μs has been allocated for this parameter. Variations in the actual propagation time are accounted for in the allowable range of aSIFSTime.

14.8.2.1.20 aTempType

The parameter aTempType defines the temperature range supported by the PHY. Type 1 equipment (X'01') supports a temperature range of 0 °C to 40 °C. Type 2 equipment (X'02') supports a temperature range of –20 °C to +55 °C. Type 3 equipment (X'03') supports a temperature range of –30 °C to +70 °C.

14.8.2.1.21 aCWmin

The parameter aCWmin defines the minimum size of the contention window, in slots. For the FH PHY, this number is 15 decimal.

14.8.2.1.22 aCWmax

The parameter aCWmin defines the maximum size of the contention window, in slots. For the FH PHY, this number is 1023 decimal.

14.8.2.1.23 aCurrentPowerState

The aCurrentPowerState attribute for the FHSS PHY allows the MAC sublayer management entity to control the power state of the PHY. This attribute can be updated using the PLMESET.request. The permissible values are ON and OFF.

14.8.2.1.24 aSupportedDataRatesTX

The aSupportedDataRatesTX attribute for the FHSS PHY is defined as a null terminated list of supported data rates in the transmit mode for this implementation. Table 51 shows the possible values appearing in the list.

Table 51—Supported data rate codes (aSupportedDataRatesTX)

Code point	Data rate
X'02'	1 Mbit/s
X'04'	2 Mbit/s
X'00'	Null terminator

14.8.2.1.25 aSupportedDataRatesRX

The aSupportedDataRatesRX attribute for the FHSS PHY is defined as a null terminated list of supported data rates in the receive mode for this implementation. Table 52 shows the possible values appearing in the list.

Table 52—Supported data rate codes (aSupportedDataRatesRX)

Code point	Data rate
X'02'	1 Mbit/s
X'04'	2 Mbit/s
X'00'	Null terminator

14.8.2.1.26 aMPDUMaxLength

The aMPDUMaximumLength attribute for the FHSS PHY is defined as the maximum PSDU, in octets, that the PHY shall ever be capable of accepting. This value for the FHSS PHY is set at 4095 octets. The recommended value for maximum PSDU length in an FHSS PHY system is 400 octets at 1 Mbit/s and 800 octets at 2 Mbit/s, which corresponds to a frame duration less than 3.5 ms. These values are optimized to achieve high performance in a variety of RF channel conditions, particularly with respect to indoor multipath, channel stability for moving STAs, and interference in the 2.4 GHz band.

14.8.2.1.27 aSupportedTxAntennas

The aSupportedTxAntennas attribute for the FHSS PHY is defined as a null terminated list of antennas that this implementation can use to transmit data. Table 53 shows the possible values appearing in the list, where $N \leq 255$.

Table 53—Number of transmit antennas

Code point	Antenna number
X'01'	Tx Antenna 1
X'02'	Tx Antenna 2
X'03'	Tx Antenna 3
...	...
N	Tx Antenna *N*
X'00'	Null terminator

14.8.2.1.28 aCurrentTxAntenna

The CurrentTxAntenna attribute for the FHSS PHY is used to describe the current antenna the implementation is using for transmission. This value should represent one of the antennas appearing in the SupportedTx-Antennas list.

14.8.2.1.29 aSupportedRxAntenna

The aSupportedRxAntennas attribute for the FHSS PHY is defined as a null terminated list of antennas that this implementation can use to receive data. In the FHSS PHY primitives, one of these values is passed as part of the PHY-RXSTART.indicate to the MAC sublayer for every received packet. Table 54 shows the possible values appearing in the list, where $N \leq 255$.

Table 54—Number of receive antennas

Code point	Antenna number
X'01'	Rx Antenna 1
X'02'	Rx Antenna 2
X'03'	Rx Antenna 3
...	...
N	Rx Antenna *N*
X'00'	Null terminator

14.8.2.1.30 aDiversitySupport

The aDiversitySupport attribute for the FHSS PHY is used to describe the implementation's diversity support. Table 55 shows the possible values appearing in the list.

Table 55—Diversity support codes

Code point	Diversity support
X'01'	Diversity available
X'02'	No diversity
X'03'	Control available

The value X'01' indicates that this implementation uses two or more antennas for diversity. The value X'02' indicates that the implementation has no diversity support. The value X'03' indicates that the choice of antennas used during diversity is programmable. (See 14.8.2.1.31.)

14.8.2.1.31 aDiversitySelectionRx

The aDiversitySelectionRx attribute for the FHSS PHY is a null terminated list describing the receive antenna or antennas currently in use during diversity and packet reception. Table 56 below shows the possible values appearing in the list, where $N \leq 255$.

Table 56—Diversity select antenna codes

Code point	Antenna number
X'01'	Rx Antenna 1
X'02'	Rx Antenna 2
X'03'	Rx Antenna 3
N	Rx Antenna *N*
X'00'	Null terminator

The null terminated list can consist of one or more of the receive antennas listed in the aSupportedRxAntennas attribute. This attribute can be changed dynamically by the LME.

14.8.2.1.32 aNumberSupportedPowerLevels

The aNumberSupportedPowerLevels attribute for the FHSS PHY describes the number of power levels this implementation supports. This attribute can be an integer of value 1 through 8, inclusive.

14.8.2.1.33 aTxPowerLevel1-8

Some implementations may provide up to eight different transmit power levels. The aTxPowerLevels attribute for the FHSS PHY is a list of up to eight power levels supported. Table 57 describes the list.

Table 57—Transmit power levels

Attribute	Power level
TxPowerLevel1	Default setting
TxPowerLevel2	Level 2
TxPowerLevel3	Level 3
TxPowerLevel4	Level 4
TxPowerLevel5	Level 5
TxPowerLevel6	Level 6
TxPowerLevel7	Level 7
TxPowerLevel8	Level 8

14.8.2.1.34 aCurrentTxPowerLevel

The aCurrentTxPowerLevel attribute for the FHSS PHY is defined as the current transmit output power level. This level shall be one of the levels implemented in the list of attributes called aTxPowerLevel*N* (where *N* is 1–8). This MIB attribute is also used to define the sensitivity of the CCA mechanism when the output power exceeds 100 mW. This MIB attribute is managed by the LME.

14.8.2.1.35 aHopTime

The aHopTime attribute for the FHSS PHY describes the time allocated for the PHY to change to a new frequency. For the FHSS PHY, this time period is 224 µs.

14.8.2.1.36 aCurrentChannelNumber

The aCurrentChannelNumber attribute for the FHSS PHY is defined as the current operating channel number of the PMD. The values of this attribute correspond to the values shown in Table 38. This MIB attribute is managed by the PLME and is updated as the result of a PLMESET.request to aCurrentSet, aCurrentPattern, or aCurrentIndex.

14.8.2.1.37 aMaxDwellTime

The aMaxDwellTime attribute for the FHSS PHY is defined as the maximum time the PMD can dwell on a channel and meet the requirements of the current regulatory domain. For the FCC regulatory domain, this number is 390 TU (FCC = 400 ms). The recommended dwell time for the FHSS PHY is 19 TU.

14.8.2.1.38 aCurrentSet

The FHSS PHY contains three sets of hopping patterns. The aCurrentSet attribute for the FHSS PHY defines what set the STA is using to determine the hopping pattern. Its value can be 1, 2, or 3. This attribute is managed by the PLME.

14.8.2.1.39 aCurrentPattern

There are up to 26 patterns in each hopping set used by the FHSS PHY. The aCurrentPattern attribute for the FHSS PHY defines what pattern the STA is using to determine the hopping sequence. Its value has various ranges, always within the overall range of 1 to 26, depending on the aCurrentRegDomain. This attribute is managed by the PLME.

14.8.2.1.40 aCurrentIndex

The FHSS PHY addresses each channel in the selected hopping pattern through an index. The aCurrentIndex attribute for the FHSS PHY defines what index the STA will use to determine the next hop-channel number. Its value has various ranges, always within the overall range of 1 to 26, depending on the aCurrentRegDomain. This attribute is managed by the PLME.

14.8.2.1.41 aCurrentPowerState

The parameter aCurrentPowerState defines the operational state of the FHSS PHY. When this attribute has a value of X'01', the PHY is "OFF." When this attribute has a value of X'02', the PHY is "ON." This attribute is managed by the PLME.

15. Direct sequence spread spectrum (DSSS) PHY specification for the 2.4 GHz band designated for ISM applications

15.1 Overview

The PHY for the direct sequence spread spectrum (DSSS) system is described in this clause. The RF LAN system is initially aimed for the 2.4 GHz band designated for ISM applications as provided in the USA according to FCC 15.247, in Europe by ETS 300–328, and in other countries according to 15.4.6.2.

The DSSS system provides a wireless LAN with both a 1 Mbit/s and a 2 Mbit/s data payload communication capability. According to the FCC regulations, the DSSS system shall provide a processing gain of at least 10 dB. This shall be accomplished by chipping the baseband signal at 11 MHz with an 11-chip PN code. The DSSS system uses baseband modulations of differential binary phase shift keying (DBPSK) and differential quadrature phase shift keying (DQPSK) to provide the 1 and 2 Mbit/s data rates, respectively.

15.1.1 Scope

The PHY services provided to the IEEE 802.11 wireless LAN MAC by the 2.4 GHz DSSS system are described in this clause. The DSSS PHY layer consists of two protocol functions:

a) A physical layer convergence function, which adapts the capabilities of the physical medium dependent (PMD) system to the PHY service. This function shall be supported by the physical layer convergence procedure (PLCP), which defines a method of mapping the IEEE 802.11 MAC sublayer protocol data units (MPDU) into a framing format suitable for sending and receiving user data and management information between two or more STAs using the associated PMD system.
b) A PMD system, whose function defines the characteristics of, and method of transmitting and receiving data through, a wireless medium (WM) between two or more STAs each using the DSSS system.

15.1.2 DSSS PHY functions

The 2.4 GHz DSSS PHY architecture is depicted in the reference model shown in Figure 11. The DSSS PHY contains three functional entities: the PMD function, the physical layer convergence function, and the layer management function. Each of these functions is described in detail in the following subclauses.

The DSSS PHY service shall be provided to the MAC through the PHY service primitives described in Clause 12.

15.1.2.1 PLCP sublayer

To allow the IEEE 802.11 MAC to operate with minimum dependence on the PMD sublayer, a physical layer convergence sublayer is defined. This function simplifies the PHY service interface to the IEEE 802.11 MAC services.

15.1.2.2 PMD sublayer

The PMD sublayer provides a means to send and receive data between two or more STAs. This clause is concerned with the 2.4 GHz ISM bands using direct sequence modulation.

15.1.2.3 Physical layer management entity (PLME)

The PLME performs management of the local PHY functions in conjunction with the MAC management entity.

15.1.3 Service specification method and notation

The models represented by figures and state diagrams are intended to be illustrations of functions provided. It is important to distinguish between a model and a real implementation. The models are optimized for simplicity and clarity of presentation; the actual method of implementation is left to the discretion of the IEEE 802.11 DSSS PHY compliant developer.

The service of a layer or sublayer is a set of capabilities that it offers to a user in the next-higher layer (or sublayer). Abstract services are specified here by describing the service primitives and parameters that characterize each service. This definition is independent of any particular implementation.

15.2 DSSS PLCP sublayer

15.2.1 Overview

This clause provides a convergence procedure in which MPDUs are converted to and from PPDUs. During transmission, the MPDU shall be prepended with a PLCP Preamble and Header to create the PPDU. At the receiver, the PLCP Preamble and header are processed to aid in demodulation and delivery of the MPDU.

15.2.2 PLCP frame format

Figure 86 shows the format for the PPDU including the DSSS PLCP Preamble, the DSSS PLCP Header, and the MPDU. The PLCP Preamble contains the following fields: Synchronization (Sync) and Start Frame Delimiter (SFD). The PLCP Header contains the following fields: IEEE 802.11 Signaling (Signal), IEEE 802.11 Service (Service), LENGTH (Length), and CCITT CRC-16. Each of these fields is described in detail in 15.2.3.

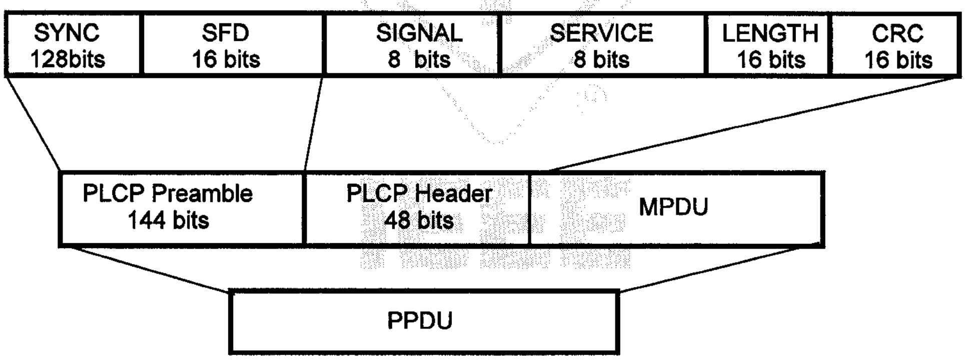


Figure 86—PLCP frame format

15.2.3 PLCP field definitions

The entire PLCP Preamble and Header shall be transmitted using the 1 Mbit/s DBPSK modulation described in 15.4.7. All transmitted bits shall be scrambled using the feedthrough scrambler described in 15.2.4.

15.2.3.1 PLCP Synchronization (SYNC) field

The SYNC field shall consist of 128 bits of scrambled 1 bit. This field shall be provided so that the receiver can perform the necessary operations for synchronization.

15.2.3.2 PLCP Start Frame Delimiter (SFD)

The SFD shall be provided to indicate the start of PHY dependent parameters within the PLCP Preamble. The SFD shall be a 16-bit field, X'F3A0' (msb to lsb). The lsb shall be transmitted first in time.

15.2.3.3 PLCP IEEE 802.11 Signal (SIGNAL) field

The 8-bit IEEE 802.11 signal field indicates to the PHY the modulation that shall be used for transmission (and reception) of the MPDU. The data rate shall be equal to the Signal field value multiplied by 100 kbit/s. The DSSS PHY currently supports two mandatory modulation services given by the following 8-bit words, where the lsb shall be transmitted first in time:

a) X'0A' (msb to lsb) for 1 Mbit/s DBPSK
b) X'14' (msb to lsb) for 2 Mbit/s DQPSK

The DSSS PHY rate change capability is described in 15.2.5. This field shall be protected by the CCITT CRC-16 frame check sequence described in 15.2.3.6.

15.2.3.4 PLCP IEEE 802.11 Service (SERVICE) field

The 8-bit IEEE 802.11 service field shall be reserved for future use. The value of X'00' signifies IEEE 802.11 device compliance. The lsb shall be transmitted first in time. This field shall be protected by the CCITT CRC-16 frame check sequence described in 15.2.3.6.

15.2.3.5 PLCP Length (LENGTH) field

The PLCP Length field shall be an unsigned 16-bit integer that indicates the number of microseconds (16 to $2^{16}-1$ as defined by aMPDUMaxLength) required to transmit the MPDU. The transmitted value shall be determined from the LENGTH parameter in the TXVECTOR issued with the PHY-TXSTART.request primitive described in 12.3.5.4. The Length field provided in the TXVECTOR is in bytes and is converted to microseconds for inclusion in the PLCP LENGTH field. The lsb shall be transmitted first in time. This field shall be protected by the CCITT CRC-16 frame check sequence described in 15.2.3.6.

15.2.3.6 PLCP CRC (CCITT CRC-16) field

The IEEE 802.11 SIGNAL, IEEE 802.11 SERVICE, and LENGTH fields shall be protected with a CCITT CRC-16 FCS (frame check sequence). The CCITT CRC-16 FCS shall be the one's complement of the remainder generated by the modulo 2 division of the protected PLCP fields by the polynomial:

$$x^{16} + x^{12} + x^5 + 1$$

The protected bits shall be processed in transmit order. All FCS calculations shall be made prior to data scrambling.

As an example, the SIGNAL, SERVICE, and LENGTH fields for a DBPSK signal with a packet length of 192 μs (24 bytes) would be given by the following:

0101 0000 0000 0000 0000 0011 0000 0000 (leftmost bit transmitted first in time)

The one's complement FCS for these protected PLCP Preamble bits would be the following:

0101 1011 0101 0111 (leftmost bit transmitted first in time)

Figure 87 depicts this example.

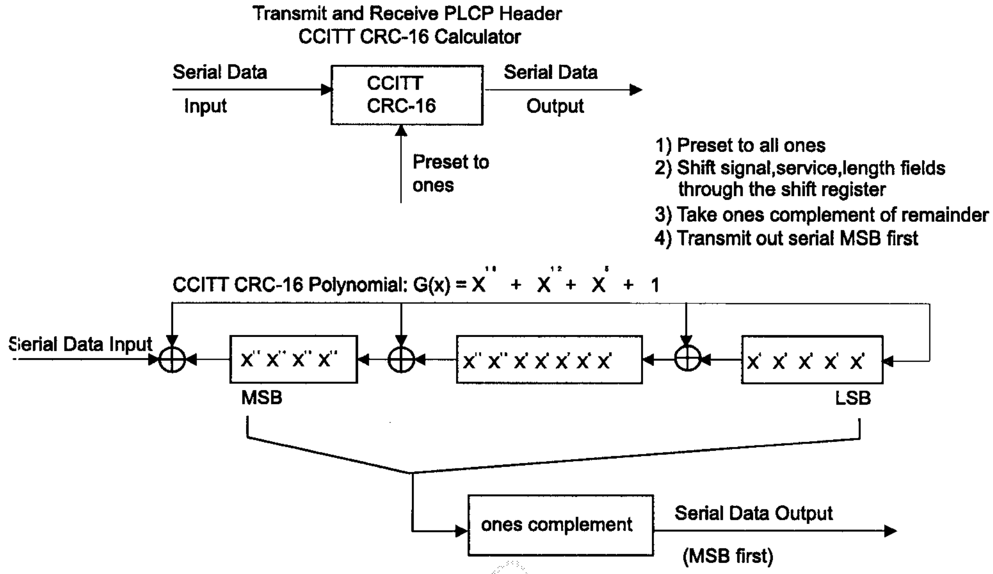


Figure 87—CCITT CRC-16 implementation

An illustrative example of the CCITT CRC-16 FCS using the information from Figure 87 follows in Figure 88.

Data	CRC registers msb lsb	
	1111111111111111	; initialize preset to 1's
0	1110111111011111	
1	1101111110111110	
0	1010111101011101	
1	0101111010111010	
0	1011110101110100	
0	0110101011001001	
0	1101010110010010	
0	1011101100000101	
0	0110011000101011	
0	1100110001010110	
0	1000100010001101	
0	0000000100111011	
0	0000001001110110	
0	0000010011101100	
0	0000100111011000	
0	0001001110110000	
0	0010011101100000	
0	0100111011000000	
0	1001110110000000	
0	0010101100100001	
0	0101011001000010	
0	1010110010000100	
1	0101100100001000	
1	1010001000110001	
0	0101010001000011	
0	1010100010000110	
0	0100000100101101	
0	1000001001011010	
0	0001010010010101	
0	0010100100101010	
0	0101001001010100	
0	1010010010101000	
	0101101101010111	; one's complement, result = CRC FCS parity

Figure 88—Example CRC calculation

15.2.4 PLCP/DSSS PHY data scrambler and descrambler

The polynomial $G(z) = z^{-7} + z^{-4} + 1$ shall be used to scramble *all* bits transmitted by the DSSS PHY. The feedthrough configuration of the scrambler and descrambler is self-synchronizing, which requires no prior knowledge of the transmitter initialization of the scrambler for receive processing. Figure 89 and Figure 90 show typical implementations of the data scrambler and descrambler, but other implementations are possible.

The scrambler should be initialized to any state except all ones when transmitting.

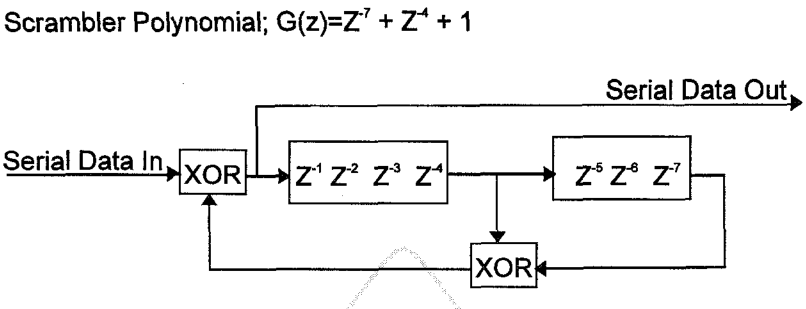


Figure 89—Data scrambler

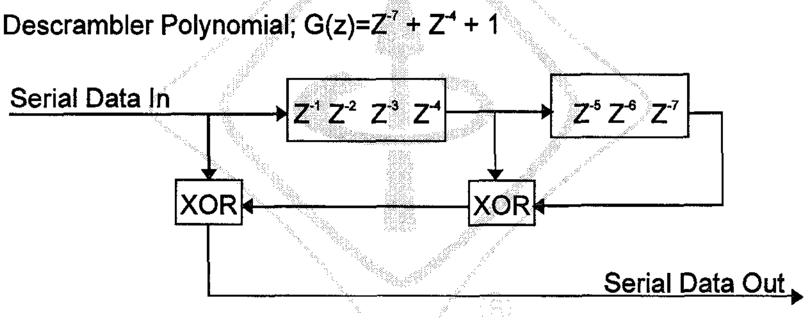


Figure 90—Data descrambler

15.2.5 PLCP data modulation and modulation rate change

The PLCP Preamble shall be transmitted using the 1 Mbit/s DBPSK modulation. The IEEE 802.11 SIGNAL field shall indicate the modulation that shall be used to transmit the MPDU. The transmitter and receiver shall initiate the modulation indicated by the IEEE 802.11 SIGNAL field starting with the first symbol (1 bit for DBPSK or 2 bits for DQPSK) of the MPDU. The MPDU transmission rate shall be set by the DATARATE parameter in the TXVECTOR issued with the PHY-TXSTART.request primitive described in 15.4.4.1.

15.2.6 PLCP transmit procedure

The PLCP transmit procedure is shown in Figure 91.

In order to transmit data, PHY-TXSTART.request shall be enabled so that the PHY entity shall be in the transmit state. Further, the PHY shall be set to operate at the appropriate channel through station management via the PLME. Other transmit parameters such as DATARATE, TX antenna, and TX power are set via the PHY-SAP with the PHY-TXSTART.request(TXVECTOR) as described in 15.4.4.2.

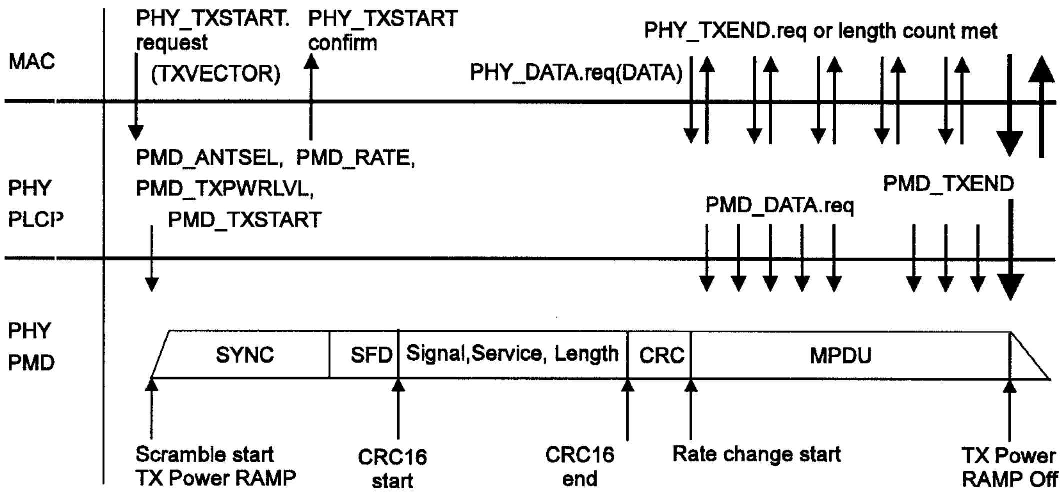


Figure 91—PLCP transmit procedure

Based on the status of clear channel assessment (CCA) indicated by PHY-CCA.indicate, the MAC will assess that the channel is clear. A clear channel shall be indicated by PHY-CCA.indicate(IDLE). If the channel is clear, transmission of the PPDU shall be initiated by issuing the PHY-TXSTART.request (TXVECTOR) primitive. The TXVECTOR elements for the PHY-TXSTART.request are the PLCP Header parameters SIGNAL (DATARATE), SERVICE, and LENGTH, and the PMD parameters of TX_ANTENNA and TXPWR_LEVEL. The PLCP Header parameter LENGTH is calculated from the TXVECTOR element by multiplying by 8 for 1 Mbit/s and by 4 for 2 Mbit/s.

The PLCP shall issue PMD_ANTSEL, PMD_RATE, and PMD_TXPWRLVL primitives to configure the PHY. The PLCP shall then issue a PMD_TXSTART.request and the PHY entity shall immediately initiate data scrambling and transmission of the PLCP Preamble based on the parameters passed in the PHY-TXSTART.request primitive. The time required for TX power on ramp described in 15.4.7.7 shall be included in the PLCP synchronization field. Once the PLCP Preamble transmission is complete, data shall be exchanged between the MAC and the PHY by a series of PHY-DATA.request(DATA) primitives issued by the MAC and PHY-DATA.confirm primitives issued by the PHY. The modulation rate change, if any, shall be initiated with the first data symbol of the MPDU as described in 15.2.5. The PHY proceeds with MPDU transmission through a series of data octet transfers from the MAC. At the PMD layer, the data octets are sent in lsb to msb order and presented to the PHY layer through PMD_DATA.request primitives. Transmission can be prematurely terminated by the MAC through the primitive PHY-TXEND.request. PHY-TXSTART shall be disabled by the issuance of the PHY-TXEND.request. Normal termination occurs after the transmission of the final bit of the last MPDU octet according to the number supplied in the DSSS PHY preamble LENGTH field. The packet transmission shall be completed and the PHY entity shall enter the receive state (i.e., PHY-TXSTART shall be disabled). It is recommended that chipping continue during power-down. Each PHY-TXEND.request is acknowledged with a PHY-TXEND.confirm primitive from the PHY.

A typical state machine implementation of the PLCP transmit procedure is provided in Figure 92.

15.2.7 PLCP receive procedure

The PLCP receive procedure is shown in Figure 93.

In order to receive data, PHY-TXSTART.request shall be disabled so that the PHY entity is in the receive state. Further, through station management via the PLME, the PHY is set to the appropriate channel and the CCA method is chosen. Other receive parameters such as receive signal strength indication (RSSI), signal quality (SQ), and indicated DATARATE may be accessed via the PHY-SAP.

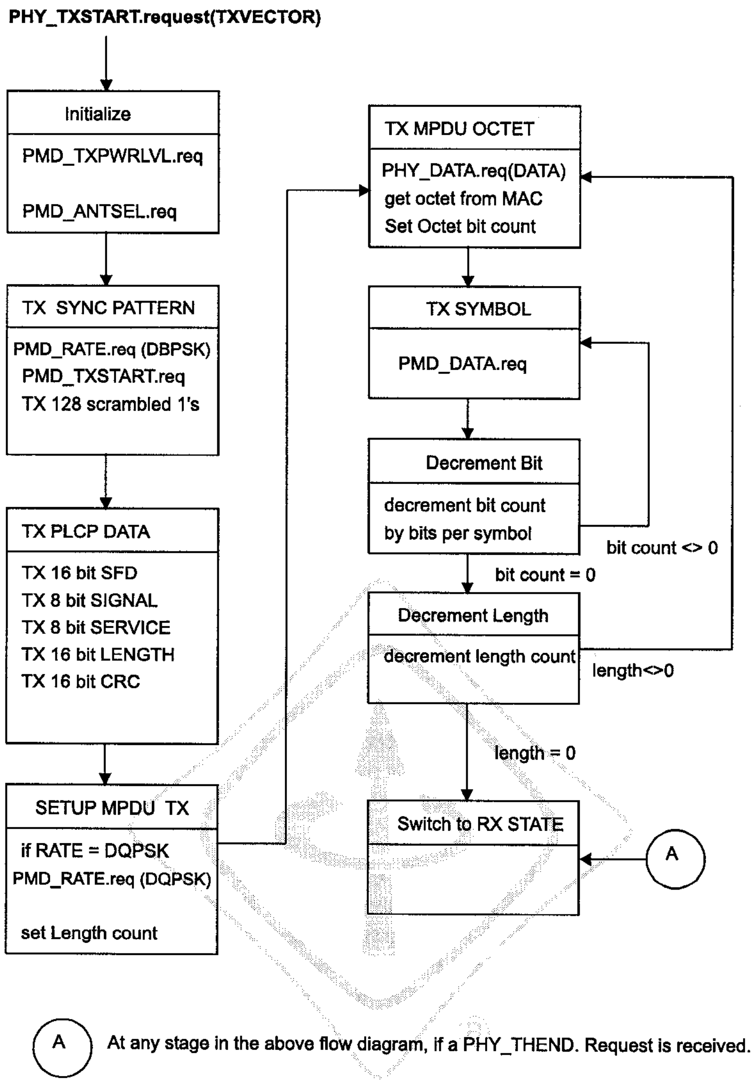


Figure 92—PLCP transmit state machine

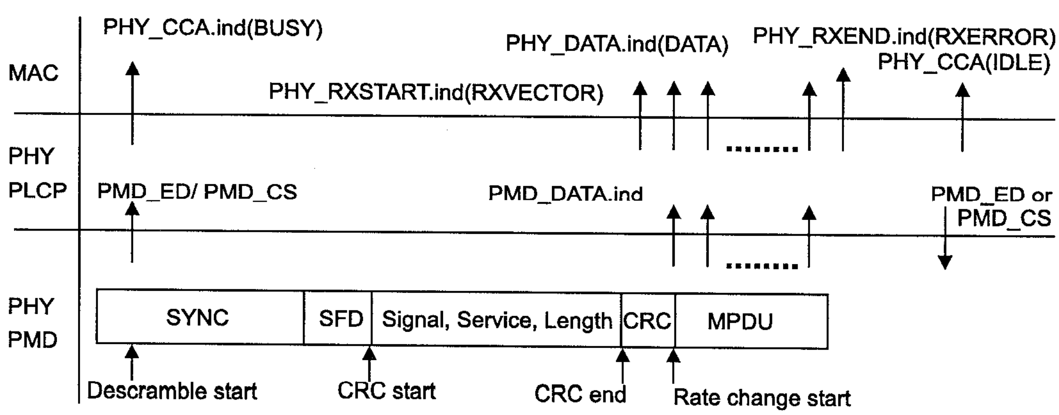


Figure 93—PLCP receive procedure

Upon receiving the transmitted energy, according to the selected CCA mode, the PMD_ED shall be enabled (according to 15.4.8.4) as the RSSI strength reaches the ED_THRESHOLD and/or PMD_CS shall be enabled after code lock is established. These conditions are used to indicate activity to the MAC via PHY-CCA.indicate according to 15.4.8.4. PHY-CCA.indicate(BUSY) shall be issued for energy detection and/or code lock prior to correct reception of the PLCP frame. The PMD primitives PMD_SQ and PMD_RSSI are issued to update the RSSI and SQ parameters reported to the MAC.

After PHY-CCA.indicate is issued, the PHY entity shall begin searching for the SFD field. Once the SFD field is detected, CCITT CRC-16 processing shall be initiated and the PLCP IEEE 802.11 SIGNAL, IEEE 802.11 SERVICE and LENGTH fields are received. The CCITT CRC-16 FCS shall be processed. If the CCITT CRC-16 FCS check fails, the PHY receiver shall return to the RX Idle state as depicted in Figure 94. Should the status of CCA return to the IDLE state during reception prior to completion of the full PLCP processing, the PHY receiver shall return to the RX Idle state.

If the PLCP Header reception is successful (and the SIGNAL field is completely recognizable and supported), a PHY-RXSTART.indicate(RXVECTOR) shall be issued. The RXVECTOR associated with this primitive includes the SIGNAL field, the SERVICE field, the MPDU length in bytes (calculated from the LENGTH field in microseconds), the antenna used for receive (RX_ANTENNA), RSSI, and SQ.

The received MPDU bits are assembled into octets and presented to the MAC using a series of PHY-DATA.indicate(DATA) primitive exchanges. The rate change indicated in the IEEE 802.11 SIGNAL field shall be initiated with the first symbol of the MPDU as described in 15.2.5. The PHY proceeds with MPDU reception. After the reception of the final bit of the last MPDU octet indicated by the PLCP Preamble LENGTH field, the receiver shall be returned to the RX Idle state as shown in Figure 94. A PHY-RXEND.indicate(NoError) primitive shall be issued. A PHY-CCA.indicate(IDLE) primitive shall be issued following a change in PHYCS (PHY carrier sense) and/or PHYED (PHY energy detection) according to the selected CCA method.

In the event that a change in PHYCS or PHYED would cause the status of CCA to return to the IDLE state before the complete reception of the MPDU as indicated by the PLCP LENGTH field, the error condition PHY-RXEND.indicate(CarrierLost) shall be reported to the MAC. The DSSS PHY will ensure that the CCA will indicate a busy medium for the intended duration of the transmitted packet.

If the PLCP Header is successful, but the indicated rate in the SIGNAL field is not receivable, a PHY-RXSTART.indicate will not be issued. The PHY shall issue the error condition PHY-RXEND.indicate(UnsupportedRate). If the PLCP Header is successful, but the SERVICE field is out of IEEE 802.11 DSSS specification, a PHY-RXSTART.indicate will not be issued. The PHY shall issue the error condition PHY-RXEND.indicate(FormatViolation). Also, in both cases, the DSSS PHY will ensure that the CCA shall indicate a busy medium for the intended duration of the transmitted frame as indicated by the Length field. The intended duration is indicated by the Length field (length$\times$1 μs).

A typical state machine implementation of the PLCP receive procedure is provided in Figure 94.

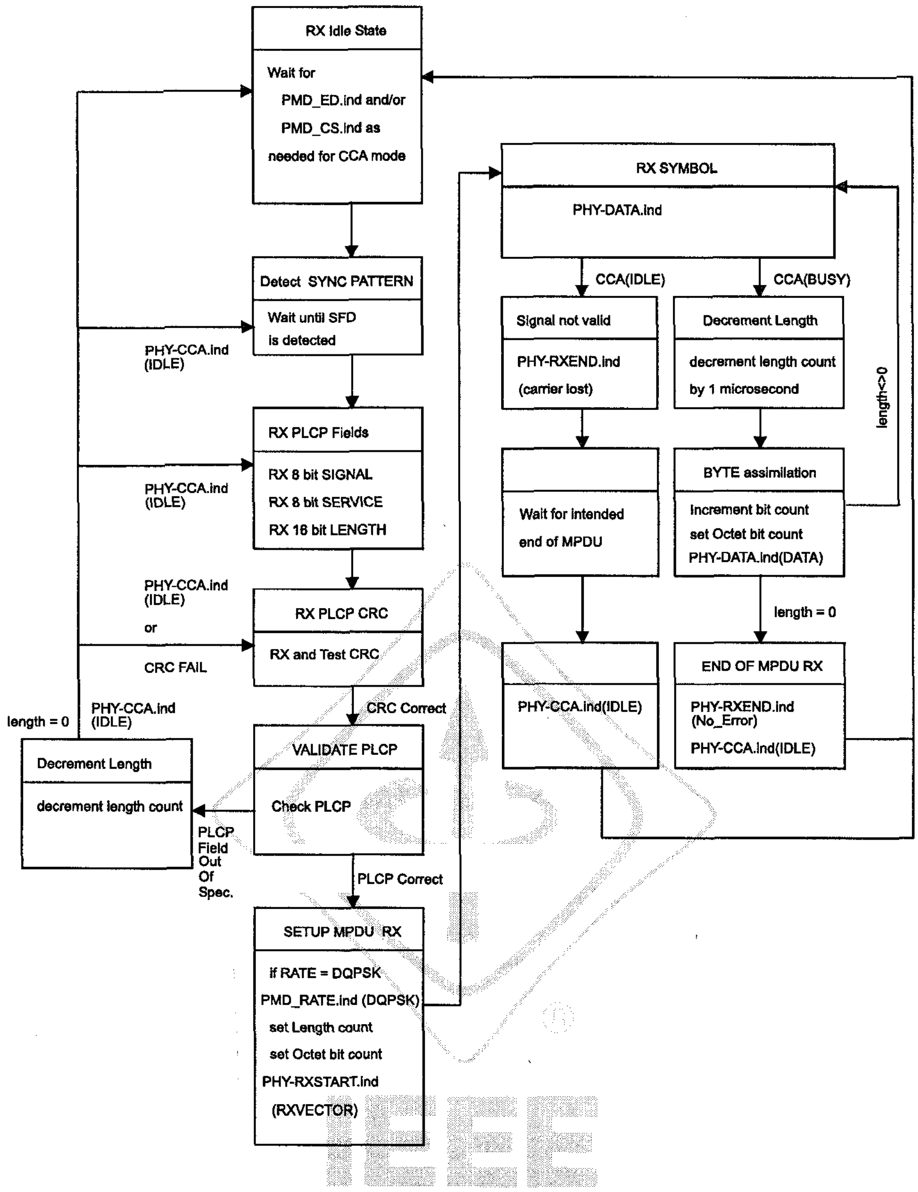


Figure 94—PLCP receive state machine

15.3 DSSS physical layer management entity (PLME)

15.3.1 PLME_SAP sublayer management primitives

Table 58 lists the MIB attributes that may be accessed by the PHY sublayer entities and intralayer of higher layer management entities (LME). These attributes are accessed via the PLME-GET, PLME-SET, and PLME-RESET primitives defined in Clause 10.

15.3.2 DSSS PHY MIB

All DSSS PHY MIB attributes are defined in Clause 12, with specific values defined in Table 58.

Table 58—MIB attribute default values/ranges

Managed object	Default value/range	Operational semantics
agPhyOperationGroup		
aPHYType	DSSS–2.4 (02)	Static
aTempType	Implementation dependent	Static
aCWmin	31	Static
aCWmax	1023	Static
aRegDomainsSupported	Implementation dependent	Static
aCurrentRegDomain	Implementation dependent	Static
aSlotTime	20 µs	Static
aCCATime	≤15 µs	Static
aRxTxTurnaroundTime	≤5 µs	Static
aTxPLCPDelay	Implementation dependent	Static
aRxTxSwitchTime	≤5 µs	Static
aTxRampOnTime	Implementation dependent	Static
aTxRFDelay	Implementation dependent	Static
aSIFSTime	10 µs	Static
aRxRFDelay	Implementation dependent	Static
aRxPLCPDelay	Implementation dependent	Static
aMACProccssingDelay	Not applicable	n/a
aTxRampOffTime	Implementation dependent	Static
aPreambleLength	144 bits	Static
aPLCPHeaderLength	48 bits	Static
agPhyRateGroup		
aSupportedDataRatesTx	X'02', X'04'	Static
aSupportedDataRatesRx	X'02', X'04'	Static
aMPDUMaxLength	$4 \leq x \leq (2^{13} - 1)$	Static
agPhyAntennaGroup		
aCurrentTxAntenna	Implementation dependent	Dynamic
aDiversitySupport	Implementation dependent	Static
agPhyTxPowerGroup		
aNumberSupportedPowerLevels	Implementation dependent	Static
aTxPowerLevel1	Implementation dependent	Static
aTxPowerLevel2	Implementation dependent	Static
aTxPowerLevel3	Implementation dependent	Static
aTxPowerLevel4	Implementation dependent	Static
aTxPowerLevel5	Implementation dependent	Static
aTxPowerLevel6	Implementation dependent	Static
aTxPowerLevel7	implementation dependent	Static
aTxPowerLevel8	Implementation dependent	Static
aCurrentTxPowerLevel	Implementation dependent	Dynamic

Table 58—MIB attribute default values/ranges *(continued)*

Managed object	Default value/range	Operational semantics
agPhyStatusGroup		
aSynthesizerLocked	Implementation dependent	Dynamic
agPhyDSSSGroup		
aCurrentChannel	Implementation dependent	Dynamic
aCCAModeSupport	Implementation dependent	Static
aCurrentCCAMode	Implementation dependent	Dynamic
aEDThreshold	Implementation dependent	Dynamic
agPhyPwrSavingGroup		
aDozeTurnonTime	Implementation dependent	Static
aCurrentPowerState	Implementation dependent	Dynamic
agAntennasListGroup		
aSupportTxAntennas	Implementation dependent	Static
aSupportRxAntennas	Implementation dependent	Static
aDiversitySelectRx	Implementation dependent	Dynamic
NOTE—The column titled "Operational semantics" contains two types: static and dynamic. Static MIB attributes are fixed and cannot be modified for a given PHY implementation. MIB attributes defined as dynamic can be modified by some management entities.		

15.4 DSSS PMD sublayer

15.4.1 Scope and field of application

This subclause describes the PMD services provided to the PLCP for the DSSS PHY. Also defined in this subclause are the functional, electrical, and RF characteristics required for interoperability of implementations conforming to this specification. The relationship of this specification to the entire DSSS physical layer is shown in Figure 95.

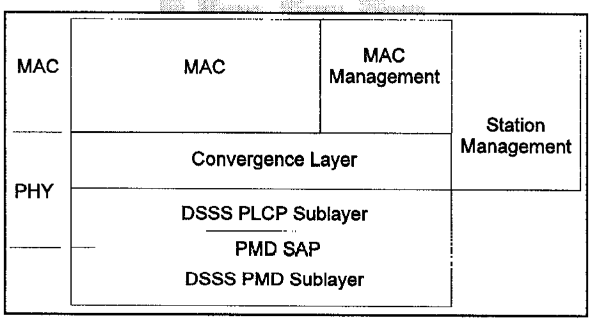


Figure 95—PMD layer reference model

15.4.2 Overview of service

The DSSS PMD sublayer accepts PLCP sublayer service primitives and provides the actual means by which data shall be transmitted or received from the medium. The combined function of DSSS PMD sublayer primitives and parameters for the receive function results in a data stream, timing information, and associated received signal parameters being delivered to the PLCP sublayer. A similar functionality shall be provided for data transmission.

15.4.3 Overview of interactions

The primitives associated with the IEEE 802.11 PLCP sublayer to the DSSS PMD fall into two basic categories:

a) Service primitives that support PLCP peer-to-peer interactions, and
b) Service primitives that have local significance and that support sublayer-to-sublayer interactions.

15.4.4 Basic service and options

All of the service primitives described in this clause are considered mandatory unless otherwise specified.

15.4.4.1 PMD_SAP peer-to-peer service primitives

Table 59 indicates the primitives for peer-to-peer interactions.

Table 59—PMD_SAP peer-to-peer service primitives

Primitive	Request	Indicate	Confirm	Response
PHY-RXSTART		X		
PHY-RXEND		X		
PHY-CCA		X		
PHY-TXSTART	X		X	
PHY-TXEND	X		X	
PHY-DATA	X	X	X	

15.4.4.2 PMD_SAP peer-to-peer service primitive parameters

Several service primitives include a parameter vector. This vector shall be actually a list of parameters that may vary depending on PHY type. Table 60 indicates the parameters required by the MAC or DSSS PHY in each of the parameter vectors used for peer-to-peer interactions.

Table 60—DSSS PMD_SAP peer-to-peer service primitives

Parameter	Associated primitive	Value
LENGTH	RXVECTOR, TXVECTOR	4 to $2^{16} - 1$
DATARATE	RXVECTOR, TXVECTOR	PHY dependent
SERVICE	RXVECTOR, TXVECTOR	PHY dependent
TXPWR_LEVEL	TXVECTOR	PHY dependent
TX_ANTENNA	TXVECTOR	PHY dependent
RSSI	RXVECTOR	PHY dependent
SQ	RXVECTOR	PHY dependent
RX_ANTENNA	RXVECTOR	PHY dependent

15.4.4.3 PMD_SAP sublayer-to-sublayer service primitives

Table 61 indicates the primitives for sublayer-to-sublayer interactions.

Table 61—PMD_SAP sublayer-to-sublayer service primitives

Primitive	Request	Indicate	Confirm	Response
PMD_TXSTART	X			
PMD_TXEND	X			
PMD_ANTSEL	X	X		
PMD_TXPWRLVL	X			
PMD_RATE	X	X		
PMD_RSSI		X		
PMD_SQ		X		
PMD_CS		X		
PMD_ED	X	X		

15.4.4.4 PMD_SAP service primitive parameters

Table 62 indicates the parameters for the PMD primitives.

Table 62—List of parameters for the PMD primitives

Parameter	Associate primitive	Value
DATA	PHY-DATA.request PHY-DATA.indicate	Octet value: X'00'–X'FF'
TXVECTOR	PHY-DATA.request	A set of parameters
RXVECTOR	PHY-DATA.indicate	A set of parameters
TXD_UNIT	PMD_DATA.request	One(1), Zero(0): DBPSK dibit combinations 00,01,11,10: DQPSK
RXD_UNIT	PMD_DATA.indicate	One(1), Zero(0): DBPSK dibit combinations 00,01,11,10: DQPSK
RF_STATE	PMD_TXE.request	Receive, Transmit
ANT_STATE	PMD_ANTSEL.indicate PMD_ANTSEL.request	1 to 256
TXPWR_LEVEL	PHY-TXSTART	0, 1, 2, 3 (max of 4 levels)
RATE	PMD_RATE.indicate PMD_RATE.request	X'0A' for 1 Mbit/s DBPSK X'14' for 2 Mbit/s DQPSK
RSSI	PMD_RSSI.indicate	0–8 bits of RSSI
SQ	PMD_SQ.indicate	0–8 bits of SQ

15.4.5 PMD_SAP detailed service specification

The following subclauses describe the services provided by each PMD primitive.

15.4.5.1 PMD_DATA.request

15.4.5.1.1 Function

This primitive defines the transfer of data from the PLCP sublayer to the PMD entity.

15.4.5.1.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_DATA.request(TXD_UNIT)

The TXD_UNIT parameter takes on the value of either one(1) or zero(0) for DBPSK modulation or the dibit combination 00, 01, 11, or 10 for DQPSK modulation. This parameter represents a single block of data, which, in turn, shall be used by the PHY to be differentially encoded into a DBPSK or DQPSK transmitted symbol. The symbol itself shall be spread by the PN code prior to transmission.

15.4.5.1.3 When generated

This primitive shall be generated by the PLCP sublayer to request transmission of a symbol. The data clock for this primitive shall be supplied by PMD layer based on the PN code repetition.

15.4.5.1.4 Effect of receipt

The PMD performs the differential encoding, PN code modulation and transmission of the data.

15.4.5.2 PMD_DATA.indicate

15.4.5.2.1 Function

This primitive defines the transfer of data from the PMD entity to the PLCP sublayer.

15.4.5.2.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_DATA.indicate(RXD_UNIT)

The RXD_UNIT parameter takes on the value of one(1) or zero(0) for DBPSK modulation or as the dibit 00, 01, 11, or 10 for DQPSK modulation. This parameter represents a single symbol that has been demodulated by the PMD entity.

15.4.5.2.3 When generated

This primitive, which is generated by the PMD entity, forwards received data to the PLCP sublayer. The data clock for this primitive shall be supplied by PMD layer based on the PN code repetition.

15.4.5.2.4 Effect of receipt

The PLCP sublayer either interprets the bit or bits that are recovered as part of the PLCP convergence procedure or passes the data to the MAC sublayer as part of the MPDU.

15.4.5.3 PMD_TXSTART.request

15.4.5.3.1 Function

This primitive, which is generated by the PHY PLCP sublayer, initiates PPDU transmission by the PMD layer.

15.4.5.3.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_TXSTART.request

15.4.5.3.3 When generated

This primitive shall be generated by the PLCP sublayer to initiate the PMD layer transmission of the PPDU. The PHY-DATA.request primitive shall be provided to the PLCP sublayer prior to issuing the PMD_TXSTART command.

15.4.5.3.4 Effect of receipt

PMD_TXSTART initiates transmission of a PPDU by the PMD sublayer.

15.4.5.4 PMD_TXEND.request

15.4.5.4.1 Function

This primitive, which is generated by the PHY PLCP sublayer, ends PPDU transmission by the PMD layer.

15.4.5.4.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_TXEND.request

15.4.5.4.3 When generated

This primitive shall be generated by the PLCP sublayer to terminate the PMD layer transmission of the PPDU.

15.4.5.4.4 Effect of receipt

PMD_TXEND terminates transmission of a PPDU by the PMD sublayer.

15.4.5.5 PMD_ANTSEL.request

15.4.5.5.1 Function

This primitive, which is generated by the PHY PLCP sublayer, selects the antenna used by the PHY for transmission or reception (when diversity is disabled).

15.4.5.5.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_ANTSEL.request(ANT_STATE)

ANT_STATE selects which of the available antennas should be used for transmit. The number of available antennas shall be determined from the MIB table parameters aSuprtRxAntennas and aSuprtTxAntennas.

15.4.5.5.3 When generated

This primitive shall be generated by the PLCP sublayer to select a specific antenna for transmission (or reception when diversity is disabled).

15.4.5.5.4 Effect of receipt

PMD_ANTSEL immediately selects the antenna specified by ANT_STATE.

15.4.5.6 PMD_ANTSEL.indicate

15.4.5.6.1 Function

This primitive, which is generated by the PHY PLCP sublayer, reports the antenna used by the PHY for reception of the most recent packet.

15.4.5.6.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_ANTSEL.indicate(ANT_STATE)

ANT_STATE reports which of the available antennas was used for reception of the most recent packet.

15.4.5.6.3 When generated

This primitive shall be generated by the PLCP sublayer to report the antenna used for the most recent packet reception.

15.4.5.6.4 Effect of receipt

PMD_ANTSEL immediately reports the antenna specified by ANT_STATE.

15.4.5.7 PMD_TXPWRLVL.request

15.4.5.7.1 Function

This primitive, which is generated by the PHY PLCP sublayer, selects the power level used by the PHY for transmission.

15.4.5.7.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_TXPWRLVL.request(TXPWR_LEVEL)

TXPWR_LEVEL selects which of the optional transmit power levels should be used for the current packet transmission. The number of available power levels shall be determined by the MIB parameter aNumber-SupportedPowerLevels. Subclause 15.4.7.3 provides further information on the optional DSSS PHY power level control capabilities.

15.4.5.7.3 When generated

This primitive shall be generated by the PLCP sublayer to select a specific transmit power. This primitive shall be applied prior to setting PMD_TXSTART into the transmit state.

15.4.5.7.4 Effect of receipt

PMD_TXPWRLVL immediately sets the transmit power level given by TXPWR_LEVEL.

15.4.5.8 PMD_RATE.request

15.4.5.8.1 Function

This primitive, which is generated by the PHY PLCP sublayer, selects the modulation rate that shall be used by the DSSS PHY for transmission.

15.4.5.8.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_RATE.request(RATE)

RATE selects which of the DSSS PHY data rates shall be used for MPDU transmission. Subclause 15.4.6.4 provides further information on the DSSS PHY modulation rates. The DSSS PHY rate change capability is fully described in 15.2.

15.4.5.8.3 When generated

This primitive shall be generated by the PLCP sublayer to change or set the current DSSS PHY modulation rate used for the MPDU portion of a PPDU.

15.4.5.8.4 Effect of receipt

The receipt of PMD_RATE selects the rate that shall be used for all subsequent MPDU transmissions. This rate shall be used for transmission only. The DSSS PHY shall still be capable of receiving all the required DSSS PHY modulation rates.

15.4.5.9 PMD_RATE.indicate

15.4.5.9.1 Function

This primitive, which is generated by the PMD sublayer, indicates which modulation rate was used to receive the MPDU portion of the PPDU. The modulation shall be indicated in the PLCP Preamble IEEE 802.11 SIGNALING field.

15.4.5.9.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_RATE.indicate(RATE)

In receive mode, the RATE parameter informs the PLCP layer which of the DSSS PHY data rates was used to process the MPDU portion of the PPDU. Subclause 15.4.6.4 provides further information on the DSSS PHY modulation rates. The DSSS PHY rate change capability is fully described in 15.2.

15.4.5.9.3 When generated

This primitive shall be generated by the PMD sublayer when the PLCP Preamble IEEE 802.11 SIGNALING field has been properly detected.

15.4.5.9.4 Effect of receipt

This parameter shall be provided to the PLCP layer for information only.

15.4.5.10 PMD_RSSI.indicate

15.4.5.10.1 Function

This optional primitive, which is generated by the PMD sublayer, provides to the PLCP and MAC entity the received signal strength.

15.4.5.10.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_RSSI.indicate(RSSI)

The RSSI shall be a measure of the RF energy received by the DSSS PHY. RSSI indications of up to 8 bits (256 levels) are supported.

15.4.5.10.3 When generated

This primitive shall be generated by the PMD when the DSSS PHY is in the receive state. It shall be continuously available to the PLCP, which, in turn, provides the parameter to the MAC entity.

15.4.5.10.4 Effect of receipt

This parameter shall be provided to the PLCP layer for information only. The RSSI may be used in conjunction with SQ as part of a CCA scheme.

15.4.5.11 PMD_SQ.indicate

15.4.5.11.1 Function

This optional primitive, which is generated by the PMD sublayer, provides to the PLCP and MAC entity the signal quality (SQ) of the DSSS PHY PN code correlation. The SQ shall be sampled when the DSSS PHY achieves code lock and held until the next code lock acquisition.

15.4.5.11.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_SQ.indicate(SQ)

The SQ shall be a measure of the PN code correlation quality received by the DSSS PHY. SQ indications of up to 8 bits (256 levels) are supported.

15.4.5.11.3 When generated

This primitive shall be generated by the PMD when the DSSS PHY is in the receive state and code lock is achieved. It shall be continuously available to the PLCP, which, in turn, provides the parameter to the MAC entity.

15.4.5.11.4 Effect of receipt

This parameter shall be provided to the PLCP layer for information only. The SQ may be used in conjunction with RSSI as part of a CCA scheme.

15.4.5.12 PMD_CS.indicate

This primitive, which is generated by the PMD, shall indicate to the PLCP layer that the receiver has acquired (locked) the PN code and data is being demodulated.

15.4.5.12.1 Function

This primitive, which is generated by the PMD, shall indicate to the PLCP layer that the receiver has acquired (locked) the PN code and data is being demodulated.

15.4.5.12.2 Semantics of the service primitive

The PMD_CS (carrier sense) primitive in conjunction with PMD_ED provide CCA status through the PLCP layer PHYCCA primitive. PMD_CS indicates a binary status of ENABLED or DISABLED. PMD_CS shall be ENABLED when the correlator SQ indicated in PMD_SQ is greater than the CS_THRESHOLD parameter. PMD_CS shall be DISABLED when the PMD_SQ falls below the correlation threshold.

15.4.5.12.3 When generated

This primitive shall be generated by the PHY sublayer when the DSSS PHY is receiving a PPDU and the PN code has been acquired.

15.4.5.12.4 Effect of receipt

This indicator shall be provided to the PLCP for forwarding to the MAC entity for information purposes through the PHYCCA indicator. This parameter shall indicate that the RF medium is busy and occupied by a

DSSS PHY signal. The DSSS PHY should not be placed into the transmit state when PMD_CS is ENABLED.

15.4.5.13 PMD_ED.indicate

15.4.5.13.1 Function

This optional primitive, which is generated by the PMD, shall indicate to the PLCP layer that the receiver has detected RF energy indicated by the PMD_RSSI primitive that is above a predefined threshold.

15.4.5.13.2 Semantics of the service primitive

The PMD_ED (energy detect) primitive, along with the PMD_SQ, provides CCA status at the PLCP layer through the PHYCCA primitive. PMD_ED indicates a binary status of ENABLED or DISABLED. PMD_ED shall be ENABLED when the RSSI indicated in PMD_RSSI is greater than the ED_THRESHOLD parameter. PMD_ED shall be DISABLED when the PMD_RSSI falls below the energy detect threshold.

15.4.5.13.3 When generated

This primitive shall be generated by the PHY sublayer when the PHY is receiving RF energy from any source that exceeds the ED_THRESHOLD parameter.

15.4.5.13.4 Effect of receipt

This indicator shall be provided to the PLCP for forwarding to the MAC entity for information purposes through the PMD_ED indicator. This parameter shall indicate that the RF medium may be busy with an RF energy source that is not DSSS PHY compliant. If a DSSS PHY source is being received, the PMD_CS function shall be enabled shortly after the PMD_ED function is enabled.

15.4.5.14 PMD_ED.request

15.4.5.14.1 Function

This optional primitive, which is generated by the PHY PLCP, sets the energy detect ED THRESHOLD value.

15.4.5.14.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_ED.request(ED_THRESHOLD)

ED_THRESHOLD sets the threshold that the RSSI indicated shall be greater than in order for PMD_ED to be enabled.

15.4.5.14.3 When generated

This primitive shall be generated by the PLCP sublayer to change or set the current DSSS PHY energy detect threshold.

15.4.5.14.4 Effect of receipt

The receipt of PMD_ED immediately changes the energy detection threshold as set by the ED_THRESHOLD parameter.

15.4.5.15 PHY-CCA.indicate

15.4.5.15.1 Function

This primitive, which is generated by the PMD, indicates to the PLCP layer that the receiver has detected RF energy that adheres to the CCA algorithm.

15.4.5.15.2 Semantics of the service primitive

The PHY-CCA primitive provides CCA status at the PLCP layer to the MAC.

15.4.5.15.3 When generated

This primitive shall be generated by the PHY sublayer when the PHY is receiving RF energy from any source that exceeds the ED_THRESHOLD parameter (PMD_ED is active), and optionally is a valid correlated DSSS PHY signal whereby PMD_CS would also be active.

15.4.5.15.4 Effect of receipt

This indicator shall be provided to the PLCP for forwarding to the MAC entity for information purposes through the PHY-CCA indicator. This parameter indicates that the RF medium may be busy with an RF energy source that may or may not be DSSS PHY compliant. If a DSSS PHY source is being received, the PMD_CS function shall be enabled shortly after the PMD_ED function is enabled.

15.4.6 PMD operating specifications, general

The following subclauses provide general specifications for the DSSS PMD sublayer. These specifications apply to both the Receive and the Transmit functions and general operation of a DSSS PHY.

15.4.6.1 Operating frequency range

The DSSS PHY shall operate in the frequency range of 2.4 GHz to 2.4835 GHz as allocated by regulatory bodies in the USA and Europe or in the 2.471 GHz to 2.497 GHz frequency band as allocated by regulatory authority in Japan.

15.4.6.2 Number of operating channels

The channel center frequencies and CHNL_ID numbers shall be as shown in Table 63. The FCC (US), IC (Canada), and ETSI (Europe) specify operation from 2.4 GHz to 2.4835 GHz. For Japan, operation is specified as 2.471 GHz to 2.497 GHz. France allows operation from 2.4465 GHz to 2.4835 GHz, and Spain allows operation from 2.445 GHz to 2.475 GHz. For each supported regulatory domain, all channels in Table 63 marked with "X" shall be supported.

Table 63—DSSS PHY frequency channel plan

		Regulatory domains					
CHNL_ID	Frequency	X'10' FCC	X'20' IC	X'30' ETSI	X'31' Spain	X'32' France	X'40' MKK
1	2412 MHz	X	X	X	—	—	—
2	2417 MHz	X	X	X	—	—	—
3	2422 MHz	X	X	X	—	—	—
4	2427 MHz	X	X	X	—	—	—
5	2432 MHz	X	X	X	—	—	—
6	2437 MHz	X	X	X	—	—	—
7	2442 MHz	X	X	X	—	—	—
8	2447 MHz	X	X	X	—	—	—
9	2452 MHz	X	X	X	—	—	—
10	2457 MHz	X	X	X	X	X	—
11	2462 MHz	X	X	X	X	X	—
12	2467 MHz	—	—	X	—	X	—
13	2472 MHz	—	—	X	—	X	—
14	2484 MHz	—	—	—	—	—	X

In a multiple cell network topology, overlapping and/or adjacent cells using different channels can operate simultaneously without interference if the distance between the center frequencies is at least 30 MHz. Channel 14 shall be designated specifically for operation in Japan.

15.4.6.3 Spreading sequence

The following 11-chip Barker sequence shall be used as the PN code sequence:

+1, –1, +1, +1, –1, +1, +1, +1, –1, –1, –1

The leftmost chip shall be output first in time. The first chip shall be aligned at the start of a transmitted symbol. The symbol duration shall be exactly 11 chips long.

15.4.6.4 Modulation and channel data rates

Two modulation formats and data rates are specified for the DSSS PHY: a *basic access rate* and an *enhanced access rate*. The basic access rate shall be based on 1 Mbit/s DBPSK modulation. The DBPSK encoder is specified in Table 64. The enhanced access rate shall be based on 2 Mbit/s DQPSK. The DQPSK encoder is specified in Table 65. (In the tables, +jω shall be defined as counterclockwise rotation.)

Table 64—1 Mbit/s DBPSK encoding table

Bit input	Phase change (+jω)
0	0
1	π

Table 65—2 Mbit/s DQPSK encoding table

Dibit pattern (d0,d1) d0 is first in time	Phase change (+jω)
00	0
01	π/2
11	π
10	3π/2 (−π/2)

15.4.6.5 Transmit and receive in-band and out-of-band spurious emissions

The DSSS PHY shall conform with in-band and out-of-band spurious emissions as set by regulatory bodies. For the USA, refer to FCC 15.247, 15.205, and 15.209. For Europe, refer to ETS 300–328.

15.4.6.6 Transmit-to-receive turnaround time

The TX-to-RX turnaround time shall be less than 10 μs, including the power-down ramp specified in 15.4.7.7.

The TX-to-RX turnaround time shall be measured at the air interface from the trailing edge of the last transmitted symbol to valid CCA detection of the incoming signal. The CCA should occur within 25 μs (10 μs for turnaround time plus 15 μs for energy detect) or by the next slot boundary occurring after the 25 μs has elapsed (refer to 15.4.8.4). A receiver input signal 3 dB above the ED threshold described in 15.4.8.4 shall be present at the receiver.

15.4.6.7 Receive-to-transmit turnaround time

The RX-to-TX turnaround time shall be measured at the MAC/PHY interface, using PHYTXSTART.request and shall be ≤5 μs. This includes the transmit power up ramp described in 15.4.7.7.

15.4.6.8 Slot time

The slot time for the DSSS PHY shall be the sum of the RX-to-TX turnaround time (5 μs) and the energy detect time (15 μs specified in 15.4.8.4). The propagation delay shall be regarded as being included in the energy detect time.

15.4.6.9 Transmit and receive antenna port impedance

The impedance of the transmit and receive antenna port(s) shall be 50 Ω if the port is exposed.

15.4.6.10 Transmit and receive operating temperature range

Three temperature ranges for full operation compliance to the DSSS PHY are specified in Clause 13. Type 1 shall be defined as 0 °C to 40 °C, and is designated for office environments. Type 2 shall be defined as –20 °C to +50 °C, and Type 3 shall be defined as –30 °C to +70 °C. These are designated for industrial environments.

15.4.7 PMD transmit specifications

The following subclauses describe the transmit functions and parameters associated with the PMD sublayer.

15.4.7.1 Transmit power levels

The maximum allowable output power as measured in accordance with practices specified by the regulatory bodies is shown in Table 66. In the USA, the radiated emissions should also conform with the ANSI uncontrolled radiation emission standards (IEEE Std C95.1-1991).

Table 66—Transmit power levels

Maximum output power	Geographic location	Compliance document
1000 mW	USA	FCC 15.247
100 mW (EIRP)	Europe	ETS 300–328
10 mW/MHz	Japan	MPT ordinance for Regulating Radio Equipment, Article 49-20

15.4.7.2 Minimum transmitted power level

The minimum transmitted power shall be no less than 1 mW.

15.4.7.3 Transmit power level control

Power control shall be provided for transmitted power greater than 100 mW. A maximum of four power levels may be provided. At a minimum, a radio capable of transmission greater than 100 mW shall be capable of switching power back to 100 mW or less.

15.4.7.4 Transmit spectrum mask

The transmitted spectral products shall be less than –30 dBr (dB relative to the SINx/x peak) for f_c – 22 MHz < f < f_c –11 MHz, f_c +11 MHz < f < f_c + 22 MHz, –50 dBr for f < f_c –22 MHz, and f > f_c + 22 MHz, where f_c is the channel center frequency. The transmit spectral mask is shown in Figure 96. The measurements shall be made using 100 kHz resolution bandwidth and a 30 kHz video bandwidth.

15.4.7.5 Transmit center frequency tolerance

The transmitted center frequency tolerance shall be ±25 ppm maximum.

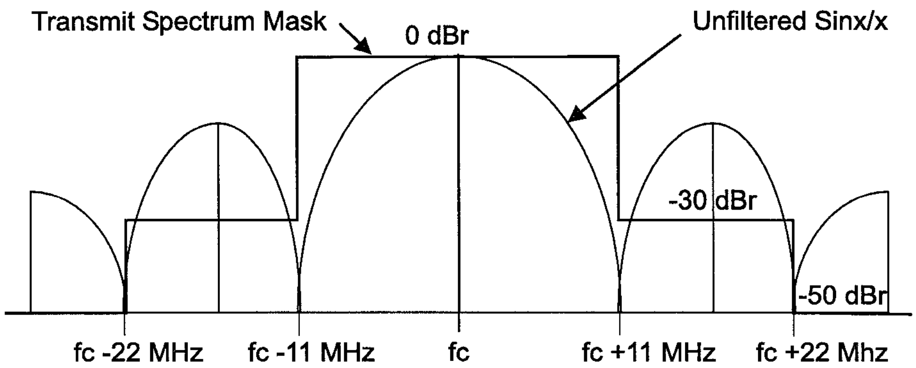


Figure 95—Transmit spectrum mask

15.4.7.6 Chip clock frequency tolerance

The PN code chip clock frequency tolerance shall be better than ±25 ppm maximum.

15.4.7.7 Transmit power-on and power-down ramp

The transmit power-on ramp for 10% to 90% of maximum power shall be no greater than 2 μs. The transmit power-on ramp is shown in Figure 96.

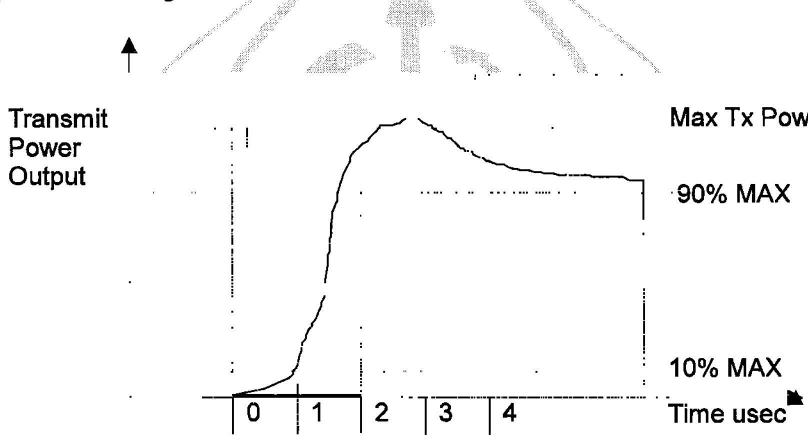


Figure 96—Transmit power-on ramp

The transmit power-down ramp for 90% to 10% maximum power shall be no greater than 2 μs. The transmit power down ramp is shown in Figure 97.

The transmit power ramps shall be constructed such that the DSSS PHY emissions conform with spurious frequency product specification defined in 15.4.6.5.

15.4.7.8 RF carrier suppression

The RF carrier suppression, measured at the channel center frequency, shall be at least 15 dB below the peak SIN(x)/x power spectrum. The RF carrier suppression shall be measured while transmitting a repetitive 01 data sequence with the scrambler disabled using DQPSK modulation. A 100 kHz resolution bandwidth shall be used to perform this measurement.

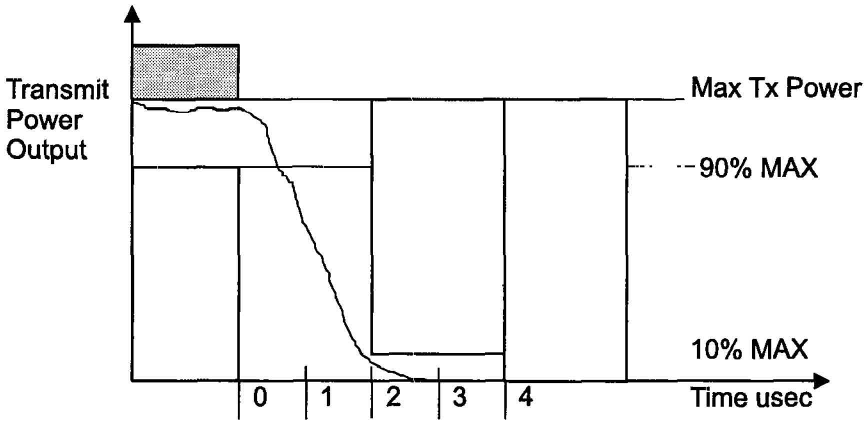


Figure 98—Transmit power-down ramp

15.4.7.9 Transmit modulation accuracy

The transmit modulation accuracy requirement for the DSSS PHY shall be based on the difference between the actual transmitted waveform and the ideal signal waveform. Modulation accuracy shall be determined by measuring the peak vector error magnitude measured during each chip period. Worst-Case vector error magnitude shall not exceeded 0.35 for the normalized sampled chip data. The ideal complex I and Q constellation points associated with DQPSK modulation (0.707,0.707), (0.707, –0.707), (–0.707, 0.707), (–0.707, –0.707) shall be used as the reference. These measurements shall be from baseband I and Q sampled data after recovery through a reference receiver system.

Figure 99 illustrates the ideal DQPSK constellation points and range of worst-case error specified for modulation accuracy.

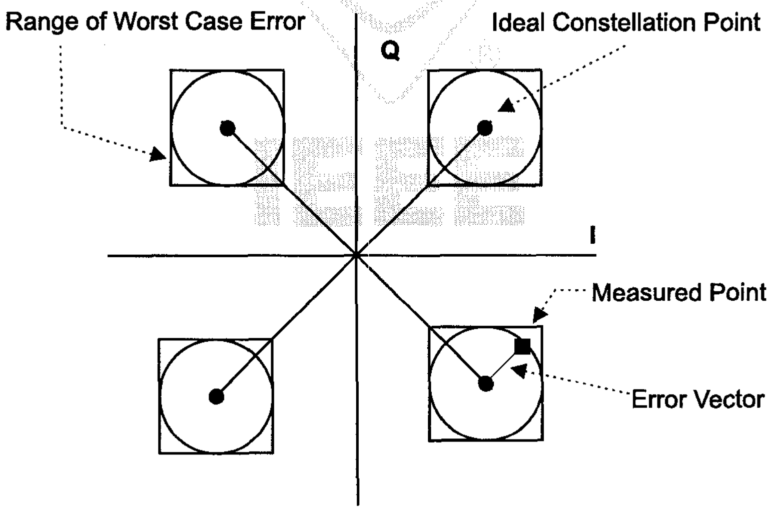


Figure 99—Modulation accuracy measurement example

Error vector measurement requires a reference receiver capable of carrier lock. All measurements shall be made under carrier lock conditions. The distortion induced in the constellation by the reference receiver shall be calibrated and measured. The test data error vectors described below shall be corrected to compensate for the reference receiver distortion.

The IEEE 802.11 vendor compatible radio shall provide an exposed TX chip clock, which shall be used to sample the I and Q outputs of the reference receiver.

The measurement shall be made under the conditions of continuous DQPSK transmission using scrambled all 1's.

The eye pattern of the I channel shall be used to determine the I and Q sampling point. The chip clock provided by the vendor radio shall be time delayed such that the samples fall at a 1/2 chip period offset from the mean of the zero crossing positions of the eye (see Figure 100). This is the ideal center of the eye and may not be the point of maximum eye opening.

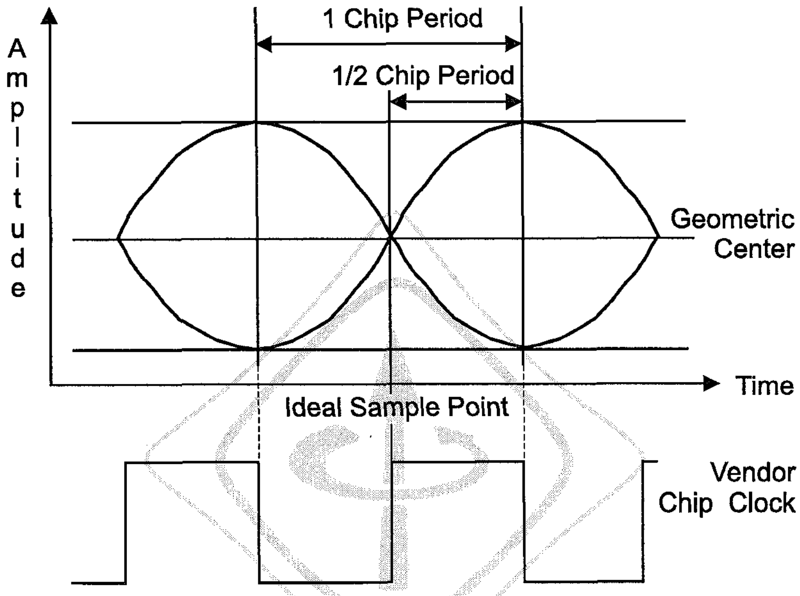


Figure 100—Chip clock alignment with baseband eye pattern

Using the aligned chip clock, 1000 samples of the I and Q baseband outputs from the reference receiver are captured. The vector error magnitudes shall be calculated as follows:

Calculate the dc offsets for I and Q samples.

$$I_{mean} = \sum_{n=0}^{1000} I(n)/1000$$

$$Q_{mean} = \sum_{n=0}^{1000} Q(n)/1000$$

Calculate the dc corrected I and Q samples for all n =1000 sample pairs.

$$I_{dc}(n) = I(n) - I_{mean}$$

$$Q_{dc}(n) = Q(n) - Q_{mean}$$

Calculate the average magnitude of I and Q samples.

$$I_{mag} = \sum_{n=0}^{1000} |I_{dc}(n)| / 1000$$

$$Q_{mag} = \sum_{n=0}^{1000} |Q_{dc}(n)| / 1000$$

Calculate the normalized error vector magnitude for the $I_{dc}(n)/Q_{dc}(n)$ pairs.

$$V_{err}(n) = \left[\frac{1}{2} \times (\{|I_{dc}(n)|/I_{mag}\}^2 + \{|Q_{dc}(n)|/Q_{mag}\}^2)\right]^{\frac{1}{2}} - V_{correction}$$

with $V_{correction}$ = error induced by the reference receiver system.

A vendor DSSS PHY implementation shall be compliant if for all n =1000 samples the following condition is met:

$V_{err}(n) < 0.35$

15.4.8 PMD receiver specifications

The following subclauses describe the receive functions and parameters associated with the PMD sublayer.

15.4.8.1 Receiver minimum input level sensitivity

The frame error ratio (FER) shall be less than 8×10^{-2} at an MPDU length of 1024 bytes for an input level of –80 dBm measured at the antenna connector. This FER shall be specified for 2 Mbit/s DQPSK modulation. The test for the minimum input level sensitivity shall be conducted with the energy detection threshold set less than or equal to –80 dBm.

15.4.8.2 Receiver maximum input level

The receiver shall provide a maximum FER of 8×10^{-2} at an MPDU length of 1024 bytes for a maximum input level of –4 dBm measured at the antenna. This FER shall be specified for 2 Mbit/s DQPSK modulation.

15.4.8.3 Receiver adjacent channel rejection

Adjacent channel rejection is defined between any two channels with ≥30 MHz separation in each channel group defined in 15.4.6.2.

The adjacent channel rejection shall be equal to or better than 35 dB with an FER of 8×10^{-2} using 2 Mbit/s DQPSK modulation described in 15.4.6.4 and an MPDU length of 1024 bytes.

The adjacent channel rejection shall be measured using the following method:

Input a 2 Mbit/s DQPSK modulated signal at a level 6 dB greater than specified in 15.4.8.1. In an adjacent channel (≥30 MHz separation as defined by the channel numbering), input a signal modulated in a similar fashion that adheres to the transmit mask specified in 15.4.7.4 to a level 41 dB above the level specified in 15.4.8.1. The adjacent channel signal shall be derived from a separate signal source. It cannot be a frequency shifted version of the reference channel. Under these conditions, the FER shall be no worse than 8×10^{-2}.

15.4.8.4 CCA

The DSSS PHY shall provide the capability to perform CCA according to at least one of the following three methods:

— *CCA Mode 1:* Energy above threshold. CCA shall report a busy medium upon detecting any energy above the ED threshold.
— *CCA Mode 2:* Carrier sense only. CCA shall report a busy medium only upon the detection of a DSSS signal. This signal may be above or below the ED threshold.
— *CCA Mode 3:* Carrier sense with energy above threshold. CCA shall report a busy medium upon the detection of a DSSS signal with energy above the ED threshold.

The energy detection status shall be given by the PMD primitive, PMD_ED. The carrier sense status shall be given by PMD_CS. The status of PMD_ED and PMD_CS is used in the PLCP convergence procedure to indicate activity to the MAC through the PHY interface primitive PHY-CCA.indicate.

A busy channel shall be indicated by PHY-CCA.indicate of class BUSY.

Clear channel shall be indicated by PHY-CCA.indicate of class IDLE.

The PHY MIB attribute aCCAModeSuprt shall indicate the appropriate operation modes. The PHY shall be configured through the PHY MIB attribute aCurrentCCAMode.

The CCA shall be TRUE if there is no energy detect or carrier sense. The CCA parameters are subject to the following criteria:

a) The energy detection threshold shall be less than or equal to –80 dBm for TX power > 100 mW, –76 dBm for 50 mW < TX power ≤ 100 mW, and –70 dBm for TX power ≤ 50 mW.
b) With a valid signal (according to the CCA mode of operation) present at the receiver antenna within 5 μs of the start of a MAC slot boundary, the CCA indicator shall report channel busy before the end of the slot time. This implies that the CCA signal is available as an exposed test point. Refer to Figure 47 for a definition of slot time boundary definition.
c) In the event that a correct PLCP Header is received, the DSSS PHY shall hold the CCA signal inactive (channel busy) for the full duration as indicated by the PLCP LENGTH field. Should a loss of carrier sense occur in the middle of reception, the CCA shall indicate a busy medium for the intended duration of the transmitted packet.

Conformance to DSSS PHY CCA shall be demonstrated by applying a DSSS compliant signal, above the appropriate ED threshold (a), such that all conditions described in b) and c) above are demonstrated.

16. Infrared (IR) PHY specification

16.1 Overview

The physical layer for the infrared system is specified in this clause. The IR PHY uses near-visible light in the 850 nm to 950 nm range for signaling. This is similar to the spectral usage of both common consumer devices such as infrared remote controls, as well as other data communications equipment, such as IrDA (Infrared Data Association) devices.

Unlike many other infrared devices, however, the IR PHY is not directed. That is, the receiver and transmitter do not have to be aimed at each other and do not need a clear line of sight. This permits the construction of a true LAN system, whereas with an aimed system, it would be difficult or impossible to install a LAN because of physical constraints.

A pair of conformant infrared devices would be able to communicate in a typical environment at a range up to about 10 m. The standard allows conformant devices to have more sensitive receivers, and this may increase range up to about 20 m.

The IR PHY relies on both reflected infrared energy as well as line-of-sight infrared energy for communications. Most designs anticipate that *all* of the energy at the receiver is reflected energy. This reliance on reflected infrared energy is called *diffuse infrared* transmission.

The standard specifies the transmitter and receiver in such a way that a conformant design will operate well in most environments where there is no line-of-sight path from the transmitter to the receiver. However, in an environment that has few or no reflecting surfaces, and where there is no line of sight, an IR PHY system may suffer reduced range.

The IR PHY will operate only in indoor environments. Infrared radiation does not pass through walls, and is significantly attenuated passing through most exterior windows. This characteristic can be used to "contain" an IR PHY in a single physical room, like a classroom or conference room. Different LANs using the IR PHY can operate in adjacent rooms separated only by a wall without interference, and without the possibility of eavesdropping.

At the time of this standard's preparation, the only known regulatory standards that apply to the use of infrared radiation are safety regulations, such as IEC 60825-1 [B2] and ANSI Z136.1 [B1]. While a conformant IR PHY device can be designed to also comply with these safety standards, conformance with this standard does not ensure conformance with other standards.

Worldwide, there are currently no frequency allocation or bandwidth allocation regulatory restrictions on infrared emissions.

Emitter (typically LED) and detector (typically PIN diode) devices for infrared communications are relatively inexpensive at the infrared wavelengths specified in the IR PHY, and at the electrical operating frequencies required by this PHY.

While many other devices in common use also use infrared emissions in the same optical band, these devices usually transmit infrared intermittently and do not interfere with the proper operation of a compliant IR PHY. If such a device does interfere, by transmitting continuously and with a very strong signal, it can be physically isolated (placing it in a different room) from the IEEE 802.11 LAN.

16.1.1 Scope

The PHY services provided to the IEEE 802.11 wireless LAN MAC by the IR system are described in this clause. The IR PHY layer consists of two protocol functions as follows:

a) A physical layer convergence function, which adapts the capabilities of the physical medium dependent (PMD) system to the PHY service. This function is supported by the physical layer convergence procedure (PLCP), which defines a method of mapping the IEEE 802.11 MAC sublayer protocol data units (MPDU) into a framing format suitable for sending and receiving user data and management information between two or more STAs using the associated PMD system.
b) A PMD system, whose function defines the characteristics of, and method of transmitting and receiving data through, the wireless medium (WM) between two or more STAs.

16.1.2 IR PHY functions

The IR PHY contains three functional entities: the PMD function, the physical layer convergence function, and the layer management function. Each of these functions is described in detail below.

The IR PHY service is provided to the MAC entity at the STA through a service access point (SAP) as described in Clause 12. For a visual guide to the relationship of the IR PHY to the remainder of a system, refer to Figure 11.

16.1.2.1 PLCP sublayer

To allow the IEEE 802.11 MAC to operate with minimum dependence on the PMD sublayer, a physical layer convergence sublayer is defined. This function simplifies the PHY service interface to the IEEE 802.11 MAC services. The PHY-specific preamble is normally associated with this convergence layer.

16.1.2.2 PMD sublayer

The PMD sublayer provides a clear channel assessment (CCA) mechanism, transmission mechanism, and reception mechanism that are used by the MAC via the PLCP to send or receive data between two or more STAs.

16.1.2.3 PHY management entity (PLME)

The PLME performs management of the local PHY functions in conjunction with the MAC management entity. Subclause 16.4 lists the MIB variables that may be accessed by the PHY sublayer entities and intralayer of higher layer management entities (LME). These variables are accessed via the PLME-GET, PLME-SET, and PLME-RESET primitives defined in Clause 10.

16.1.3 Service specification method and notation

The models represented by figures and state diagrams are intended as the illustrations of functions provided. It is important to distinguish between a model and a real implementation. The models are optimized for simplicity and clarity of presentation; the actual method of implementation is left to the discretion of the IEEE 802.11 IR PHY compliant developer. Conformance to the standard is not dependent on following the model, and an implementation that follows the model closely may not be conformant.

Abstract services are specified here by describing the service primitives and parameters that characterize each service. This definition is independent of any particular implementation. In particular, the PHY-SAP operations are defined and described as instantaneous; however, this may be difficult to achieve in an implementation.

16.2 IR PLCP sublayer

While the PLCP sublayer and the PMD sublayer are described separately, the separation and distinction between these sublayers is artificial, and is not meant to imply that the implementation must separate these functions. This distinction is made primarily to provide a point of reference from which to describe certain functional components and aspects of the PMD. The functions of the PLCP can be subsumed by a PMD sublayer; in this case, the PMD will incorporate the PHY-SAP as its interface, and will not offer a PMD-SAP.

16.2.1 Overview

A convergence procedure is provided by which MPDUs are converted to and from PLCPDUs. During transmission, the MPDU (PLCSDU) is prepended with a PLCP Preamble and PLCP Header to create the PLCPDU. At the receiver, the PLCP Preamble is processed and the internal data fields are processed to aid in demodulation and delivery of the MPDU (PSDU).

16.2.2 PLCP frame format

Figure 101 shows the format for the PLCPDU including the PLCP Preamble, the PLCP Header, and the PSDU. The PLCP Preamble contains the following fields: Synchronization (SYNC) and Start Frame Delimiter (SFD). The PLCP Header contains the following fields: Data Rate (DR), DC Level Adjustment (DCLA), Length (LENGTH), and Cyclic Redundancy Check (CRC). Each of these fields is described in detail in 16.2.4.

PLCP Preamble		PLCP Header				PSDU
SYNC	SFD	DR	DCLA	LENGTH	CRC	
57-73 slots	4 slots	3 slots	32 slots	16 bits	16 bits	variable number of octets

Figure 101—PLCPDU frame format

16.2.3 PLCP modulation and rate change

The PLCP Preamble shall be transmitted using the basic pulse defined in 16.3.3.2. The PLCSDU, LENGTH, and CRC fields shall be transmitted using pulse position modulation (PPM). PPM maps bits in the octet into symbols: 16-PPM maps four bits into a 16-position symbol, and 4-PPM maps two bits into a 4-position symbol. The basic L-PPM time unit is the slot. A slot corresponds to one of the L positions of a symbol and has a 250 ns duration. The PLCSDU, LENGTH, and CRC fields are transmitted at one of two bit rates: 1 Mbit/s or 2 Mbit/s. The Data Rate field indicates the data rate that will be used to transmit the PLCSDU, LENGTH, and CRC fields. The 1 Mbit/s data rate uses 16-PPM (basic access rate), and the 2 Mbit/s data rate uses 4-PPM (enhanced access rate). The transmitter and receiver will initiate the modulation or demodulation indicated by the DR field starting with the first 4 bits (in 16-PPM) or 2 bits (in 4-PPM) of the LENGTH field. The PSDU transmission rate is set by the DATARATE parameter in the PHY-TXSTART.request primitive. Any conformant IR PHY shall be capable of receiving at 1 Mbit/s and 2 Mbit/s. Transmission at 2 Mbit/s is optional.

A PHY-TXSTART.request that specifies a data rate which is not supported by a PHY instance will cause the PHY to indicate an error to its MAC instance. A PHY is not permitted under any circumstance to transmit at a different rate than the requested rate.

16.2.4 PLCP field definitions

16.2.4.1 PLCP Synchronization (SYNC) field

The SYNC field consists of a sequence of alternated presence and absence of a pulse in consecutive slots. The SYNC field has a minimum length of 57 L-PPM slots and a maximum length of 73 L-PPM slots and shall terminate with the absence of a pulse in the last slot. This field is provided so that the receiver can perform clock recovery (slot synchronization), automatic gain control (optional), signal-to-noise ratio estimation (optional), and diversity selection (optional).

The SYNC field is not modulated using L-PPM, but instead consists of transitions in L-PPM slots that would otherwise constitute an illegal symbol. See 16.3.2.1 for legal symbols.

16.2.4.2 PLCP Start Frame Delimiter (SFD) field

The SFD field length is four L-PPM slots and consists of the binary sequence 1001, where 1 indicates a pulse in the L-PPM slot and 0 indicates no pulse in the L-PPM slot. The leftmost bit shall be transmitted first. The SFD field is provided to indicate the start of the PLCP Preamble and to perform bit and symbol synchronization.

The SFD field is not modulated using L-PPM, but instead consists of transitions in L-PPM slots that would otherwise constitute an illegal symbol.

16.2.4.3 PLCP Data Rate (DR) field

The DR field indicates to the PHY the data rate that shall be used for the transmission or reception of the PLCSDU, LENGTH, and CRC fields. The transmitted value shall be provided by the PHY-TXSTART.request primitive as described in Clause 12. The DR field has a length of three L-PPM slots. The leftmost bit, as shown below, shall be transmitted first. The IR PHY currently supports two data rates defined by the slot pattern shown for the three L-PPM slots following the SFD, where 1 indicates a pulse in the L-PPM slot and 0 indicates no pulse in the L-PPM slot:

1 Mbit/s: 000
2 Mbit/s: 001

The DR field is not modulated using L-PPM, but instead consists of transitions in L-PPM slots that would otherwise constitute an illegal symbol.

16.2.4.4 PLCP DC Level Adjustment (DCLA) field

The DCLA field is required to allow the receiver to stabilize the dc level after the SYNC, SFD, and DR fields. The leftmost bit, as shown below, shall be transmitted first. The length of the DCLA field is 32 L-PPM slots and consists of the contents shown, where 1 indicates a pulse in the L-PPM slot and 0 indicates no pulse in the L-PPM slot:

1 Mbit/s: 00000000100000000000000010000000
2 Mbit/s: 00100010001000100010001000100010

The DCLA field is not modulated using L-PPM, but instead consists of transitions in L-PPM slots that would otherwise constitute an illegal symbol.

16.2.4.5 PLCP LENGTH field

The LENGTH field is an unsigned 16-bit integer that indicates the number of octets to be transmitted in the PSDU. The transmitted value shall be provided by the PHYTXSTART.request primitive as described in Clause 12. The lsb shall be transmitted first. This field is modulated and sent in L-PPM format. This field is protected by the CRC described in 16.2.4.6.

16.2.4.6 PLCP CRC field

The LENGTH field shall be protected by a 16-bit CRC-CCITT. The CRC-CCITT is the one's complement of the remainder generated by the modulo 2 division of the LENGTH field by the polynomial:

$$x^{16}+x^{12}+x^{5}+1$$

The protected bits will be processed in transmit order. The msb of the 16-bit CRC-CCITT shall be transmitted first. This field shall be modulated and sent in L-PPM format. All CRC-CCITT calculations shall be made prior to L-PPM encoding on transmission and after L-PPM decoding on reception.

16.2.4.7 PSDU field

This field is composed of a variable number of octets. The minimum is 0 (zero) and the maximum is 2500. The lsb of each octet shall be transmitted first. All the octets of this field shall be modulated and sent in L-PPM format.

16.2.5 PLCP procedures

16.2.5.1 PLCP transmit procedure

All commands issued by the MAC require that a confirmation primitive be issued by the PHY. The confirmation primitives provide flow control between the MAC and the PHY.

The steps below are the transmit procedure:

a) Based on the status of CCA, the MAC shall determine whether the channel is clear.
b) If the channel is clear, transmission of the PSDU shall be initiated by a PHY-TXSTART.request with parameters LENGTH and DATARATE.
c) The PHY entity shall immediately initiate transmission of the PLCP Preamble and PLCP Header based on the LENGTH and DATARATE parameters passed in the PHY-TXSTART.request. Once the PLCP Preamble and PLCP Header transmission is completed, the PHY entity shall issue a PHY-TXSTART.confirm.
d) Each octet of the PSDU is passed from the MAC to the PHY by a single PHY-DATA.request primitive. Each PHY-DATA.request shall be confirmed by the PHY with a PHY-DATA.confirm before the next request can be made.
e) At the PHY layer each PSDU octet shall be divided into symbols of 2 or 4 bits each. The symbols shall be modulated using L-PPM and transmitted into the medium.
f) Transmission is terminated by the MAC through the primitive PHY-TXEND.request. The PHY shall confirm the resulting end of transmission with a PHY-TXEND.confirm.

16.2.5.2 PLCP receive procedure

The steps below are the receive procedure:

a) CCA is provided to the MAC via the PHY-CCA.indicate primitive. When the PHY senses activity on the medium, it shall indicate that the medium is busy with a PHY-CCA.indicate with a value of BUSY. This will normally occur during the SYNC field of the PLCP Preamble.

b) The PHY entity shall begin searching for the SFD field. Once the SFD field is detected, the PHY entity shall attempt to receive the PLCP Header. After receiving the DR and DCLA fields, the PHY shall initiate processing of the received CRC and LENGTH fields. The data rate indicated in the DR field applies to all symbols in the latter part of the received PHYSDU, commencing with the first symbol of the LENGTH field. The CRC-CCITT shall be checked for correctness immediately after its reception.

c) If the CRC-CCITT check fails, or the value received in the DR field is not one supported by the PHY, then a PHY-RXSTART.indicate shall not be issued to the MAC. When the medium is again free, the PHY shall issue a PHY-CCA.indicate with a value of IDLE.

d) If the PLCP Preamble and PLCP Header reception is successful, the PHY shall send a PHY-RXSTART.indicate to the MAC; this includes the parameters DATARATE and LENGTH.

   In the absence of errors, the receiving PHY shall report the same length to its local MAC, in the RXVECTOR parameter of the PHY-RXSTART.indicate primitive, that the peer MAC presented to its local PHY entity in the TXVECTOR parameter of its respective PHY-TXSTART.request.

e) The received PLCSDU L-PPM symbols shall be assembled into octets and presented to the MAC using a series of PHY-DATA.indicate primitives, one per octet.

f) Reception shall be terminated after the reception of the final symbol of the last PLCSDU octet indicated by the PLCP Header's LENGTH field. After the PHY-DATA.indicate for that octet is issued, the PHY shall issue a PHY-RXEND.indicate primitive to its MAC.

g) After issuing the PHY-RXEND.indicate primitive, and when the medium is no longer busy, the PHY shall issue a PHY-CCA.indicate primitive with a value of IDLE.

16.2.5.3 CCA procedure

CCA is provided to the MAC via the PHY-CCA.indicate primitive.

The steps below are the CCA procedure:

a) When the PHY senses activity on the medium, a PHY-CCA.indicate primitive with a value of BUSY shall be issued. This will normally occur during reception of the SYNC field of the PLCP Preamble.

b) When the PHY senses that the medium is free, a PHY-CCA.indicate primitive with a value of IDLE shall be issued.

c) At any time, the MAC may issue a PHY-CCARESET.request primitive, which will reset the PHY's internal CCA detection mechanism to the medium not-busy (IDLE) state. This primitive will be acknowledged with a PHY-CCARESET.confirm primitive.

16.2.5.4 PMD_SAP peer-to-peer service primitive parameters

Several service primitives include a parameter vector. This vector shall be actually a list of parameters that may vary depending on PHY type. Table 67 indicates the parameters required by the MAC or IR PHY in each of the parameter vectors used for peer-to-peer interactions.

Table 67—IR PMD_SAP peer-to-peer service primitives

Parameter	Associated primitive	Value
LENGTH	RXVECTOR, TXVECTOR	4 to $2^{16} - 1$
DATARATE	RXVECTOR, TXVECTOR	PHY dependent

16.3 IR PMD sublayer

The IR PMD sublayer does not define PMD SAPs. The mechanism for communications between the PLCP and PMD sublayers, as well as the distinction between these two sublayers, if any, is left to implementors. In particular, it is possible to design and implement in a conformant way a single sublayer that subsumes the functions of both the PLCP and PMD, presenting only the PHY-SAP.

16.3.1 Overview

The PMD functional, electrical, and optical characteristics required for interoperability of implementations conforming to this specification are described in this subclause. The relationship of this specification to the entire IR physical layer is shown in Figure 11.

16.3.2 PMD operating specifications, general

General specifications for the IR PMD sublayer are provided in this subclause. These specifications apply to both the receive and transmit functions and general operation of a compliant IR PHY.

16.3.2.1 Modulation and channel data rates

Two modulation formats and data rates are specified for the IR PHY: a *basic access rate* and an *enhanced access rate*. The basic access rate is based on 1 Mbit/s 16-PPM modulation. The 16-PPM encoding is specified in Table 68. Each group of 4 data bits is mapped to one of the 16-PPM symbols. The enhanced access rate is based on 2 Mbit/s 4-PPM. The 4-PPM encoding is specified in Table 69. Each group of 2 data bits is mapped to one of the 4-PPM symbols. Transmission order of the symbol slots is from left to right, as shown in the table, where a 1 indicates in-band energy in the slot, and a 0 indicates the absence of in-band energy in the slot.

The data in Table 68 and Table 69 have been arranged (gray coded) so that a single out-of-position-by-one error in the medium, caused, for example, by intersymbol interference, results in only a single bit error in the received data, rather than in a multiple bit error.

Table 68—Sixteen-PPM basic rate mapping

Data	16-PPM symbol
0000	0000000000000001
0001	0000000000000010
0011	0000000000000100
0010	0000000000001000
0110	0000000000010000
0111	0000000000100000
0101	0000000001000000
0100	0000000010000000
1100	0000000100000000
1101	0000001000000000
1111	0000010000000000
1110	0000100000000000
1010	0001000000000000
1011	0010000000000000
1001	0100000000000000
1000	1000000000000000

Table 69—Four-PPM enhanced rate mapping

Data	4-PPM symbol
00	0001
01	0010
11	0100
10	1000

16.3.2.2 Octet partition and PPM symbol generation procedure

Since PPM is a block modulation method, with the block size less than a full octet, octets have to be partitioned prior to modulation (mapping into PPM symbols).

Octet partition depends on the PPM order being used.

Assume an octet is formed by eight bits numbered 7 6 5 4 3 2 1 0, where bit 0 is the lsb. Partition the octet as follows:

For 16-PPM, create two PPM symbols:

— The symbol using bits 3 2 1 0 shall be transmitted onto the medium first
— The symbol using bits 7 6 5 4 shall be transmitted onto the medium last

For 4-PPM, create four PPM symbols:

— The symbol using bits 1 0 shall be transmitted onto the medium first
— The symbol using bits 3 2 shall be transmitted onto the medium second
— The symbol using bits 5 4 shall be transmitted onto the medium third
— The symbol using bits 7 6 shall be transmitted onto the medium last

16.3.2.3 Operating environment

The IR PHY will operate only in indoor environments. IR PHY interfaces cannot be exposed to direct sunlight. The IR PHY relies on reflected infrared energy and does not require a line of sight between emitter and receiver in order to work properly. The range and bit-error-rate of the system may vary with the geometry of the environment and with natural and artificial illumination conditions.

16.3.2.4 Operating temperature range

The temperature range for full operation compliance with the IR PHY is specified as 0 °C to 40 °C.

16.3.3 PMD transmit specifications

The following subclauses describe the transmit functions and parameters associated with the PMD sublayer.

16.3.3.1 Transmitted peak optical power

The peak optical power of an emitted pulse shall be as specified in Table 70.

Table 70—Peak optical power as a function of emitter radiation pattern mask

Emitter radiation pattern mask	Peak optical power
Mask 1	2 W ± 20%
Mask 2	0.55 W ± 20%

16.3.3.2 Basic pulse shape and parameters

The basic pulse width, measured between the 50% amplitude points, shall be 250 ± 10 ns. The pulse rise time, measured between the 10% and 90% amplitude points, shall be no more than 40 ns. The pulse fall time, measured between the 10% and 90% amplitude points, shall be no more than 40 ns. The edge jitter, defined as the absolute deviation of the edge from its correct position, shall be no more than 10 ns. The basic pulse shape is shown in Figure 102.

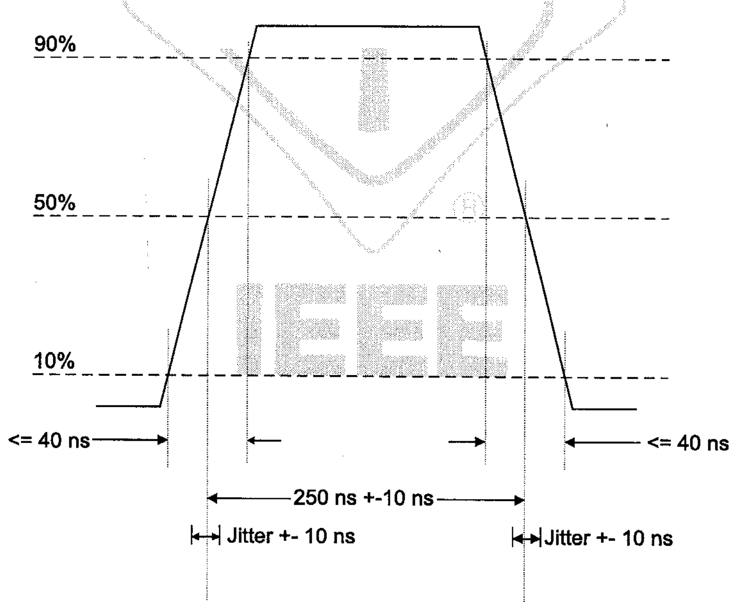


Figure 102—Basic pulse shape

16.3.3.3 Emitter radiation pattern mask

Currently the standard contains two emitter radiation pattern masks. Mask 1 is defined in Table 71 and illustrated in Figure 103. Mask 2 is defined in Table 72 and illustrated in Figure 105.

Table 71—Definition of the emitter radiation pattern mask 1

Declination angle	Normalized irradiance
$\alpha \leq 60°$	$> 3.5e{-}6$
$\alpha \leq 29°$	$\leq 2.2e{-}5$
$29° < \alpha \leq 43°$	$\leq -1.06e{-}4 + (0.44e{-}5)\ \alpha$
$43° < \alpha \leq 57°$	$\leq 1.15e{-}4 - (7.1e{-}7)\ \alpha$
$57° < \alpha \leq 74°$	$\leq 2.98e{-}4 - (3.9e{-}6)\ \alpha$
$74° < \alpha \leq 90°$	$\leq 4.05e{-}5 - (4.5e{-}7)\ \alpha$

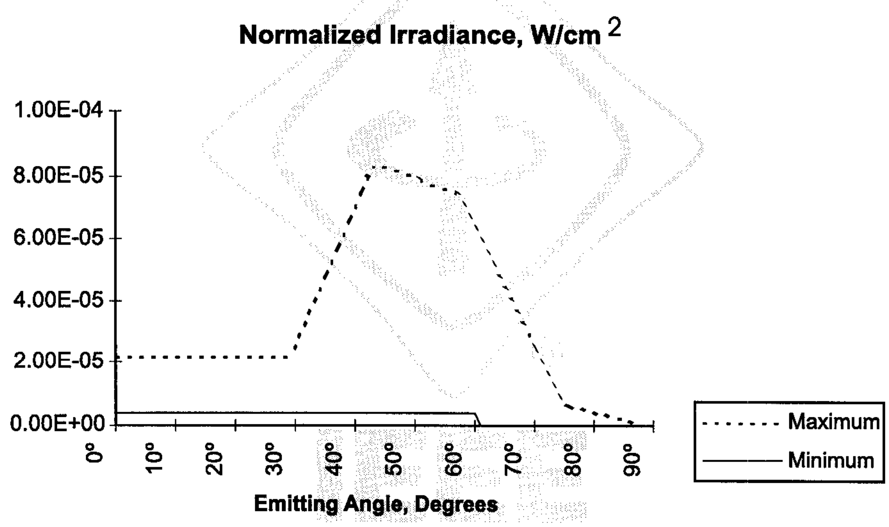


Figure 103—Emitter radiation pattern mask 1

Following is a description of how to interpret the Mask 1 table and figure. Position the conformant Mask 1 device in its recommended attitude. Define the conformant Mask 1 device axis as the axis passing through the emitter center and having the direction of the vertical from the floor. The mask represents the irradiance normalized to the total peak emitted power, as a function of the angle between the conformant Mask 1 device axis and the axis from the emitter center to the test receiver center (declination angle). The distance between emitter and test receiver is 1 m. The test receiver normal is always aimed at the emitter center. The azimuth angle is a rotation angle on the conformant device axis.

A device is conformant if for any azimuth angle its radiation pattern as a function of declination angle falls within the pattern mask.

Figure 104 is a description of how to interpret the Mask 2 table with reference to Figure 105.

Table 72—Definition of emitter radiation pattern mask 2

Declination angle	Pitch angle	Normalized irradiance
$\alpha \leq 60$	$\alpha = 0$	$0.05 \pm 15\%$
$\alpha \leq 90$	$\alpha = 0$	$0.025 \pm 15\%$
$\alpha \geq 100$	$\alpha = 0$	≤ 0.015
$0 \leq \alpha \leq 60$	$0 \leq \alpha \leq 10$	$0.035 \leq I \leq 0.055$
$0 \leq \alpha \leq 60$	$10 \leq \alpha \leq 20$	$0.0225 \leq I \leq 0.05$
$0 \leq \alpha \leq 60$	$\alpha \geq 30$	≤ 0.015

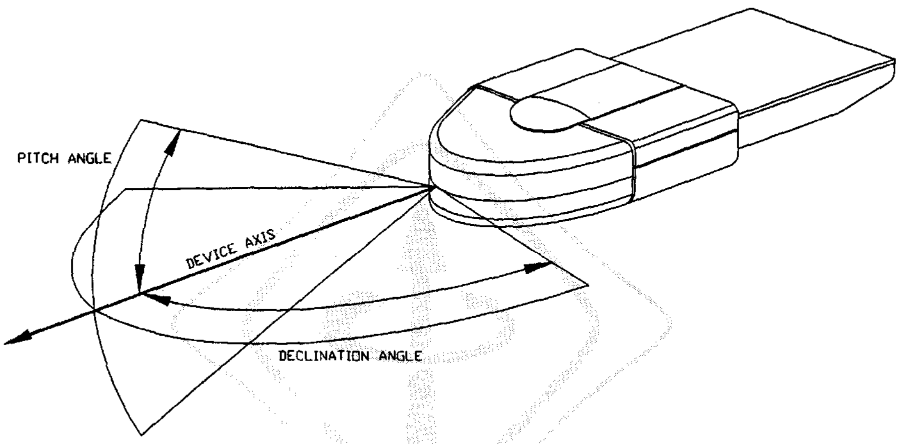


Figure 104—Mask 2 device orientation drawing

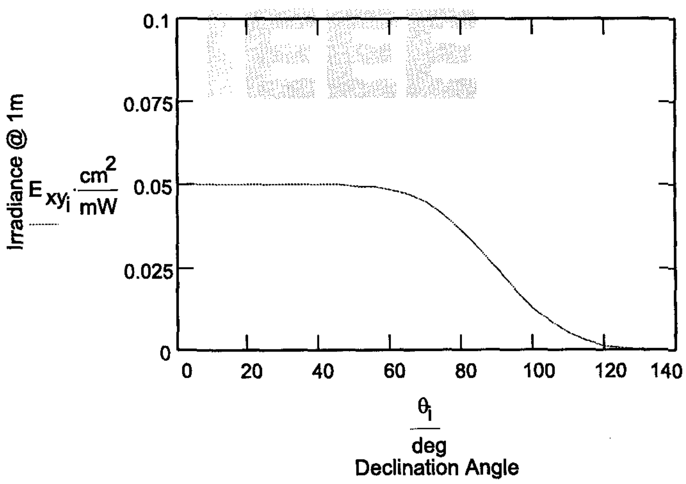


Figure 105—Emitter radiation pattern mask 2

Position the conformant Mask 2 device in its recommended attitude. Define the conformant Mask 2 device axis as passing through the emitter center and having the direction relative to the device as defined by the manufacturer. The declination angle plane is as defined by the manufacturer. The mask represents the irradiance normalized to the peak emitted power on the conformant Mask 2 device axis, as a function of the angle between the conformant device axis and the axis from the emitter center to the test receiver center (declination angle) in the declination plane. The distance between emitter and test receiver is 1 m. The test receiver normal is always aimed at the emitter center. The pitch angle is an angle relative to the conformant device axis which is perpendicular to the declination plane.

The device is conformant if, for a pitch angle of 0 degrees, at any declination angle from 0 to 100 degrees, and if, for any declination angle from 0 to 60 degrees, at any pitch angle from 0 to 20 degrees, its radiation pattern as a function of angle falls within the pattern mask.

Other radiation patterns are for future study.

16.3.3.4 Optical emitter peak wavelength

The optical emitter peak wavelength shall be between 850 and 950 nm.

16.3.3.5 Transmit spectrum mask

Define the transmit spectrum of a transmitter as the Fourier Transform, or equivalent, of a voltage (or current) signal whose amplitude, as a function of time, is proportional to the transmitted optical power.

The transmit spectrum of a conformant transmitter shall be 20 dB below its maximum for all frequencies above 15 MHz. The transmit spectrum mask is shown in Figure 106.

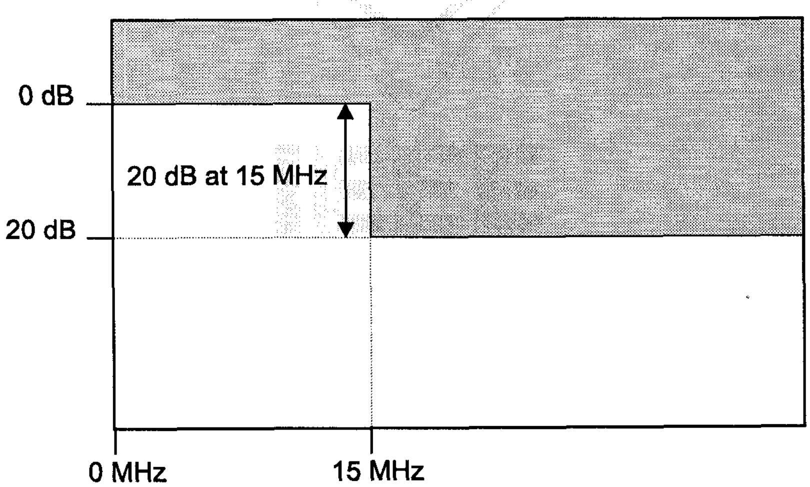


Figure 106—Transmit spectrum mask

16.3.4 PMD receiver specifications

The following subclauses describe the receive functions and parameters associated with the PMD sublayer.

16.3.4.1 Receiver sensitivity

The receiver sensitivity, defined as the minimum irradiance (in mW/cm^2) at the photodetector plane required for a frame error ratio (FER) of 4×10^{-5} with a PLCSDU of 512 octets and with an unmodulated background IR source between 800 nm and 1000 nm with a level of 0.1 mW/cm^2, shall be

1 Mbit/s: $2 \times 10^{-5} mW/cm^2$
2 Mbit/s: $8 \times 10^{-5} mW/cm^2$

16.3.4.2 Receiver dynamic range

The receiver dynamic range, defined as the ratio between the maximum and minimum irradiance at the plane normal to the receiver axis that assures an FER lower than or equal to 4×10^{-5} with a PLCSDU of 512 octets and with an unmodulated background IR source between 800 nm and 1000 nm with a level of 0.1 mW/cm^2, shall be ≥30 dB.

16.3.4.3 Receiver field-of-view (FOV)

The receiver axis is defined as the direction of incidence of the optical signal at which the received optical power is maximum.

The received optical power shall be greater than the values given in Table 73, at the angles indicated, where "angle of incidence" is the angle of incidence of the optical signal relative to the receiver axis, and "received power" is the received optical power as a percentage of that measured at the receiver axis.

Table 73—Definition of the receiver field of view

Angle of incidence	Received power
$\alpha \leq 20°$	≥ 65%
$\alpha \leq 40°$	≥ 55%
$\alpha \leq 60°$	≥ 35%
$\alpha \leq 80°$	≥ 10%

16.3.5 Energy Detect, Carrier Sense, and CCA definitions

16.3.5.1 Energy Detect (ED) signal

The ED signal shall be set true when IR energy variations in the band between 1 MHz and 10 MHz exceed 0.001 mW/cm^2.

The ED shall operate independently of the CS. ED shall not be asserted at the minimum signal level specified in 16.3.4.1, which is below the level specified in this subclause.

This signal is not directly available to the MAC.

16.3.5.2 Carrier Sense (CS) signal

The CS shall be asserted by the PHY when it detects and locks onto an incoming PLCP Preamble signal. Conforming PHYs shall assert this condition within the first 12 µs of signal reception, at the minimum signal level equal to the receiver sensitivity specified in 16.3.4.1, with a background IR level as specified in 16.3.4.1.

The CS shall be deasserted by the PHY when the receiving conformant device loses carrier lock.

NOTE—The 12 μs specification is somewhat less than the minimum length of the PLCP SYNC interval, which is 14.25 μs.

The CS shall operate independently of ED and shall not require a prior ED before the acquisition and assertion of CS. This permits reception of signals at the minimum signal level specified in 16.3.4.1, even though these signals fall below the ED level.

This signal is not directly available to the MAC.

16.3.5.3 CCA

CCA shall be asserted "IDLE" by the PHY when the CS and the ED are both false, or when ED has been continuously asserted for a period of time defined by the product of aCCAWatchdogTimerMax and aCCAWatchdogCountMax without CS becoming active. When either CS or ED go true, CCA is indicated as "BUSY" to the MAC via the primitive PHY-CCA.indicate. CS and DE behavior are defined in 16.3.5.2.

Normally, CCA will be held "BUSY" throughout the period of the PLCP Header. After receiving the last PLCP bit and the first data octet, the PHY shall signal PHY-RXSTART.indicate with the parameters LENGTH and RATE. CCA shall be held "BUSY" until the number of octets specified in the decoded PLCP Header are received. At that time the PHY shall signal PHY-RXEND.indicate. The CCA may remain "BUSY" after the end of data if some form of energy is still being detected. The PHY will signal PHY-CCA.indicate with a value of IDLE only when the CCA goes "CLEAR."

The transition of CCA from "BUSY" to "IDLE" is indicated to the MAC via the primitive PHY-CCA.indicate.

If CS and ED go false before the PHY signals PHY-RXSTART.indicate, CCA is set to "IDLE" and *immediately* signaled to the MAC via PHY-CCA.indicate with a value of IDLE. If CS and ED go false after the PHY has signaled PHY-RXSTART.indicate, implying that the PLCP Header has been properly decoded, then the PHY shall not signal a change in state of CCA until the proper interval has passed for the number of octets indicated by the received PLCP LENGTH. At that time, the PHY shall signal PHY-RXEND.indicate with an RXERROR parameter of CarrierLost followed by PHY-CCA.indicate with a value of IDLE.

The transition of CCA from "CLEAR" to "BUSY" resets the CCA watchdog timer and CCA watchdog counter. aCCAWatchdogTimerMax and aCCAWatchdogCountMax are parameters available via MIB entries and can be read and set via the LME.

Rise and fall times of CCA relative to the OR'ing of the CS and ED signals shall be less than 30 ns. CS and ED are both internal signals to the PHY and are not available directly to the MAC, nor are they defined at any exposed interface.

16.3.5.4 CHNL_ID

For the IR PHY, CHNL_ID = X'01' is defined as the baseband modulation method. All other values are not defined.

16.4 PHY attributes

PHY attributes have allowed values and default values that are PHY-dependent. Table 74 describes those values, and further specifies whether they are permitted to vary from implementation to implementation.

Table 74 does not provide the definition of the attributes, but only provides the IR PHY-specific values for the attributes whose definitions are in Clause 13 of this standard.

Table 74—IR PHY MIB attributes

PHY MIB object	Default value	Operational semantics	Operational behavior
aCCATime	5 μs	Static	Identical for all conformant PHY
aRxTxTurnaroundTime	0 μs	Static	Identical for all conformant PHY
aSlotTime	8 μs	Static	Identical for all conformant PHY
aRxTxSwitchTime	0	Static	Identical for all conformant PHY
aTxRampOnTime	0	Static	Identical for all conformant PHY
aRxPLCPDelay	1 μs	Static	Identical for all conformant PHY
aTxPLCPDelay	Implementation dependent	Static	Identical for all conformant PHY
aRxRFDelay	Implementation dependent	Static	Identical for all conformant PHY
aTxRFDelay	Implementation dependent	Static	Identical for all conformant PHY
aCCAWatchdogTimerMax	Implementation dependent	Dynamic	A conformant PHY may set this via the LME
aCCAWatchdogCountMax	Implementation dependent	Dynamic	A conformant PHY may set this via the LME
aCCAWatchdogTimerMin	22 μs	Static	Identical for all conformant PHY
aCCAWatchdogCountMin	1	Static	Identical for all conformant PHY
aMACProcessingtDelay	2 μs	Static	Identical for all conformant PHY
aTxRampOffTime	0 μs	Static	Identical for all conformant PHY
aMPDUMaxLength	2500 octets	Static	Identical for all conformant PHY
aSIFSTime	7 μs	Static	Identical for all conformant PHY
aSupportedRatesTx	Implementation dependent	Static	All conformant PHY must include the value X'02' (1 Mbit/s).
aSupportedRatesRx	Implementation dependent	Static	All conformant PHY must include the values X'02' (1 Mbit/s) and X'04' (2 Mbit/s).
aPHYType	03	Static	Identical for all conformant PHY
aCWmin	63	Static	Identical for all conformant PHY
aCWmax	1023	Static	Identical for all conformant PHY
aPLCPHeaderLength	41 μs (1 Mbit/s) 25 μs (2 Mbit/s)	Static	Identical for all conformant PHY
aPreambleLength	16 μs (1 Mbit/s) 20 μs (2 Mbit/s)	Static	Identical for all conformant PHY

Annex A

(normative)

Protocol Implementation Conformance Statement (PICS) proforma

A.1 Introduction

The supplier of a protocol implementation that is claimed to conform to IEEE Std 802.11-1997 shall complete the following PICS proforma.

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 PICS proformas);
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 labeled by the same numeral <n> is required
pred:	conditional symbol, including predicate identification

A.2.2 General abbreviations

N/A	not applicable
AD	address function capability
CF	implementation under test (IUT) configuration
FR	MAC frame capability
FS	frame sequence capability
PC	protocol capability
PICS	protocol implementation conformance statement

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 questionnaire, divided into 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 a range of values. (Note that there are some items where two or more choices from a set of possible answers may apply. All relevant choices are to be marked, in these cases.)

Each item is identified by an item reference in the first column. The second column contains the question to be answered. The third column contains the reference or references to the material that specifies the item in the main body of IEEE Std 802.11-1997. The remaining columns record the status of each item, i.e., whether support is mandatory, optional, or conditional, and provide the space for the answers (see also A.3.4). Marking an item as supported is to be interpreted as a statement that all relevant requirements of the subclauses and normative annexes, cited in the References column for the item, are met by the implementation.

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 labeled A<*I*> or X<*I*>, 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 or presentation.

The PICS proforma for a station consists of A.4.1, through A.4.4 inclusive, and at least one of A.4.5, A.4.6, or A.4.7 corresponding to the PHY implemented.

A completed PICS proforma, including any Additional Information and Exception Information, is the PICS 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 capabilities, if this 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 in the interpretation of the PICS. It is not intended or expected that a large quantity of information will be supplied, and a PICS can be considered complete without any such information. Examples of such Additional Information might be a outline of the ways in which an (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 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 happen occasionally 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 preprinted 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 X<*i*> reference to an item of Exception Information, and shall provide the appropriate rationale in the Exception Information item itself.

An implementation for which an Exception Information item is required in this way does not conform to IEEE Std 802.11-1997.

NOTE—A possible reason for the situation described above is that a defect in IEEE Std 802.11-1997 has been reported, a correction for which is expected to change the requirement not met by the implementation.

A.3.4 Conditional status

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 below, and "<S>" is one of the status symbols M or O.

If the value of the predicate is true, the conditional item is applicable, and its status is given by S: the support column is to be completed in the usual way. Otherwise, the conditional item is not relevant and the Not Applicable (N/A) answer is to be marked.

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 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, and is false otherwise.

Each item referenced in a predicate, or in a preliminary question for grouped conditional items, is indicated by an asterisk in the Item column.

A.4 PICS proforma—IEEE Std 802.11-1997[7]

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 the machines and/or operating systems(s), system names	

NOTES

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.

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 802.11-1997

Identification of protocol standard	IEEE Std 802.11-1997
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this 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 802.11-1997.)	Yes ❑ No ❑

Date of statement (dd/mm/yy)	

[7] *Copyright release for PICS proforma:* 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.4.3 IUT configuration

Item	IUT configuration	References	Status	Support
	What is the configuration of the IUT?			
* CF1	Access Point (AP)	5.2	O.1	Yes ❑ No ❑
* CF2	Independent station (*not* an AP)	5.2	O.1	Yes ❑ No ❑
* CF3	Frequency-Hopping spread spectrum (FHSS) PHY for the 2.4 GHz band		O.2	Yes ❑ No ❑
* CF4	Direct Sequence Spread Spectrum (DSSS) PHY for the 2.4 GHz band		O.2	Yes ❑ No ❑
* CF5	Infrared PHY		O.2	Yes ❑ No ❑

A.4.4 MAC protocol

A.4.4.1 MAC protocol capabilities

Item	Protocol capability	References	Status	Support
	Are the following MAC protocol capabilities supported?			
PC1	Authentication service	5.4.3.1, 5.4.3.2, 5.7.6, 5.7.7, 8.1, Annex C	M	Yes ❑ No ❑
PC1.1	Authentication state	5.5	M	Yes ❑ No ❑
PC1.2	Open System authentication	8.1.1	M	Yes ❑ No ❑
PC1.3	Shared Key authentication	8.1.2, 8.3	PC2:M	Yes ❑ No ❑ N/A ❑
* PC2	WEP algorithm	5.4.3.3, 8.2, Annex C	O	Yes ❑ No ❑
PC2.1	WEP Encryption procedure	8.2.3, 8.2.4, 8.2.5	PC2:M	Yes ❑ No ❑ N/A ❑
PC2.2	WEP Decryption procedure	8.2.3, 8.2.4, 8.2.5	PC2:M	Yes ❑ No ❑ N/A ❑
PC2.3	Security services management	8.3	M	Yes ❑ No ❑
PC3	Distributed Coordination function	9.1, 9.2, Annex C	M	Yes ❑ No ❑
PC3.1	Net Allocation Vector (NAV) function	9.2.1, 9.2.5, 9.3.2.2	M	Yes ❑ No ❑
PC3.2	Interframe space usage and timing	9.2.3, 9.2.5, 9.2.10	M	Yes ❑ No ❑
PC3.3	Random Backoff function	9.2.4	M	Yes ❑ No ❑
PC3.4	DCF Access procedure	9.2.5.1, 9.2.5.5	M	Yes ❑ No ❑
PC3.5	Random Backoff procedure	9.2.5.2	M	Yes ❑ No ❑
PC3.6	Recovery procedures and retransmit limits	9.2.5.3	M	Yes ❑ No ❑
PC3.7	RTS/CTS procedure	9.2.5.4, 9.2.5.6, 9.2.5.7	M	Yes ❑ No ❑

A.4.4.1 MAC protocol capabilities *(continued)*

Item	Protocol capability	References	Status	Support
PC3.8	Directed MPDU transfer	9.2.6	M	Yes ❑ No ❑
PC3.9	Broadcast and multicast MPDU transfer	9.2.7	M	Yes ❑ No ❑
PC3.10	MAC level acknowledgment	9.2.2, 9.2.8	M	Yes ❑ No ❑
PC3.11	Duplicate detection and recovery	9.2.9	M	Yes ❑ No ❑
* PC4	Point coordinator (PC)	9.1, 9.3, Annex C	CF1:O	Yes ❑ No ❑ N/A ❑
PC4.1	Maintenance of CFP structure and timing	9.3.1, 9.3.2	PC4:M	Yes ❑ No ❑ N/A ❑
PC4.2	PCF MPDU transfer from PC	9.3.3	PC4:M	Yes ❑ No ❑ N/A ❑
* PC4.3	PCF MPDU transfer to PC	9.3.3	PC4:O	Yes ❑ No ❑ N/A ❑
PC4.4	Overlapping PC provisions	9.3.3.2	PC4:M	Yes ❑ No ❑ N/A ❑
PC4.5	Polling list maintenance	9.3.4	PC4.3:M	Yes ❑ No ❑ N/A ❑
* PC5	CF-Pollable	9.1, 9.3, Annex C	CF2:O	Yes ❑ No ❑ N/A ❑
PC5.1	Interpretation of CFP structure and timing	9.3.1, 9.3.2	PC5:M	Yes ❑ No ❑ N/A ❑
PC5.2	PCF MPDU transfer to/from and CF-Pollable STA	9.3.3	PC5:M	Yes ❑ No ❑ N/A ❑
PC5.3	Polling list update	9.3.4	PC5:M	Yes ❑ No ❑ N/A ❑
PC6	Fragmentation	9.2, 9.4, Annex C	M	Yes ❑ No ❑
PC7	Defragmentation	9.2, 9.5, Annex C	M	Yes ❑ No ❑
PC8	MAC data service	9.1.5, 9.8, Annex C	M	Yes ❑ No ❑
PC8.1	Reorderable-Multicast service class	9.8	M	Yes ❑ No ❑
PC8.2	StrictlyOrdered service class	9.8	O	Yes ❑ No ❑
PC9	Multirate support	9.6, Annex C	M	Yes ❑ No ❑
* PC10	Multiple outstanding MSDU support	9.8, Annex C	O	Yes ❑ No ❑
PC10.1	Multiple outstanding MSDU transmission restrictions	9.8	PC10:M	Yes ❑ No ❑ N/A ❑
PC11	Timing synchronization	11.1, Annex C	M	Yes ❑ No ❑
PC11.1	Timing in an infrastructure network	11.1.1.1, 11.1.4	CF1:M	Yes ❑ No ❑ N/A ❑
PC11.2	Timing in an Independent BSS (IBSS)	11.1.1.2, 11.1.4	CF2:M	Yes ❑ No ❑ N/A ❑
PC11.3	Beacon Generation function	11.1.2	M	Yes ❑ No ❑ N/A ❑
PC11.5	TSF synchronization and accuracy	11.1.2	M	Yes ❑ No ❑
PC11.5	Infrastructure BSS initialization	11.1.3	CF1:M	Yes ❑ No ❑ N/A ❑
PC11.6	Independent BSS initialization	11.1.3	CF2:M	Yes ❑ No ❑ N/A ❑
PC11.7	Passive scanning	11.1.3	CF2:M	Yes ❑ No ❑ N/A ❑
PC11.8	Active scanning	11.1.3	CF2:M	Yes ❑ No ❑ N/A ❑

A.4.4.1 MAC protocol capabilities *(continued)*

Item	Protocol capability	References	Status	Support
PC11.9	Probe response	11.1.3	M	Yes ❑ No ❑
PC11.10	Hop Synchronization function	11.1.5	CF3:M	Yes ❑ No ❑ N/A ❑
PC12	Infrastructure power management	11.2.1, Annex C	M	Yes ❑ No ❑
PC12.1	Station power management modes	11.2.1.1, 11.2.1.8	CF2:M	Yes ❑ No ❑ N/A ❑
PC12.2	TIM transmission	11.2.1.2, 11.2.1.3	CF1:M	Yes ❑ No ❑ N/A ❑
PC12.3	AP function during CP	11.2.1.4	CF1:M	Yes ❑ No ❑ N/A ❑
PC12.4	AP function during CFP	11.2.1.5	PC4:M	Yes ❑ No ❑ N/A ❑
PC12.5	Receive function during CP	11.2.1.6	CF2:M	Yes ❑ No ❑ N/A ❑
PC12.6	Receive function during CFP	11.2.1.7	PC5:M	Yes ❑ No ❑ N/A ❑
PC12.7	Aging function	11.2.1.9	CF1:M	Yes ❑ No ❑ N/A ❑
PC13	IBSS power management	11.2.2, Annex C	CF2:M	Yes ❑ No ❑ N/A ❑
PC13.1	Initialization of power management	11.2.2.2	CF2:M	Yes ❑ No ❑ N/A ❑
PC13.2	STA power state transitions	11.2.2.3	CF2:M	Yes ❑ No ❑ N/A ❑
PC13.3	ATIM and frame transmission	11.2.2.4	CF2:M	Yes ❑ No ❑ N/A ❑
PC14	Association and reassociation	5.4, 5.7, 11.3, Annex C	M	Yes ❑ No ❑
PC14.1	Association state	5.5	M	Yes ❑ No ❑
PC14.2	STA association procedure	11.3.1	CF2:M	Yes ❑ No ❑ N/A ❑
PC14.3	AP association procedure	11.3.2	CF1:M	Yes ❑ No ❑ N/A ❑
PC14.4	STA reassociation procedure	11.3.3	CF2:M	Yes ❑ No ❑ N/A ❑
PC14.5	AP reassociation procedure	11.3.4	CF1:M	Yes ❑ No ❑ N/A ❑
PC15	Management information base (MIB)	11.4, Annex C	M	Yes ❑ No ❑
PC15.1	SMT object class	11.4.2.1	M	Yes ❑ No ❑
* PC15.2	Privacy package	11.4.2.1	PC2:M	Yes ❑ No ❑ N/A ❑
PC15.3	MAC object class	11.4.2.2	M	Yes ❑ No ❑
* PC15.4	MAC statistics package	11.4.2.2	O	Yes ❑ No ❑
PC15.3	Resource type object class	11.4.2.3	M	Yes ❑ No ❑

A.4.4.2 MAC frames

Item	MAC frame	References	Status	Support
	Is transmission of the following MAC frames supported?	7, Annex C		
FT1	Association request	7	CF2:M	Yes ❑ No ❑ N/A ❑
FT2	Association response	7	CF1:M	Yes ❑ No ❑ N/A ❑
FT3	Reassociation request	7	CF2:M	Yes ❑ No ❑ N/A ❑
FT4	Reassociation response	7	CF1:M	Yes ❑ No ❑ N/A ❑
FT5	Probe request	7	CF2:M	Yes ❑ No ❑ N/A ❑
FT6	Probe response	7	M	Yes ❑ No ❑
FT7	Beacon	7	M	Yes ❑ No ❑
FT8	ATIM	7	CF2:M	Yes ❑ No ❑ N/A ❑
FT9	Disassociation	7	M	Yes ❑ No ❑
FT10	Authentication	7	M	Yes ❑ No ❑
FT11	Deauthentication	7	M	Yes ❑ No ❑
FT12	PS-Poll	7	CF2:M	Yes ❑ No ❑ N/A ❑
FT13	RTS	7	M	Yes ❑ No ❑
FT14	CTS	7	M	Yes ❑ No ❑
FT15	ACK	7	M	Yes ❑ No ❑
FT16	CF-End	7	PC4:M	Yes ❑ No ❑ N/A ❑
FT17	CF End+CF-Ack	7	PC4:M	Yes ❑ No ❑ N/A ❑
FT18	Data	7	M	Yes ❑ No ❑
FT19	Data + CF-Ack	7	(PC4 OR PC5):M	Yes ❑ No ❑ N/A ❑
FT20	Data + CF-Poll	7	PC4.3:M	Yes ❑ No ❑ N/A ❑
FT21	Data + CF-Ack+CF-Poll	7	PC4.3:M	Yes ❑ No ❑ N/A ❑
FT22	Null	7	M	Yes ❑ No ❑
FT23	CF-Ack (no data)	7	(PC4 OR PC5):M	Yes ❑ No ❑ N/A ❑
FT24	CF-Poll (no data)	7	PC4.3:M	Yes ❑ No ❑ N/A ❑
FT25	CF-Ack+CF-Poll (no data)	7	PC4.3:M	Yes ❑ No ❑ N/A ❑
	Is reception of the following MAC frames supported?	7, Annex C		
FR1	Association request	7	CF1:M	Yes ❑ No ❑ N/A ❑
FR2	Association response	7	CF2:M	Yes ❑ No ❑ N/A ❑
FR3	Reassociation request	7	CF1:M	Yes ❑ No ❑ N/A ❑
FR4	Reassociation response	7	CF2:M	Yes ❑ No ❑ N/A ❑
FR5	Probe request	7	M	Yes ❑ No ❑
FR6	Probe response	7	M	Yes ❑ No ❑
FR7	Beacon	7	M	Yes ❑ No ❑
FR8	ATIM	7	CF2:M	Yes ❑ No ❑ N/A ❑
FR9	Disassociation	7	M	Yes ❑ No ❑
FR10	Authentication	7	M	Yes ❑ No ❑

A.4.4.2 MAC frames *(continued)*

Item	MAC frame	References	Status	Support
FR11	Deauthentication	7	M	Yes ❑ No ❑
FR12	PS-Poll	7	CF1:M	Yes ❑ No ❑ N/A ❑
FR13	RTS	7	M	Yes ❑ No ❑
FR14	CTS	7	M	Yes ❑ No ❑
FR15	ACK	7	M	Yes ❑ No ❑
FR16	CF-End	7	M	Yes ❑ No ❑
FR17	CF End+CF-Ack	7	M	Yes ❑ No ❑
FR18	Data	7	M	Yes ❑ No ❑
FR19	Data + CF-Ack	7	M	Yes ❑ No ❑
FR20	Data + CF-Poll	7	PC5:M	Yes ❑ No ❑ N/A ❑
FR21	Data + CF-Ack+CF-Poll	7	PC5:M	Yes ❑ No ❑ N/A ❑
FR22	Null	7	M	Yes ❑ No ❑
FR23	CF-Ack (no data)	7	(PC4 OR PC5):M	Yes ❑ No ❑ N/A ❑
FR24	CF-Poll (no data)	7	PC5:M	Yes ❑ No ❑ N/A ❑
FR25	CF-Ack+CF-Poll (no data)	7	PC5:M	Yes ❑ No ❑ N/A ❑

A.4.4.3 Frame exchange sequences

Item	Frame exchange sequence	References	Status	Support
	Are the following frame sequences supported?			
FS1	Basic frame sequences	9.7, Annex C	M	Yes ❑ No ❑
FS2	CF-Frame sequences	9.7, Annex C	(PC4 OR PC5):M	Yes ❑ No ❑ N/A ❑

A.4.4.4 MAC addressing functions

Item	MAC Address function	References	Status	Support
	Are the following MAC Addressing functions supported?			
AD1	STA universal individual IEEE802 address	5.3.3, 7.1.3.3	M	Yes ❑ No ❑
AD2	BSS identifier generation	7.1.3.3, 11.1.3, Annex C	M	Yes ❑ No ❑
AD3	Receive address matching	7.1.3.3, 7.2.2, Annex C	M	Yes ❑ No ❑ N/A ❑

A.4.5 Frequency-Hopping PHY functions

Item	Protocol feature	References	Status	Support
	Which requirements and options does the PHY support?			
FH1	PHY service primitive parameters			
FH1.1	TXVECTOR parameter: LENGTH	14.2.2.1	M	Yes ❑ No ❑
FH1.2	TXVECTOR parameter: PLCPBITRATE	14.2.2.2	M	Yes ❑ No ❑
FH1.2.1	PLCPBITRATE = X'00' (1.0 Mbit/s)	14.2.2.2	M	Yes ❑ No ❑
* FH1.2.2	PLCPBITRATE = X'02' (2.0 Mbit/s)	14.2.2.2	O	Yes ❑ No ❑
FH1.3	RXVECTOR parameter: LENGTH	14.2.3.1	M	Yes ❑ No ❑
FH1.4	RXVECTOR parameter: RSSI	14.2.3.2	O	Yes ❑ No ❑
FH2	PLCP frame format			
FH2.1	PLCP Preamble: Sync	14.3.2.1.1	M	Yes ❑ No ❑
FH2.2	PLCP Preamble: Start Frame Delimiter	14.3.2.1.2	M	Yes ❑ No ❑
FH2.3	PLCP Header: Length Word	14.3.2.2.1	M	Yes ❑ No ❑
FH2.4	PLCP Header: Signaling field	14.3.2.2.2	M	Yes ❑ No ❑
FH2.5	PLCP Header: Header Error Check	14.3.2.2.3	M	Yes ❑ No ❑
FH2.6	PLCP Data Whitener: Scrambling and bias suppression encoding	14.3.2.3, 14.3.3.1.1	M	Yes ❑ No ❑
FH3	PLCP Transmit procedure			
FH3.1	Transmit: transmit on MAC request	14.3.3.1.1	M	Yes ❑ No ❑
FH3.2	Transmit: format and whiten frame	14.3.3.1.1	M	Yes ❑ No ❑
FH3.3	Transmit: Timing	14.3.3.1.1	M	Yes ❑ No ❑
FH4	PLCP CS/CCA procedure			
FH4.1	CS/CCA: perform on a minimum of one antenna	14.3.3.2.1	M	Yes ❑ No ❑
FH4.2.	CS/CCA: Detect preamble starting up to 20 µs after start of slot time	14.3.3.2.1	M	Yes ❑ No ❑
FH4.3	CS/CCA: Detect preamble starting at least 16 µs prior to end of slot time	14.3.3.2.1	M	Yes ❑ No ❑
FH4.4	CS/CCA: Detect random data	14.3.3.2.1	M	Yes ❑ No ❑
FH4.5	CS/CCA: Perform on antenna with essentially same gain and pattern as transmit antenna	14.3.3.2.1	M	Yes ❑ No ❑
FH4.6	CS/CCA: Detect valid SFD and PLCP header	14.3.3.2.1	M	Yes ❑ No ❑
FH4.7	CS/CCA: Maintain BUSY indication until end of length contained in valid PLCP header	14.3.3.2.1	M	Yes ❑ No ❑
FH5	PLCP Receive procedure			
FH5.1	Receive: Receive and dewhiten frame	14.3.3.3.1	M	Yes ❑ No ❑
FH6	PHY LME			
FH6.1	PLME: Support FH sync	14.4.2.2	M	Yes ❑ No ❑
FH6.2	PLME: Support PLME primitives	14.4.3.2	O	Yes ❑ No ❑

A.4.5 Frequency-Hopping PHY functions *(continued)*

Item	Protocol feature	References	Status	Support
FH7	Geographic area specific requirements			
* FH7.1	Geographic areas			
FH7.1.1	North America	14.6.2	O.1	Yes ❑ No ❑
FH7.1.2	Most of Europe	14.6.2	O.1	Yes ❑ No ❑
FH7.1.3	Japan	14.6.2	O.1	Yes ❑ No ❑
FH7.1.4	Spain	14.6.2	O.1	Yes ❑ No ❑
FH7.1.5	France	14.6.2	O.1	Yes ❑ No ❑
FH7.2	Operating frequency range	14.6.3	FH7.1:M	Yes ❑ No ❑
FH7.3	Number of operating channels	14.6.4	FH7.1:M	Yes ❑ No ❑
FH7.4	Operating channel frequencies	14.6.5	FH7.1:M	Yes ❑ No ❑
FH7.5	Occupied channel bandwidth	14.6.6	FH7.1:M	Yes ❑ No ❑
FH7.6	Minimum hop rate	14.6.7	FH7.1:M	Yes ❑ No ❑
FH7.7	Hop sequences	14.6.8	FH7.1:M	Yes ❑ No ❑
FH7.8	Unwanted emissions	14.6.9	FH7.1:M	Yes ❑ No ❑
FH8	1 Mbit/s PMD			
FH8.1	Modulation 2GFSK, BT=0.5, 1=positive freq. dev, 0=negative freq. dev.	14.6.10	M	Yes ❑ No ❑
FH8.2	Peak frequency deviation	14.6.10	M	Yes ❑ No ❑
FH8.3	Zero-Crossing error	14.6.10	M	Yes ❑ No ❑
FH8.4	Nominal channel data rate	14.6.11	M	Yes ❑ No ❑
FH8.5	Channel switching/settling time	14.6.12	M	Yes ❑ No ❑
FH8.6	Receive to transmit switch time	14.6.13	M	Yes ❑ No ❑
FH8.7	Nominal transmit power	14.6.14.1	M	Yes ❑ No ❑
FH8.8	Transmit power levels	14.6.14.2	M	Yes ❑ No ❑
FH8.9	Transmit power level control to <100 mW	14.6.14.3	M	Yes ❑ No ❑
FH8.10	Transmit spectrum shape	14.6.14.4	M	Yes ❑ No ❑
FH8.11	Transmit center frequency tolerance	14.6.14.5	M	Yes ❑ No ❑
FH8.12	Transmitter ramp periods	14.6.14.6	M	Yes ❑ No ❑
FH8.13	Receiver input dynamic range	14.6.15.1	M	Yes ❑ No ❑
FH8.14	Receiver center frequency acceptance range	14.6.15.2	M	Yes ❑ No ❑
FH8.15	Clear channel assessment power threshold for Pdet 90% (preamble)/70% (random data) for 100 mW units	14.6.15.3	M	Yes ❑ No ❑
FH8.16	Clear channel assessment power threshold for units >100 mW; sensitivity threshold is 1/2 dB lower for every dB above 20 dBm	14.6.15.3	M	Yes ❑ No ❑
FH8.17	Minimum receiver sensitivity at FER=3% with 400 octet frames	14.6.15.4	M	Yes ❑ No ❑
FH8.18	Intermodulation protection	14.6.15.5	M	Yes ❑ No ❑
FH8.19	Desensitization	14.6.15.6	M	Yes ❑ No ❑

A.4.5 Frequency-Hopping PHY functions *(continued)*

Item	Protocol feature	References	Status	Support
FH8.20	Operating temperature range	14.6.16	M	Yes ❑ No ❑
FH8.20.1	Temperature type 1	14.6.16	O	Yes ❑ No ❑
FH8.20.2	Temperature type 2	14.6.16	O	Yes ❑ No ❑
FH8.20.3	Temperature type 3	14.6.16	O	Yes ❑ No ❑
FH9	2 Mbit/s PMD			
FH9.1	All 1M PMD requirements	14.7.1	FH1.2.2:M	Yes ❑ No ❑ N/A ❑
FH9.2	Modulation 4GFSK, BT=0.5	14.7.2	FH1.2.2:M	Yes ❑ No ❑ N/A ❑
FH9.3	Frame structure for 2M PHY	14.7.2.1	FH1.2.2:M	Yes ❑ No ❑ N/A ❑
FH9.4	Nominal channel data rate	14.7.3	FH1.2.2:M	Yes ❑ No ❑ N/A ❑
FH9.5	Input dynamic range	14.7.4	FH1.2.2:M	Yes ❑ No ❑ N/A ❑
FH9.6	Minimum receiver sensitivity at FER=3% with 400 octet frames	14.7.5	FH1.2.2:M	Yes ❑ No ❑ N/A ❑
FH9.7	Intermodulation protection	14.7.6	FH1.2.2:M	Yes ❑ No ❑ N/A ❑
FH9.8	Desensitization	14.7.7	FH1.2.2:M	Yes ❑ No ❑ N/A ❑
FH10	MIB	13.1, 14.8, Annex C	M	Yes ❑ No ❑
FH10.1	PHY object class	13.1,14.8	M	Yes ❑ No ❑

A.4.6 Direct sequence PHY functions

Item	PHY feature	References	Status	Support
	PLCP sublayer procedures	15.2		
DS1	Preamble prepend on TX	15.2.1	M	Yes ❑ No ❑
DS1.1	PLCP frame format	15.2.2, 15.2.3	M	Yes ❑ No ❑
DS1.2	PLCP integrity check generation	15.2.3, 15.2.3.6	M	Yes ❑ No ❑
DS1.3	TX rate change capability	15.2.3.3, 15.2.5	M	Yes ❑ No ❑
DS1.4	Supported data rates	15.1, 15.2.3.3	M	Yes ❑ No ❑
DS1.5	Data whitener scrambler	15.2.4	M	Yes ❑ No ❑
DS1.6	Scrambler initialization	15.2.4	M	Yes ❑ No ❑
DS2	Preamble process on RX	15.2.1		
DS2.1	PLCP frame format	15.2.2, 15.2.3	M	Yes ❑ No ❑
DS2.2	PLCP integrity check verify	15.2.3, 15.2.3.6	M	Yes ❑ No ❑
DS2.3	RX Rate change capability	15.2.3.3, 15.2.5	M	Yes ❑ No ❑
DS2.4	Data whitener descrambler	15.2.4	M	Yes ❑ No ❑
DS3	PN code sequence	15.4.6.3	M	Yes ❑ No ❑
DS4	Chipping continue on power down	15.2.6	O	Yes ❑ No ❑
*DS5	Operating channel capability	15.2.6, 15.4.6.2		
* DS5.1	North America (FCC)	15.2.6, 15.4.6.2	DS5:O.1	Yes ❑ No ❑ N/A ❑
DS5.1.1	channel 1	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑

A.4.6 Direct sequence PHY functions *(continued)*

Item	PHY feature	References	Status	Support
DS5.1.2	channel 2	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
DS5.1.3	channel 3	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
DS5.1.4	channel 4	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
DS5.1.5	channel 5	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
DS5.1.6	channel 6	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
DS5.1.7	channel 7	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
DS5.1.8	channel 8	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
DS5.1.9	channel 9	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
DS5.1.10	channel 10	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
DS5.1.11	channel 11	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
* DS5.2	Canada (IC)	15.2.6, 15.4.6.2	DS5:O.1	Yes ❑ No ❑ N/A ❑
DS5.2.1	channel 1	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.2	channel 2	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.3	channel 3	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.4	channel 4	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.5	channel 5	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.6	channel 6	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.7	channel 7	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.8	channel 8	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.9	channel 9	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.10	channel 10	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.11	channel 11	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
* DS5.3	Europe (ETSI)	15.2.6, 15.4.6.2	DS5:O.1	Yes ❑ No ❑ N/A ❑
DS5.3.1	channel 1	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.2	channel 2	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.3	channel 3	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.4	channel 4	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.5	channel 5	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.6	channel 6	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.7	channel 7	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.8	channel 8	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.9	channel 9	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.10	channel 10	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.11	channel 11	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.12	channel 12	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.13	channel 13	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
* DS5.4	France	15.2.6, 15.4.6.2	DS5:O.1	Yes ❑ No ❑ N/A ❑
DS5.4.1	channel 10	15.2.6, 15.4.6.2	DS5.4:M	Yes ❑ No ❑ N/A ❑
DS5.4.2	channel 11	15.2.6, 15.4.6.2	DS5.4:M	Yes ❑ No ❑ N/A ❑
DS5.4.3	channel 12	15.2.6, 15.4.6.2	DS5.4:M	Yes ❑ No ❑ N/A ❑

A.4.6 Direct sequence PHY functions *(continued)*

Item	PHY feature	References	Status	Support
DS5.4.4	channel 13	15.2.6, 15.4.6.2	DS5.4:M	Yes ❑ No ❑ N/A ❑
* DS5.5	Spain	15.2.6, 15.4.6.2	DS5:O.1	Yes ❑ No ❑ N/A ❑
DS5.5.1	channel 10	15.2.6, 15.4.6.2	DS5.5:M	Yes ❑ No ❑ N/A ❑
DS5.5.2	channel 11	15.2.6, 15.4.6.2	DS5.5:M	Yes ❑ No ❑ N/A ❑
* DS5.6	Japan (RCR)	15.2.6, 15.4.6.2	DS5:O.1	Yes ❑ No ❑ N/A ❑
DS6	Bits to symbol mapping	15.4.6.4		
DS6.1	1 Mbit/s	15.4.6.4	M	Yes ❑ No ❑
DS6.2	2 Mbit/s	15.4.6.4	M	Yes ❑ No ❑
*DS7	CCA functionality	15.4.8.4		
DS7.1	Energy Only (RSSI above threshold)	15.4.8.4	DS7:O.2	Yes ❑ No ❑
DS7.2	IEEE 802.11 DSSS correlation	15.4.8.4	DS7:O.2	Yes ❑ No ❑
DS7.3	Both methods	15.4.8.4	DS7:O.2	Yes ❑ No ❑
DS7.4	Hold CCA busy for packet duration of a correctly received PLCP but carrier lost during reception of MPDU	15.2.7	M	Yes ❑ No ❑
DS7.5	Hold CCA busy for packet duration of a correctly received but out of spec PLCP	15.2.7	M	Yes ❑ No ❑
DS8	Transmit antenna selection	15.4.5.5, 15.4.5.6	O	Yes ❑ No ❑
DS9	Receive antenna diversity	15.4.5.5, 15.4.5.6, 15.4.5.7	O	Yes ❑ No ❑
*DS10	Antenna port(s) availability	15.4.6.9	O	Yes ❑ No ❑
DS10.1	50 Ω impedance	15.4.6.9	DS10:M	Yes ❑ No ❑ N/A ❑
*DS11	Transmit power level support	15.4.5.8, 15.4.7.3	O	Yes ❑ No ❑
DS11.1	If greater than 100 mW capability	15.4.7.3	DS11:M	Yes ❑ No ❑ N/A ❑
*DS12	Radio type (temperature range)	15.4.6.10		
DS12.1	Type 1	15.4.6.10	DS12:O.3	Yes ❑ No ❑ N/A ❑
DS12.2	Type 2	15.4.6.10	DS12:O.3	Yes ❑ No ❑ N/A ❑
DS13	Spurious emissions conformance	15.4.6.5	M	Yes ❑ No ❑
DS14	TX-RX turnaround time	15.4.6.6	M	Yes ❑ No ❑
DS15	RX-TX turnaround time	15.4.6.7	M	Yes ❑ No ❑
DS16	Slot time	15.4.6.8	M	Yes ❑ No ❑
DS17	ED reporting time	15.4.6.8, 15.4.8.4	M	Yes ❑ No ❑
DS18	Minimum transmit power level	15.4.7.2	M	Yes ❑ No ❑
DS19	Transmit spectral mask conformance	15.4.7.4	M	Yes ❑ No ❑
DS20	Transmitted center frequency tolerance	15.4.7.5	M	Yes ❑ No ❑

A.4.6 Direct sequence PHY functions *(continued)*

Item	PHY feature	References	Status	Support
DS21	Chip clock frequency tolerance	15.4.7.6	M	Yes ❑ No ❑
DS22	Transmit power on ramp	15.4.7.7	M	Yes ❑ No ❑
DS23	Transmit power down ramp	15.4.7.7	M	Yes ❑ No ❑
DS24	RF carrier suppression	15.4.7.8	M	Yes ❑ No ❑
DS25	Transmit modulation accuracy	15.4.7.9	M	Yes ❑ No ❑
DS26	Receiver minimum input level sensitivity	15.4.8.1	M	Yes ❑ No ❑
DS27	Receiver maximum input level	15.4.8.2	M	Yes ❑ No ❑
DS28	Receiver adjacent channel rejection	15.4.8.3	M	Yes ❑ No ❑
DS29	MIB	13.1, 15.3.4, Annex C	M	Yes ❑ No ❑
DS29.1	PHY object class	13.1, 15.3.4	M	Yes ❑ No ❑

A.4.7 Infrared baseband PHY functions

Item	Feature	References	Status	Support
IR1	Is the transmitted SYNC field length in the range of required number of PPM slots, with the absence of a pulse in the last slot of the field?	16.2.4.1	M	Yes ❑
IR2	Is the transmitted SYNC field entirely populated by alternating presence and absence of pulses in consecutive PPM slots, with the absence of a pulse in the last slot of the field?	16.2.4.1	M	Yes ❑
IR3	Is the transmitted SFD field the binary sequence 1001, where 1 indicates a pulse in the PPM slot and 0 indicates no pulse in the PPM slot?	16.2.4.2	M	Yes ❑
IR4	Is the transmitted DR field pulse sequence equal to the correct value for the data rate provided by the TXVECTOR parameter PLCP BITRATE, where 1 indicates a pulse in the PPM slot and 0 indicates no pulse in the PPM slot?	16.2.4.3	M	Yes ❑
IR5	Is the transmitted DCLA field 32 PPM slots long with the specified sequence for 1 Mbit/s, where 1 indicates a pulse in the PPM slot and 0 indicates no pulse in the PPM slot? 1 Mbit/s: 00000000100000000000000010000000	16.2.4.4	M	Yes ❑
* IR5a	Does the unit support 2 Mbit/s transmission?	16.2.4.4	O	Yes ❑ No ❑
IR5b	If the unit supports 2 Mbit/s transmission, is the transmitted DCLA field 32 PPM slots long with the specified sequence for 2 Mbit/s, where 1 indicates a pulse in the PPM slot and 0 indicates no pulse in the PPM slot? 2 Mbit/s: 00100010001000100010001000100010	16.2.4.4	IR5a:M	Yes ❑ No ❑ N/A ❑

A.4.7 Infrared baseband PHY functions *(continued)*

Item	Feature	References	Status	Support
IR6	Is the transmitted LENGTH field the correct PPM representation of the unsigned 16-bit binary integer, lsb transmitted first, equal to the correct value provided by the TXVECTOR parameter LENGTH?	16.2.4.5	M	Yes ❑
IR7	Is the transmitted CRC field the correct PPM representation of the CRC value calculated as per reference subclause, transmitted lsb first?	16.2.4.6	M	Yes ❑
IR8	Is the transmitted PSDU field the correct PPM representation of the PSDU, transmitted lsb first?	16.2.4.7	M	Yes ❑
IR9	When the CCA is false does transmission begin based on PHYTXSTART.request?	16.2.5.1	M	Yes ❑
IR10	Does the PHY issue a PHYTXSTART.confirm after the transmission of the PLCP header?	16.2.5.1	M	Yes ❑
IR11	Does the PHY accept each octet of the PSDU in a PHYDATA.request and answer with a PHYDATA.confirm?	16.2.5.1	M	Yes ❑
IR12	Does the PHY cease transmission in response to a PHYTXEND.request and answer with a PHYTXEND.confirm?	16.2.5.1	M	Yes ❑
IR13	Does the PHY of a receiving STA send a PHYCCA.indicate during reception of the SYNC field?	16.2.5.2	M	Yes ❑
IR14	Does the PHY of a receiving STA properly receive a transmission that changes data rate according to the DR field?	16.2.5.2	M	Yes ❑
IR15	Does the PHY of a receiving STA properly reject an incorrect CRC?	16.2.5.2	M	Yes ❑
IR16	Does the PHY of a receiving STA properly reject a DR field other than those specified in reference subclause?	16.2.5.2, 16.2.4.3	M	Yes ❑
IR17	Does the PHY of a receiving STA send PHYRXSTART.indicate with correct RATE and LENGTH parameters after proper reception of PLCP preamble and PLCP header?	16.2.5.2	M	Yes ❑
IR18	Does the PHY of a receiving STA forward receive octets in PHYDATA.indicate primitives?	16.2.5.2	M	Yes ❑
IR19	Does the PHY of a receiving STA send a PHYRXEND.indicate after the final octet indicated by the LENGTH field?	16.2.5.2	M	Yes ❑
IR20	Does the PHY of a receiving STA send a PHYCCA.indicate with a state value of IDLE after the PHYRXEND.indicate?	16.2.5.2	M	Yes ❑
IR21	Does the PHY reset its CCA detection mechanism upon receiving a PHYCCARST.request, and respond with a PHYCCARST.indicate?	16.2.5.3	M	Yes ❑

A.4.7 Infrared baseband PHY functions *(continued)*

Item	Feature	References	Status	Support
IR22	When transmitting at 1 Mbit/s does the PHY transmit PPM symbols according the 16-PPM Basic Rate Mapping table, transmitting from left to right?	16.3.2.1, 16.3.2.2	M	Yes ❑
IR23	When transmitting at 2 Mbit/s does the PHY transmit PPM symbols according to the 4-PPM Enhanced Rate Mapping table, transmitting from left to right?	16.3.2.1, 16.3.2.2	IR5a:M	Yes ❑
IR24	Does the PHY operate over a temperature range of 0 to 40 °C?	16.3.2.4	M	Yes ❑
* IR25	If the unit is conformant to emitter radiation mask 1, is the peak optical power of an emitted pulse within the specification range averaged over the pulse width?	16.3.3.1	O.1	Yes ❑ No ❑ N/A ❑
* IR26	If the unit is conformant to emitter radiation mask 2, is the peak optical power of an emitted pulse within the specification range averaged over the pulse width?	16.3.3.1	O.1	Yes ❑ No ❑ N/A ❑
IR27	Does the transmitted pulse shape conform to the description of the reference subclause?	16.3.3.2	M	Yes ❑
IR28	Does the emitter radiation pattern as a function of angle conform to the requirements of the reference subclause as applicable based on conformance to emitter radiation mask 1?	16.3.3.3	IR25:M	Yes ❑ No ❑ N/A ❑
IR28a	Does the emitter radiation pattern as a function of angle conform to the requirements of the reference subclause as applicable based on conformance to emitter radiation mask 2?	16.3.3.3	IR26:M	Yes ❑ No ❑ N/A ❑
IR29	Is the peak emitter optical output as a function of wavelength in the range specified?	16.3.3.4	M	Yes ❑
IR30	Does the spectrum of the transmit signal amplitude as a voltage or current meet the requirements of the reference subclause?	16.3.3.5	M	Yes ❑
IR31	Does the receiver sensitivity meet the requirements of the reference subclause for receive signals of both 1 and 2 Mbit/s?	16.3.4.1	M	Yes ❑
IR32	Does the receiver exhibit a dynamic range as specified in reference subclause?	16.3.4.2	M	Yes ❑
IR33	Does the receiver field of view conform to the requirements of the reference subclause?	16.3.4.3	M	Yes ❑
IR34	When it is known that the conditions are such that the Carrier Detect Signal and the Energy Detect Signal are false is the CCA asserted IDLE?	16.3.5.1	M	Yes ❑

A.4.7 Infrared baseband PHY functions *(continued)*

Item	Feature	References	Status	Support
IR35	When the conditions are such that Energy Detect is true for greater then the time defined in reference subclause, does CCA become IDLE?	16.3.5.1	M	Yes ❑
IR36	When conditions are such that either Carrier Detect or Energy Detect go true, does CCA go BUSY?	16.3.5.1	M	Yes ❑
IR37	Is the MIB completely supported?	16.4	M	Yes ❑

Annex B

(informative)

Hopping sequences

The following tables pertain to the hopping sequences for North America and ETSI.

Table B.1—Hopping sequence set 1

index	0	3	6	9	12	15	18	21	24	27	30	33	36
1	2	5	8	11	14	17	20	23	26	29	32	35	38
2	25	28	31	34	37	40	43	46	49	52	55	58	61
3	64	67	70	73	76	79	3	6	9	12	15	18	21
4	10	13	16	19	22	25	28	31	34	37	40	43	46
5	45	48	51	54	57	60	63	66	69	72	75	78	2
6	18	21	24	27	30	33	36	39	42	45	48	51	54
7	73	76	79	3	6	9	12	15	18	21	24	27	30
8	49	52	55	58	61	64	67	70	73	76	79	3	6
9	21	24	27	30	33	36	39	42	45	48	51	54	57
10	63	66	69	72	75	78	2	5	8	11	14	17	20
11	78	2	5	8	11	14	17	20	23	26	29	32	35
12	31	34	37	40	43	46	49	52	55	58	61	64	67
13	61	64	67	70	73	76	79	3	6	9	12	15	18
14	24	27	30	33	36	39	42	45	48	51	54	57	60
15	54	57	60	63	66	69	72	75	78	2	5	8	11
16	65	68	71	74	77	80	4	7	10	13	16	19	22
17	28	31	34	37	40	43	46	49	52	55	58	61	64
18	79	3	6	9	12	15	18	21	24	27	30	33	36
19	33	36	39	42	45	48	51	54	57	60	63	66	69
20	4	7	10	13	16	19	22	25	28	31	34	37	40
21	20	23	26	29	32	35	38	41	44	47	50	53	56
22	13	16	19	22	25	28	31	34	37	40	43	46	49
23	38	41	44	47	50	53	56	59	62	65	68	71	74
24	74	77	80	4	7	10	13	16	19	22	25	28	31
25	56	59	62	65	68	71	74	77	80	4	7	10	13
26	71	74	77	80	4	7	10	13	16	19	22	25	28
27	23	26	29	32	35	38	41	44	47	50	53	56	59
28	5	8	11	14	17	20	23	26	29	32	35	38	41
29	39	42	45	48	51	54	57	60	63	66	69	72	75
30	12	15	18	21	24	27	30	33	36	39	42	45	48
31	36	39	42	45	48	51	54	57	60	63	66	69	72
32	68	71	74	77	80	4	7	10	13	16	19	22	25
33	9	12	15	18	21	24	27	30	33	36	39	42	45
34	70	73	76	79	3	6	9	12	15	18	21	24	27
35	77	80	4	7	10	13	16	19	22	25	28	31	34
36	6	9	12	15	18	21	24	27	30	33	36	39	42
37	62	65	68	71	74	77	80	4	7	10	13	16	19
38	29	32	35	38	41	44	47	50	53	56	59	62	65
39	14	17	20	23	26	29	32	35	38	41	44	47	50

Table B.1—Hopping sequence set 1 *(continued)*

index	0	3	6	9	12	15	18	21	24	27	30	33	36
40	27	30	33	36	39	42	45	48	51	54	57	60	63
41	16	19	22	25	28	31	34	37	40	43	46	49	52
42	59	62	65	68	71	74	77	80	4	7	10	13	16
43	43	46	49	52	55	58	61	64	67	70	73	76	79
44	76	79	3	6	9	12	15	18	21	24	27	30	33
45	34	37	40	43	46	49	52	55	58	61	64	67	70
46	72	75	78	2	5	8	11	14	17	20	23	26	29
47	11	14	17	20	23	26	29	32	35	38	41	44	47
48	60	63	66	69	72	75	78	2	5	8	11	14	17
49	80	4	7	10	13	16	19	22	25	28	31	34	37
50	47	50	53	56	59	62	65	68	71	74	77	80	4
51	22	25	28	31	34	37	40	43	46	49	52	55	58
52	75	78	2	5	8	11	14	17	20	23	26	29	32
53	66	69	72	75	78	2	5	8	11	14	17	20	23
54	41	44	47	50	53	56	59	62	65	68	71	74	77
55	15	18	21	24	27	30	33	36	39	42	45	48	51
56	35	38	41	44	47	50	53	56	59	62	65	68	71
57	67	70	73	76	79	3	6	9	12	15	18	21	24
58	52	55	58	61	64	67	70	73	76	79	3	6	9
59	58	61	64	67	70	73	76	79	3	6	9	12	15
60	44	47	50	53	56	59	62	65	68	71	74	77	80
61	50	53	56	59	62	65	68	71	74	77	80	4	7
62	17	20	23	26	29	32	35	38	41	44	47	50	53
63	7	10	13	16	19	22	25	28	31	34	37	40	43
64	19	22	25	28	31	34	37	40	43	46	49	52	55
65	8	11	14	17	20	23	26	29	32	35	38	41	44
66	69	72	75	78	2	5	8	11	14	17	20	23	26
67	51	54	57	60	63	66	69	72	75	78	2	5	8
68	42	45	48	51	54	57	60	63	66	69	72	75	78
69	3	6	9	12	15	18	21	24	27	30	33	36	39
70	30	33	36	39	42	45	48	51	54	57	60	63	66
71	57	60	63	66	69	72	75	78	2	5	8	11	14
72	37	40	43	46	49	52	55	58	61	64	67	70	73
73	55	58	61	64	67	70	73	76	79	3	6	9	12
74	26	29	32	35	38	41	44	47	50	53	56	59	62
75	46	49	52	55	58	61	64	67	70	73	76	79	3
76	53	56	59	62	65	68	71	74	77	80	4	7	10
77	40	43	46	49	52	55	58	61	64	67	70	73	76
78	32	35	38	41	44	47	50	53	56	59	62	65	68
79	48	51	54	57	60	63	66	69	72	75	78	2	5

Table B.1—Hopping sequence set 1 *(continued)*

index	39	42	45	48	51	54	57	60	63	66	69	72	75
1	41	44	47	50	53	56	59	62	65	68	71	74	77
2	64	67	70	73	76	79	3	6	9	12	15	18	21
3	24	27	30	33	36	39	42	45	48	51	54	57	60
4	49	52	55	58	61	64	67	70	73	76	79	3	6
5	5	8	11	14	17	20	23	26	29	32	35	38	41
6	57	60	63	66	69	72	75	78	2	5	8	11	14
7	33	36	39	42	45	48	51	54	57	60	63	66	69
8	9	12	15	18	21	24	27	30	33	36	39	42	45
9	60	63	66	69	72	75	78	2	5	8	11	14	17
10	23	26	29	32	35	38	41	44	47	50	53	56	59
11	38	41	44	47	50	53	56	59	62	65	68	71	74
12	70	73	76	79	3	6	9	12	15	18	21	24	27
13	21	24	27	30	33	36	39	42	45	48	51	54	57
14	63	66	69	72	75	78	2	5	8	11	14	17	20
15	14	17	20	23	26	29	32	35	38	41	44	47	50
16	25	28	31	34	37	40	43	46	49	52	55	58	61
17	67	70	73	76	79	3	6	9	12	15	18	21	24
18	39	42	45	48	51	54	57	60	63	66	69	72	75
19	72	75	78	2	5	8	11	14	17	20	23	26	29
20	43	46	49	52	55	58	61	64	67	70	73	76	79
21	59	62	65	68	71	74	77	80	4	7	10	13	16
22	52	55	58	61	64	67	70	73	76	79	3	6	9
23	77	80	4	7	10	13	16	19	22	25	28	31	34
24	34	37	40	43	46	49	52	55	58	61	64	67	70
25	16	19	22	25	28	31	34	37	40	43	46	49	52
26	31	34	37	40	43	46	49	52	55	58	61	64	67
27	62	65	68	71	74	77	80	4	7	10	13	16	19
28	44	47	50	53	56	59	62	65	68	71	74	77	80
29	78	2	5	8	11	14	17	20	23	26	29	32	35
30	51	54	57	60	63	66	69	72	75	78	2	5	8
31	75	78	2	5	8	11	14	17	20	23	26	29	32
32	28	31	34	37	40	43	46	49	52	55	58	61	64
33	48	51	54	57	60	63	66	69	72	75	78	2	5
34	30	33	36	39	42	45	48	51	54	57	60	63	66
35	37	40	43	46	49	52	55	58	61	64	67	70	73
36	45	48	51	54	57	60	63	66	69	72	75	78	2
37	22	25	28	31	34	37	40	43	46	49	52	55	58
38	68	71	74	77	80	4	7	10	13	16	19	22	25
39	53	56	59	62	65	68	71	74	77	80	4	7	10

Table B.1—Hopping sequence set 1 *(continued)*

index	39	42	45	48	51	54	57	60	63	66	69	72	75
40	66	69	72	75	78	2	5	8	11	14	17	20	23
41	55	58	61	64	67	70	73	76	79	3	6	9	12
42	19	22	25	28	31	34	37	40	43	46	49	52	55
43	3	6	9	12	15	18	21	24	27	30	33	36	39
44	36	39	42	45	48	51	54	57	60	63	66	69	72
45	73	76	79	3	6	9	12	15	18	21	24	27	30
46	32	35	38	41	44	47	50	53	56	59	62	65	68
47	50	53	56	59	62	65	68	71	74	77	80	4	7
48	20	23	26	29	32	35	38	41	44	47	50	53	56
49	40	43	46	49	52	55	58	61	64	67	70	73	76
50	7	10	13	16	19	22	25	28	31	34	37	40	43
51	61	64	67	70	73	76	79	3	6	9	12	15	18
52	35	38	41	44	47	50	53	56	59	62	65	68	71
53	26	29	32	35	38	41	44	47	50	53	56	59	62
54	80	4	7	10	13	16	19	22	25	28	31	34	37
55	54	57	60	63	66	69	72	75	78	2	5	8	11
56	74	77	80	4	7	10	13	16	19	22	25	28	31
57	27	30	33	36	39	42	45	48	51	54	57	60	63
58	12	15	18	21	24	27	30	33	36	39	42	45	48
59	18	21	24	27	30	33	36	39	42	45	48	51	54
60	4	7	10	13	16	19	22	25	28	31	34	37	40
61	10	13	16	19	22	25	28	31	34	37	40	43	46
62	56	59	62	65	68	71	74	77	80	4	7	10	13
63	46	49	52	55	58	61	64	67	70	73	76	79	3
64	58	61	64	67	70	73	76	79	3	6	9	12	15
65	47	50	53	56	59	62	65	68	71	74	77	80	4
66	29	32	35	38	41	44	47	50	53	56	59	62	65
67	11	14	17	20	23	26	29	32	35	38	41	44	47
68	2	5	8	11	14	17	20	23	26	29	32	35	38
69	42	45	48	51	54	57	60	63	66	69	72	75	78
70	69	72	75	78	2	5	8	11	14	17	20	23	26
71	17	20	23	26	29	32	35	38	41	44	47	50	53
72	76	79	3	6	9	12	15	18	21	24	27	30	33
73	15	18	21	24	27	30	33	36	39	42	45	48	51
74	65	68	71	74	77	80	4	7	10	13	16	19	22
75	6	9	12	15	18	21	24	27	30	33	36	39	42
76	13	16	19	22	25	28	31	34	37	40	43	46	49
77	79	3	6	9	12	15	18	21	24	27	30	33	36
78	71	74	77	80	4	7	10	13	16	19	22	25	28
79	8	11	14	17	20	23	26	29	32	35	38	41	44

Table B.2—Hopping sequence set 2

index	1	4	7	10	13	16	19	22	25	28	31	34	37
1	3	6	9	12	15	18	21	24	27	30	33	36	39
2	26	29	32	35	38	41	44	47	50	53	56	59	62
3	65	68	71	74	77	80	4	7	10	13	16	19	22
4	11	14	17	20	23	26	29	32	35	38	41	44	47
5	46	49	52	55	58	61	64	67	70	73	76	79	3
6	19	22	25	28	31	34	37	40	43	46	49	52	55
7	74	77	80	4	7	10	13	16	19	22	25	28	31
8	50	53	56	59	62	65	68	71	74	77	80	4	7
9	22	25	28	31	34	37	40	43	46	49	52	55	58
10	64	67	70	73	76	79	3	6	9	12	15	18	21
11	79	3	6	9	12	15	18	21	24	27	30	33	36
12	32	35	38	41	44	47	50	53	56	59	62	65	68
13	62	65	68	71	74	77	80	4	7	10	13	16	19
14	25	28	31	34	37	40	43	46	49	52	55	58	61
15	55	58	61	64	67	70	73	76	79	3	6	9	12
16	66	69	72	75	78	2	5	8	11	14	17	20	23
17	29	32	35	38	41	44	47	50	53	56	59	62	65
18	80	4	7	10	13	16	19	22	25	28	31	34	37
19	34	37	40	43	46	49	52	55	58	61	64	67	70
20	5	8	11	14	17	20	23	26	29	32	35	38	41
21	21	24	27	30	33	36	39	42	45	48	51	54	57
22	14	17	20	23	26	29	32	35	38	41	44	47	50
23	39	42	45	48	51	54	57	60	63	66	69	72	75
24	75	78	2	5	8	11	14	17	20	23	26	29	32
25	57	60	63	66	69	72	75	78	2	5	8	11	14
26	72	75	78	2	5	8	11	14	17	20	23	26	29
27	24	27	30	33	36	39	42	45	48	51	54	57	60
28	6	9	12	15	18	21	24	27	30	33	36	39	42
29	40	43	46	49	52	55	58	61	64	67	70	73	76
30	13	16	19	22	25	28	31	34	37	40	43	46	49
31	37	40	43	46	49	52	55	58	61	64	67	70	73
32	69	72	75	78	2	5	8	11	14	17	20	23	26
33	10	13	16	19	22	25	28	31	34	37	40	43	46
34	71	74	77	80	4	7	10	13	16	19	22	25	28
35	78	2	5	8	11	14	17	20	23	26	29	32	35
36	7	10	13	16	19	22	25	28	31	34	37	40	43
37	63	66	69	72	75	78	2	5	8	11	14	17	20
38	30	33	36	39	42	45	48	51	54	57	60	63	66
39	15	18	21	24	27	30	33	36	39	42	45	48	51

Table B.2—Hopping sequence set 2 *(continued)*

index	1	4	7	10	13	16	19	22	25	28	31	34	37
40	28	31	34	37	40	43	46	49	52	55	58	61	64
41	17	20	23	26	29	32	35	38	41	44	47	50	53
42	60	63	66	69	72	75	78	2	5	8	11	14	17
43	44	47	50	53	56	59	62	65	68	71	74	77	80
44	77	80	4	7	10	13	16	19	22	25	28	31	34
45	35	38	41	44	47	50	53	56	59	62	65	68	71
46	73	76	79	3	6	9	12	15	18	21	24	27	30
47	12	15	18	21	24	27	30	33	36	39	42	45	48
48	61	64	67	70	73	76	79	3	6	9	12	15	18
49	2	5	8	11	14	17	20	23	26	29	32	35	38
50	48	51	54	57	60	63	66	69	72	75	78	2	5
51	23	26	29	32	35	38	41	44	47	50	53	56	59
52	76	79	3	6	9	12	15	18	21	24	27	30	33
53	67	70	73	76	79	3	6	9	12	15	18	21	24
54	42	45	48	51	54	57	60	63	66	69	72	75	78
55	16	19	22	25	28	31	34	37	40	43	46	49	52
56	36	39	42	45	48	51	54	57	60	63	66	69	72
57	68	71	74	77	80	4	7	10	13	16	19	22	25
58	53	56	59	62	65	68	71	74	77	80	4	7	10
59	59	62	65	68	71	74	77	80	4	7	10	13	16
60	45	48	51	54	57	60	63	66	69	72	75	78	2
61	51	54	57	60	63	66	69	72	75	78	2	5	8
62	18	21	24	27	30	33	36	39	42	45	48	51	54
63	8	11	14	17	20	23	26	29	32	35	38	41	44
64	20	23	26	29	32	35	38	41	44	47	50	53	56
65	9	12	15	18	21	24	27	30	33	36	39	42	45
66	70	73	76	79	3	6	9	12	15	18	21	24	27
67	52	55	58	61	64	67	70	73	76	79	3	6	9
68	43	46	49	52	55	58	61	64	67	70	73	76	79
69	4	7	10	13	16	19	22	25	28	31	34	37	40
70	31	34	37	40	43	46	49	52	55	58	61	64	67
71	58	61	64	67	70	73	76	79	3	6	9	12	15
72	38	41	44	47	50	53	56	59	62	65	68	71	74
73	56	59	62	65	68	71	74	77	80	4	7	10	13
74	27	30	33	36	39	42	45	48	51	54	57	60	63
75	47	50	53	56	59	62	65	68	71	74	77	80	4
76	54	57	60	63	66	69	72	75	78	2	5	8	11
77	41	44	47	50	53	56	59	62	65	68	71	74	77
78	33	36	39	42	45	48	51	54	57	60	63	66	69
79	49	52	55	58	61	64	67	70	73	76	79	3	6

Table B.2—Hopping sequence set 2 ***(continued)***

index	40	43	46	49	52	55	58	61	64	67	70	73	76
1	42	45	48	51	54	57	60	63	66	69	72	75	78
2	65	68	71	74	77	80	4	7	10	13	16	19	22
3	25	28	31	34	37	40	43	46	49	52	55	58	61
4	50	53	56	59	62	65	68	71	74	77	80	4	7
5	6	9	12	15	18	21	24	27	30	33	36	39	42
6	58	61	64	67	70	73	76	79	3	6	9	12	15
7	34	37	40	43	46	49	52	55	58	61	64	67	70
8	10	13	16	19	22	25	28	31	34	37	40	43	46
9	61	64	67	70	73	76	79	3	6	9	12	15	18
10	24	27	30	33	36	39	42	45	48	51	54	57	60
11	39	42	45	48	51	54	57	60	63	66	69	72	75
12	71	74	77	80	4	7	10	13	16	19	22	25	28
13	22	25	28	31	34	37	40	43	46	49	52	55	58
14	64	67	70	73	76	79	3	6	9	12	15	18	21
15	15	18	21	24	27	30	33	36	39	42	45	48	51
16	26	29	32	35	38	41	44	47	50	53	56	59	62
17	68	71	74	77	80	4	7	10	13	16	19	22	25
18	40	43	46	49	52	55	58	61	64	67	70	73	76
19	73	76	79	3	6	9	12	15	18	21	24	27	30
20	44	47	50	53	56	59	62	65	68	71	74	77	80
21	60	63	66	69	72	75	78	2	5	8	11	14	17
22	53	56	59	62	65	68	71	74	77	80	4	7	10
23	78	2	5	8	11	14	17	20	23	26	29	32	35
24	35	38	41	44	47	50	53	56	59	62	65	68	71
25	17	20	23	26	29	32	35	38	41	44	47	50	53
26	32	35	38	41	44	47	50	53	56	59	62	65	68
27	63	66	69	72	75	78	2	5	8	11	14	17	20
28	45	48	51	54	57	60	63	66	69	72	75	78	2
29	79	3	6	9	12	15	18	21	24	27	30	33	36
30	52	55	58	61	64	67	70	73	76	79	3	6	9
31	76	79	3	6	9	12	15	18	21	24	27	30	33
32	29	32	35	38	41	44	47	50	53	56	59	62	65
33	49	52	55	58	61	64	67	70	73	76	79	3	6
34	31	34	37	40	43	46	49	52	55	58	61	64	67
35	38	41	44	47	50	53	56	59	62	65	68	71	74
36	46	49	52	55	58	61	64	67	70	73	76	79	3
37	23	26	29	32	35	38	41	44	47	50	53	56	59
38	69	72	75	78	2	5	8	11	14	17	20	23	26
39	54	57	60	63	66	69	72	75	78	2	5	8	11

Table B.2—Hopping sequence set 2 *(continued)*

index	40	43	46	49	52	55	58	61	64	67	70	73	76
40	67	70	73	76	79	3	6	9	12	15	18	21	24
41	56	59	62	65	68	71	74	77	80	4	7	10	13
42	20	23	26	29	32	35	38	41	44	47	50	53	56
43	4	7	10	13	16	19	22	25	28	31	34	37	40
44	37	40	43	46	49	52	55	58	61	64	67	70	73
45	74	77	80	4	7	10	13	16	19	22	25	28	31
46	33	36	39	42	45	48	51	54	57	60	63	66	69
47	51	54	57	60	63	66	69	72	75	78	2	5	8
48	21	24	27	30	33	36	39	42	45	48	51	54	57
49	41	44	47	50	53	56	59	62	65	68	71	74	77
50	8	11	14	17	20	23	26	29	32	35	38	41	44
51	62	65	68	71	74	77	80	4	7	10	13	16	19
52	36	39	42	45	48	51	54	57	60	63	66	69	72
53	27	30	33	36	39	42	45	48	51	54	57	60	63
54	2	5	8	11	14	17	20	23	26	29	32	35	38
55	55	58	61	64	67	70	73	76	79	3	6	9	12
56	75	78	2	5	8	11	14	17	20	23	26	29	32
57	28	31	34	37	40	43	46	49	52	55	58	61	64
58	13	16	19	22	25	28	31	34	37	40	43	46	49
59	19	22	25	28	31	34	37	40	43	46	49	52	55
60	5	8	11	14	17	20	23	26	29	32	35	38	41
61	11	14	17	20	23	26	29	32	35	38	41	44	47
62	57	60	63	66	69	72	75	78	2	5	8	11	14
63	47	50	53	56	59	62	65	68	71	74	77	80	4
64	59	62	65	68	71	74	77	80	4	7	10	13	16
65	48	51	54	57	60	63	66	69	72	75	78	2	5
66	30	33	36	39	42	45	48	51	54	57	60	63	66
67	12	15	18	21	24	27	30	33	36	39	42	45	48
68	3	6	9	12	15	18	21	24	27	30	33	36	39
69	43	46	49	52	55	58	61	64	67	70	73	76	79
70	70	73	76	79	3	6	9	12	15	18	21	24	27
71	18	21	24	27	30	33	36	39	42	45	48	51	54
72	77	80	4	7	10	13	16	19	22	25	28	31	34
73	16	19	22	25	28	31	34	37	40	43	46	49	52
74	66	69	72	75	78	2	5	8	11	14	17	20	23
75	7	10	13	16	19	22	25	28	31	34	37	40	43
76	14	17	20	23	26	29	32	35	38	41	44	47	50
77	80	4	7	10	13	16	19	22	25	28	31	34	37
78	72	75	78	2	5	8	11	14	17	20	23	26	29
79	9	12	15	18	21	24	27	30	33	36	39	42	45

Table B.3—Hopping sequence set 3

index	2	5	8	11	14	17	20	23	26	29	32	35	38
1	4	7	10	13	16	19	22	25	28	31	34	37	40
2	27	30	33	36	39	42	45	48	51	54	57	60	63
3	66	69	72	75	78	2	5	8	11	14	17	20	23
4	12	15	18	21	24	27	30	33	36	39	42	45	48
5	47	50	53	56	59	62	65	68	71	74	77	80	4
6	20	23	26	29	32	35	38	41	44	47	50	53	56
7	75	78	2	5	8	11	14	17	20	23	26	29	32
8	51	54	57	60	63	66	69	72	75	78	2	5	8
9	23	26	29	32	35	38	41	44	47	50	53	56	59
10	65	68	71	74	77	80	4	7	10	13	16	19	22
11	80	4	7	10	13	16	19	22	25	28	31	34	37
12	33	36	39	42	45	48	51	54	57	60	63	66	69
13	63	66	69	72	75	78	2	5	8	11	14	17	20
14	26	29	32	35	38	41	44	47	50	53	56	59	62
15	56	59	62	65	68	71	74	77	80	4	7	10	13
16	67	70	73	76	79	3	6	9	12	15	18	21	24
17	30	33	36	39	42	45	48	51	54	57	60	63	66
18	2	5	8	11	14	17	20	23	26	29	32	35	38
19	35	38	41	44	47	50	53	56	59	62	65	68	71
20	6	9	12	15	18	21	24	27	30	33	36	39	42
21	22	25	28	31	34	37	40	43	46	49	52	55	58
22	15	18	21	24	27	30	33	36	39	42	45	48	51
23	40	43	46	49	52	55	58	61	64	67	70	73	76
24	76	79	3	6	9	12	15	18	21	24	27	30	33
25	58	61	64	67	70	73	76	79	3	6	9	12	15
26	73	76	79	3	6	9	12	15	18	21	24	27	30
27	25	28	31	34	37	40	43	46	49	52	55	58	61
28	7	10	13	16	19	22	25	28	31	34	37	40	43
29	41	44	47	50	53	56	59	62	65	68	71	74	77
30	14	17	20	23	26	29	32	35	38	41	44	47	50
31	38	41	44	47	50	53	56	59	62	65	68	71	74
32	70	73	76	79	3	6	9	12	15	18	21	24	27
33	11	14	17	20	23	26	29	32	35	38	41	44	47
34	72	75	78	2	5	8	11	14	17	20	23	26	29
35	79	3	6	9	12	15	18	21	24	27	30	33	36
36	8	11	14	17	20	23	26	29	32	35	38	41	44
37	64	67	70	73	76	79	3	6	9	12	15	18	21
38	31	34	37	40	43	46	49	52	55	58	61	64	67
39	16	19	22	25	28	31	34	37	40	43	46	49	52

Table B.3—Hopping sequence set 3 *(continued)*

index	2	5	8	11	14	17	20	23	26	29	32	35	38
40	29	32	35	38	41	44	47	50	53	56	59	62	65
41	18	21	24	27	30	33	36	39	42	45	48	51	54
42	61	64	67	70	73	76	79	3	6	9	12	15	18
43	45	48	51	54	57	60	63	66	69	72	75	78	2
44	78	2	5	8	11	14	17	20	23	26	29	32	35
45	36	39	42	45	48	51	54	57	60	63	66	69	72
46	74	77	80	4	7	10	13	16	19	22	25	28	31
47	13	16	19	22	25	28	31	34	37	40	43	46	49
48	62	65	68	71	74	77	80	4	7	10	13	16	19
49	3	6	9	12	15	18	21	24	27	30	33	36	39
50	49	52	55	58	61	64	67	70	73	76	79	3	6
51	24	27	30	33	36	39	42	45	48	51	54	57	60
52	77	80	4	7	10	13	16	19	22	25	28	31	34
53	68	71	74	77	80	4	7	10	13	16	19	22	25
54	43	46	49	52	55	58	61	64	67	70	73	76	79
55	17	20	23	26	29	32	35	38	41	44	47	50	53
56	37	40	43	46	49	52	55	58	61	64	67	70	73
57	69	72	75	78	2	5	8	11	14	17	20	23	26
58	54	57	60	63	66	69	72	75	78	2	5	8	11
59	60	63	66	69	72	75	78	2	5	8	11	14	17
60	46	49	52	55	58	61	64	67	70	73	76	79	3
61	52	55	58	61	64	67	70	73	76	79	3	6	9
62	19	22	25	28	31	34	37	40	43	46	49	52	55
63	9	12	15	18	21	24	27	30	33	36	39	42	45
64	21	24	27	30	33	36	39	42	45	48	51	54	57
65	10	13	16	19	22	25	28	31	34	37	40	43	46
66	71	74	77	80	4	7	10	13	16	19	22	25	28
67	53	56	59	62	65	68	71	74	77	80	4	7	10
68	44	47	50	53	56	59	62	65	68	71	74	77	80
69	5	8	11	14	17	20	23	26	29	32	35	38	41
70	32	35	38	41	44	47	50	53	56	59	62	65	68
71	59	62	65	68	71	74	77	80	4	7	10	13	16
72	39	42	45	48	51	54	57	60	63	66	69	72	75
73	57	60	63	66	69	72	75	78	2	5	8	11	14
74	28	31	34	37	40	43	46	49	52	55	58	61	64
75	48	51	54	57	60	63	66	69	72	75	78	2	5
76	55	58	61	64	67	70	73	76	79	3	6	9	12
77	42	45	48	51	54	57	60	63	66	69	72	75	78
78	34	37	40	43	46	49	52	55	58	61	64	67	70
79	50	53	56	59	62	65	68	71	74	77	80	4	7

Table B.3—Hopping sequence set 3 *(continued)*

index	41	44	47	50	53	56	59	62	65	68	71	74	77
1	43	46	49	52	55	58	61	64	67	70	73	76	79
2	66	69	72	75	78	2	5	8	11	14	17	20	23
3	26	29	32	35	38	41	44	47	50	53	56	59	62
4	51	54	57	60	63	66	69	72	75	78	2	5	8
5	7	10	13	16	19	22	25	28	31	34	37	40	43
6	59	62	65	68	71	74	77	80	4	7	10	13	16
7	35	38	41	44	47	50	53	56	59	62	65	68	71
8	11	14	17	20	23	26	29	32	35	38	41	44	47
9	62	65	68	71	74	77	80	4	7	10	13	16	19
10	25	28	31	34	37	40	43	46	49	52	55	58	61
11	40	43	46	49	52	55	58	61	64	67	70	73	76
12	72	75	78	2	5	8	11	14	17	20	23	26	29
13	23	26	29	32	35	38	41	44	47	50	53	56	59
14	65	68	71	74	77	80	4	7	10	13	16	19	22
15	16	19	22	25	28	31	34	37	40	43	46	49	52
16	27	30	33	36	39	42	45	48	51	54	57	60	63
17	69	72	75	78	2	5	8	11	14	17	20	23	26
18	41	44	47	50	53	56	59	62	65	68	71	74	77
19	74	77	80	4	7	10	13	16	19	22	25	28	31
20	45	48	51	54	57	60	63	66	69	72	75	78	2
21	61	64	67	70	73	76	79	3	6	9	12	15	18
22	54	57	60	63	66	69	72	75	78	2	5	8	11
23	79	3	6	9	12	15	18	21	24	27	30	33	36
24	36	39	42	45	48	51	54	57	60	63	66	69	72
25	18	21	24	27	30	33	36	39	42	45	48	51	54
26	33	36	39	42	45	48	51	54	57	60	63	66	69
27	64	67	70	73	76	79	3	6	9	12	15	18	21
28	46	49	52	55	58	61	64	67	70	73	76	79	3
29	80	4	7	10	13	16	19	22	25	28	31	34	37
30	53	56	59	62	65	68	71	74	77	80	4	7	10
31	77	80	4	7	10	13	16	19	22	25	28	31	34
32	30	33	36	39	42	45	48	51	54	57	60	63	66
33	50	53	56	59	62	65	68	71	74	77	80	4	7
34	32	35	38	41	44	47	50	53	56	59	62	65	68
35	39	42	45	48	51	54	57	60	63	66	69	72	75
36	47	50	53	56	59	62	65	68	71	74	77	80	4
37	24	27	30	33	36	39	42	45	48	51	54	57	60
38	70	73	76	79	3	6	9	12	15	18	21	24	27
39	55	58	61	64	67	70	73	76	79	3	6	9	12

Table B.3—Hopping sequence set 3 *(continued)*

index	41	44	47	50	53	56	59	62	65	68	71	74	77
40	68	71	74	77	80	4	7	10	13	16	19	22	25
41	57	60	63	66	69	72	75	78	2	5	8	11	14
42	21	24	27	30	33	36	39	42	45	48	51	54	57
43	5	8	11	14	17	20	23	26	29	32	35	38	41
44	38	41	44	47	50	53	56	59	62	65	68	71	74
45	75	78	2	5	8	11	14	17	20	23	26	29	32
46	34	37	40	43	46	49	52	55	58	61	64	67	70
47	52	55	58	61	64	67	70	73	76	79	3	6	9
48	22	25	28	31	34	37	40	43	46	49	52	55	58
49	42	45	48	51	54	57	60	63	66	69	72	75	78
50	9	12	15	18	21	24	27	30	33	36	39	42	45
51	63	66	69	72	75	78	2	5	8	11	14	17	20
52	37	40	43	46	49	52	55	58	61	64	67	70	73
53	28	31	34	37	40	43	46	49	52	55	58	61	64
54	3	6	9	12	15	18	21	24	27	30	33	36	39
55	56	59	62	65	68	71	74	77	80	4	7	10	13
56	76	79	3	6	9	12	15	18	21	24	27	30	33
57	29	32	35	38	41	44	47	50	53	56	59	62	65
58	14	17	20	23	26	29	32	35	38	41	44	47	50
59	20	23	26	29	32	35	38	41	44	47	50	53	56
60	6	9	12	15	18	21	24	27	30	33	36	39	42
61	12	15	18	21	24	27	30	33	36	39	42	45	48
62	58	61	64	67	70	73	76	79	3	6	9	12	15
63	48	51	54	57	60	63	66	69	72	75	78	2	5
64	60	63	66	69	72	75	78	2	5	8	11	14	17
65	49	52	55	58	61	64	67	70	73	76	79	3	6
66	31	34	37	40	43	46	49	52	55	58	61	64	67
67	13	16	19	22	25	28	31	34	37	40	43	46	49
68	4	7	10	13	16	19	22	25	28	31	34	37	40
69	44	47	50	53	56	59	62	65	68	71	74	77	80
70	71	74	77	80	4	7	10	13	16	19	22	25	28
71	19	22	25	28	31	34	37	40	43	46	49	52	55
72	78	2	5	8	11	14	17	20	23	26	29	32	35
73	17	20	23	26	29	32	35	38	41	44	47	50	53
74	67	70	73	76	79	3	6	9	12	15	18	21	24
75	8	11	14	17	20	23	26	29	32	35	38	41	44
76	15	18	21	24	27	30	33	36	39	42	45	48	51
77	2	5	8	11	14	17	20	23	26	29	32	35	38
78	73	76	79	3	6	9	12	15	18	21	24	27	30
79	10	13	16	19	22	25	28	31	34	37	40	43	46

IEEE

Annex C

(normative)

Formal description of MAC operation

This annex contains formal descriptions of the behavior of MAC station (STA) and access point (AP) entities. These descriptions also describe the frame formats and the generation and interpretation of information encoded in MAC frames, in the parameters of service primitives supported by the MAC, and in MIB attributes used or generated by the MAC. The MAC is described using the 1992 version of the ITU Specification and Description Language (SDL-92). SDL-92 is defined in ITU-T Recommendation Z.100 (03/93). An update to Z.100 was approved in 1996 (SDL-96), but none of the SDL facilities used in this annex were modified. An introduction to the MAC formal description is provided in Clause C.1. Definitions of the data types and operators used by the MAC state machines are provided in Clause C.2. An SDL system describing MAC operation at an IEEE 802.11 station is contained in Clause C.3. Finally, a subset of an SDL system describing the aspects of MAC operation at an IEEE 802.11 AP that differ from operation at a non-AP station is provided in Clause C.4.

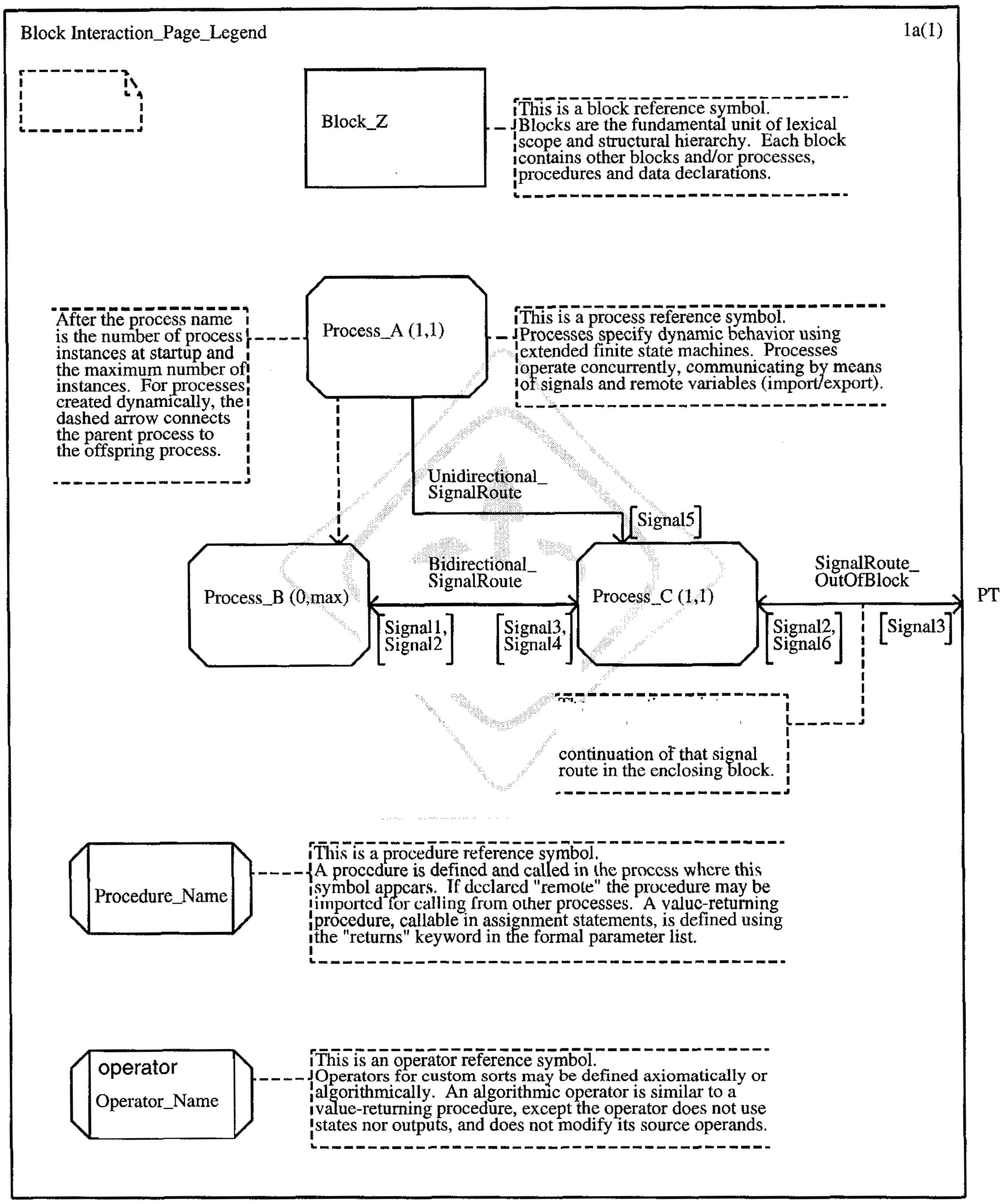
In Annex D, the MAC and PHY management information bases are described in Abstract Syntax Notation One (ASN.1), defined in ISO/IEC 8824 and ISO/IEC 8825. ITU-T Recommendation Z.105 (03/95) defines the use of SDL in conjunction with ASN.1, allowing system behavior to be defined using SDL and data types to be defined using ASN.1. Incomplete tool support precluded the use of Z.105 in this annex. However, within the limits of Z.100, the data types in Clause C.2 are defined in a similar manner to Z.105. Annex E contains a listing of available documentation.

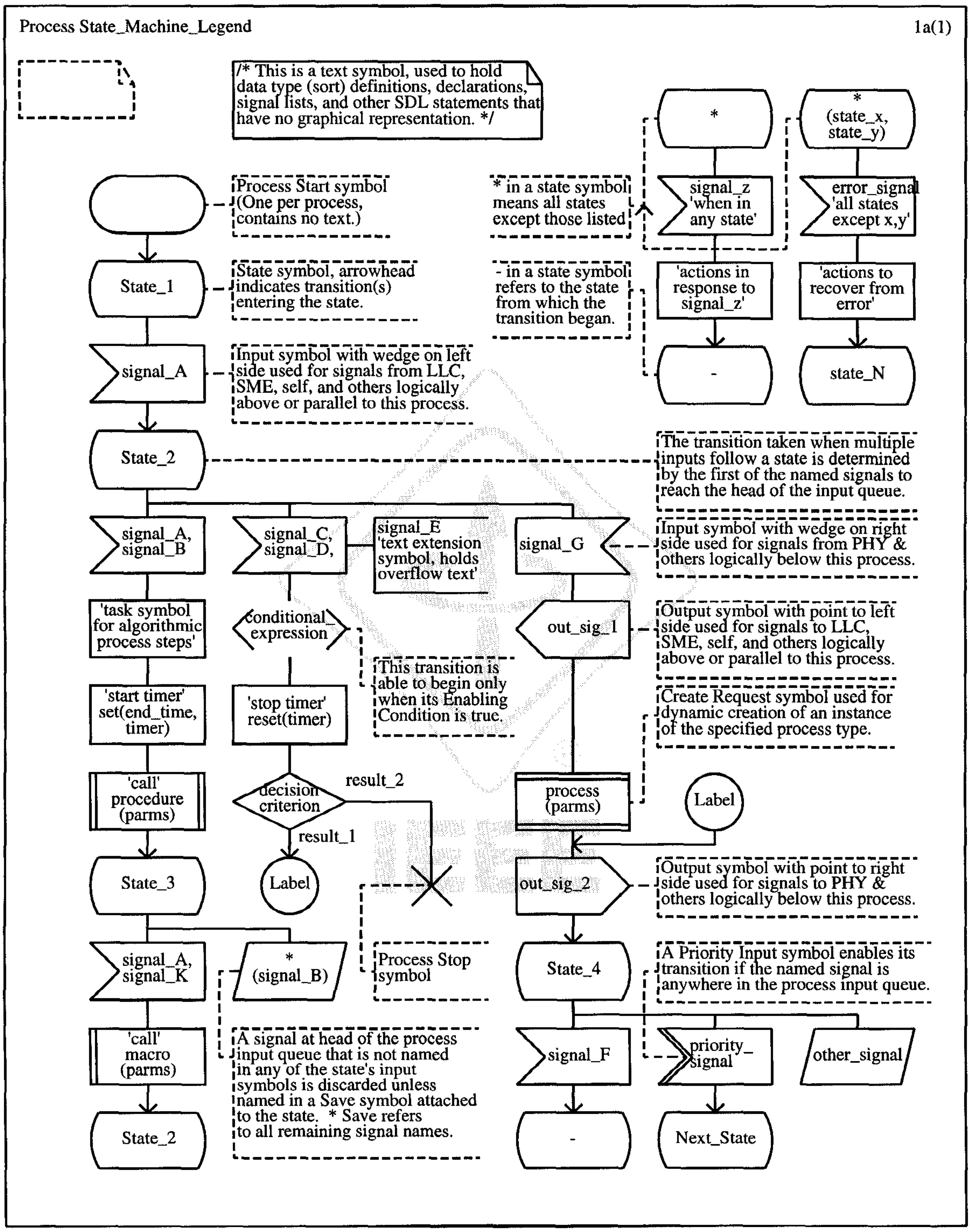
NOTES

1—Software for generating, analyzing, verifying, and simulating SDL system descriptions is available from several sources. The SDL code in this annex was generated using *SDT/PC version 3.02*; from Telelogic AB, Malmo, Sweden (+46-40-174700; internet: telelogic.se); USA office in Princeton, NJ (+1-609-520-1935; internet: telelogic.com). Telelogic offers SDT for several workstation platforms in addition to SDT/PC.

2—The SDL definitions in this annex should be usable with any SDL tool that supports the 1993 version or 1996 update of Recommendation Z.100. The use of Telelogic's product to prepare this annex does not constitute an endorsement of SDT by the IEEE LAN/MAN Standards Committee or by the IEEE.

2—The diagrams on the next two pages show most of the symbols of SDL graphical syntax (SDL-GR) used in the MAC formal description. The symbols in these diagrams have labels and comments that explain their meanings. These diagrams are intended to serve as a legend for the SDL-GR symbols that comprise most of the process interaction and state transition diagrams. These diagrams are neither a complete SDL system, nor a complete presentation of SDL-GR symbology. Also, this state machine fragment exists to illustrate the SDL graphical syntax, and does not describe any useful behavior.


Block Interaction_Page_Legend
1a(1)
Block_Z
This is a block reference symbol.
Blocks are the fundamental unit of lexical scope and structural hierarchy. Each block contains other blocks and/or processes, procedures and data declarations.
Process_A (1,1)
After the process name is the number of process instances at startup and the maximum number of instances. For processes created dynamically, the dashed arrow connects the parent process to the offspring process.
This is a process reference symbol.
Processes specify dynamic behavior using extended finite state machines. Processes operate concurrently, communicating by means of signals and remote variables (import/export).
Unidirectional_
SignalRoute
[Signal5]
Process_B (0,max)
Bidirectional_
SignalRoute
Process_C (1,1)
SignalRoute_
OutOfBlock
PT
[Signal1,
Signal2]
[Signal3,
Signal4]
[Signal2,
Signal6]
[Signal3]
continuation of that signal route in the enclosing block.
Procedure_Name
This is a procedure reference symbol.
A procedure is defined and called in the process where this symbol appears. If declared "remote" the procedure may be imported for calling from other processes. A value-returning procedure, callable in assignment statements, is defined using the "returns" keyword in the formal parameter list.
operator
Operator_Name
This is an operator reference symbol.
Operators for custom sorts may be defined axiomatically or algorithmically. An algorithmic operator is similar to a value-returning procedure, except the operator does not use states nor outputs, and does not modify its source operands.



Process State_Machine_Legend
1a(1)
/* This is a text symbol, used to hold data type (sort) definitions, declarations, signal lists, and other SDL statements that have no graphical representation. */
Process Start symbol (One per process, contains no text.)
State_1
State symbol, arrowhead indicates transition(s) entering the state.
signal_A
Input symbol with wedge on left side used for signals from LLC, SME, self, and others logically above or parallel to this process.
State_2
*
signal_z 'when in any state'
'actions in response to signal_z'
-
* in a state symbol means all states except those listed
- in a state symbol refers to the state from which the transition began.
* (state_x, state_y)
error_signal 'all states except x,y'
'actions to recover from error'
state_N
The transition taken when multiple inputs follow a state is determined by the first of the named signals to reach the head of the input queue.
signal_A, signal_B
signal_C, signal_D,
signal_E 'text extension symbol, holds overflow text'
signal_G
Input symbol with wedge on right side used for signals from PHY & others logically below this process.
'task symbol for algorithmic process steps'
conditional expression
This transition is able to begin only when its Enabling Condition is true.
out_sig_1
Output symbol with point to left side used for signals to LLC, SME, self, and others logically above or parallel to this process.
'start timer' set(end_time, timer)
'stop timer' reset(timer)
Create Request symbol used for dynamic creation of an instance of the specified process type.
'call' procedure (parms)
decision criterion
result_2
result_1
process (parms)
Label
State_3
Label
out_sig_2
Output symbol with point to right side used for signals to PHY & others logically below this process.
signal_A, signal_K
* (signal_B)
Process Stop symbol
State_4
A Priority Input symbol enables its transition if the named signal is anywhere in the process input queue.
'call' macro (parms)
A signal at head of the process input queue that is not named in any of the state's input symbols is discarded unless named in a Save symbol attached to the state. * Save refers to all remaining signal names.
signal_F
priority_ signal
other_signal
State_2
-
Next_State

C.1 Introduction to the MAC formal description

This formal description defines the behavior of IEEE 802.11 MAC entities. The MAC protocol functional decomposition used herein facilitates explicit description of the reference points and durations of the various timed intervals; the bases for generation and/or validation of header fields, service parameters, and MIB attributes; and the interpretation of each value in cases where enumerated data types are used in service parameters.

C.1.1 Fundamental assumptions

The MAC protocol is described as an SDL system, which is a set of extended finite state machines. Each state machine is a set of independent processes, all of which operate concurrently. All variable data-holding entities and procedures exist solely within the context of a single process. In SDL all interprocess communication is done with signals (there are no global variables). Signals may be sent and received explicitly, using SDL's output and input symbols, or implicitly, using SDL's export/import mechanism (only if the variables or procedures are declared "remote"). By default, signals incur delays when traversing channels between blocks; however, only nondelaying channels and signal routes are used in the MAC state machines, and all remote variables and procedures are declared with the "nodelay" property.

State transitions, procedure calls, and tasks (assignment statements and other algorithmic processing steps) are assumed to require zero time. This permits the time intervals that are part of the normative MAC behavior to be defined explicitly, using SDL timers. One unit of system time (a 1.0 change in the value of "now") is assumed to represent one microsecond of real time. Usec (microsecond) and TU (time unit) data types are defined, with operators to convert Usec and TU values to SDL time or duration when necessary.

The SDL system boundary encloses the MAC entities. The LLC, SME, PHY, and distribution system are part of the environment. SDL generally assumes that entities in the environment operate as specified; however, the MAC state machines that communicate with the various SAPs attempt to validate inputs from the environment, and to handle cases where a pair of communicating entities, one within the system and the other outside the system boundary, have different local views of the medium, station, or service state. All stations in an IEEE 802.11 service set are assumed to exhibit the behaviors described herein. Nevertheless, because of the open nature of the wireless medium, the MAC state machines check for error cases that can arise only when an entity on the wireless medium is transmitting IEEE 802.11 PDUs, but is not obeying the communication protocols specified by this standard.

C.1.2 Notation conventions

When practical, names used in the clauses of this standard are spelled identically in this annex. The principal exceptions are those names that conflict with one of SDL's reserved words (such as power management mode "active" is renamed "sta_active" in SDL). To help fit the SDL text into the graphic symbols, acronyms with multiple, sequential capital letters are written with only the first letter capitalized (e.g., "MSDU" is written "Msdu" and "MLMEJoin.request" is written "MlmeJoin.request").

SDL reserved words and the names of variables and synonyms (named constants) begin with lowercase letters. The names of sorts (data types), signals, signal routes, channels, blocks, and processes begin with uppercase letters. The names of certain groups of variables and/or synonyms begin with a particular lowercase letter, followed by the remainder of the name, beginning with an uppercase letter. These groups are

"aNameOfAttribute"	MIB attributes
"cNameOfCapability"	Capability bits, also used for internal values exported as MIB counters
"dNameOfDuration"	Duration (relative time) values, declared as Usec, TU, or Duration
"eNameOfElement"	Element ID values

"mNameOfVariable"	Remote variables used for intra-MAC communication, but not part of the MIB. Most of these variables are exported from the MLME block.
"sNameOfStaticValue"	Synonyms for static data values used within the MAC
"tNameOfTime"	Time (absolute time) values, declared as Usec, TU, or Time. The names of timers begin with "T."

C.1.3 Modeling techniques

State machines are grouped according to defined function sets that are visible, directly or indirectly, at an exposed interface. The emphasis in the organization of the state machines is explicitly to show initiation of and response to events at the exposed interfaces, and time-related actions, including those dependent on the absence of external events (e.g., response timeouts) and intervals measured in derived units (e.g., backoff "time" in units of slots during which the wireless medium is idle). The operations associated with the various state transitions emphasize communication functions. Most of the details regarding insertion, extraction, and encoding of information in fields of the protocol data units is encapsulated with the definitions of those fields. This approach, which relies heavily on SDL's abstract data type and inheritance mechanisms, permits the behavior of the data-holding entities to be precisely defined, without obscuring process flow by adding in-line complexity to the individual state transitions.

The modeling of protocol and service data units requires sorts such as octet strings, and operators such as bitwise boolean functions, which are not predefined in SDL. These sorts and operators are defined in Package macsorts, which appears in Clause C.2.

Protocol and service data unit sorts are based on the **Bit** sort. Bit is a subtype of SDL's predefined Boolean sort. As a result, Bit literals "0" and "1"are alternative names for "false" and "true," and have no numeric significance. To use "0" or "1" as integer values requires a conversion operation. Items of the **Bitstring** sort are 0-origin, variable-length strings of Bits. With Bitstring operands, operators "and," "or," "xor," and "not"operate bitwise, with the length of the result equal to the length of the longest (or only) source string. The **Octet** sort is a subtype of Bitstring that adds conversion operators to and from Integer. Each item of the Octet sort has length=8 {by usage convention in Z.100, enforced in Z.105}. Items of the **Octetstring** sort are 0-origin, variable-length strings of Octets. The **Frame** sort is a subtype of Octetstring that adds operators to extract and to modify all MAC header fields and most other MAC frame fields and elements. Most MAC fields and elements that contain named values with specific value assignments or enumerations are defined as subtypes of Frame, Octetstring, or Bitstring with the names added as literals or synonyms, so the state machines can refer to the names without introducing ambiguity about the value encodings.

Where communication at a SAP or between processes is strictly FIFO, the (implicit) input queue of the SDL processes is used. When more sophisticated queue management is needed, a queue whose entries are instances of one, specified sort is created using the **Queue** generator. Entries on Queue sorts may be added and removed at either the tail or the head, and the number of queue entries may be determined. The contents of a Queue may also be searched to locate entries with particular parameter values.

Clause C.2 contains an SDL-92 Package (a named collection of SDL definitions that can be included by reference into SDL System specification), which is a formal description of the formats and data encodings used in IEEE 802.11 service data units, protocol data units, and the parameters of the service primitives used at each of the service access points supported by the IEEE 802.11 MAC. This package also contains definitions for some data structures and operators used internally by one or more of the MAC state machines.

The behaviors of many intra-MAC operators are part of the normative description of the MAC protocol because results of the specified operations are visible, directly or indirectly, at exposed interfaces. For example, custom operators are used to define the generation of the CRC-32 value used in the FCS field (operator crc32, page 330), the calculation of frame transmission time used as part of the value in the Duration/ID field in certain types of frames (operator calcDur, page 343), the comparison of the values of particular fields of a received MAC header with cached data values as part of the procedure for detecting duplicate frames (operator searchTupleCache, page 320), and numerous other aspects of frame formats and information encoding. On the other hand, data structures used solely for intra-MAC storage or for transferring of information between different state machines of a single station or access point, are only normative to the extent that they define items of internal state and the temporal sequence necessary for proper operation of the MAC protocol. The specific structures and encodings used for internal data storage and communication functions in this formal description do *not* constrain MAC implementations, provided those implementations exhibit the specified behaviors at the defined service access points and, in conjunction with an appropriate PHY, on the wireless medium.

C.2 Data type and operator definitions for the MAC state machines

This clause is in SDL/PR (phrase notation), with the exception of procedural operators, which are defined in SDL/GR (graphic notation). Package macsorts contains the definitions of the sorts (data types with associated operators and literals) and synonyms (named constants) used by the MAC state machines. Package macmib defines data types for attributes in the MAC MIB, and portions of the PHY MIB, accessed by the MAC state machines. Package macmib exists solely to satisfy SDL's strong type checking in the absence of an SDL tool that fully supports Z.105 (the combined use of SDL with ASN.1).

```
/*******************************************************************/
Package macsorts;


/*******************************************************************
 *     Enumerated types used within the MAC state machines
 *******************************************************************/
newtype ChangeType                  /* type of change due at the next boundary */
  literals          dwell,          /* dwell (only with FH PHY) */
                    mocp;           /* medium occupancy (only with PCF) */
endnewtype ChangeType;

newtype Imed        /* priority for queuing MMPDUs, relative to MSDUs */
  literals          head,           /* place MMPDU at head of transmit queue */
                    norm;           /* place MMPDU at tail of transmit queue */
endnewtype Imed;

newtype NavSrc      /* source of duration in SetNav & ClearNav signals */
  literals     rts,                          /* RTS frame */
               cfpBss,  cfendBss,            /* start/end of CFP in own BSS */
               cfpOther,  cfendOther,        /* start/end of CFP in other BSS */
               cswitch,                      /* channel switch */
               misc,                         /* durId from other frame types */
               nosrc;                        /* non-reception events */
endnewtype NavSrc;

newtype PsMode        /* power-save mode of a station (PsResponse signal) */
  literals  sta_active,  power_save,  unknown;   endnewtype PsMode;

newtype PsState       /* power-save state of this station */
  literals  awake,  doze;   endnewtype PsState;

newtype StateErr      /* requests disasoc or deauth (MmIndicate signal) */
  literals  noerr,  class2,  class3;   endnewtype StateErr;

newtype StationState /* asoc/auth state of sta (SsResponse signal) */
  literals  not_auth,  auth_open,  auth_key,  asoc,  dis_asoc;
endnewtype StationState;

newtype TxResult      /* transmission attempt status (PduConfirm signal) */
```



```
  literals  successful,  partial,  retryLimit,  txLifetime
  atimAck,  atimNak;   endnewtype TxResult;

/*******************************************************************
 *     Enumerated types used in PHY service primitives
 *******************************************************************/
newtype CcaStatus    /* <state> parameter of PhyCca.indication */
  literals  busy,  idle;   endnewtype CcaStatus;

newtype PhyRxStat    /* <rxerror> parameter of PhyRxEnd.indication */
  literals  no_error,  fmt_violation,  carrier_lost,  unsupt_rate;
endnewtype PhyRxStat;

/*******************************************************************
 *     PLACEHOLDERS FOR MLME/PLME GET/SET PARAMETER VALUES
 *******************************************************************/
      /* MibAtrib (placeholder in MlmeGet/Set definitions) */
syntype MibAtrib = Charstring   endsyntype MibAtrib;

      /* MibValue (placeholder in MlmeGet/Set definitions) */
syntype MibValue = Integer   endsyntype MibValue;

/*******************************************************************
 *     Enumerated types used in MAC and MLME service primitives
 *******************************************************************/
newtype AuthTyp    /* <authentication type> parm in Mlme primitives */
  inherits Octetstring  operators all;
  adding  literals open_system,  shared_key;
  axioms   open_system == mkOS(0, 2);  shared_key == mkOS(1, 2);
endnewtype AuthType;

newtype AuthTypeSet  powerset( AuthType);   endnewtype AuthTypeSet;

newtype BssType      /* <BSS type> parameter & BSS description element */
  literals  infrastructure,  independent,  any_bss;   endnewtype BssType;

newtype BssTypeSet  powerset( BssType);   endnewtype BssTypeSet;

newtype CfPriority  /* <priority> parameter of various requests */
  literals  contention,  contentionFree;   endnewtype CfPriority;

newtype MibStatus   /* <status> parm of Mlme/Plme Get/Set.confirm */
  literals  success,   invalid,   write_only,   read_only;
endnewtype MibStatus;

newtype MlmeStatus  /* <status> parm of Mlme operation confirm */
  literals  success,  invalid,  timeout,  refused,
             tomany_req,  already_bss;   endnewtype MlmeStatus;

newtype PwrSave      /* <power save mode> parameter of MlmePowerMgt */
  literals  sta_active,  power_save;   endnewtype PwrSave;

newtype Routing      /* <routing info> parameter for MAC data service */
  literals  null_rt;   endnewtype Routing;

newtype RxStatus     /* <reception status> parm of MaUnitdata indication */
  literals  rx_success,  rx_failure;   endnewtype RxStatus;

newtype ScanType     /* <scan type> parameter of MlmeScan.request */
  literals  active_scan,  passive_scan;   endnewtype ScanType;

newtype ServiceClass  /* <service class> parameter for MaUnitdata */
  literals  reorderable,  strictlyOrdered;   endnewtype ServiceClass;
```

```
newtype TxStatus  /* <transmission status> parm of MaUnitdataStatus */
  literals  successful,   retryLimit,   txLifetime,   noBss,
            excessiveDataLength,       nonNullSourceRouting,
            unsupportedPriority,       unavailablePriority,
            unsupportedServiceClass,   unavailableServiceClass,
            unavailableKeyMapping;     endnewtype TxStatus;

/*******************************************************************
 *    Intra-MAC remote variables (names of form mXYZ)
 *******************************************************************/
remote mActingAsAp Boolean nodelay;    /* =true if STA started BSS */
remote mAId  AsocId nodelay;           /* AID assigned to STA by AP */
remote mAssoc  Boolean nodelay;        /* =true if STA associated w/BSS */
remote mAtimW  Boolean nodelay;        /* =true if ATIM window in prog */
remote mBkIP  Boolean nodelay;         /* =true if backoff in prog */
remote mBrates Ratestring nodelay;     /* basic rate set for this sta */
remote mBssId  MacAddr nodelay;        /* identifier of current (I)BSS */
remote mCap  Octetstring nodelay;      /* capability info from MlmeJoin */
remote mCfp  Boolean nodelay;          /* =true if CF period in progress */
remote mDisable  Boolean nodelay;      /* =true if not in any BSS; then */
                                       /* TX only sends probe_req; RX */
                                       /* only accepts beacon, probe_rsp */
remote mDtimCount  Integer nodelay;    /* =0 at Tbtt of Beacon with DTIM */
remote mDtimPeriod Integer nodelay;    /* beacon periods per DTIM period */
remote mFxIP  Boolean nodelay;         /* =true during frame exchange seq */
remote mIbss  Boolean nodelay;         /* =true if STA is member of IBSS */
remote mListenInt  Integer nodelay;    /* beacons between wake up @TBTT */
remote mNavEnd  Time nodelay;          /* NAV end Time, <=now when idle */
remote mNextBdry  Time nodelay;        /* next boundary Time; =0 if none */
remote mNextTbtt  Time nodelay;        /* Time next beacon due to occur */
remote mOrates Ratestring nodelay;     /* operational rate set in use */
remote mPcAvail  Boolean nodelay;      /* =true if point coord in BSS */
remote mPcDlvr  Boolean nodelay;       /* =true if CF delivery only */
remote mPcPoll  Boolean nodelay;       /* =true if CF delivery & polling */
remote mPdly  Usec nodelay;            /* probe delay from start or join */
remote mPsm  PwrSave nodelay;          /* power save mode of STA */
remote mPss  PsState nodelay;          /* power save state of STA */
remote mRxA  Boolean nodelay;          /* =true if RX indicated by PHY */
remote mSsId  Octetstring nodelay;     /* name of the current (I)BSS */
remote procedure TSF nodelay;          /* read & update 64-bit TSF timer */
    fpar Integer, Boolean;   returns Integer;

/*******************************************************************
 *    Named static data values (names of form sXYZ)
 *******************************************************************/
synonym sMaxMsduLng Integer = 2304;    /* max octets in an MSDU */
synonym sMacHdrLng  Integer = 24;      /* octets in data header, no WEP */
synonym sWepHdrLng  Integer = 28;      /* octets in data header with WEP */
synonym sWepAddLng  Integer = 8;       /* octets added for WEP */
synonym sWdsAddLng  Integer = 6;       /* octets added for WDS (addr4) */
synonym sCrcLng  Integer = 4;          /* octets for crc32 (FCS, ICV) */
synonym sMaxMpduLng  Integer =         /* max octets in an MPDU */
      (sMaxMsduLng + sMacHdrLng + sWdsAddLng + sWepAddLng + sCrcLng);
syntype FrameIndexRange = Integer      /* index range for octets in MPDU */
      constants 0 : sMaxMpduLng   endsyntype FrameIndexRange;
synonym sTsOctet  Integer = 24;        /* first octet of Timestamp field */
synonym sMinFragLng Integer = 256;     /* min value for aMpduMaxLength */
synonym sMaxFragNum Integer =          /* maximum fragment number */
      (sMaxMsduLng / (sMinFragLng - sMacHdrLng - sCrcLng));
synonym sAckCtsLng Integer = 112;      /* bits in ACK and CTS frames */
```

```
/********************************************************************
 *     Station configuration flags (static, supplementary to MIB)
 ********************************************************************/
synonym sVersion  Integer = 0;        /* supported Protocol Version */
synonym sCanBeAp  Boolean = false;    /* =true if STA can operate as AP */
synonym sCanBePc  Boolean = false;    /* =true if AP can be Point Coord */
synonym sCfPollable Boolean =true;    /* =true if responds to CF-polls */

/********************************************************************
 *     Discrete microsecond and Time Unit sorts
 ********************************************************************/
       /* SDL does not define the relationship between its concept
       /* of Time and physical time in the system being described.
       /* An abstraction is needed to establish this relationship,
       /* because Time in SDL uses the semantics of Real, whereas
       /* time in the MAC protocol is discrete, with the semantics
       /* of Natural and a step size (resolution) of 1 micosecond. */

       /* Most MAC times are defined using the subtypes of Integer
       /* Usec and TU.  These have operators for explicit conversion
       /* to SDL Time (tUsec, tTU), SDL Duration (dUsec, dTU), and
       /* from SDL Time (uTime, tuTime) as needed to comply with SDL's
       /* strong type checking.  Where the MAC state machines need to
       /* access the contents of the TSF timer, SDL's "now" (current
       /* time) is used.  This yields readable time-dependent code,
       /* but the value of "now" cannot be modified by an SDL program,
       /* so adopting the TSF time from timestamps in received Beacons
       /* or Probe Responses is shown as a call to remote procedure TSF. */

       /* Microsecond sort -- also has operators tmin and tmax */
newtype Usec   inherits Integer   operators all;
  adding  operators
    dUsec : Usec -> Duration;
    tUsec : Usec -> Time;
    uTime : Time -> Usec;
    tmax  : Usec, Usec -> Usec;
    tmin  : Usec, Usec -> Usec;
  axioms
    for all u, w in Usec(
      u >= w ==> tmax(u, w) == u;
      u < w ==> tmax(u, w) == w;
      u >= w ==> tmin(u, w) == w;
      u < w ==> tmin(u, w) == u;
      for all t in Time(     for all r in Real(
          r = float(u) ==> tUsec(u) == Time!(Duration!(r));
          t = Time!(Duration!(r)) and u = fix(r) ==> u == uTime(t);));
      for all d in Duration(     for all r in Real(
          r = float(u) ==> dUsec(u) == Duration!(r); )));
  constants >= 0 /* constrain value range to be non-negative */
endnewtype Usec;

       /* Time Unit sort -- (1 * TU) = (1024 * Usec) */
newtype TU  inherits Integer  operators all;
  adding  operators
    dTU    : TU -> Duration;
    tTU    : TU -> Time;
    tuTime : Time -> TU;
    u2TU   : Usec -> TU;
    tu2U   : TU -> Usec;
  axioms
    for all k in TU(     for all t in Time(     for all r in Real(
          r = float(k) ==> tTU(k) == Time!(Duration!(1024 * r));
          t = Time!(Duration!(r)) and k = (fix(r) / 1024) ==>
              k == tuTime(t);));
```

```
      for all d in Duration(     for all r in Real(
          r = float(k) ==> dTU(k) == Duration!(1024 * r);));
      for all u in Usec(
        u2TU(u) == u / 1024;        k2U(k) == k * 1024; ));
  constants >= 0 /* constrain value range to be non-negative */
endnewtype TU;

/*******************************************************************
 *     Generator for 0-origin String sorts (adapted from Z.105, Annex A)
 *******************************************************************/
      /* String0(sort, nullSymbol) can define strings of any sort.
      /* These strings are indexed starting from 0 rather than 1.
      /* Sorts defined by String0 have the normal String operators, plus
      /* Tail (all but first item), Head (all but last item), and
      /* aggregators S2, S3, S4, S6, S8 (make fixed length strings). */
generator String0(type Item, literal Emptystring)
  literals Emptystring;
  operators
    MkString : Item -> String0;    /* make a string from an item */
    Length : String0 -> Integer;   /* length of string */
    First : String0 -> Item;       /* first item in string */
    Tail : String0 -> String0;     /* all but first item in string */
    Last : String0 -> Item;        /* last item in string */
    head : String0 -> String0;     /* all but last item in string */
    "//" : String0, String0 -> String0;          /* concatenation */
    Extract! : String0, Integer -> Item;         /* get item from string */
    Modify! : String0, Integer, Item -> String0;  /* modify string */
    SubStr : String0, Integer, Integer -> String0;
      /* SubStr(s,i,j) is string0 of length j starting at string0(i) */
    S2 : Item, Item -> String0;        S3 : Item, Item, Item -> String0;
    S4 : Item, Item, Item, Item -> String0;
    S6 : Item, Item, Item, Item, Item, Item -> String0;
    S8 : Item, Item, Item, Item, Item, Item, Item, Item -> String0;
  axioms
    for all item0,item1,item2,item3,item4,item5,item6,item7 in Item(
      for all s, s1, S2, S3 in String0(      for all i, j in Integer(
      /* constructors are Emptystring, MkString, and "//"; */
      /* equalities between constructor terms */
          s // Emptystring == s;
          Emptystring // s == s;
          (s1 // S2) // S3 == s1 // (S2 // S3);
      /* definition of Length by applying it to all constructors */
           type String Length(Emptystring) == 0;
           type String Length(MkString(item0)) == 1;
           type String Length(s1 // S2) == Length(s1) + Length(S2);
      /* definition of Extract! by applying it to all constructors, */
          Extract!(MkString(item0), 0) == item0;
          i < Length(s1) ==> Extract!(s1 // S2, i) == Extract!(s1, i);
          i >= Length(s1)
            ==> Extract!(s1 // S2, i) == Extract!(S2, i - Length(s1));
          i < 0 or i >= Length(s) ==> Extract!(s, i) == error!;
      /* definition of First and Last by other operations */
          First(s) == Extract!(s, 0);
          Last(s) == Extract!(s, Length(s) - 1);
      /* definition of substr(s,i,j) by induction on j, */
          i >= 0 and i <= Length(s) ==> SubStr(s, i, 0) == Emptystring;
          i >= 0 and j > 0 and i + j <= Length(s)
            ==> SubStr(s, i, j) ==
              SubStr(s, i, j - 1) // MkString(Extract!(s, i + j - 1));
          i < 0 or j < 0 or i + j > Length(s) ==> SubStr(s, i, j) ==
              error!;
      /* definition of Modify!, Head, Tail, Sx by other operations */
          Modify!(s, i, item0) ==
            SubStr(s, 0, i) // MkString(item0) //
```



```
                SubStr(s, i + 1, Length(s) - i - 1);
            head(s) == SubStr(s, 0, Length(s) - 1);
            Tail(s) == SubStr(s, 1, Length(s) - 1);
            S2(item0, item1) == MkString(item0) // MkString(item1);
            S3(item0, item1, item2) ==
              MkString(item0) // MkString(item1) // MkString(item2);
            S4(item0, item1, item2, item3) ==
              MkString(item0) // MkString(item1) // MkString(item2) //
                MkString(item3);
            S6(item0, item1, item2, item3, item4, item5) ==
              MkString(item0) // MkString(item1) // MkString(item2) //
                MkString(item3) // MkString(item4) // MkString(item5);
            S8(item0, item1, item2, item3, item4, item5, item6, item7) ==
              MkString(item0) // MkString(item1) // MkString(item2) //
                MkString(item3) // MkString(item4) // MkString(item5) //
                MkString(item6) // MkString(item7); )));
endgenerator String0;

/*****************************************************************
 *     ASN.1-style BIT sort (from Z.105, Annex A)
 *****************************************************************/
        /* Bit is a subtype of Boolean -- bit values 0 and 1 are
        /* not numerals and cannot be used with Integer operators */
newtype Bit   inherits Boolean
    literals  0 = false,  1 = true;   operators all;   endnewtype Bit;

/*****************************************************************
 *     ASN.1-style BIT STRING sort (adapted from Z.105, Annex A)
 *****************************************************************/
        /* Bitstrings are 0-origin strings of Bit.  Z.105 uses ASN.1-style
        /* literals in binary ('1011'B) or hexadecimal ('D3'H), but this
        /* syntax is not accepted for Z.100 string literals.  Therefore,
        /* this version provides only hexadecimal literals 0x00-0xFF.
        /* Bitstring operators '=>', 'not', 'and', 'or', and 'xor' act
        /* bitwise, with the length of the result string equal to the
        /* length of the longest (or only) source string. */
newtype Bitstring String0(Bit, '')
  adding literals     macro Hex_Literals;
  operators
    "not" : Bitstring -> Bitstring;
    "and" : Bitstring, Bitstring -> Bitstring;
    "or" : Bitstring, Bitstring -> Bitstring;
    "xor" : Bitstring, Bitstring -> Bitstring;
    "=>" : Bitstring, Bitstring -> Bitstring;     noequality;
  axioms     macro Hex_Axioms;
    for all s, s1, S2, S3 in Bitstring(
      s = s == true;
      s1 = S2 == S2 = s1;
      s1 /= S2 == not (s1 = S2);
      s1 = S2 == true ==> s1 == S2;
      ((s1 = S2) and (S2 = S3)) ==> s1 = S3 == true;
      ((s1 = S2) and (S2 /= S3)) ==> s1 = S3 == false;
      for all b, b1, b2 in Bit(
        not ('') == '';
        not (MkString(b) // s) == MkString(not (b)) // not (s);
'' and '' == '';
        Length(s) > 0 ==> '' and s == MkString(0) and s;
        Length(s) > 0 ==> s and '' == s and MkString(0);
        (MkString(b1) // s1) and (MkString(b2) // S2) ==
          MkString(b1 and b2) // (s1 and S2);
        s1 or S2 == not (not s1 and not S2);
        s1 xor S2 == (s1 or S2) and not (s1 and S2);
        s1 => S2 == not (not s1 and S2);));
  map  for all b1, b2 in Bitstring literals(
```

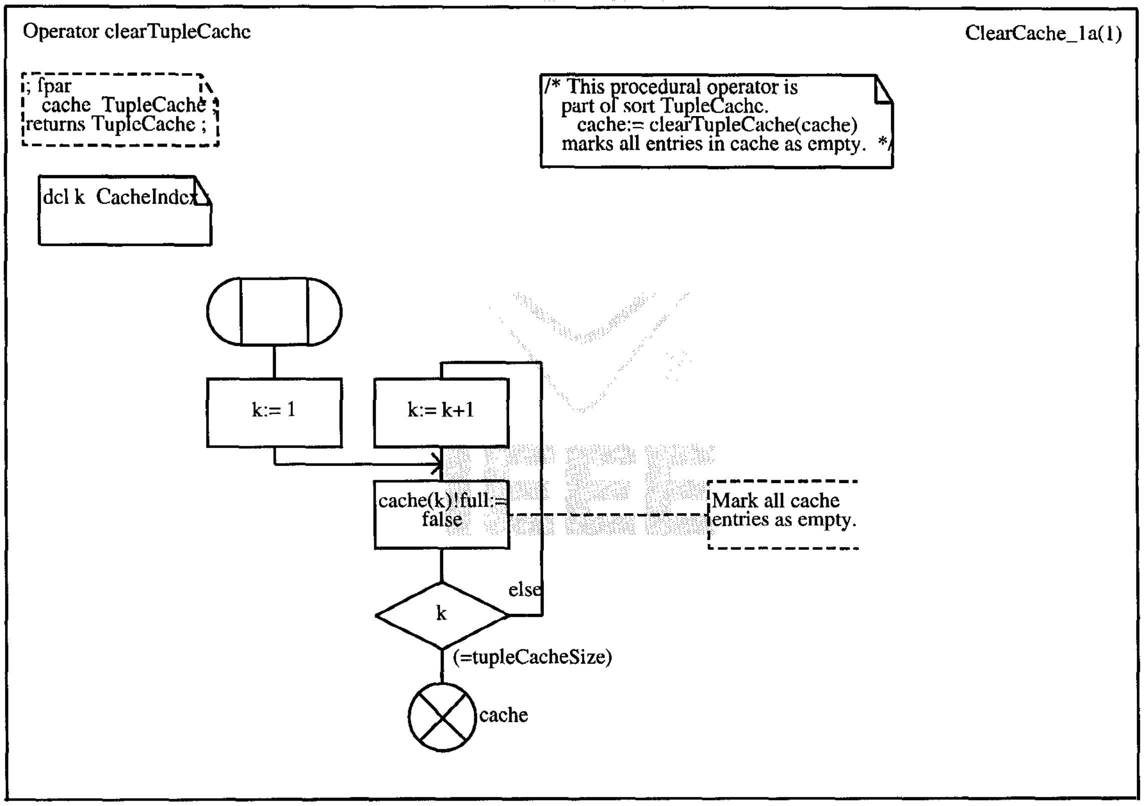
```
      for all bs1, bs2 in Charstring literals(
/* connection to the String generator */
        for all b in Bit literals(
            spelling(b1) = '''' // bs1 // bs2 // '''',
          spelling(b2) = '''' // bs2 // '''',  spelling(b) = bs1
            ==> b1 == MkString(b) // b2; )));
endnewtype Bitstring;

/*******************************************************************
 *     OCTET sort          (influenced by Z.105, Annex A)
 *******************************************************************/
       /* Octet is a subtype of Bitstring where length always =8.
       /* Z.105 adds a "size "keyword to SDL and defines Octet with
       /* "... constants size (8) ... " to impose this length constraint.
       /* Here Octet relies on proper use maintain lengths as multiples
       /* of 8.  Proper length strings are created by the hexadecimal
       /* Bitstring literals (e.g. 0xD5) and operator mkOctet:
       /*     o:= mkOctet(i)      converts a non-negative Integer (mod 256)
       /*                         to an Octet (exactly 8 bits)
       /*     i:= octetVal(o)     converts an Octet to an Integer (0:255)
       /*     o:= flip(o)         reverses bit order of the Octet
       /*                         (0<-->7, 1<-->6, 2<-->5, 3<-->4) */
newtype Octet   inherits Bitstring   operators all;
  adding  operators
    mkOctet  : Integer -> Octet;
    octetVal : Octet -> Integer;
    flip     : Octet -> Octet;
  axioms
    for all i in Integer(     for all z in Octet(
        i = 0 ==> mkOctet(i) == S8(0, 0, 0, 0, 0, 0, 0, 0);
        i = 1 ==> mkOctet(i) == S8(1, 0, 0, 0, 0, 0, 0, 0);
        i > 1 and i <= 255 ==> mkOctet(i) ==
            SubStr((First(mkOctet(i mod 2)) // mkOctet(i / 2)), 0, 8);
        i > 255 ==> mkOctet(i) == mkOctet(i mod 256);
        i < 0 ==> mkOctet(i) == error!;
        z = MkString(0) ==> octetVal(z) == 0;
        z = MkString(1) ==> octetVal(z) == 1;
        Length(z) > 1 and Length(z) <= 8 ==>
            octetVal(z) == octetVal(First(z)) +
                (2 * (octetVal(SubStr(z, 1, Length(z) - 1))));
        Length(z) > 8 ==> octetVal(z) == error!;
        flip(z) == S8(z(7),z(6),z(5),z(4),z(3),z(2),z(1),z(0)); ));
endnewtype Octet;

/*******************************************************************
 *     OCTET STRING sort   (somewhat influenced by Z.105, Annex A)
 *******************************************************************/
       /* Octetstrings are 0-ORIGIN strings of Octet, NOT 1-ORIGIN
       /* strings like Octet_String in Z.105 (hence the name change).
       /* Octetstring has conversion operators to and from Bitstring,
       /* and integer to Octetstring.  Octetstring literals are "null"
       /* and 1-4, 6, 8 item 0x00 strings O1, O2, O3, O4, O6, O8. */
newtype Octetstring String0(Octet, null)
  adding  literals O1, O2, O3, O4, O6, O8;
  operators
    B_S  : Octetstring -> Bitstring;          /* name changed from Z.105 */
    O_S  : Bitstring -> Octetstring;          /* name changed from Z.105 */
    mkOS : Integer,Integer -> Octetstring;    /* mkOS(i1,i2) returns */
             /* mkstring(mkOctet(i1)) padded (0x00) to length i2 */
    mk2octets : Integer -> Octetstring; /* 16-bit integer to 2-octets */
  axioms
    for all b, b1, b2 in Bitstring(
      for all s in Octetstring(     for all o in Octet(
          B_S(null) == '';
```

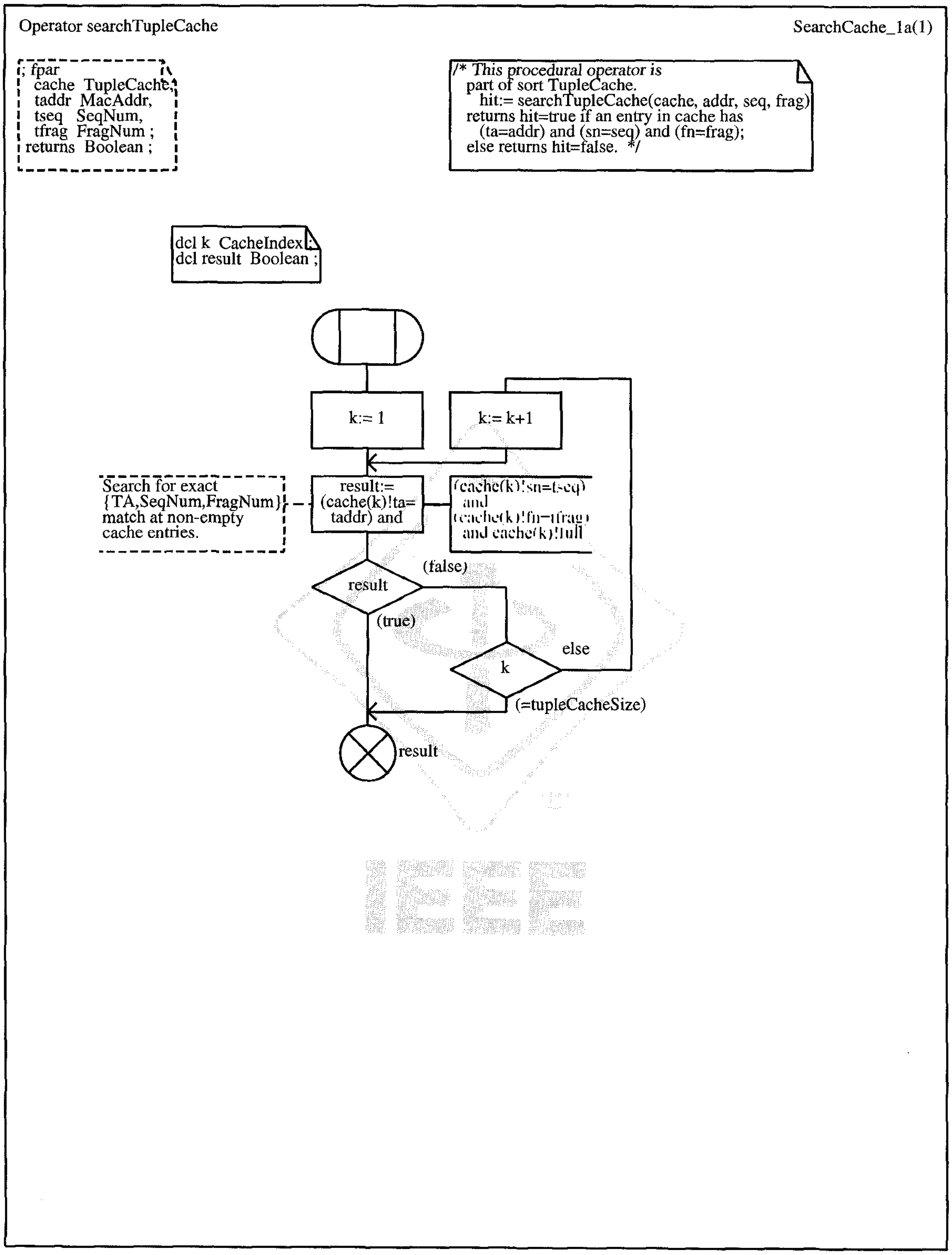


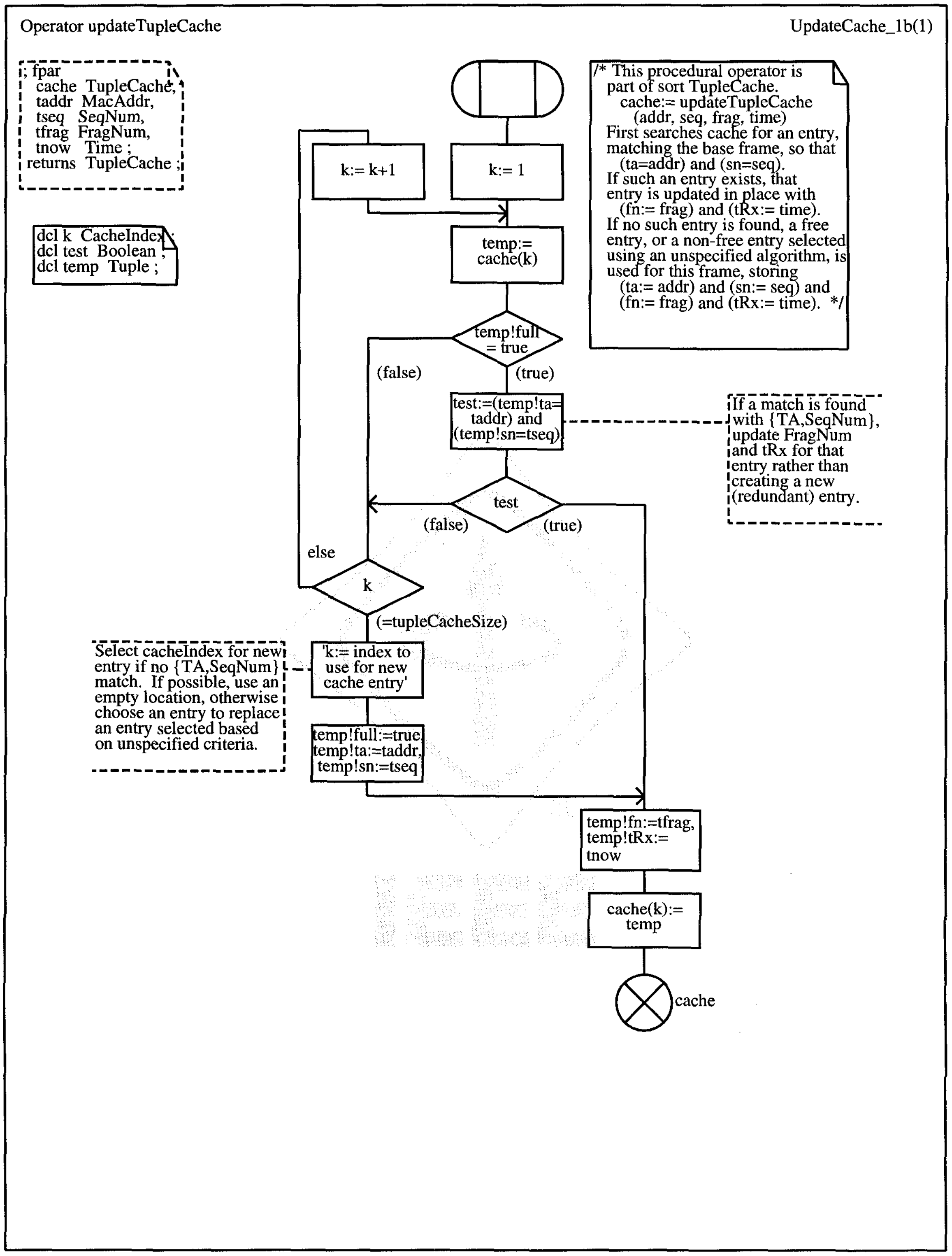
```
          B_S(MkString(o) // s) == o // B_S(s);
          O_S('') == null;
          Length(b1) > 0, Length(b1) < 8 ==>
            O_S(b1) == MkString(b1 or 0x00);  /* expand b1 to 8 bits */
          b == b1 // b2, Length(b1) = 8 ==>
            O_S(b) == MkString(b1) // O_S(b2);
          for all i, k in Integer(
            k = 1 ==> mkOS(i, k) == MkString(mkOctet(i));
            k > 1 ==> mkOS(i, k) == mkOS(i, k - 1) // MkString(0x00);
            k <= 0 ==> error!;
            mk2octets(i) == MkString(mkOctet(i mod 256)) //
              MkString(mkOctet(i / 256)); );
          O1 == MkString(0x00);      O2 == O1 // O1;
          O3 == O2 // O1;            O4 == O2 // O2;
          O6 == O4 // O2;            O8 == O4 // O4; )));
  map  for all O1, O2 in Octetstring literals(
      for all b1, b2 in Bitstring literals(
         spelling(O1) =  spelling(b1),  spelling(O2) =  spelling(b2)
          ==> O1 = O2 == b1 = b2; ));
endnewtype Octetstring;

/*******************************************************************
 *     MAC Address sorts
 *******************************************************************/
      /* MacAddr is a subtype of Octetstring with added operators:
      /*     isGroup(m) =true if given a group address
      /*     isBcst(m)  =true if given the broadcast address
      /*     isLocal(m) =true if given a locally-administered address
      /*     adrOs(m)   converts MacAddr to Octetstring
      /* MAC addresses must be defined to be exactly 6 octets long,
      /* typically using the S6 operator or nullAddr synonym. */
newtype MacAddr   inherits Octetstring   operators all;
  adding  operators
    isGroup : MacAddr -> Boolean;
    isBcst  : MacAddr -> Boolean;
    isLocal : MacAddr -> Boolean;
    adrOs   : MacAddr -> Octetstring;
  axioms
    for all m in MacAddr(
      (Length(m) = 6) and ((Extract!(m,0) and 0x01) = 0x01)
        ==> isGroup(m) == true;
      (Length(m) = 6) and ((Extract!(m,0) and 0x01) = 0x00)
        ==> isGroup(m) == false;
      (Length(m) = 6) and (m = S6(0xFF,0xFF,0xFF,0xFF,0xFF,0xFF))
        ==> isBcst == true;
      (Length(m) = 6) and (m /= S6(0xFF,0xFF,0xFF,0xFF,0xFF,0xFF))
        ==> isBcst == false;
      (Length(m) = 6) and ((Extract!(m,0) and 0x02) = 0x02)
        ==> isLocal == true;
      (Length(m) = 6) and ((Extract!(m,0) and 0x02) = 0x00)
        ==> isLocal == false;
      Length(m) /= 6 ==> error! /* common error! term */;
      for all o in Octetstring(m = MacAddr!(o) == adrOs(m) = o; ));
endnewtype MacAddr;

newtype MacAddrSet   powerset( MacAddr)   endnewtype MacAddrSet;

synonym bcstAddr  MacAddr =      /* Broadcast Address */
             <<type MacAddr>>  S6(0xFF,0xFF,0xFF,0xFF,0xFF,0xFF);

synonym nullAddr  MacAddr =      /* Null Address */
             << type MacAddr>>  S6(0x00,0x00,0x00,0x00,0x00,0x00);
```

```
/******************************************************************
 *     BSS description sorts
 ******************************************************************/
        /* BssDscr is used with MlmeScan.confirm and MlmeJoin.request */
newtype BssDscr  struct
    bdBssId        MacAddr;
    bdSsId         Octetstring;     /* 1 <= length <= 32 */
    bdType         BssType;
    bdBcnPer       TU;              /* beacon period in Time Units */
    bdDtimPer      Integer;         /* DTIM period in beacon periods */
    bdTstamp       Octetstring;     /* 8 Octets from ProbeRsp/Beacon */
    bdStartTs      Octetstring;     /* 8 Octets TSF when rx Tstamp */
    bdPhyParms     PhyParms;        /* empty if not needed by PHY */
    bdCfParms      CfParms;         /* empty if not CfPollable/no PCF */
    bdIbssParms    IbssParms;       /* empty if infrastructure BSS */
    bdCap          Capability;      /* capability information */
    bdBrates       Ratestring;      /* BSS basic rate set */
endnewtype BssDscr;

newtype BssDscrSet   powerset( BssDscr)   endnewtype BssDscrSet;

/******************************************************************
 *     Duplicate filtering support sorts
 ******************************************************************/
syntype FragNum = Integer   /* Range of possible fragment numbers */
      constants 0:sMaxFragNum  endsyntype FragNum;
syntype SeqNum = Integer    /* Range of possible sequence numbers */
      constants 0:4095  endsyntype SeqNum;
newtype Tuple  struct /* for duplicate filtering & defragmentation */
    full      Boolean;        /* =true if Tuple contains valid info */
    ta        MacAddr;        /* transmitting station address (Addr2) */
    sn        SeqNum;         /* Msdu/Mmpdu sequence number */
    fn        FragNum;        /* most recent Mpdu fragment number */
    tRx       Time;           /* reception time (endRx of fragment) */
  default (. false, nullAddr, 0, 0, 0 .);
endnewtype Tuple;

/******************************************************************
 *     TupleCache support sorts
 ******************************************************************/
        /* Number of TupleCache entries and associated index range */
synonym tupleCacheSize Integer = 32;        /* this value is an example,
                           TupleCache size is implementation dependent */
syntype CacheIndex = Integer  constants 1:tupleCacheSize
  endsyntype CacheIndex;

        /* TupleCache array
        /*     cache:= ClearTupleCache(cache)  to initialize cache
        /*     cache:= UpdateTupleCache(cache, addr, seq, frag, endRx)
        /*            if <addr,seq> is already cached, updates frag
        /*            if <addr,seq> not cached, fills an empty entry
        /*              or replaces an entry using an unspecified algorithm
        /*     SearchTupleCache(cache, addr, seq, frag)
        /*            returns true if specified <addr,seq,frag> in cache */
newtype TupleCache  Array( CacheIndex, Tuple);
  adding  operators
    ClearTupleCache  : TupleCache -> TupleCache;
    SearchTupleCache : TupleCache, MacAddr, SeqNum, FragNum -> Boolean;
    UpdateTupleCache : TupleCache, MacAddr, SeqNum, FragNum, Time ->
        TupleCache;
  operator ClearTupleCache;
    fpar cache TupleCache;  returns TupleCache;  referenced;
  operator SearchTupleCache;
    fpar cache TupleCache,  taddr MacAddr,  tseq SeqNum,  tfrag FragNum;
```

```
    returns Boolean;  referenced;
  operator UpdateTupleCache;
    fpar cache TupleCache,  taddr MacAddr,  tseq SeqNum,  tfrag FragNum,
    tnow Time;  returns TupleCache;  referenced;
endnewtype TupleCache;

/*******************************************************************
 *     32-bit Counter sort and Integer string sort
 *******************************************************************/
       /* This sort used for MIB counters, needed because SDL Integers
       /* have no specified maximum value.  inc(counter) increments the
       /* counter value by 1, with wraparound from (2^32)-1 to 0. */
newtype Counter32   inherits Integer   operators all;
  adding  operators
    inc : Counter32 -> Counter32;
  axioms
    for all c in Counter32(
      c < 4294967295 ==> inc(c) == c + 1;
      c >= 4294967295 ==> inc(c) == 0; );
endnewtype Counter32;

       /* String (1-origin) of Integer */
newtype Intstring   String( Integer, noInt);   endnewtype Intstring;
```



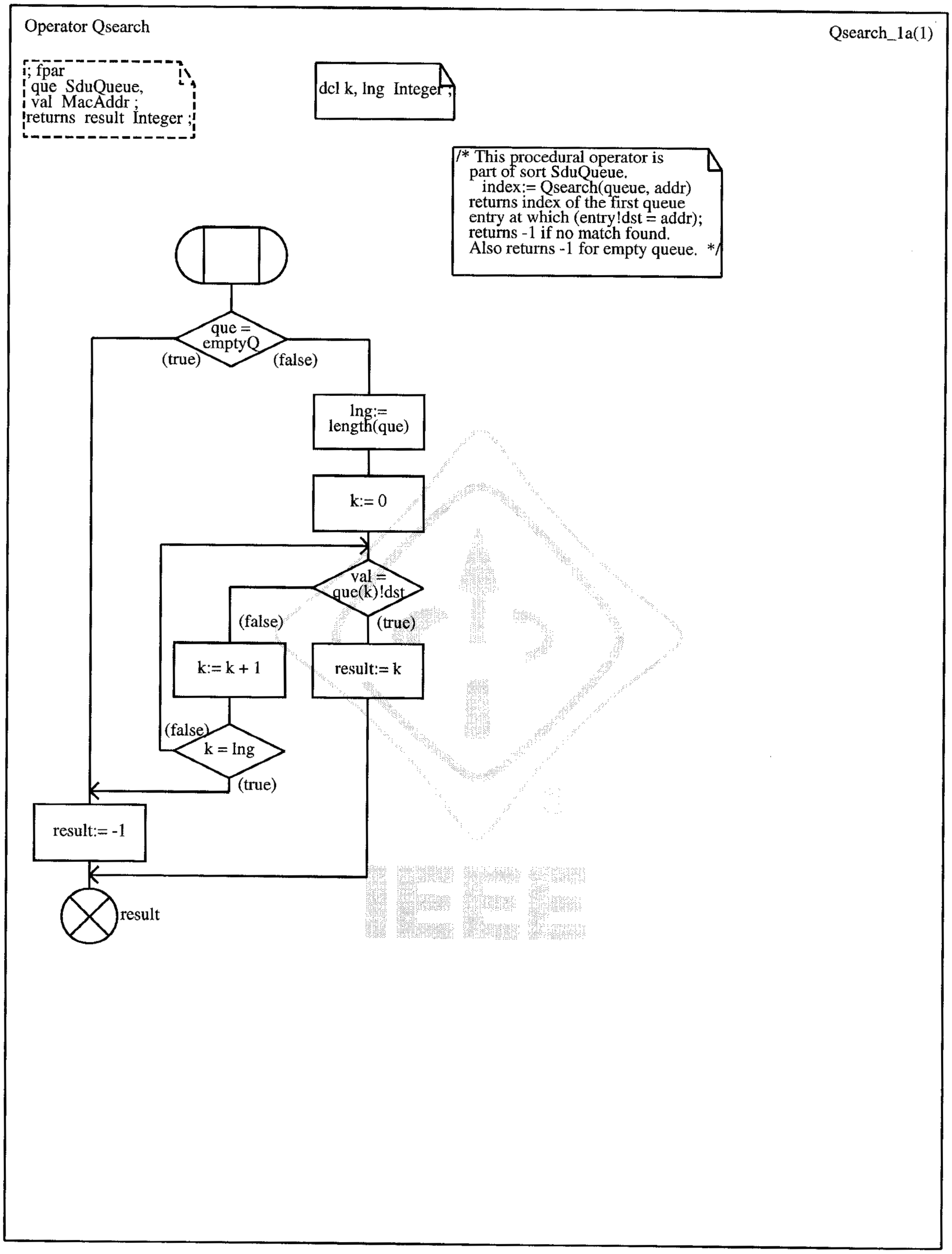

Operator searchTupleCache
SearchCache_1a(1)
; fpar
cache TupleCache,
taddr MacAddr,
tseq SeqNum,
tfrag FragNum ;
returns Boolean ;
/* This procedural operator is
part of sort TupleCache.
hit:= searchTupleCache(cache, addr, seq, frag)
returns hit=true if an entry in cache has
(ta=addr) and (sn=seq) and (fn=frag);
else returns hit=false. */
dcl k CacheIndex ;
dcl result Boolean ;
k:= 1
k:= k+1
Search for exact
{TA,SeqNum,FragNum}
match at non-empty
cache entries.
result:=
(cache(k)!ta=
taddr) and
(cache(k)!sn=tseq)
and
(cache(k)!fn=tfrag)
and cache(k)!full
result
(false)
(true)
k
else
(=tupleCacheSize)
result



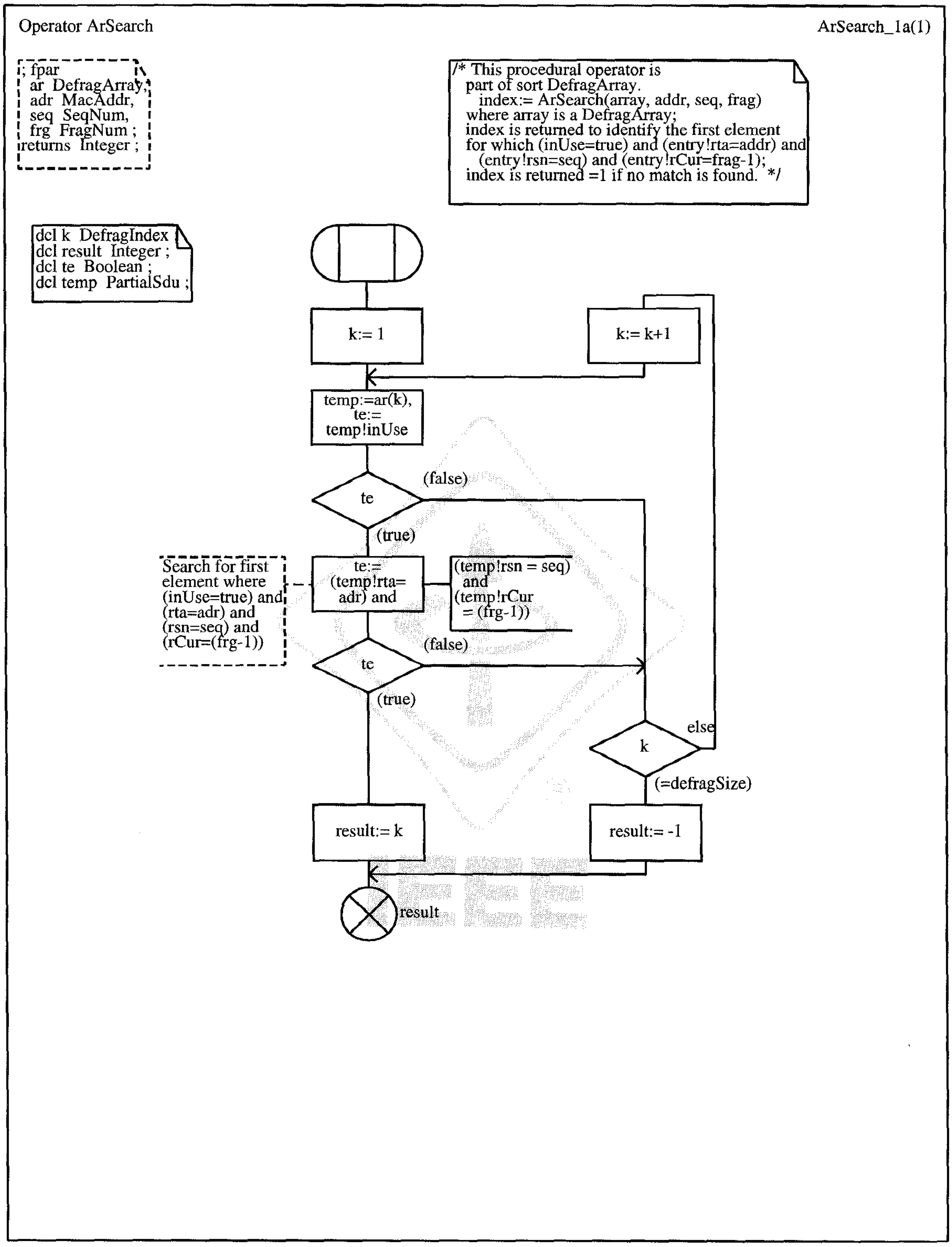
Operator updateTupleCache
UpdateCache_1b(1)
; fpar
cache TupleCache,
taddr MacAddr,
tseq SeqNum,
tfrag FragNum,
tnow Time ;
returns TupleCache ;
dcl k CacheIndex ;
dcl test Boolean ;
dcl temp Tuple ;
/* This procedural operator is part of sort TupleCache.
cache:= updateTupleCache (addr, seq, frag, time)
First searches cache for an entry, matching the base frame, so that (ta=addr) and (sn=seq).
If such an entry exists, that entry is updated in place with (fn:= frag) and (tRx:= time).
If no such entry is found, a free entry, or a non-free entry selected using an unspecified algorithm, is used for this frame, storing (ta:= addr) and (sn:= seq) and (fn:= frag) and (tRx:= time). */
k:= k+1
k:= 1
temp:= cache(k)
temp!full = true
(false)
(true)
test:=(temp!ta= taddr) and (temp!sn=tseq)
If a match is found with {TA,SeqNum}, update FragNum and tRx for that entry rather than creating a new (redundant) entry.
test
(false)
(true)
else
k
(=tupleCacheSize)
Select cacheIndex for new entry if no {TA,SeqNum} match. If possible, use an empty location, otherwise choose an entry to replace an entry selected based on unspecified criteria.
'k:= index to use for new cache entry'
temp!full:=true
temp!ta:=taddr,
temp!sn:=tseq
temp!fn:=tfrag,
temp!tRx:=
tnow
cache(k):=
temp
cache

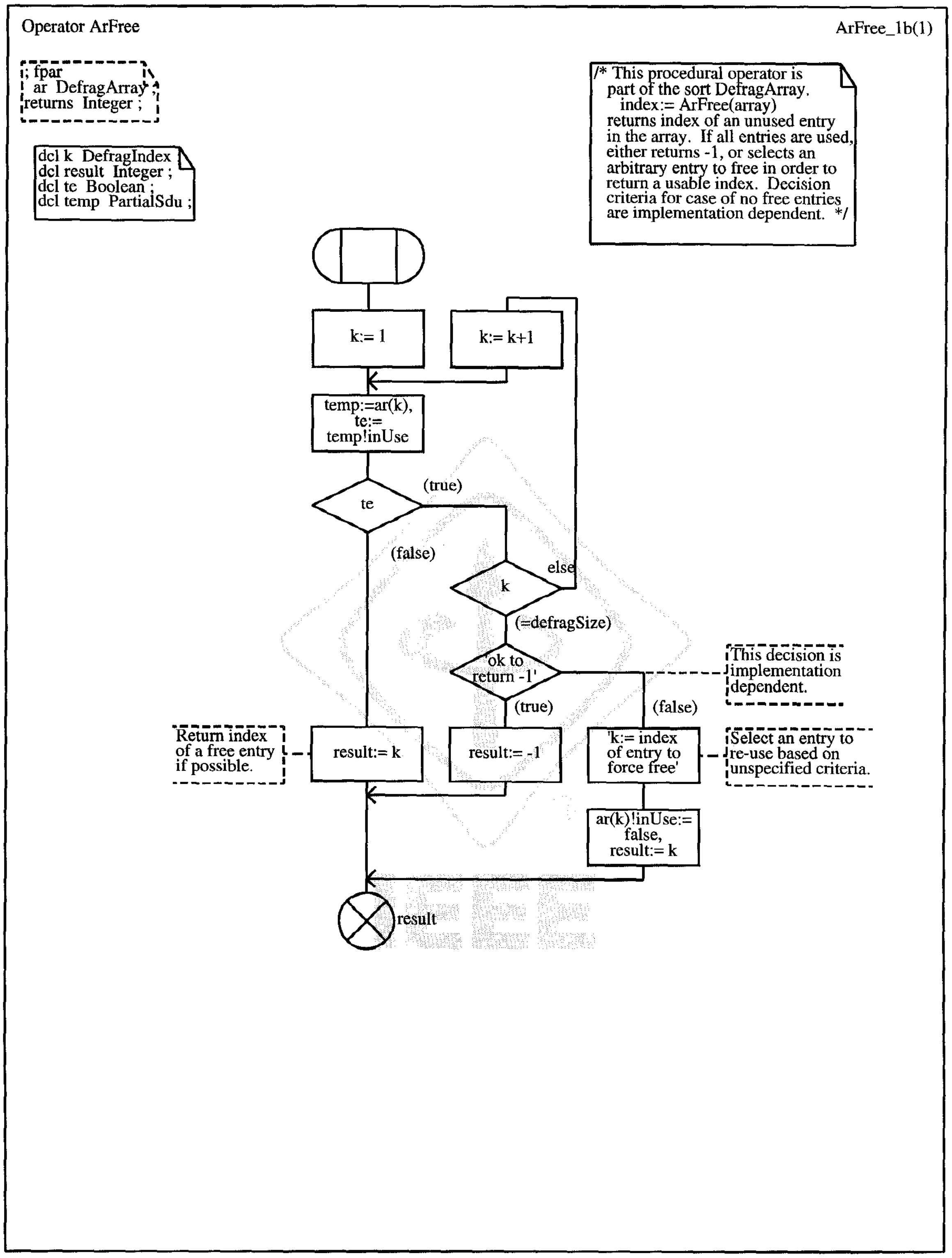
```
/*****************************************************************
 *     Generator for Queue sorts
 *****************************************************************/
       /* The Queue generator is derived from the String0 generator
       /* to create Queues of any sort.  Queues operators are:
       /*    Qfirst(queue,item)  adds item as the first queue element
       /*    Qlast(queue,item)   adds item as the last queue element
       /* and the String0 operators Length, //, First, Last, Head, Tail
       /* Because operators can only return a single value, removing an
       /* element from a queue is a 2-step process:
       /*    dequeue first:   item:=First(queue);   queue:=Tail(queue);
       /*    dequeue last:    item:=Last(queue);    queue:=Head(queue); */
generator Queue(type Item, literal Emptyqueue)
  literals Emptyqueue;
  operators
    MkQ      : Item -> Queue;            /* make a queue from an item */
    Lengt    : Queue -> Integer;         /* number of items on queue */
    First    : Queue -> Item;            /* first item on queue */
    Qfirst   : Queue,Item -> Queue;      /* add item as first on queue */
    Tail     : Queue -> Queue;           /* all but first item on queue */
    Last     : Queue -> Item;            /* last item on queue */
    Qlast    : Queue,Item -> Queue;      /* add item as last on queue */
    head     : Queue -> Queue;           /* all but last item on queue */
    "//"     : Queue,Queue -> Queue;     /* concatenation */
    Extract! : Queue,Integer -> Item;    /* copy item from queue */
    Modify!  : Queue,Integer,Item -> Queue; /* modify item in queue */
    SubQ     : Queue,Integer,Integer -> Queue;
      /* SubQ(q,i,j) queue of length j starting from queue(i) */
  axioms
    for all item0 in Item(     for all q, q1, q2, q3 in Queue(
        for all i, j in Integer(
      /* constructors are Emptyqueue, MkQueue, and "//"; */
      /* equalities between constructor terms */
         q // Emptyqueue == q;
         Emptyqueue // q == q;
         (q1 // q2) // q3 == q1 // (q2 // q3);
      /* definition of Length by applying it to all constructors */
          type Queue Length(Emptyqueue) == 0;
          type Queue Length(MkQueue(item0)) == 1;
          type Queue Length(q1 // q2) == Length(q1) + Length(q2);
      /* definition of Extract! by applying it to all constructors, */
        Extract!(MkQueue(item0), 0) == item0;
         i < Length(q1) ==> Extract!(q1 // q2, i) == Extract!(q1, i);
         i >= Length(q1)
           ==> Extract!(q1 // q2, i) == Extract!(q2, i - Length(q1));
         i < 0 or i >= Length(q) ==> Extract!(q, i) == error!;
      /* definition of First and Last by other operations */
         First(q) == Extract!(q, 0);
         Last(q) == Extract!(q, Length(q) - 1);
      /* definition of SubQ(q,i,j) by induction on j, */
         i >= 0 and i <= Length(q) ==> SubQ(q, i, 0) == Emptyqueue;
         i >= 0 and j > 0 and i + j <= Length(q) ==> SubQ(q, i, j) ==
             SubQ(q, i, j - 1) // MkQueue(Extract!(q, i + j - 1));
         i < 0 or j < 0 or i + j > Length(q) ==> SubQ(q,i,j) == error!;
      /* define Modify!, Head, Tail, Qfirst, Qlast by other ops */
         Modify!(q, i, item0) == SubQ(q, 0, I) //
             MkQueue(item0) // SubQ(q, i + 1, Length(q) - i - 1);
         head(q) == SubQ(q, 0, Length(q) - 1);
         Tail(q) == SubQ(q, 1, Length(q) - 1);
         Qfirst(q, item0) == MkQueue(item0) // q;
         Qlast(q, item0) == q // MkQueue(item0); )));
endgenerator Queue;
```

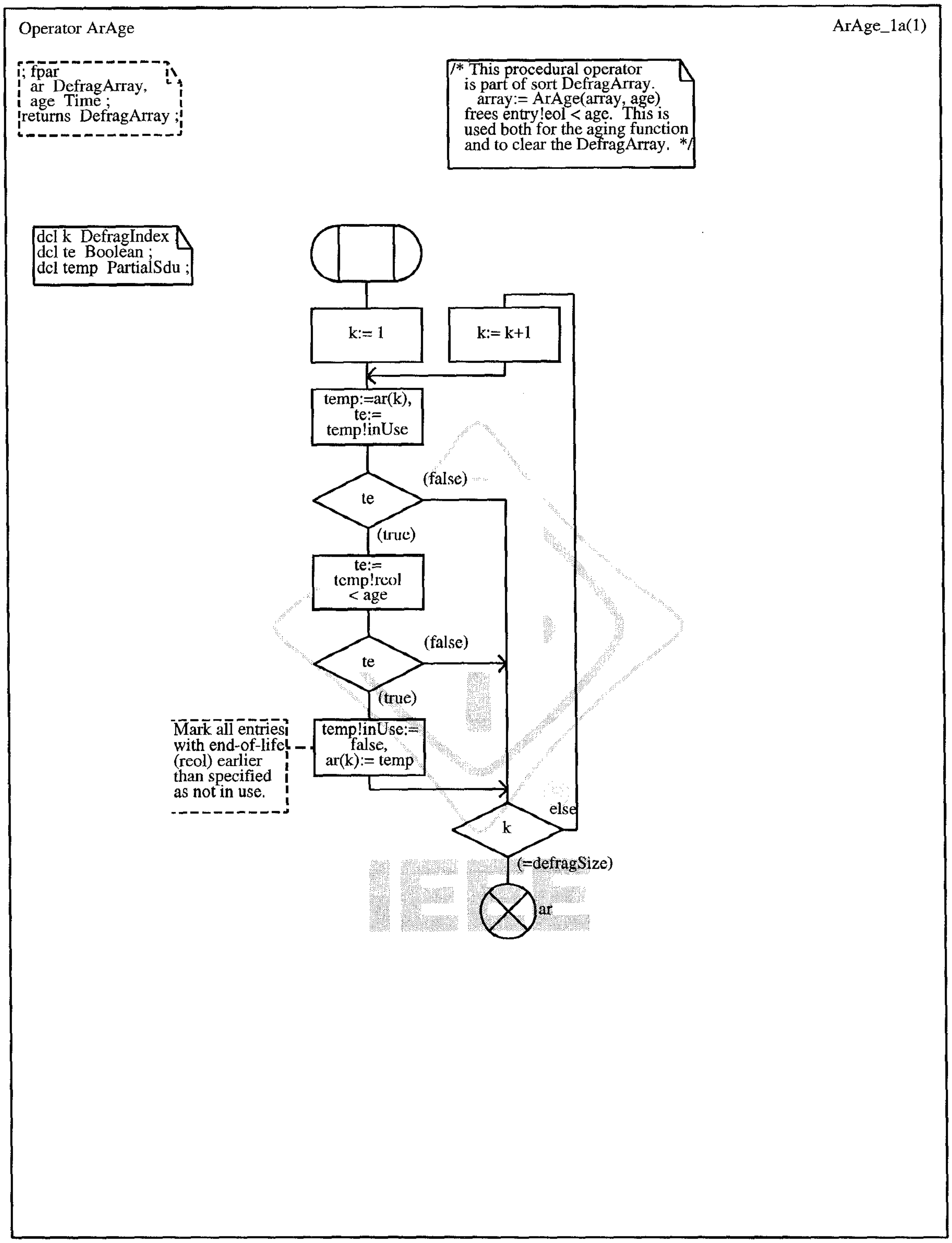
```
/*****************************************************************
 *     Fragmentation support sorts
 *****************************************************************/
       /* Array to hold up to FragNum fragments of an Msdu/Mmpdu */
newtype FragArray  Array(FragNum, Frame);   endnewtype FragArray;

       /* FragSdu structure is for OUTGOING MSDUs/MMPDUs (called SDUs)
       /* Each SDU, even if not fragmented, is held in an instance of
       /* this structure awaiting its (re)transmission attempt(s).
       /* Transmit queue(s) are ordered lists of FragSdu instances. */
newtype FragSdu struct
    fTot      FragNum;       /* number of fragments in pdus FragArray */
    fCur      FragNum;       /* next fragment number to send */
    fAnc      FragNum;       /* next fragment to announce in ATIM or TIM
                                when fAnc > fCur, pdus(fCur)+ may be sent */
    eol       Time;          /* set to (now + dUsec(aMaxTxMsduLifetime))
                                when the entry is created */
    sqf       SeqNum;        /* SDU sequence number, set at 1st Tx attempt */
    src       Integer;       /* short retry counter for this SDU */
    lrc       Integer;       /* long retry counter for this SDU */
    dst       MacAddr;       /* destinaton address */
    grpa      Boolean;       /* =true if RA (not DA) is a group address */
    psm       Boolean;       /* =true if RA (not DA) may be in pwr_save */
    resume    Boolean;       /* =true if fragment burst being resumed */
    cnfTo     PId;           /* address to which confirmation is sent */
    txrate    Rate;          /* data rate used for initial fragment */
    cf        CfPriority;    /* requested priority (from LLC) */
    pdus      FragArray;     /* array of Frame to hold fragments */
endnewtype FragSdu;

       /* Queue of FragSdu
       /* for power save buffers, etc., searchable with Qsearch operator:
       /*     index:= Qsearch(queue, addr)      where queue is an SduQueue,
       /* index identifies the first queue entry at which
       /* entry!dst = addr; or as -1 if no match (or queue empty). */
newtype SduQueue Queue(FragSdu, emptyQ);
  adding  operators
    qSearch : SduQueue, MacAddr -> Integer;
  operator qSearch;
    fpar que SduQueue,  val MacAddr;  returns Integer; referenced;
endnewtype SduQueue;
```


Operator Qsearch
Qsearch_1a(1)
; fpar
que SduQueue,
val MacAddr ;
returns result Integer ;
dcl k, lng Integer ;
/* This procedural operator is part of sort SduQueue.
index:= Qsearch(queue, addr)
returns index of the first queue entry at which (entry!dst = addr);
returns -1 if no match found.
Also returns -1 for empty queue. */
que = emptyQ
(true)
(false)
lng:= length(que)
k:= 0
val = que(k)!dst
(false)
(true)
k:= k + 1
result:= k
(false)
k = lng
(true)
result:= -1
result

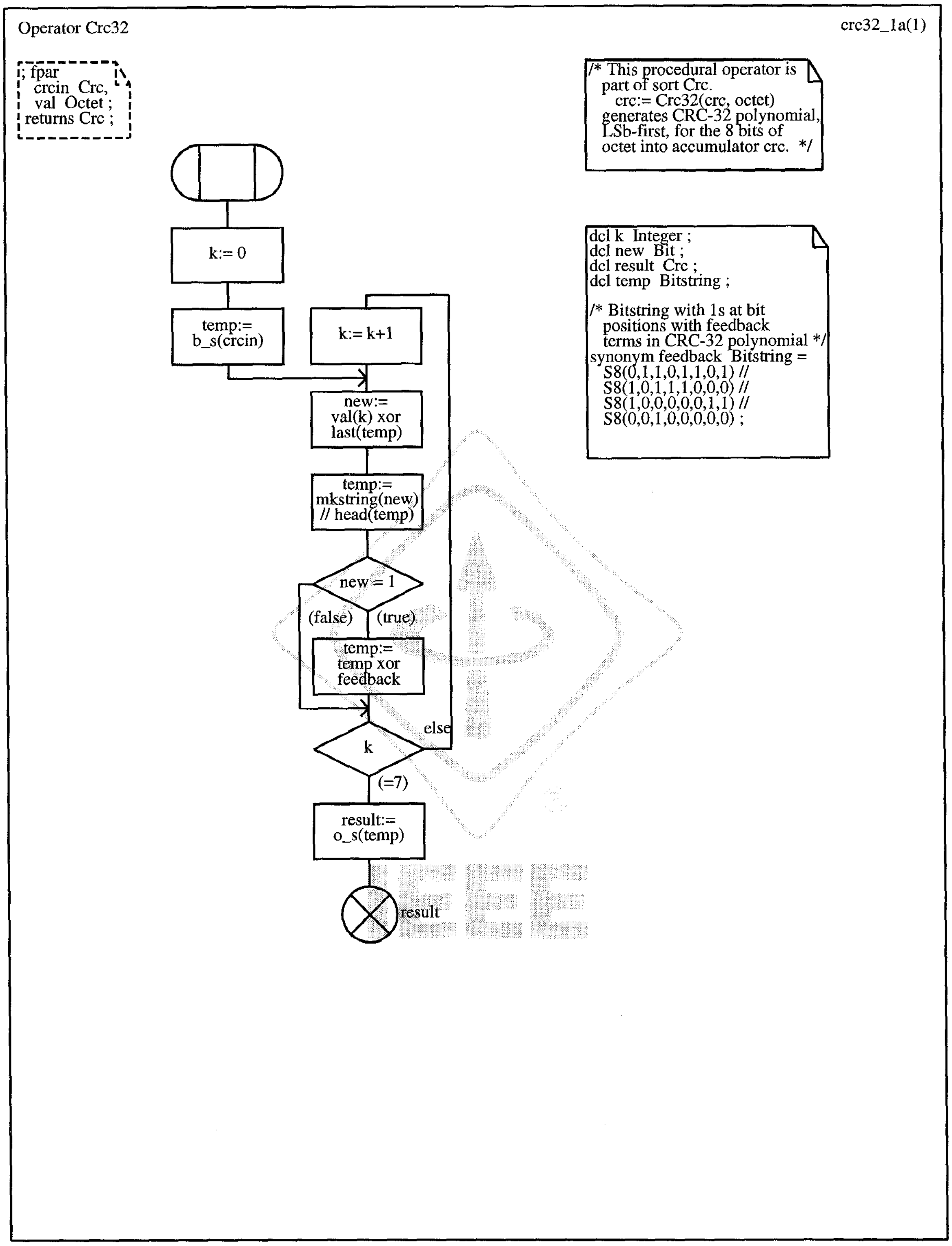
```
/*******************************************************************
 *     Defragmentation support sorts
 *******************************************************************/

      /* The PartialSdu structure is for INCOMPLETE MSDUs/MMPDUs
      /* (generically SDUs) for which at least 1 fragment has been
      /* received.  Unfragmented SDUs are reported upward immediately,
      /* and are never stored in instances of this structure.  */
newtype PartialSdu struct
    inUse     Boolean;      /* =true if this instance holds any fragments */
    rta       MacAddr;      /* transmitting station (Addr2) */
    rsn       SeqNum;       /* SDU sequence number */
    rCur      FragNum;      /* fragment number of most recent Mpdu */
    reol      Time;         /* (now+dUsec(aMaxReceiveLifetime) @ 1st Mpdu */
    rsdu      Frame;        /* buffer where Mpdus are concatenated */
  default (. false, nullAddr, 0, 0, 0, null .);
endnewtype PartialSdu;

newtype PartialSduKeys struct      /* if aPrivacyOptionImplemented=true *
    wDefKeys      KeyVector;       /* default keys when 1st frag received */
    wKeyMap       KeyMapArray;     /* key mappings when 1st frag received */
    wExclude      Boolean;         /* aExcludeUnencrypted @ 1st frag rx */
endnewtype PartialSduKeys;

      /* Number of entries in defragmentation array at this station.
      /* The value is implementation dependent (min=3, see 9.5). */
synonym defragSize Integer = 6;
syntype defragIndex = Integer   constants 1:defragSize
endsyntype defragIndex;

      /* Array of PartialSdu for use defragmenting Msdus and Mmpdus.
      /* Searchable using the ArSearch operator
      /*     index:= ArSearch(array, addr, seq, frag)
      /* where index is returned to identify the first element for which
      /* ((inUse = true) and (entry!rta = addr) and (entry!rsn = seq)
      /*  and (entry!rCur = (frag-1));   or as =1 if no match found.
      /*     index:= ArFree(array)  returns the index of a free entry,
      /* or -1 if no entries free.  May free an entry, selected using
      /* an unspecified algorithm, to avoid returning -1.
      /*     array:= ArAge(array, age)
      /* frees where (entry!eol < age), also used to clear array. */
newtype DefragArray  Array( defragIndex, PartialSdu);
  adding  operators
    ArSearch : DefragArray, MacAddr, SeqNum, FragNum -> Integer;
    ArFree   : DefragArray -> Integer;
    ArAge    : DefragArray, Time -> DefragArray;
  operator ArSearch;
    fpar  ar DefragArray,   adr MacAddr,   seq SeqNum,   frg FragNum;
    returns Integer; referenced;
  operator ArFree;  fpar  ar DefragArray;   returns Integer; referenced;
  operator ArAge;  fpar  ar DefragArray,   age Time;
    returns DefragArray; referenced;
endnewtype DefragArray;

newtype DefragKeysArray  Array( defragIndex, PartialSduKeys);
endnewtype DefragKeysArray;
```

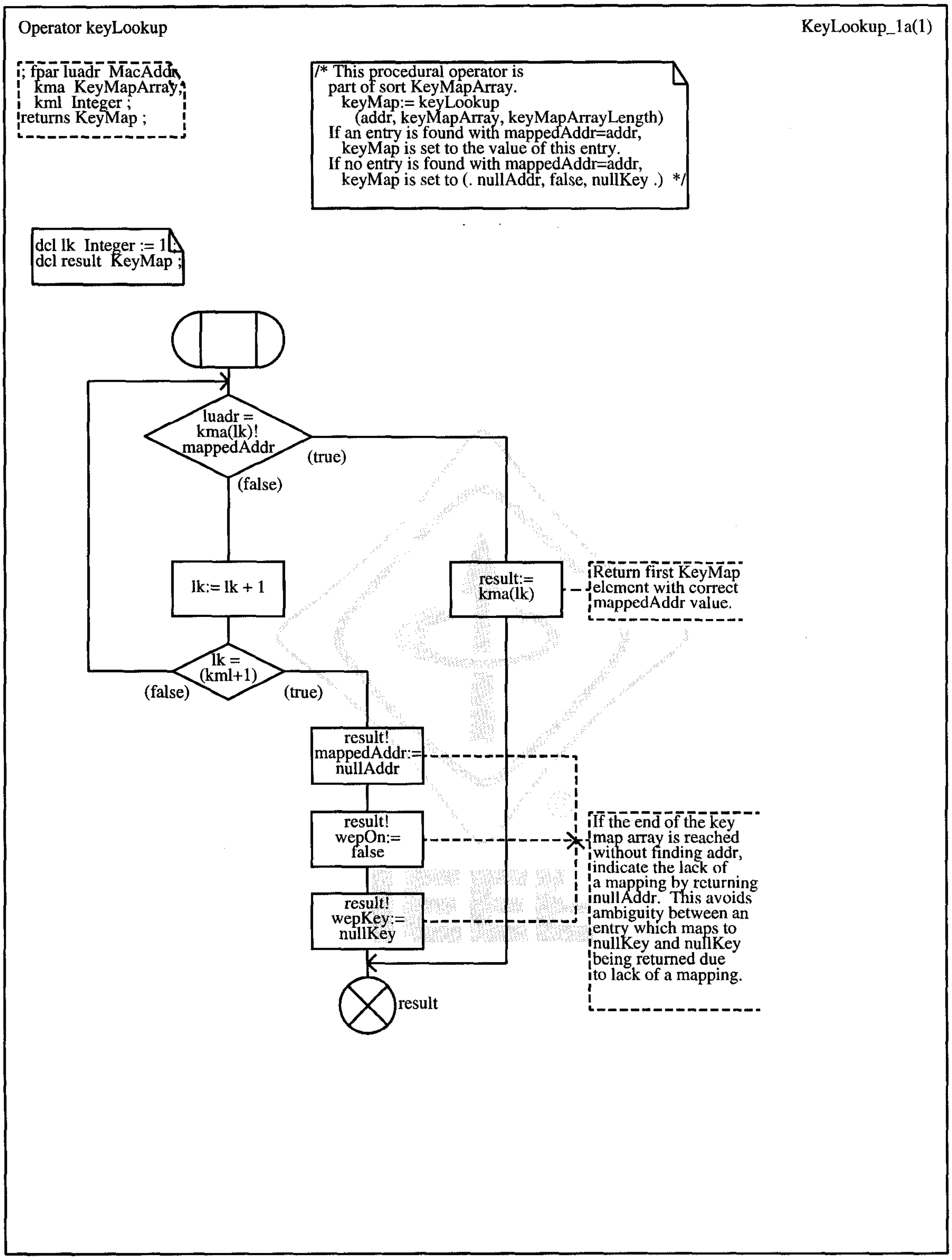

Operator ArSearch
ArSearch_1a(1)
; fpar
ar DefragArray,
adr MacAddr,
seq SeqNum,
frg FragNum ;
returns Integer ;
/* This procedural operator is
part of sort DefragArray.
index:= ArSearch(array, addr, seq, frag)
where array is a DefragArray;
index is returned to identify the first element
for which (inUse=true) and (entry!rta=addr) and
(entry!rsn=seq) and (entry!rCur=frag-1);
index is returned =1 if no match is found. */
dcl k DefragIndex
dcl result Integer ;
dcl te Boolean ;
dcl temp PartialSdu ;
k:= 1
k:= k+1
temp:=ar(k),
te:=
temp!inUse
te
(false)
(true)
Search for first
element where
(inUse=true) and
(rta=adr) and
(rsn=seq) and
(rCur=(frg-1))
te:=
(temp!rta=
adr) and
(temp!rsn = seq)
and
(temp!rCur
= (frg-1))
te
(false)
(true)
else
k
(=defragSize)
result:= k
result:= -1
result


Operator ArFree
ArFree_1b(1)
; fpar
ar DefragArray ;
returns Integer ;
dcl k DefragIndex
dcl result Integer ;
dcl te Boolean ;
dcl temp PartialSdu ;
/* This procedural operator is part of the sort DefragArray.
index:= ArFree(array)
returns index of an unused entry in the array. If all entries are used, either returns -1, or selects an arbitrary entry to free in order to return a usable index. Decision criteria for case of no free entries are implementation dependent. */
k:= 1
k:= k+1
temp:=ar(k), te:= temp!inUse
te
(true)
(false)
k
else
(=defragSize)
'ok to return -1'
This decision is implementation dependent.
(true)
(false)
Return index of a free entry if possible.
result:= k
result:= -1
'k:= index of entry to force free'
Select an entry to re-use based on unspecified criteria.
ar(k)!inUse:= false, result:= k
result


Operator ArAge
ArAge_1a(1)
; fpar
ar DefragArray,
age Time ;
returns DefragArray ;
/* This procedural operator
is part of sort DefragArray.
array:= ArAge(array, age)
frees entry!eol < age. This is
used both for the aging function
and to clear the DefragArray. */
dcl k DefragIndex ;
dcl te Boolean ;
dcl temp PartialSdu ;
k:= 1
k:= k+1
temp:=ar(k),
te:=
temp!inUse
(false)
te
(true)
te:=
temp!reol
< age
(false)
te
(true)
Mark all entries
with end-of-life
(reol) earlier
than specified
as not in use.
temp!inUse:=
false,
ar(k):= temp
else
k
(=defragSize)
ar


```
/*****************************************************************
 *    CRC-32 sorts (for FCS and ICV)
 *****************************************************************/

      /* Crc is a subtype of Octetstring with added operators:
      /*     crc:= Crc32(crc,octet)
      /* updates the crc value to include the new octet, and
      /*     Mirror(crc), which returns a Crc value with the order
      /* of the octets, and of the bits in each octet, reversed for
      /* MSb-first transmission (see 7.1.1).  Crc variables must have
      /* exactly 4 octets, which is done using initCrc or S4. */
newtype Crc   inherits Octetstring   operators all;
  adding  operators
    Crc32  : Crc, Octet -> Crc;
    mirror : Crc -> Octetstring;
  operator Crc32;  fpar crcin Crc, val Octet;  returns Crc;  referenced;
  axioms     for all c in Crc(
      mirror(c) == S4(flip(c(3)),flip(c(2)),flip(c(1)), flip(c(0))); );
endnewtype Crc;
synonym initCrc Crc =    /* Initial Crc value (all 1s) */
      << type Crc>> S4(0xFF,0xFF,0xFF,0xFF);
synonym goodCrc Crc =    /* Unique remainder for valid CRC-32 */
      << type Crc>> S4(0x7B,0xDD,0x04,0xC7);


/*****************************************************************
 *    WEP support sorts
 *****************************************************************/
syntype KeyIndex = Integer   constants 0:3   endsyntype KeyIndex;
newtype PrngKey inherits Octetstring   operators all;
  adding literals nullKey;  /* nullKey is not any of 2^40 key values */
  axioms nullKey == null;   default nullKey;   endnewtype PrngKey;
newtype KeyVector        /* vector of default WEP keys */
  Array( KeyIndex, PrngKey);   endnewtype KeyVector;
      /* Number of entries in aWepKeyMappings array at this station.
      /* implementation dependent value, minimum=10 (see 8.3.2). */
synonym sWepKeyMappingLength Integer = 10;
syntype KeyMappingRange = Integer
      constants 1:sWepKeyMappingLength   endsyntype KeyMappingRange;
newtype KeyMap  struct  /* structure used for entries in KeyMapArray */
    mappedAddr    MacAddr;
    wepOn         Boolean;
    wepKey        PrngKey;
endnewtype KeyMap;
      /* KeyMapArray -- used for aWepKeyMapping table;
      /* an array of KeyMap indexed by KeyMappingRange, with operator
      /*     KeyMap := keyLookup(addr, keyMapArray, keyMapArrayLength)
      /* returns the KeyMap entry for the specified addr, or
      /* (. nullAddr, false, nullKey .) if no mapping for addr. */
newtype KeyMapArray  Array( KeyMappingRange, KeyMap);
  adding  operators
    keyLookup : MacAddr, KeyMapArray, Integer -> KeyMap;
  operator keyLookup;
    fpar  luadr MacAddr,   kma KeyMapArray,   kml Integer;
    returns KeyMap; referenced;
endnewtype KeyMapArray;
```


Operator Crc32
crc32_1a(1)
; fpar
crcin Crc,
val Octet ;
returns Crc ;
/* This procedural operator is
part of sort Crc.
crc:= Crc32(crc, octet)
generates CRC-32 polynomial,
LSb-first, for the 8 bits of
octet into accumulator crc. */
dcl k Integer ;
dcl new Bit ;
dcl result Crc ;
dcl temp Bitstring ;
/* Bitstring with 1s at bit
positions with feedback
terms in CRC-32 polynomial */
synonym feedback Bitstring =
S8(0,1,1,0,1,1,0,1) //
S8(1,0,1,1,1,0,0,0) //
S8(1,0,0,0,0,0,1,1) //
S8(0,0,1,0,0,0,0,0) ;
k:= 0
temp:=
b_s(crcin)
k:= k+1
new:=
val(k) xor
last(temp)
temp:=
mkstring(new)
// head(temp)
new = 1
(false)
(true)
temp:=
temp xor
feedback
k
else
(=7)
result:=
o_s(temp)
result



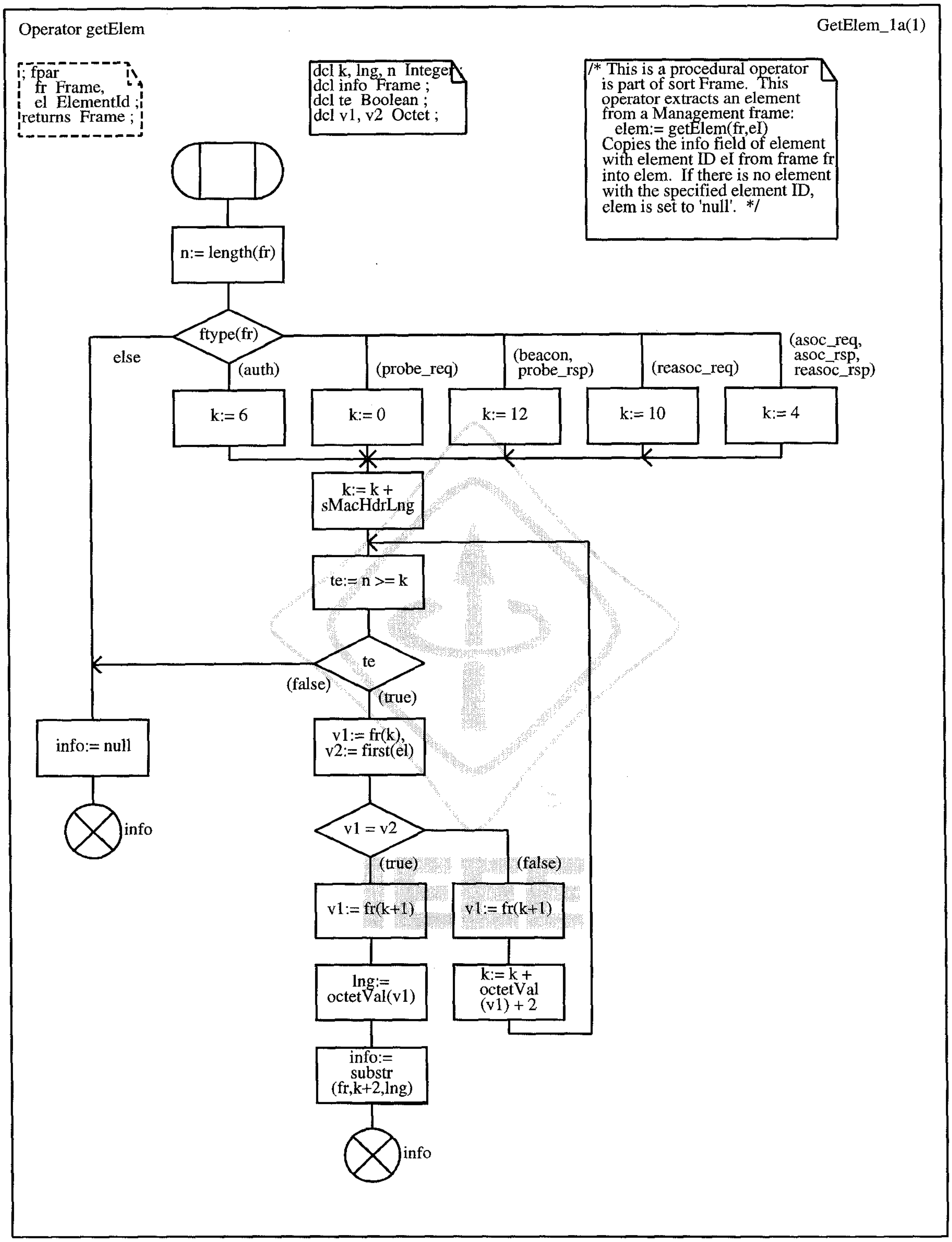
Operator keyLookup
KeyLookup_1a(1)
; fpar luadr MacAddr,
kma KeyMapArray,
kml Integer ;
returns KeyMap ;
/* This procedural operator is
part of sort KeyMapArray.
keyMap:= keyLookup
(addr, keyMapArray, keyMapArrayLength)
If an entry is found with mappedAddr=addr,
keyMap is set to the value of this entry.
If no entry is found with mappedAddr=addr,
keyMap is set to (. nullAddr, false, nullKey .) */
dcl lk Integer := 1;
dcl result KeyMap ;
luadr = kma(lk)! mappedAddr
(true)
(false)
lk:= lk + 1
result:= kma(lk)
Return first KeyMap element with correct mappedAddr value.
lk = (kml+1)
(false)
(true)
result! mappedAddr:= nullAddr
result! wepOn:= false
result! wepKey:= nullKey
If the end of the key map array is reached without finding addr, indicate the lack of a mapping by returning nullAddr. This avoids ambiguity between an entry which maps to nullKey and nullKey being returned due to lack of a mapping.
result


```
/******************************************************************
 *     FRAME sort (the basic definition of fields in MAC frames)
 ******************************************************************/

      /* Frame is a subtype of Octetstring with operators for creating
      /* MAC headers, extracting each of the header fields and some
      /* management frame fields, and modifying most of these fields.
      /* There are operators to create and extract management frame
      /* elements, but no operators for the frame body, IV, ICV, and FCS
      /* fields, which are handled directly as Octetstrings. */
newtype Frame   inherits Octetstring   operators all;
  adding  operators
    mkFrame         : TypeSubtype, MacAddr, MacAddr, Octetstring -> Frame;
    mkCtl           : TypeSubtype, Octetstring, MacAddr -> Frame;
    protocolVer     : Frame -> Integer;   /* Protocol version (2 bits) */
    basetype        : Frame -> BasicType; /* Type field (2 bits) */
    ftype           : Frame -> TypeSubtype; /* Type & Subtype (6 bits) */
    setFtype        : Frame, TypeSubtype -> Frame;
    toDs            : Frame -> Bit;   /* To DS bit (1 bit) */
    setToDs         : Frame, Bit -> Frame;
    frDs            : Frame -> Bit;   /* From DS bit (1 bit) */
    setFrDs         : Frame, Bit -> Frame;
    moreFrag        : Frame -> Bit;   /* More Fragments bit (1 bit) */
    setMoreFrag     : Frame, Bit -> Frame;
    retryBit        : Frame -> Bit;   /* Retry bit (1 bit) */
    setRetryBit     : Frame, Bit -> Frame;
    pwrMgt          : Frame -> Bit;   /* Power Management bit (1 bit) */
    setPwrMgt       : Frame, Bit -> Frame;
    moreData        : Frame -> Bit;   /* More Data bit (1 bit) */
    setMoreData     : Frame, Bit -> Frame;
    wepBit          : Frame -> Bit;   /* WEP bit (1 bit) */
    setWepBit       : Frame, Bit -> Frame;
    orderBit        : Frame -> Bit;   /* {strictly}Order{ed} (1 bit) */
    setOrderBit     : Frame, Bit -> Frame;
    durId           : Frame -> Integer; /* Duration/ID field (2) */
    setDurId        : Frame, Integer -> Frame;
    addr1           : Frame -> MacAddr; /* Address 1 [DA/RA] field (6) */
    setAddr1        : Frame, MacAddr -> Frame;
    addr2           : Frame -> MacAddr; /* Address 2 [SA/TA] field (6) */
    setAddr2        : Frame, MacAddr -> Frame;
    addr3           : Frame -> MacAddr; /* Address 3 [Bss/DA/SA] field */
    setAddr3        : Frame, MacAddr -> Frame;
    addr4           : Frame -> MacAddr; /* Address 4 [WDS-SA] field (6) */
    insAddr4        : Frame, MacAddr -> Frame;
    seq             : Frame -> SeqNum; /* Sequence Number (12 bits) */
    setSeq          : Frame, SeqNum -> Frame;
    frag            : Frame -> FragNum; /* Fragment Number (4 bits) */
    setFrag         : Frame, FragNum -> Frame;
    ts              : Frame -> Time; /* Timestamp field (8) */
    setTs           : Frame, Time -> Frame;
    mkElem          : ElementID, Octetstring -> Frame; /* make element */
    GetElem         : Frame, ElementID -> Frame; /* get element if aval */
    status          : Frame -> StatusCode; /* Status Code field (2) */
    setStatus       : Frame, StatusCode -> Frame;
    authStat        : Frame -> StatusCode; /* Status Code in Auth frame */
    reason          : Frame -> ReasonCode; /* Reason Code field (2) */
    authSeqNum      : Frame -> Integer; /* Auth Sequence Number (2) */
    authAlg         : Frame -> AuthType; /* Auth Algorithm field (2) */
    beaconInt       : Frame -> TU;     /* Beacon Interval field (2) */
    listenInt       : Frame -> TU;     /* Listen Interval field (2) */
    AId             : Frame -> AsocId; /* Association ID field (2) */
    setAId          : Frame, AsocId -> Frame;
    curApAddr       : Frame -> MacAddr; /* Current AP Addr field (6) */
    capA            : Frame, Capability -> Bit; /* Capability (Re)Asoc */
    setCapA         : Frame, Capability, Bit -> Frame;
```

```
    capB            : Frame, Capability -> Bit; /* Capability Bcn/Probe */
    setCapB         : Frame, Capability, Bit -> Frame;
    keyId           : Frame -> KeyIndex; /* Key ID subfield (2 bits) */
    setKeyId: Frame, KeyIndex -> Frame;
  operator GetElem;
    fpar   fr Frame,   el ElementID;   returns Frame;   referenced;
  axioms
    for all f in Frame(    for all a, sa, da, ra, ta, bssa in MacAddr(
     for all body, dur, sid, info in Octetstring(
      addr1(f) == SubStr(f,4,6);
      setAddr1(f,a) == SubStr(f,0,4) // a // SubStr(f,10,Length(f)-10);
      addr2(f) == SubStr(f,10,6);
      setAddr2(f,a) == SubStr(f,0,10) // a // SubStr(f,16,Length(f)-16);
      addr3(f) == SubStr(f,16,6);
      setAddr3(f,a) == SubStr(f,0,16) // a // SubStr(f,22,Length(f)-22);
      addr4(f) == SubStr(f,24,6);
      insAddr4(f,a) == SubStr(f,0,24) // a // SubStr(f,24,Length(f)-24);
      curApAddr(f) == SubStr(f,28,6);
        for all ft in TypeSubtype(
          mkFrame(ft, da, bssa, body) ==
              ft // O3 // da // aMacAddress // bssa // O2 // body;
          (ft = rts) ==> mkCtl(ft, dur, ra) ==
              ft // O1 // dur // ra // aStationID;
          (ft = ps_poll) ==> mkCtl(ft, sid, bssa) ==
              ft // O1 // sid // bssa // aStationID;
          (ft = cts) or (ft = ack) ==> mkCtl(ft, dur, ra) ==
              ft // O1 // dur // ra;
          (ft = cfend) or (ft = cfend_ack) ==> mkCtl(ft, bssa, ra) ==
              ft // O3 // ra // bssa;
          ftype(f) == MkString(f(0) and 0xFC);
          setFtype(f, ft) == Modify!(f, 0, MkString((f(0) and 0x03) or
              ft)); );
        for all bt in BasicType(   basetype(f) == f(0) and 0x0C;   );
        for all i in Integer(
          protocolVer(f) == octetVal(f(0) and 0x03);
          authSeqNum(f) == octetVal(f(26)) + (octetVal(f(27)) * 256);
          durId(f) == octetVal(f(2)) + (octetVal(f(3)) * 256);
          setDurId(f, i) == SubStr(f, 0, 2) // mkOS(i mod 256, 1) //
              mkOS(i / 256, 1) // SubStr(f, 4, Length(f) - 4); );
        for all e in ElementID(
          mkElem(e, info) == e // mkOS(Length(info) + 2, 1) // info; );
        for all b in Bit(
          toDs(f) == if (f(1) and 0x01) then 1 else 0 fi;
          setToDs(f, b) ==
              Modify!(f, 1, (f(1) and 0xFE) or S8(0,0,0,0,0,0,0,b));
          frDs(f) == if (f(1) and 0x02) then 1 else 0 fi;
          setFrDs(f, b) ==
              Modify!(f, 1, (f(1) and 0xFD) or S8(0,0,0,0,0,0,b,0));
          moreFrag(f) == if (f(1) and 0x04) then 1 else 0 fi;
          setMoreFrag(f, b) ==
              Modify!(f, 1, (f(1) and 0xFB) or S8(0,0,0,0,0,b,0,0));
          retryBit(f) == if (f(1) and 0x08) then 1 else 0 fi;
          setRetryBit(f, b) ==
              Modify!(f, 1, (f(1) and 0xF7) or S8(0,0,0,0,b,0,0,0));
          pwrMgt(f) == if (f(1) and 0x10) then 1 else 0 fi;
          setPwrMgt(f, b) ==
              Modify!(f, 1, (f(1) and 0xFB) or S8(0,0,0,b,0,0,0,0));
          moreData(f) == if (f(1) and 0x20) then 1 else 0 fi;
          setMoreData(f, b) ==
              Modify!(f, 1, (f(1) and 0xFB) or S8(0,0,b,0,0,0,0,0));
          wepBit(f) == if (f(1) and 0x40) then 1 else 0 fi;
          setWepBit(f, b) ==
              Modify!(f, 1, (f(1) and 0xFB) or S8(0,b,0,0,0,0,0,0));
          orderBit(f) == if (f(1) and 0x80) then 1 else 0 fi;
```

```
        setOrderBit(f, b) ==
            Modify!(f, 1, (f(1) and 0xFB) or S8(b,0,0,0,0,0,0,0));
        for all c in Capability(
         capA(f,c) == if (B_S(SubStr(f,24,2)) and c) then 1 else 0 fi;
         setCapA(f,c,b) == SubStr(f,0,24) // (B_S(SubStr(f,24,2) and
              (not c)) or (if b then c else O2 fi)) //
              SubStr(f,26,Length(f) - 26);
         capB(f,c) == if (B_S(SubStr(f,34,2)) and c) then 1 else 0 fi;
         setCapB(f,c,b) == SubStr(f,0,34) // (B_S(SubStr(f,34,2) and
              (not c)) or (if b then c else O2 fi)) //
              SubStr(f,36,Length(f) - 36); ));
      for all sq in SeqNum(
        seq(f) == (octetVal(f(22) and 0xF0)/16)+(octetVal(f(23)*16));
        setSeq(f, sq) == SubStr(f, 0, 22) // MkString((f(22) and 0x0F)
            or mkOctet((sq mod 16) * 16)) // mkOS(sq / 16, 1) //
            SubStr(f, 24, Length(f) - 24); );
      for all fr in FragNum(
        frag(f) == octetVal(f(22) and 0x0F);
        setFrag(f, fr) ==
            SubStr(f, 0, 22) // MkString((f(22) and 0xF0) or
            mkOctet(fr)) // SubStr(f, 23, Length(f) - 23); );
      for all tm in Time(
        ts(f) == tUsec( Usec!(octetVal(f(24)) +
            (256 * (octetVal(f(25)) +
              (256 * (octetVal(f(26)) +
                (256 * (octetVal(f(27)) +
                  (256 * (octetVal(f(28)) +
                    (256 * (octetVal(f(29)) +
                      (256 * (octetVal(f(30)) +
                        (256 * octetVal(f(31)))))))))))))) ) );
        setTs(f, tm) == SubStr(f, 0, 24) // mkOS(fix(tm), 1) //
            mkOS((fix(tm) / 256), 1) // mkOS((fix(tm) / 65536), 1) //
            mkOS((fix(tm) / 16777216), 1) //
            mkOS((fix(tm) / 4294967296), 1) //
            mkOS(((fix(tm) / 4294967296) / 256), 1) //
            mkOS(((fix(tm) / 4294967296) / 65536), 1) //
            mkOS(((fix(tm) / 4294967296) / 16777216), 1) //
            SubStr(f, 32, Length(f) - 32); );
      for all stat in StatusCode(
        status(f) == SubStr(f, 26, 2);
        setStatus(f, stat) ==
            SubStr(f, 0, 26) // stat // SubStr(f, 28, Length(f) - 28);
        authStat(f) == SubStr(f, 28, 2); );
      for all rea in ReasonCode(   reason(f) == SubStr(f, 24, 2);   );
      for all alg in AuthType(   AuthType(f) == SubStr(f, 24, 2);   );
      for all u in TU(
        beaconInt(f) == octetVal(f(32)) + (octetVal(f(33)) * 256);
        listenInt(f) == octetVal(f(26)) + (octetVal(f(27)) * 256); );
      for all sta in AssocId(
        AId(f) == octetVal(f(28)) + (octetVal(f(29)) * 256);
        setAId(f, sta) == SubStr(f, 0, 28) // mkOS(sta mod 256, 1) //
            mkOS(sta / 256, 1) // SubStr(f, 30, Length(f) - 30); );
      for all kid in KeyIndexRange(
        keyId(f) == octetVal(f(27)) / 64;
        setKeyId(f, kid) == Modify!(f, 27, mkOS(kid * 64)); );   )));
endnewtype Frame;
```

```
/*******************************************************************
 *     ReasonCode sort
 *******************************************************************/
newtype ReasonCode   inherits Octetstring   operators all;
  adding  literals unspec_reason,   auth_not_valid,   deauth_lv_ss,
    inactivity,   ap_overload,   class2_err,   class3_err,
    disas_lv_ss,   asoc_not_auth;
  axioms
    unspec_reason    == mkOS(1, 2);      auth_not_valid      == mkOS(2, 2);
    deauth_lv_ss     == mkOS(3, 2);      inactivity          == mkOS(4, 2);
    ap_overload      == mkOS(5, 2);      class2_err          == mkOS(6, 2);
    class3_err       == mkOS(7, 2);      disas_lv_ss         == mkOS(8, 2);
    asoc_not_auth    == mkOS(9, 2);
endnewtype ReasonCode;

/*******************************************************************
 *     StatusCode sort
 *******************************************************************/
newtype StatusCode   inherits Octetstring   operators all;
  adding  literals successful,   unspec_fail,   unsup_cap,
    reasoc_no_asoc,   fail_other, unsupt_alg, auth_seq_fail,
    chlng_fail,   auth_timeout,   ap_full,   unsup_rate;
  axioms
    successful       == mkOS(0, 2);      unspec_failure      == mkOS(1, 2);
    unsup_cap        == mkOS(10, 2);     reasoc_no_asoc      == mkOS(11, 2);
    fail_other       == mkOS(12, 2);     unsupt_alg          == mkOS(13, 2);
    auth_seq_fail    == mkOS(14, 2);     chlng_fail          == mkOS(15, 2);
    auth_timeout     == mkOS(16, 2);     ap_full             == mkOS(17, 2);
    unsup_rate       == mkOS(18, 2);
endnewtype StatusCode;
```


Operator getElem
GetElem_1a(1)
; fpar
fr Frame,
el ElementId ;
returns Frame ;
dcl k, lng, n Integer;
dcl info Frame ;
dcl te Boolean ;
dcl v1, v2 Octet ;
/* This is a procedural operator
is part of sort Frame. This
operator extracts an element
from a Management frame:
elem:= getElem(fr,eI)
Copies the info field of element
with element ID eI from frame fr
into elem. If there is no element
with the specified element ID,
elem is set to 'null'. */
n:= length(fr)
ftype(fr)
else
(auth)
(probe_req)
(beacon,
probe_rsp)
(reasoc_req)
(asoc_req,
asoc_rsp,
reasoc_rsp)
k:= 6
k:= 0
k:= 12
k:= 10
k:= 4
k:= k +
sMacHdrLng
te:= n >= k
te
(false)
(true)
info:= null
info
v1:= fr(k),
v2:= first(el)
v1 = v2
(true)
(false)
v1:= fr(k+1)
v1:= fr(k+1)
lng:=
octetVal(v1)
k:= k +
octetVal
(v1) + 2
info:=
substr
(fr,k+2,lng)
info

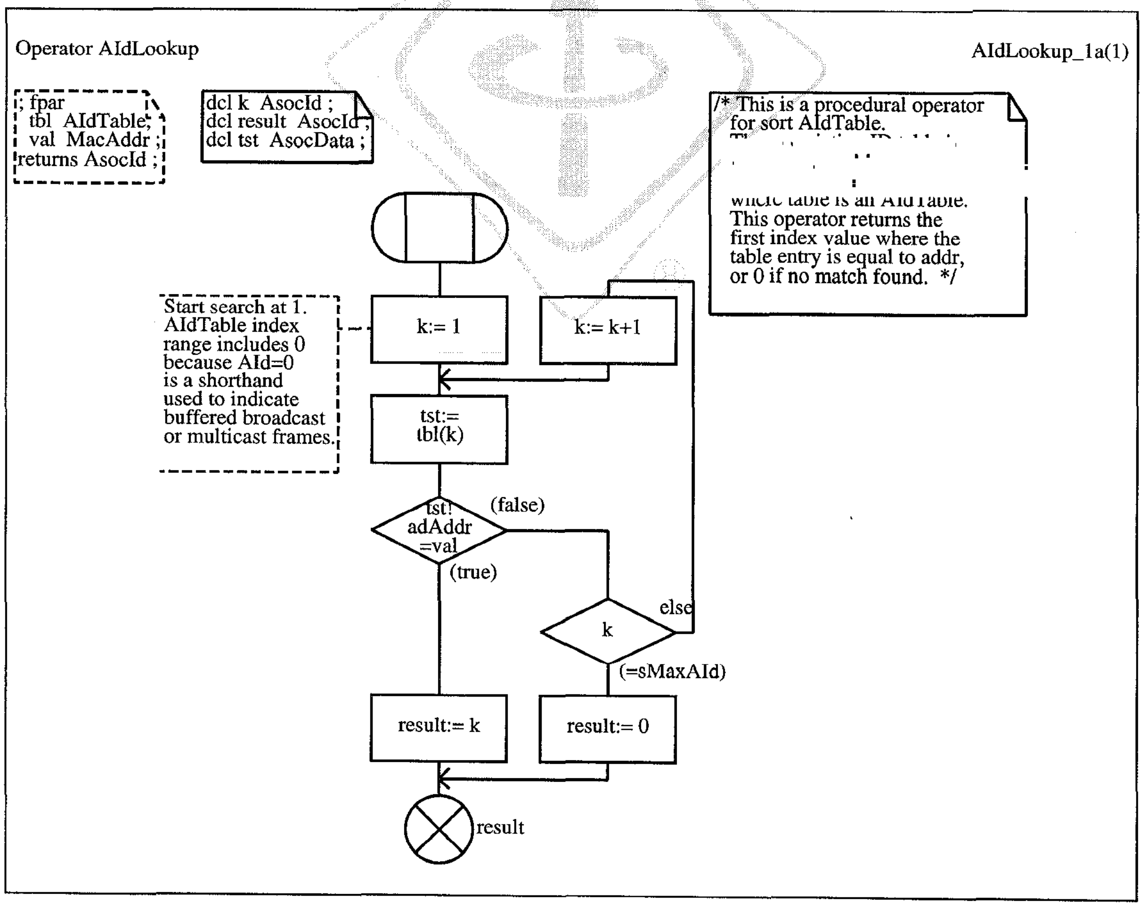
```
/*********************************************************************
 *     Frame Type sorts
 *********************************************************************/
      /* TypeSubtype defines the full, 6-bit frame type identifiers.
      /* These values are useful with ftype operator of Frame sort. */
newtype TypeSubtype   inherits Octetstring   operators all;
  adding  literals         asoc_req,     asoc_rsp,     reasoc_req,   reasoc_rsp,
             probe_req,    probe_rsp,    beacon,       atim,         disasoc,
             auth,         deauth,       ps_poll,      rts,          cts,
             ack,          cfend,        cfend_ack,    data,         data_ack,
             data_poll,    data_poll_ack,              null_frame,   cfack,
             cfpoll,       cfpoll_ack;
  axioms
    asoc_req         == MkString(S8(0,0,0,0,0,0,0,0));
    asoc_rsp         == MkString(S8(0,0,0,0,1,0,0,0));
    reasoc_req       == MkString(S8(0,0,0,0,0,1,0,0));
    reasoc_rsp       == MkString(S8(0,0,0,0,1,1,0,0));
    probe_req        == MkString(S8(0,0,0,0,0,0,1,0));
    probe_rsp        == MkString(S8(0,0,0,0,1,0,1,0));
    beacon           == MkString(S8(0,0,0,0,0,0,0,1));
    atim             == MkString(S8(0,0,0,0,1,0,0,1));
    disasoc          == MkString(S8(0,0,0,0,0,1,0,1));
    auth             == MkString(S8(0,0,0,0,1,1,0,1));
    deauth           == MkString(S8(0,0,0,0,0,0,1,1));
    ps_poll          == MkString(S8(0,0,1,0,0,1,0,1));
    rts              == MkString(S8(0,0,1,0,1,1,0,1));
    cts              == MkString(S8(0,0,1,0,0,0,1,1));
    ack              == MkString(S8(0,0,1,0,1,0,1,1));
    cfend            == MkString(S8(0,0,1,0,0,1,1,1));
    cfend_ack        == MkString(S8(0,0,1,0,1,1,1,1));
    data             == MkString(S8(0,0,0,1,0,0,0,0));
    data_ack         == MkString(S8(0,0,0,1,1,0,0,0));
    data_poll        == MkString(S8(0,0,0,1,0,1,0,0));
    data_poll_ack    == MkString(S8(0,0,0,1,1,1,0,0));
    null_frame       == MkString(S8(0,0,0,1,0,0,1,0));
    cfack            == MkString(S8(0,0,0,1,1,0,1,0));
    cfpoll           == MkString(S8(0,0,0,1,0,1,1,0));
    cfpoll_ack       == MkString(S8(0,0,0,1,1,1,1,0));
endnewtype TypeSubtype;

      /* BasicTypes defines the 2-bit frame type groups */
newtype BasicType   inherits Bitstring   operators all;
  adding  literals   control,   data,   management,   reserved;
  axioms
    control          == S8(0,0,1,0,0,0,0,0);
    data             == S8(0,0,0,1,0,0,0,0);
    management       == S8(0,0,0,0,0,0,0,0);
    reserved         == S8(0,0,1,1,0,0,0,0);
endnewtype BasicType;
```

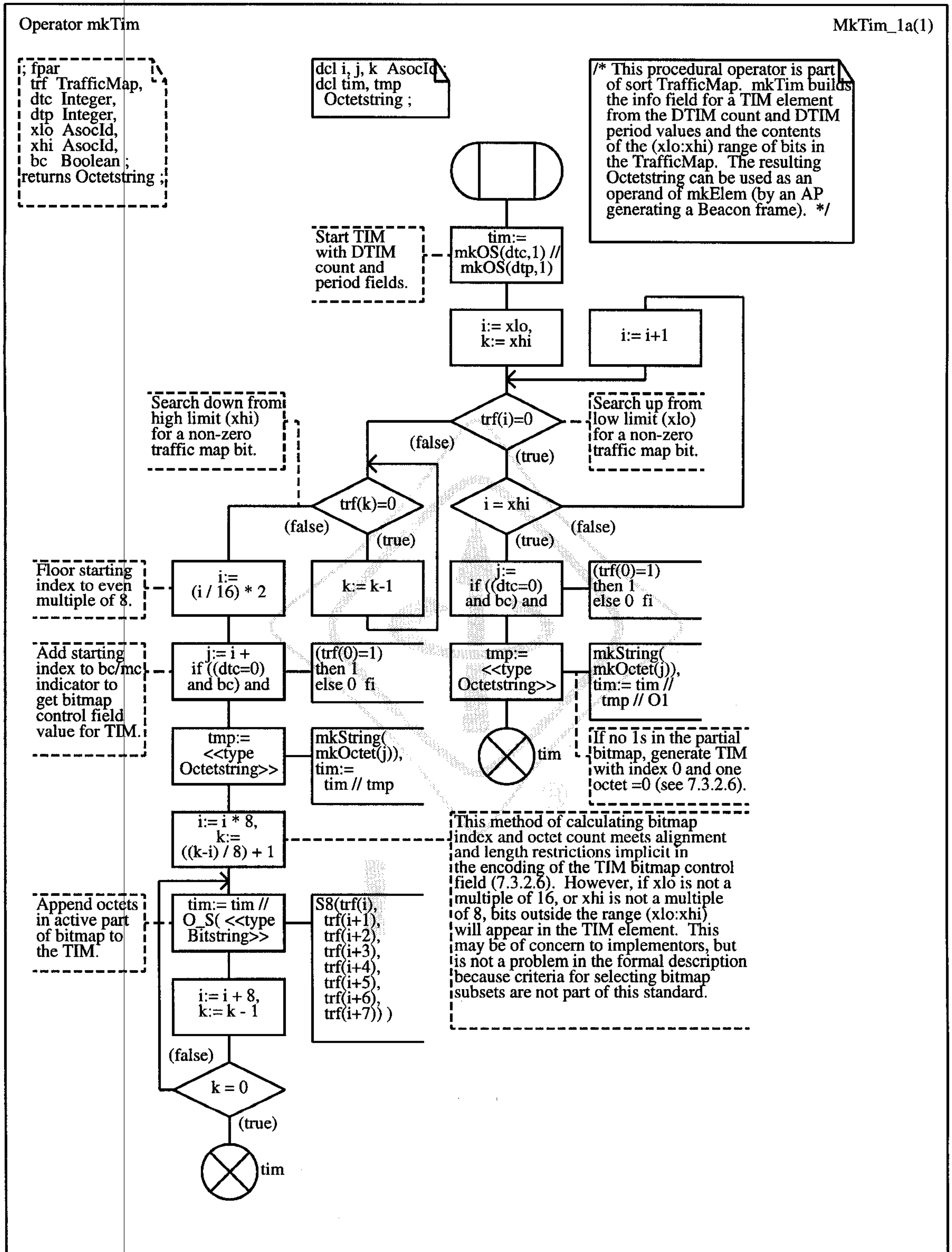
```
/******************************************************************
 *     ElementID sort
 ******************************************************************/
newtype ElementID   inherits Octetstring   operators all;
  adding  literals eSsId,   eSupRates,   eFhParms,   eDsParms,
    eCfParms,   eTim,   eIbParms,   eCtext;
  axioms
    eSsId      == mkOS(0, 1);  /* service set identifier (0:32) */
    eSupRates == mkOS(1, 1);  /* supported rates (1:8) */
    eFhParms  == mkOS(2, 1);  /* FH parameter set (5) */
    eDsParms  == mkOS(3, 1);  /* DS parameter set (1) */
    eCfParms  == mkOS(4, 1);  /* CF parameter set (6) */
    eTim       == mkOS(5, 1);  /* Traffic Information Map (4:254) */
    eIbParms  == mkOS(6, 1);  /* IBSS parameter set (2) */
    eCtext     == mkOS(16, 1); /* challenge text (128, see 8.1.2.2) */
endnewtype ElementID;

/******************************************************************
 *     Capability field bit assignments sort
 ******************************************************************/
newtype Capability   inherits Bitstring   operators all;
  adding  literals cEss, cIbss, cPollable, cPollReq, cPrivacy;
  axioms
    cEss ==          S8(1,0,0,0,0,0,0,0) // 0x00; /* ESS capability */
    cIbss ==         S8(0,1,0,0,0,0,0,0) // 0x00; /* IBSS capability */
    cPollable ==     S8(0,0,1,0,0,0,0,0) // 0x00; /* CF-pollable (sta),
                                                    PC present (ap) */
    cPollReq ==      S8(0,0,0,1,0,0,0,0) // 0x00; /* not CF poll req (sta),
                                                    PC polls (ap) */
    cPrivacy ==      S8(0,0,0,0,1,0,0,0) // 0x00; /* WEP required */
endnewtype Capability;

/******************************************************************
 *     IBSS parameter set sort
 ******************************************************************/
newtype IbssParms   inherits Octetstring   operators all;
  adding  operators
    atimWin : IbssParms -> TU;
    setAtimWin : IbssParms, TU -> IbssParms;
  axioms
    for all ib in IbssParms(     for all u in TU(
        atimWin(ib) == octetVal(ib(0)) + (octetVal(ib(1)) * 256);
        setAtimWin(ib, u) == mkOS(u mod 256, 1) // mkOS(u / 256, 1); ));
endnewtype IbssParms;

/******************************************************************
 *     CF parameter set sort
 ******************************************************************/
newtype CfParms   inherits Octetstring   operators all;
  adding  operators
    cfpCount         : CfParms -> Integer; /* CfpCount field (1) */
    setCfpCount      : CfParms, Integer -> CfParms;
    cfpPeriod        : CfParms -> Integer; /* CfpPeriod field (1) */
    setCfpPeriod     : CfParms, Integer -> CfParms;
    cfpMaxDur        : CfParms -> TU;      /* CfpMaxDuration field (2) */
    setCfpMaxDur     : CfParms, TU -> CfParms;
    cfpDurRem        : CfParms -> TU;      /* CfpDurRemaining field (2) */
    setCfpDurRem     : CfParms, TU -> CfParms;
  axioms
    for all cf in CfParms(   for all i in Integer(   for all u in TU(
          cfpCount(cf) == octetVal(cf(0));
          setCfpCount(cf, i) == mkOS(i, 1) // Tail(cf);
          cfpPeriod(cf) == octetVal(cf(1));
          setCfpPeriod(cf, i) == cf(0) // mkOS(i, 1) // SubStr(cf,2,4);
```

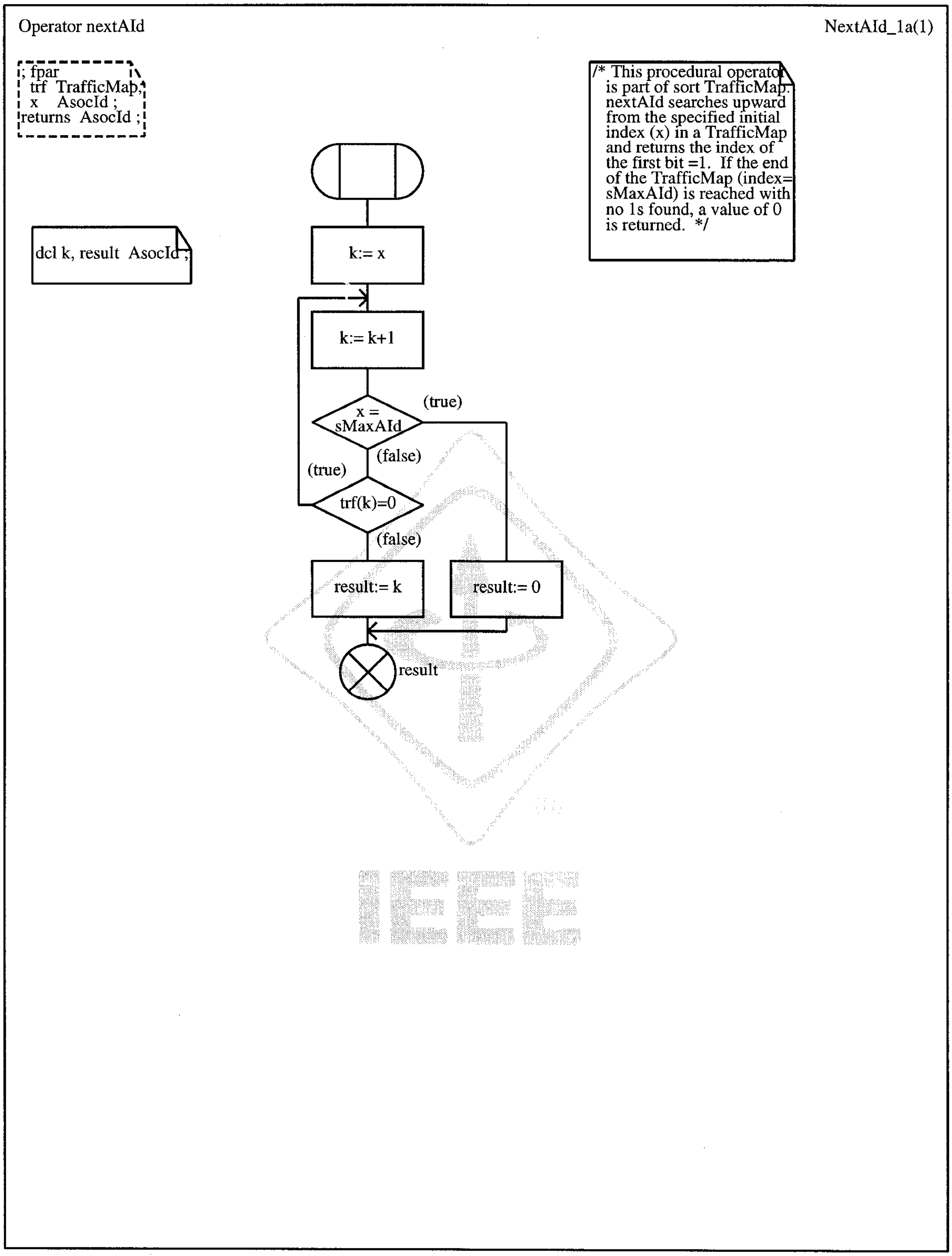


```
        cfpMaxDur(cf) == octetVal(cf(2)) + (octetVal(cf(3)) * 256);
        setCfpMaxDur(cf, u) == SubStr(cf, 0, 2) // mkOS(u mod 256, 1)
            // mkOS(u / 256, 1) // SubStr(cf, 4, 2);
        cfpDurRem(cf) == octetVal(cf(4)) + (octetVal(cf(5)) * 256);
        setCfpDurRem(cf, u) == SubStr(cf, 0, 4) // mkOS(u mod 256, 1)
            // mkOS(u / 256, 1); )));
endnewtype CfParms;

/*********************************************************************
 *     Sorts for association management at AP
 *********************************************************************/
synonym sMaxAId Integer = 2007;  /* 2007 is largest allowable value */
                                 /* implementation limit may be lower */
syntype AsocId = Integer   constants 0:sMaxAId   endsyntype AsocId;
      /* Station Association Record -- only used at APs */
newtype AsocData struct
    adAddr     MacAddr;     /* address of associated station */
    adPsm      PwrSave;     /* power save mode of the station */
    adCfPoll   Boolean;     /* true if station is CfPollable */
    adPollRq   Boolean;     /* true if station requested polling */
    adNoPoll   Boolean;     /* true if station requested no polling */
    adMsduIP   Boolean;     /* true if partial Msdu outstanding to sta */
    adAuth     AuthType;    /* authentication type used by station */
    adRates    RateSet;     /* supported rates from association request */
    adAge      Time;        /* time of association */
endnewtype AsocData;
       /* Association table -- array of AsocData, only used at APs
       /*     index:= AIdLookup(table, addr)
       /* returns the index of location where table(x)!adAddr=addr
       /* or 0 if no such location found. */
newtype AIdTable   Array(AsocId, AsocData);
  adding operators
    AIdLookup : AIdTable, MacAddr -> AsocId;
  operator AIdLookup;
    fpar tbl AIdTable,   val MacAddr;   returns AsocId;   referenced;
endnewtype AIdTable;

/*********************************************************************
 *     Traffic Information Map (TIM) support sorts
 *********************************************************************/
       /* TrafficMap is an Array of Bit indexed by AId.
       /* Bits =1 in TrafficMap denote the presence of buffered frame(s)
       /* for the station assigned that AId.  TrafficMap operators are:
       /*     mkTim(trafficMap, dtimCnt, dtimPer, lowAId, highAId, bcst)
       /* returns Octetstring to use as the info field of a TIM element
       /* The TIM will contain bits =1 for TrafficMap locations in the
       /* range (lowAId):(highAId).  Buffered broadcasts and multicasts
       /* (AId 0) are indicated if dtimCnt=0 and if bcst=true.
       /*     nextAId(trafficMap, currentAId)
       /* returns index greater than currentAId at which TrafficMap=1.
       /* If no locations before sMaxAId are =1, returns 0. */
newtype TrafficMap  Array( AsocId, Bit);
  adding operators
    mkTim : TrafficMap, Integer, Integer, AsocId, AsocId, Boolean ->
        Octetstring;
    nextAId : TrafficMap, AsocId -> AsocId;
  operator mkTim;
    fpar trf TrafficMap,  dtc Integer,  dtp Integer,  xlo AsocId,
    xhi AsocId,  bc Boolean;  returns Octetstring;  referenced;
  operator nextAId;
    fpar trf TrafficMap,  x AsocId;  returns AsocId;  referenced;
endnewtype TrafficMap;
       /* TIM is a subtype of Octetstring with operators:
       /*     bufFrame(tim,AId)   returns true if the TIM info field
```

```
        /*            (obtained using getElem) is =1 at tim(AId).
        /*      bufBcst(tim)   returns true if the TIM info field
        /*             indicates buffered broadcast/multicast traffic
        /*      dtCount(tim)   returns DTIM count value from TIM
        /*      dtPeriod(tim)  returns DTIM period value from TIM */
newtype TIM   inherits Octetstring   operators all;
  adding  operators
    bufFrame : TIM, AsocId -> Boolean;
    bufBcst  : TIM -> Boolean;
    dtCount  : TIM -> Integer;
    dtPeriod : TIM -> Integer;
  axioms
    for all el in TIM(     for all a in AsocId(
      bufFrame(el, a) ==
        if a < (octetVal(el(2) and 0xFE) * 8) then false
          else
          if a >= ((octetVal(el(2) and 0xFE)*8) + ((Length(el)-3)*8))
            then false
            else
              Extract!(B_S(el), (a-(octetVal(el(2) and 0xFE)*8)+24)) = 1
          fi fi;
      bufBcst(el) == (el(2) and 0x01) = 0x01;
      dtCount(el) == octetVal(el(0));
      dtPeriod(el) == octetVal(el(1)); ));
endnewtype TIM;
```




Operator mkTim
MkTim_1a(1)
; fpar
trf TrafficMap,
dtc Integer,
dtp Integer,
xlo AsocId,
xhi AsocId,
bc Boolean ;
returns Octetstring ;
dcl i, j, k AsocId;
dcl tim, tmp
Octetstring ;
/* This procedural operator is part of sort TrafficMap. mkTim builds the info field for a TIM element from the DTIM count and DTIM period values and the contents of the (xlo:xhi) range of bits in the TrafficMap. The resulting Octetstring can be used as an operand of mkElem (by an AP generating a Beacon frame). */
Start TIM with DTIM count and period fields.
tim:= mkOS(dtc,1) // mkOS(dtp,1)
i:= xlo, k:= xhi
i:= i+1
Search down from high limit (xhi) for a non-zero traffic map bit.
trf(i)=0
(false)
(true)
Search up from low limit (xlo) for a non-zero traffic map bit.
trf(k)=0
(false)
(true)
i = xhi
(true)
(false)
Floor starting index to even multiple of 8.
i:= (i / 16) * 2
k:= k-1
j:= if ((dtc=0) and bc) and
(trf(0)=1) then 1 else 0 fi
Add starting index to bc/mc indicator to get bitmap control field value for TIM.
j:= i + if ((dtc=0) and bc) and
(trf(0)=1) then 1 else 0 fi
tmp:= <<type Octetstring>>
mkString(mkOctet(j)), tim:= tim // tmp // O1
tim
If no 1s in the partial bitmap, generate TIM with index 0 and one octet =0 (see 7.3.2.6).
tmp:= <<type Octetstring>>
mkString(mkOctet(j)), tim:= tim // tmp
i:= i * 8, k:= ((k-i) / 8) + 1
This method of calculating bitmap index and octet count meets alignment and length restrictions implicit in the encoding of the TIM bitmap control field (7.3.2.6). However, if xlo is not a multiple of 16, or xhi is not a multiple of 8, bits outside the range (xlo:xhi) will appear in the TIM element. This may be of concern to implementors, but is not a problem in the formal description because criteria for selecting bitmap subsets are not part of this standard.
Append octets in active part of bitmap to the TIM.
tim:= tim // O_S(<<type Bitstring>>
S8(trf(i), trf(i+1), trf(i+2), trf(i+3), trf(i+4), trf(i+5), trf(i+6), trf(i+7)))
i:= i + 8, k:= k - 1
(false)
k = 0
(true)
tim


Operator nextAId
NextAId_1a(1)
; fpar
trf TrafficMap;
x AsocId ;
returns AsocId ;
/* This procedural operator is part of sort TrafficMap. nextAId searches upward from the specified initial index (x) in a TrafficMap and returns the index of the first bit =1. If the end of the TrafficMap (index= sMaxAId) is reached with no 1s found, a value of 0 is returned. */
dcl k, result AsocId ;
k:= x
k:= k+1
x = sMaxAId
(true)
(false)
(true)
trf(k)=0
(false)
result:= k
result:= 0
result

```
/****************************************************************
 *     Multi-rate support sorts
 ****************************************************************/
newtype Rate   inherits Octet   operators all;
  adding  operators
    calcDur : Rate, Integer -> Integer;
                /* converts (rate,bitCount) to integer microseconds */
    rateVal : Rate -> Rate; /* clears high-order bit */
    basicRate : Rate -> Rate; /* sets high-order bit */
    isBasic : Rate -> Boolean; /* true if high-order bit set */
  axioms
    for all r in Rate(    for all i in Integer(    for all b in Boolean(
        calcDur(r, i) == ((((10000000 + (octetVal(r and 0x7F) - 1)) /
            (500 * octetVal(r and 0x7F))) * i) + 9999) / 10000;
        rateVal(r) == r and 0x7F;
        basicRate(r) == r or 0x80;
        isBasic(r) == (r and 0x80) = 0x80; )));
endnewtype Rate;

syntype RateString = Octetstring   endsyntype RateString;

/****************************************************************
 *     MPDU duration factor support sort
 ****************************************************************/
      /* These operators support the encoding used to allow
      /* an Integer to represent the value of aMpduDurationFactor.
      /*     calcDF(PlcpBits, MpduBits)  returns an Integer which is
      /* the fractional part of ((PlcpBits/MpduBits)-1)*(1e9).
      /*     stuff(durFactor, MpduBits)  returns the number of PlcpBits
      /* which result from MpduBits at the specified durFactor. */
newtype DurFactor   inherits Integer   operators all;
  adding  operators
    calcDF : Integer, Integer -> DurFactor;
    stuff  : DurFactor, Integer -> Integer;
  axioms
    for all df in DurFactor(     for all mb, pb in Integer(
        calcDF(pb, mb) == ((pb * 1000000000) / mb) - 1000000000;
        stuff(df, mb) == ((mb * df) + (mb - 1)) / 1000000000; ));
endnewtype DurFactor;

/****************************************************************
 *     Generic PHY parameter set sort
 ****************************************************************/
      /* Generic PHY parameter element for signals related to Beacons
      /* and Probe Responses that are PHY-type independent. */
syntype PhyParms = Octetstring   endsyntype PhyParms;

/****************************************************************
 *     FH parameter set sort
 ****************************************************************/
newtype FhParms   inherits Octetstring   operators all;
  adding  operators
    dwellTime       : FhParms -> TU;      /* Dwell Time field (2) */
    setDwellTime    : FhParms, TU -> FhParms;
    hopSet          : FhParms -> Integer; /* Hop Set field (1) */
    setHopSet       : FhParms, Integer -> FhParms;
    hopPattern      : FhParms -> Integer; /* Hop Pattern field (1) */
    setHopPattern   : FhParms, Integer -> FhParms;
    hopIndex        : FhParms -> Integer; /* Hop Index field (1) */
    setHopIndex     : FhParms, Integer -> FhParms;
  axioms
    for all fh in FhParms(   for all i in Integer(   for all u in TU(
      dwellTime(fh) == octetVal(fh(0)) + (octetVal(fh(1)) * 256);
      setDwellTime(fh, u) ==
```

```
            mkOS(u mod 256, 1) // mkOS(u / 256, 1) // SubStr(fh, 2, 3);
        hopSet(fh) == octetVal(fh(2));
        setHopSet(fh,i) == SubStr(fh,0,2) // mkOS(i,1) // SubStr(fh,3,2);
        hopPattern(fh) == octetVal(fh(3));
        setHopPattern(fh, i) == SubStr(fh,0,3) // mkOS(i,1) // Last(fh);
        hopIndex(fh) == octetVal(fh(4));
        setHopIndex(fh, i) == SubStr(fh, 0, 4) // mkOS(i, 1);)));
endnewtype FhParms;

/*******************************************************************
 *     DS parameter set sort
 *******************************************************************/
newtype DsParms   inherits Octetstring   operators all;
  adding  operators
    curChannel: DsParms -> Integer; /* Current Channel (1) */
    setCurChannel: DsParms, Integer -> DsParms;
  axioms
    for all ds in DsParms(     for all i in Integer(
        curChannel(ds) == octetVal(ds(0));
        setCurChannel(ds, i) == mkOS(i); ));
endnewtype DsParms;

endpackage;
/********************************************************************/


/********************************************************************/
use macsorts;
Package macmib;

       /* This Package contains definitions of the MAC MIB attributes
       /* and the subset of the PHY MIB attributes used by the MAC state
       /* machines.  These are needed under Z.100 to permit analysis of
       /* the state machine definitions.  In future revisions these may
       /* be replaced with the ASN.1 MIB definition, from Annex D, if
       /* a Z.105-compliant SDL tool is available. */

/*******************************************************************
 *     StationConfig Group
 *******************************************************************/
remote aMediumOccupancyLimit  TU nodelay;
remote aReceiveDTIMs  Boolean nodelay;
synonym aCfPollable  Boolean = <<package macsorts>> sCfPollable;
remote aCfpPeriod  Integer nodelay;
remote aCfpMaxDuration  TU nodelay;
remote aAuthenticationResponseTimeout  TU nodelay;
remote aAuthenticationType  AuthTypeSet nodelay;
remote aWepUndecryptableCount  Counter32 nodelay;

/*******************************************************************
 *     AuthenticationAlgorithms Table
 *******************************************************************/
synonym aAuthenticationAlgorithms  AuthTypeSet =
    incl(open_system, incl(shared_key));
             /* NOTE:  Only  include shared_key in this set
                if aPrivacyOptionImplemented=true. */

/*******************************************************************
 *     WepDefaultKeys Table      (only if aPrivacyOptionImplemented=true)
 *******************************************************************/
remote aWepDefaultKeys  KeyVector nodelay;

/*******************************************************************
 *     WepKeyMappings Table      (only if aPrivacyOptionImplemented=true)
```



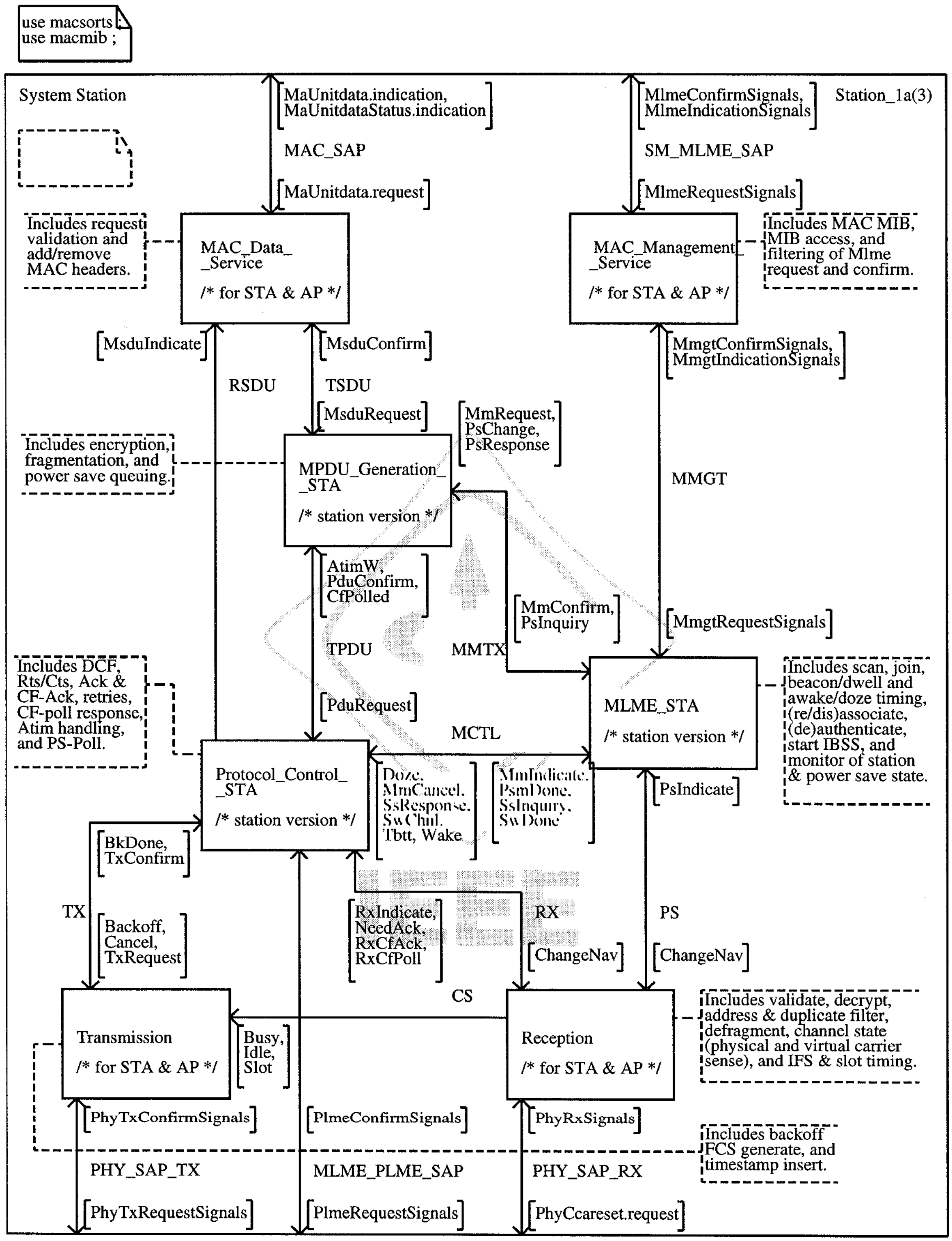
```
 ******************************************************************/
remote aWepKeyMappings  KeyMapArray nodelay;
synonym aWepKeyMappingLength  Integer =
    <<package macsorts>> sWepKeyMappingLength;

/******************************************************************
 *     Privacy Group     (only 1 item if aPrivacyOptionImplemented=false)
 ******************************************************************/
synonym aPrivacyOptionImplemented  Boolean = true;
remote aPrivacyInvoked  Boolean nodelay;
remote aWepDefaultKeyId  KeyIndex nodelay;
remote aExcludeUnencrypted  Boolean nodelay;
remote aWepIcvErrorCount  Counter32 nodelay;
remote aWepExcludedCount  Counter32 nodelay;

/******************************************************************
 *     Operation Group
 ******************************************************************/
synonym aMacAddress MacAddr =
  <<type MacAddr>> S6(0x00, 0x11, 0x22, 0x33, 0x44, 0x55);
    /* each station has a unique globally administered address */
    /* Value may be overwritten with locally administered address at */
    /* MlmeReset, but is always a static value during MAC operation */
remote aRtsThreshold  Integer nodelay;
remote aShortRetryLimit  Integer nodelay;
remote aLongRetryLimit  Integer nodelay;
remote aFragmentationThreshold  Integer nodelay;
remote aMaxTransmitMsduLifetime  TU nodelay;
remote aMaxReceiveLifetime  TU nodelay;
synonym aManufacturerId  Charstring = 'name of manufacturer';
synonym aProductId  Charstring = 'identifier unique to manufacturer';

/******************************************************************
 *     GroupAddresses Table
 ******************************************************************/
remote aGroupAddresses  MacAddrSet nodelay;

/******************************************************************
 *     Counters Group
 ******************************************************************/
remote aTransmittedFragmentCount  Counter32 nodelay;
remote aMulticastTransmittedFrameCount  Counter32 nodelay;
remote aFailedCount  Counter32 nodelay;
remote aRetryCountCounter32 nodelay;
remote aMultipleRetryCount  Counter32 nodelay;
remote aRtsSuccessCount  Counter32 nodelay;
remote aRtsFailureCount  Counter32 nodelay;
remote aAckFailureCount  Counter32 nodelay;
remote aReceivedFragmentCount  Counter32 nodelay;
remote aMulticastReceivedFrameCount  Counter32 nodelay;
remote aFcsErrorCount  Counter32 nodelay;
remote aFrameDuplicateCount  Counter32 nodelay;

/******************************************************************
 *     PhyOperation Group     (values shown are mostly for FH PHY)
 ******************************************************************/
synonym aPHYType  Integer = 01;
synonym RegDomainSupported Octetstring = S4(0x10, 0x20, 0x30, 0x00);
remote aCurrentRegDomain  Integer nodelay;
synonym aSlotTime  Usec = (aCcaTime + aRxTxTurnaroundTime +
    aAirPropagationTime + aMacProcessingTime);
synonym aCcaTime  Usec = 27;
synonym aRxTxTurnaroundTime Usec = (aTxPlcpDelay + aRxTxSwitchTime +
    aTxRampOnTime + aTxRfDelay);
```

```
synonym aTxPlcpDelay  Usec = 1;
synonym aRxTxSwitchTime  Usec = 10;
synonym aTxRampOnTime  Usec = 8;
synonym aTxRfDelay  Usec = 1;
synonym aSifsTime Usec = (aRxRfDelay + aRxPlcpDelay +
    aMacProcessingTime + aRxTxTurnaroundTime);
synonym aRxRfDelay  Usec = 4;
synonym aRxPlcpDelay  Usec = 2;
synonym aMacProcessingTime  Usec = 2;
synonym aTxRampOffTime  Usec = 8;
synonym aPreambleLength  Usec = 96;
synonym aPlcpHeaderLength  Usec = 32;
synonym aMpduDurationFactor  <<package macsorts>> DurFactor = 31250000;
synonym aAirPropagationTime  Usec = 1;
synonym aTempType  Integer = 01;
synonym aCWmax  Integer = 1023;
synonym aCWmin  Integer = 15;

/********************************************************************
 *     PhyRate Group     (values shown are mostly for FH PHY)
 ********************************************************************/
synonym aSupportedRatesTx  Octetstring = S3(0x82, 0x04, 0x00);
synonym aSupportedRatesRx  Octetstring = S3(0x82, 0x04, 0x00);
synonym aMpduMaxLength  Integer = 4095;

/********************************************************************
 *     PhyFHSS Group     (only used with FH PHY)
 ********************************************************************/
synonym aHopTime  Usec = 224;
remote aCurrentChannelNumber  Integer nodelay;
synonym aMaxDwellTime  TU = 390;
remote aCurrentSet  Integer nodelay;
remote aCurrentPattern  Integer nodelay;
remote aCurrentIndex  Integer nodelay;

/********************************************************************/
      /*     The MAC state machines do not reference any attributes in
      /*            PhyAntennaGroup,    PhyTxPowerGroup,    PhyDsssGroup,
      /*            PhyStatusGroup,     PhyPowerSavingGroup,
      /*            PhyIR Group,        AntennasList  */

endpackage;
/********************************************************************/
```


C.3 State machines for MAC stations

The following SDL-92 system specification defines operation of the MAC protocol at an IEEE 802.11 STA. Many aspects of STA operation also apply to AP operation. These are defined in blocks and processes referenced from both the STA and AP system specifications. Blocks and processes used in both STA and AP are identifiable by the SDL comment /* for STA & AP */ below the block or process name. Blocks and processes specific to STA operation are identifiable by the SDL comment /* station version */ below the block or process name. The definitions of all blocks and processes referenced in the station system specification appear in Clause C.3.

The remainder of Clause C.3 is the formal description, in SDL/GR, of an IEEE 802.11 STA.

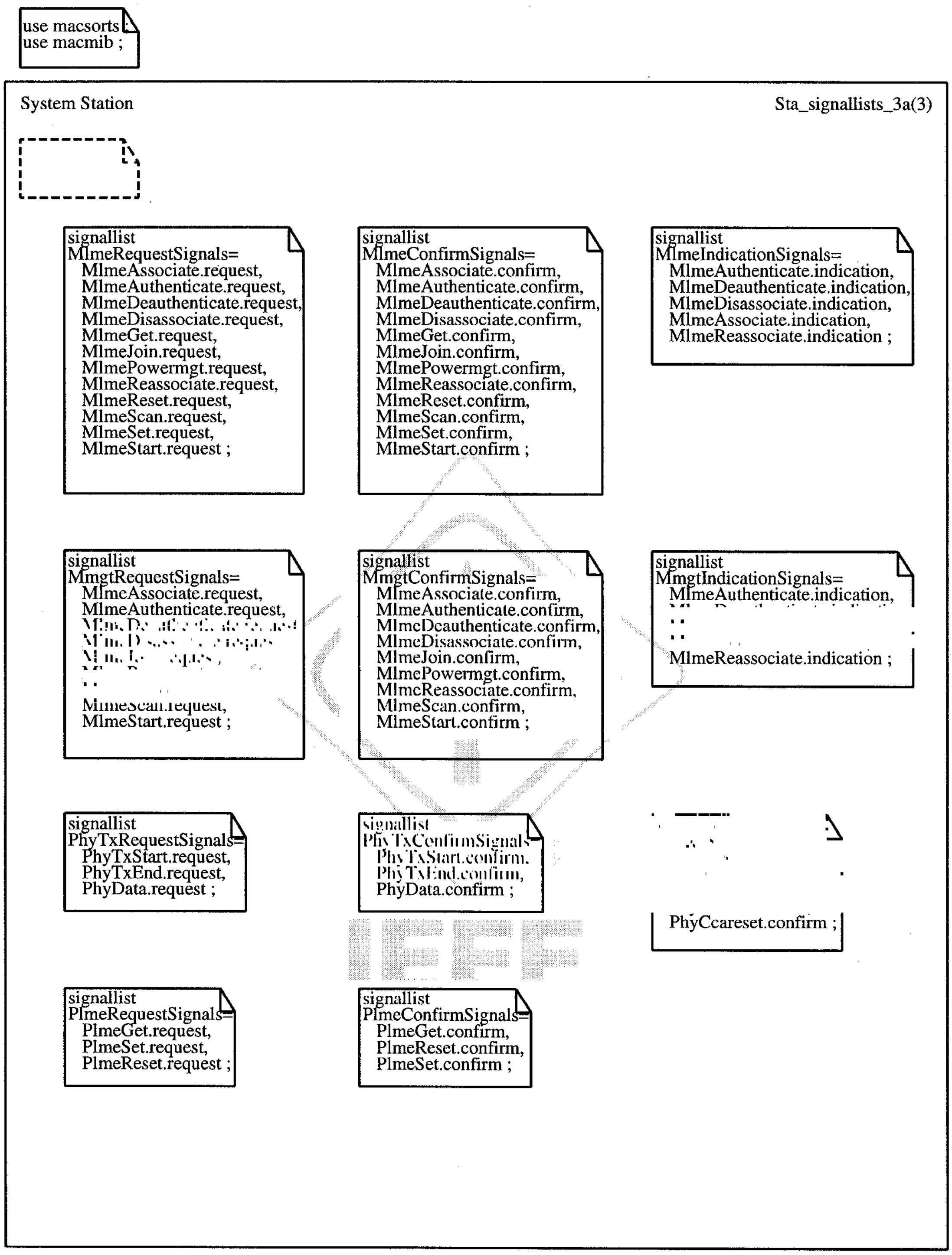

use macsorts ;
use macmib ;
System Station
Station_1a(3)
MaUnitdata.indication,
MaUnitdataStatus.indication
MAC_SAP
MaUnitdata.request
MlmeConfirmSignals,
MlmeIndicationSignals
SM_MLME_SAP
MlmeRequestSignals
Includes request validation and add/remove MAC headers.
MAC_Data_
_Service
/* for STA & AP */
MAC_Management_
_Service
/* for STA & AP */
Includes MAC MIB, MIB access, and filtering of Mlme request and confirm.
MsduIndicate
MsduConfirm
RSDU
TSDU
MsduRequest
MmRequest,
PsChange,
PsResponse
MmgtConfirmSignals,
MmgtIndicationSignals
MMGT
Includes encryption, fragmentation, and power save queuing.
MPDU_Generation_
_STA
/* station version */
AtimW,
PduConfirm,
CfPolled
TPDU
MMTX
MmConfirm,
PsInquiry
MmgtRequestSignals
Includes DCF, Rts/Cts, Ack & CF-Ack, retries, CF-poll response, Atim handling, and PS-Poll.
PduRequest
MCTL
MLME_STA
/* station version */
Includes scan, join, beacon/dwell and awake/doze timing, (re/dis)associate, (de)authenticate, start IBSS, and monitor of station & power save state.
Protocol_Control_
_STA
/* station version */
Doze,
MmCancel,
SsResponse,
SwChnl,
Tbtt, Wake
MmIndicate,
PsmDone,
SsInquiry,
SwDone
PsIndicate
BkDone,
TxConfirm
TX
Backoff,
Cancel,
TxRequest
RxIndicate,
NeedAck,
RxCfAck,
RxCfPoll
RX
ChangeNav
PS
ChangeNav
CS
Transmission
/* for STA & AP */
Busy,
Idle,
Slot
Reception
/* for STA & AP */
Includes validate, decrypt, address & duplicate filter, defragment, channel state (physical and virtual carrier sense), and IFS & slot timing.
PhyTxConfirmSignals
PlmeConfirmSignals
PhyRxSignals
Includes backoff FCS generate, and timestamp insert.
PHY_SAP_TX
MLME_PLME_SAP
PHY_SAP_RX
PhyTxRequestSignals
PlmeRequestSignals
PhyCcareset.request

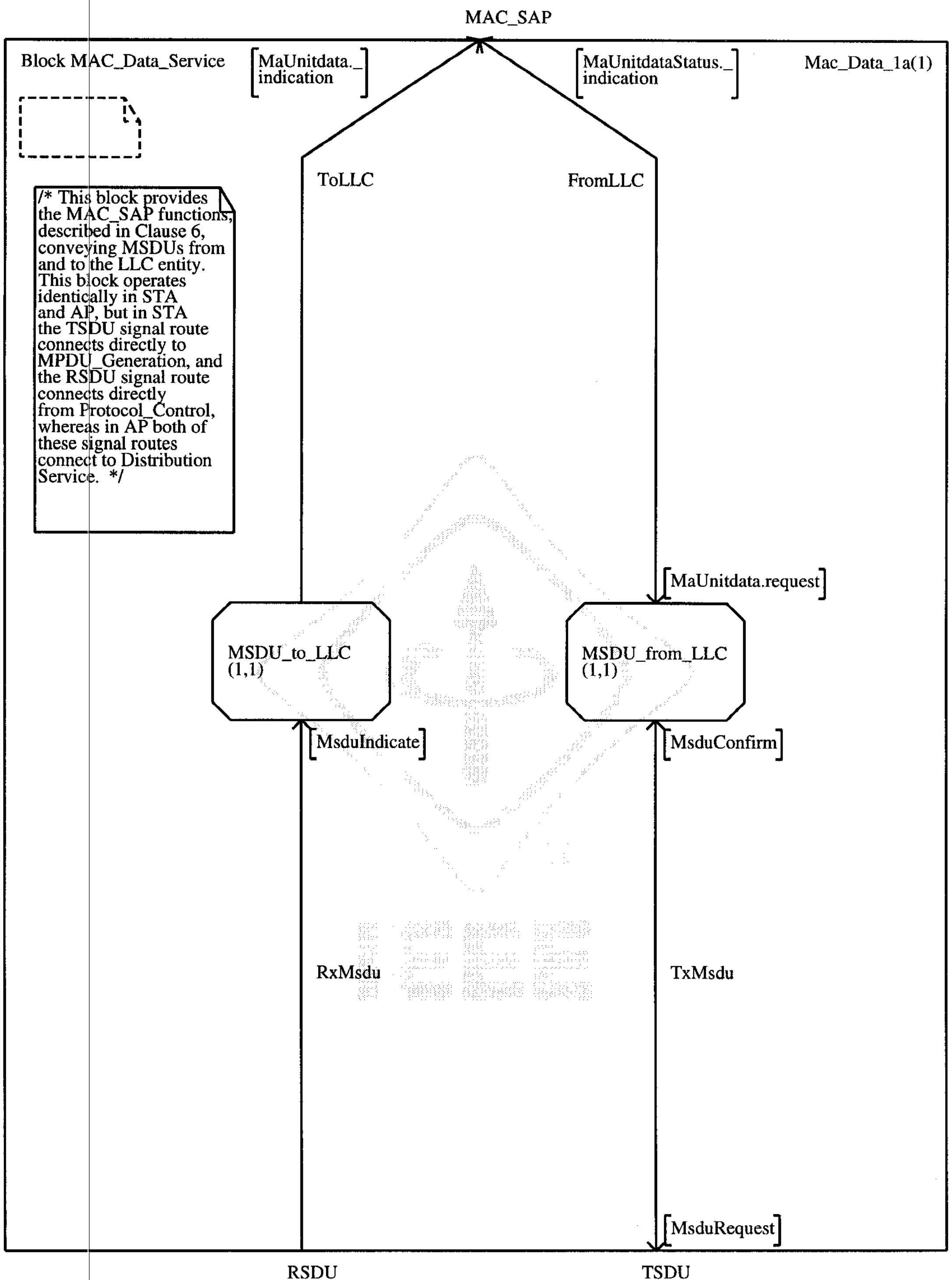
```
use macsorts ;
use macmib ;
```

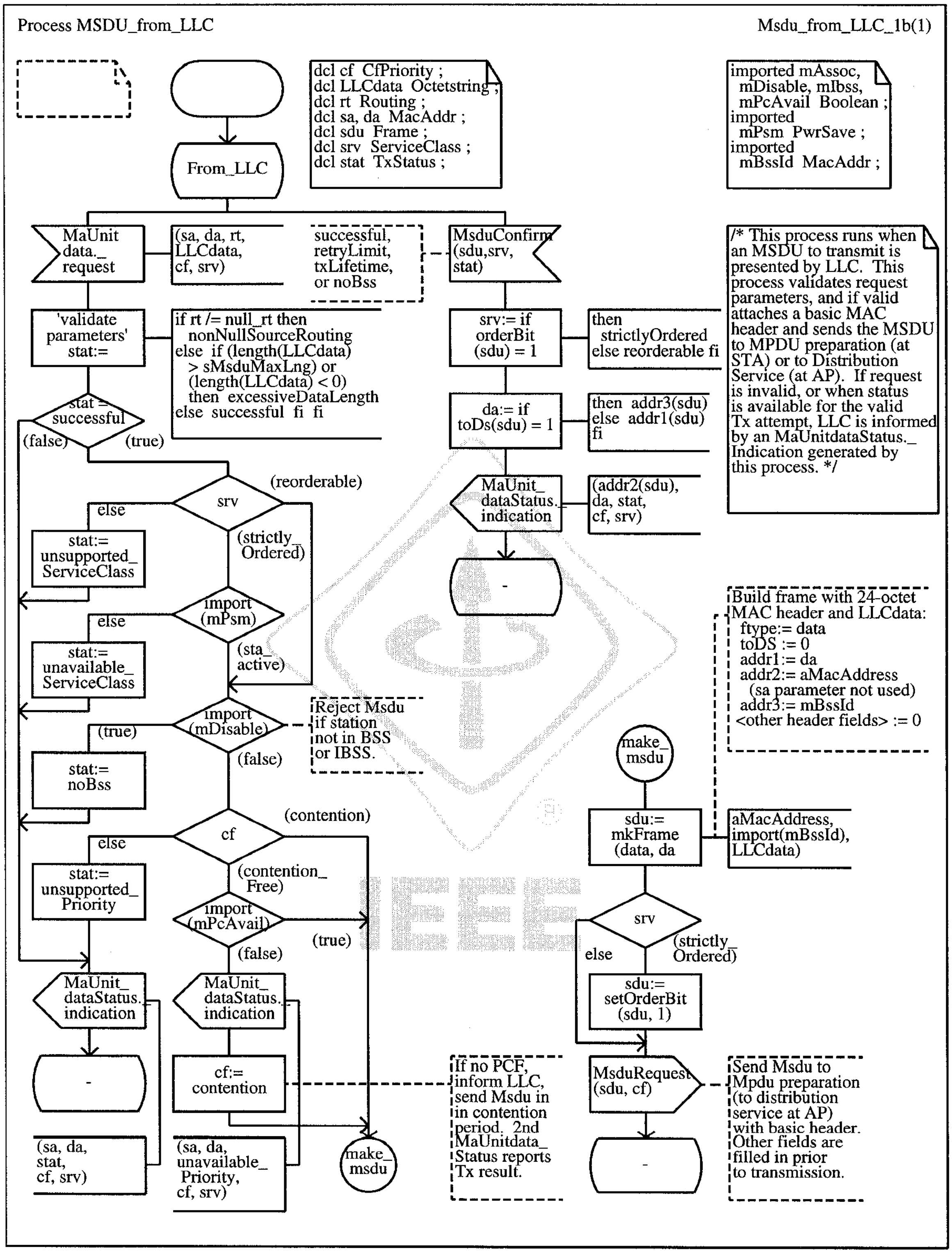
System Station — Sta_signals_2b(3)

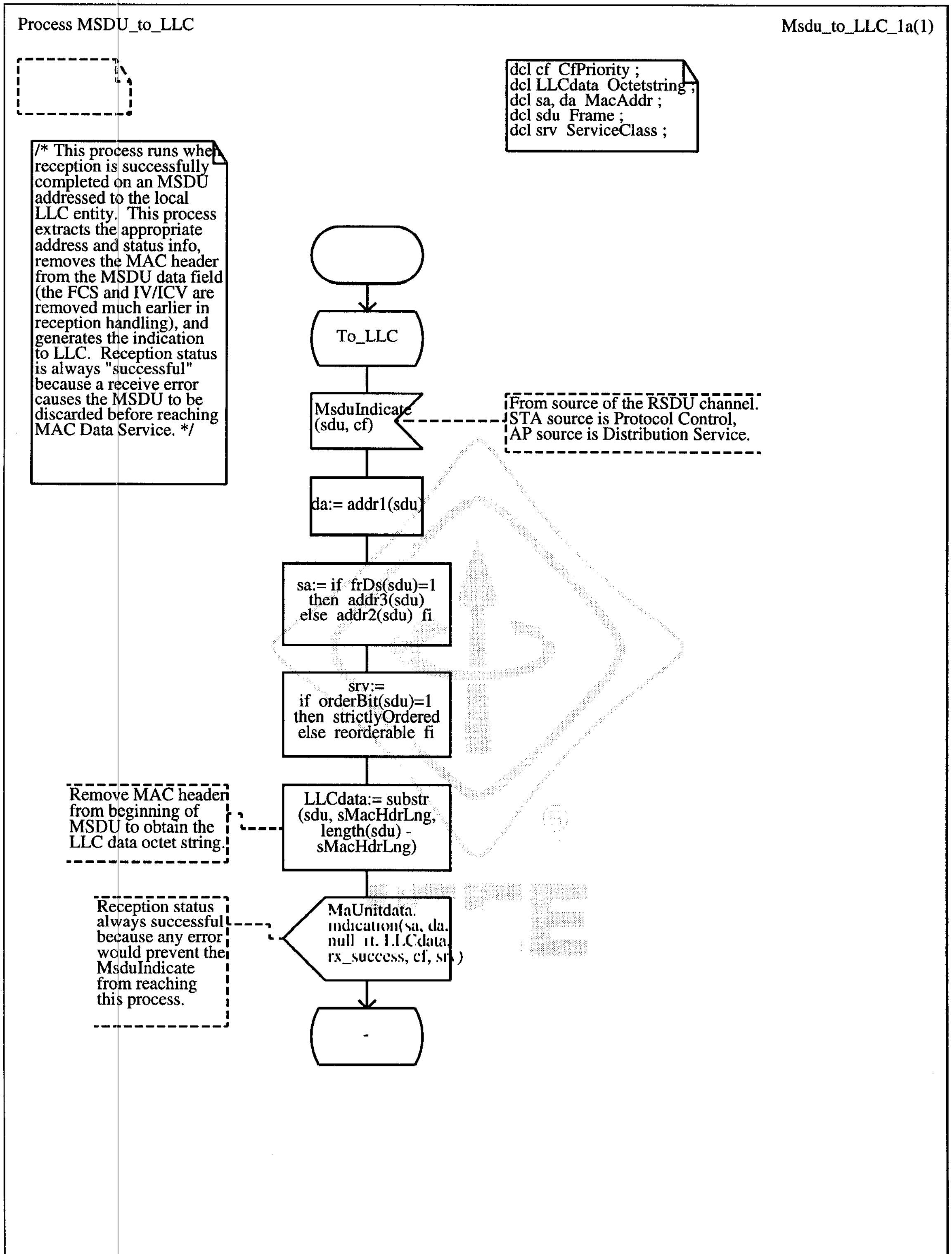
```
signal
  AtimW,
  Backoff(Integer,Integer),
  BkDone(Integer),
  Busy,
  Cancel,
  CfPolled,
  ChangeNav(Time,Duration,NavSrc),
  Doze,
  Idle,
  MaUnitdata.indication(MacAddr,MacAddr,
    Routing,Octetstring,RxStatus,
    CfPriority,ServiceClass),
  MaUnitdata.request(MacAddr,MacAddr,
    Routing,Octetstring,CfPriority,ServiceClass),
  MaUnitdataStatus.indication(MacAddr,
    MacAddr,TxStatus,CfPriority,ServiceClass),
  MlmeAssociate.confirm(MlmeStatus),
  MlmeAssociate.indication(MacAddr),
  MlmeAssociate.request(MacAddr,TU,Capability,Integer),
  MlmeAuthenticate.confirm
  [illegible]
```

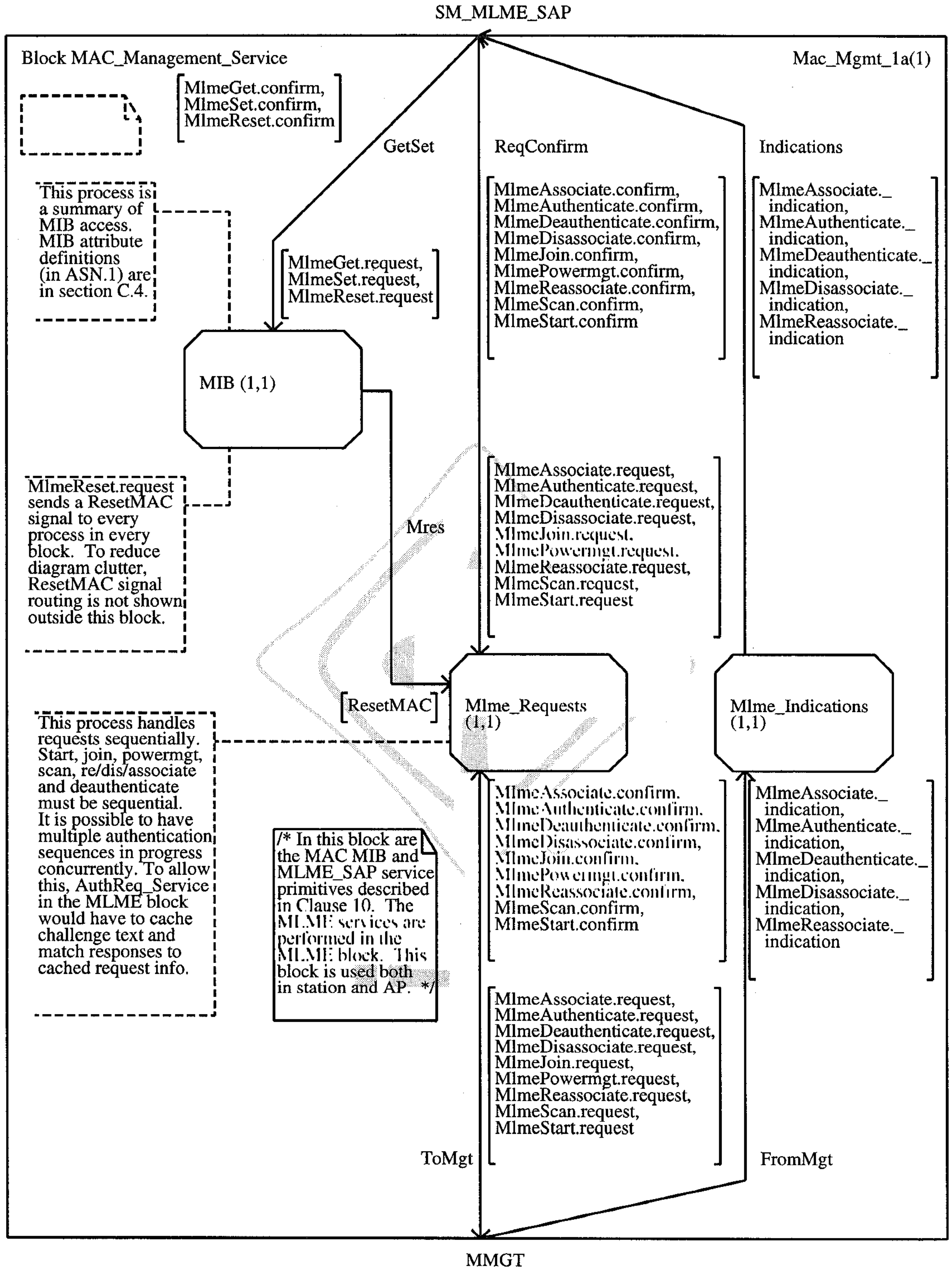
```
signal
  MmCancel,
  MmConfirm(Frame,TxStatus),
  MmIndicate(Frame,Time,Time,StateErr),
  MmRequest(Frame,Imed,Rate),
  MsduConfirm(Frame,CfPriority,TxStatus),
  MsduIndicate(Frame,CfPriority),
  MsduRequest(Frame,CfPriority),
  NeedAck(MacAddr,Time,Duration,Rate),
  PduConfirm(FragSdu,TxResult),
  PduRequest(FragSdu),
  PhyCca.indication(Ccastatus),
  PhyCcarst.confirm,
  PhyCcarst.request,
  PhyData.confirm,
  PhyData.indication(Octet),
  PhyData.request(Octet),
  PhyRxEnd.indication(PhyRxStat),
  PhyRxStart.indication(Integer,Rate),
  PhyTxEnd.confirm,
  PhyTxEnd.request,
  PhyTxStart.confirm,
  PhyTxStart.request(Integer,Rate),
  PlmeGet.confirm(MibStatus,
    MibAtrib,MibValue),
  [illegible]
  [illegible]
  [illegible]
  [illegible]
  [illegible]
  [illegible]
  PsChange(MacAddr,PsMode),
  PsIndicate(MacAddr,PsMode),
  PsInquiry(MacAddr),
  PsResponse(MacAddr,PsMode),
  ResetMAC,
  RxCfAck(MacAddr),
  RxIndicate(Frame,Time,Time,Rate),
  Slot,
  SsInquiry(MacAddr),
  SsResponse(MacAddr,
    StationState,StationState),
  SwChnl(Integer,Boolean),
  SwDone,
  TBTT,
  TxConfirm,
  TxRequest(Frame,Rate),
  Wake ;
```

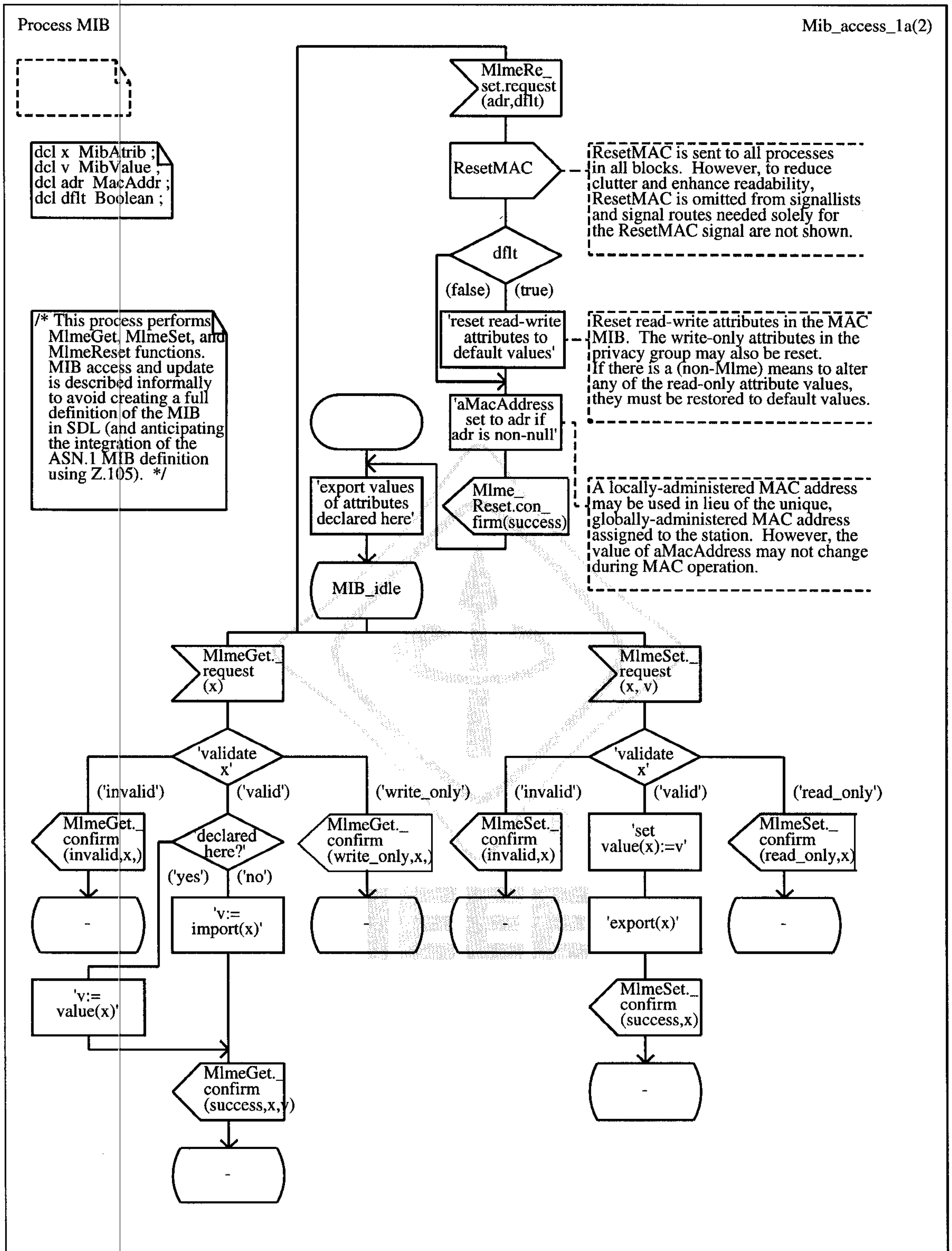

use macsorts
use macmib ;
System Station
Sta_signallists_3a(3)
signallist
MlmeRequestSignals=
MlmeAssociate.request,
MlmeAuthenticate.request,
MlmeDeauthenticate.request,
MlmeDisassociate.request,
MlmeGet.request,
MlmeJoin.request,
MlmePowermgt.request,
MlmeReassociate.request,
MlmeReset.request,
MlmeScan.request,
MlmeSet.request,
MlmeStart.request ;
signallist
MlmeConfirmSignals=
MlmeAssociate.confirm,
MlmeAuthenticate.confirm,
MlmeDeauthenticate.confirm,
MlmeDisassociate.confirm,
MlmeGet.confirm,
MlmeJoin.confirm,
MlmePowermgt.confirm,
MlmeReassociate.confirm,
MlmeReset.confirm,
MlmeScan.confirm,
MlmeSet.confirm,
MlmeStart.confirm ;
signallist
MlmeIndicationSignals=
MlmeAuthenticate.indication,
MlmeDeauthenticate.indication,
MlmeDisassociate.indication,
MlmeAssociate.indication,
MlmeReassociate.indication ;
signallist
MmgtRequestSignals=
MlmeAssociate.request,
MlmeAuthenticate.request,
MlmeStart.request ;
signallist
MmgtConfirmSignals=
MlmeAssociate.confirm,
MlmeAuthenticate.confirm,
MlmeDeauthenticate.confirm,
MlmeDisassociate.confirm,
MlmeJoin.confirm,
MlmePowermgt.confirm,
MlmeReassociate.confirm,
MlmeScan.confirm,
MlmeStart.confirm ;
signallist
MmgtIndicationSignals=
MlmeAuthenticate.indication,
MlmeReassociate.indication ;
signallist
PhyTxRequestSignals=
PhyTxStart.request,
PhyTxEnd.request,
PhyData.request ;
signallist
PhyTxConfirmSignals=
PhyTxStart.confirm,
PhyTxEnd.confirm,
PhyData.confirm ;
PhyCcareset.confirm ;
signallist
PlmeRequestSignals=
PlmeGet.request,
PlmeSet.request,
PlmeReset.request ;
signallist
PlmeConfirmSignals=
PlmeGet.confirm,
PlmeReset.confirm,
PlmeSet.confirm ;


MAC_SAP
Block MAC_Data_Service
MaUnitdata._ indication
MaUnitdataStatus._ indication
Mac_Data_1a(1)
/* This block provides the MAC_SAP functions, described in Clause 6, conveying MSDUs from and to the LLC entity. This block operates identically in STA and AP, but in STA the TSDU signal route connects directly to MPDU_Generation, and the RSDU signal route connects directly from Protocol_Control, whereas in AP both of these signal routes connect to Distribution Service. */
ToLLC
FromLLC
MaUnitdata.request
MSDU_to_LLC (1,1)
MSDU_from_LLC (1,1)
MsduIndicate
MsduConfirm
RxMsdu
TxMsdu
MsduRequest
RSDU
TSDU


Process MSDU_from_LLC
Msdu_from_LLC_1b(1)
dcl cf CfPriority ;
dcl LLCdata Octetstring ;
dcl rt Routing ;
dcl sa, da MacAddr ;
dcl sdu Frame ;
dcl srv ServiceClass ;
dcl stat TxStatus ;
imported mAssoc, mDisable, mIbss, mPcAvail Boolean ; imported mPsm PwrSave ; imported mBssId MacAddr ;
From_LLC
MaUnit data._ request
(sa, da, rt, LLCdata, cf, srv)
successful, retryLimit, txLifetime, or noBss
MsduConfirm (sdu,srv, stat)
/* This process runs when an MSDU to transmit is presented by LLC. This process validates request parameters, and if valid attaches a basic MAC header and sends the MSDU to MPDU preparation (at STA) or to Distribution Service (at AP). If request is invalid, or when status is available for the valid Tx attempt, LLC is informed by an MaUnitdataStatus._ Indication generated by this process. */
'validate parameters' stat:=
if rt /= null_rt then nonNullSourceRouting else if (length(LLCdata) > sMsduMaxLng) or (length(LLCdata) < 0) then excessiveDataLength else successful fi fi
srv:= if orderBit (sdu) = 1
then strictlyOrdered else reorderable fi
da:= if toDs(sdu) = 1
then addr3(sdu) else addr1(sdu) fi
stat = successful
(false)
(true)
srv
(reorderable)
else
(strictly_ Ordered)
stat:= unsupported_ ServiceClass
MaUnit_ dataStatus. indication
(addr2(sdu), da, stat, cf, srv)
-
import (mPsm)
(sta_ active)
stat:= unavailable_ ServiceClass
Build frame with 24-octet MAC header and LLCdata: ftype:= data toDS := 0 addr1:= da addr2:= aMacAddress (sa parameter not used) addr3:= mBssId <other header fields> := 0
import (mDisable)
(true)
(false)
Reject Msdu if station not in BSS or IBSS.
stat:= noBss
make_ msdu
cf
(contention)
(contention_ Free)
stat:= unsupported_ Priority
sdu:= mkFrame (data, da
aMacAddress, import(mBssId), LLCdata)
import (mPcAvail)
(true)
(false)
srv
else
(strictly_ Ordered)
sdu:= setOrderBit (sdu, 1)
MaUnit_ dataStatus. indication
cf:= contention
If no PCF, inform LLC, send Msdu in in contention period. 2nd MaUnitdata_ Status reports Tx result.
MsduRequest (sdu, cf)
Send Msdu to Mpdu preparation (to distribution service at AP) with basic header. Other fields are filled in prior to transmission.
(sa, da, stat, cf, srv)
(sa, da, unavailable_ Priority, cf, srv)
make_ msdu


Process MSDU_to_LLC
Msdu_to_LLC_1a(1)
dcl cf CfPriority ;
dcl LLCdata Octetstring ;
dcl sa, da MacAddr ;
dcl sdu Frame ;
dcl srv ServiceClass ;
/* This process runs when reception is successfully completed on an MSDU addressed to the local LLC entity. This process extracts the appropriate address and status info, removes the MAC header from the MSDU data field (the FCS and IV/ICV are removed much earlier in reception handling), and generates the indication to LLC. Reception status is always "successful" because a receive error causes the MSDU to be discarded before reaching MAC Data Service. */
To_LLC
MsduIndicate
(sdu, cf)
From source of the RSDU channel.
STA source is Protocol Control,
AP source is Distribution Service.
da:= addr1(sdu)
sa:= if frDs(sdu)=1
then addr3(sdu)
else addr2(sdu) fi
srv:=
if orderBit(sdu)=1
then strictlyOrdered
else reorderable fi
Remove MAC header from beginning of MSDU to obtain the LLC data octet string.
LLCdata:= substr
(sdu, sMacHdrLng,
length(sdu) -
sMacHdrLng)
Reception status always successful because any error would prevent the MsduIndicate from reaching this process.
MaUnitdata.
indication(sa, da,
rx_success, cf,
-


SM_MLME_SAP
Block MAC_Management_Service
Mac_Mgmt_1a(1)
[MlmeGet.confirm, MlmeSet.confirm, MlmeReset.confirm]
GetSet
ReqConfirm
Indications
This process is a summary of MIB access. MIB attribute definitions (in ASN.1) are in section C.4.
[MlmeAssociate.confirm, MlmeAuthenticate.confirm, MlmeDeauthenticate.confirm, MlmeDisassociate.confirm, MlmeJoin.confirm, MlmePowermgt.confirm, MlmeReassociate.confirm, MlmeScan.confirm, MlmeStart.confirm]
[MlmeAssociate._ indication, MlmeAuthenticate._ indication, MlmeDeauthenticate._ indication, MlmeDisassociate._ indication, MlmeReassociate._ indication]
[MlmeGet.request, MlmeSet.request, MlmeReset.request]
MIB (1,1)
MlmeReset.request sends a ResetMAC signal to every process in every block. To reduce diagram clutter, ResetMAC signal routing is not shown outside this block.
Mres
[MlmeAssociate.request, MlmeAuthenticate.request, MlmeDeauthenticate.request, MlmeDisassociate.request, MlmeJoin.request, MlmePowermgt.request, MlmeReassociate.request, MlmeScan.request, MlmeStart.request]
[ResetMAC]
Mlme_Requests (1,1)
Mlme_Indications (1,1)
This process handles requests sequentially. Start, join, powermgt, scan, re/dis/associate and deauthenticate must be sequential. It is possible to have multiple authentication sequences in progress concurrently. To allow this, AuthReq_Service in the MLME block would have to cache challenge text and match responses to cached request info.
/* In this block are the MAC MIB and MLME_SAP service primitives described in Clause 10. The MLME services are performed in the MLME block. This block is used both in station and AP. */
[MlmeAssociate.confirm, MlmeAuthenticate.confirm, MlmeDeauthenticate.confirm, MlmeDisassociate.confirm, MlmeJoin.confirm, MlmePowermgt.confirm, MlmeReassociate.confirm, MlmeScan.confirm, MlmeStart.confirm]
[MlmeAssociate._ indication, MlmeAuthenticate._ indication, MlmeDeauthenticate._ indication, MlmeDisassociate._ indication, MlmeReassociate._ indication]
[MlmeAssociate.request, MlmeAuthenticate.request, MlmeDeauthenticate.request, MlmeDisassociate.request, MlmeJoin.request, MlmePowermgt.request, MlmeReassociate.request, MlmeScan.request, MlmeStart.request]
ToMgt
FromMgt
MMGT


Process MIB
Mib_access_1a(2)
dcl x MibAtrib ;
dcl v MibValue ;
dcl adr MacAddr ;
dcl dflt Boolean ;
/* This process performs MlmeGet, MlmeSet, and MlmeReset functions. MIB access and update is described informally to avoid creating a full definition of the MIB in SDL (and anticipating the integration of the ASN.1 MIB definition using Z.105). */
MlmeRe_ set.request (adr,dflt)
ResetMAC
ResetMAC is sent to all processes in all blocks. However, to reduce clutter and enhance readability, ResetMAC is omitted from signallists and signal routes needed solely for the ResetMAC signal are not shown.
dflt
(false)
(true)
'reset read-write attributes to default values'
Reset read-write attributes in the MAC MIB. The write-only attributes in the privacy group may also be reset. If there is a (non-Mlme) means to alter any of the read-only attribute values, they must be restored to default values.
'aMacAddress set to adr if adr is non-null'
A locally-administered MAC address may be used in lieu of the unique, globally-administered MAC address assigned to the station. However, the value of aMacAddress may not change during MAC operation.
Mlme_ Reset.con_ firm(success)
'export values of attributes declared here'
MIB_idle
MlmeGet._ request (x)
MlmeSet._ request (x, v)
'validate x'
('invalid')
('valid')
('write_only')
('read_only')
MlmeGet._ confirm (invalid,x,)
'declared here?'
('yes')
('no')
MlmeGet._ confirm (write_only,x,)
MlmeSet._ confirm (invalid,x)
'set value(x):=v'
MlmeSet._ confirm (read_only,x)
-
'v:= import(x)'
'export(x)'
'v:= value(x)'
MlmeSet._ confirm (success,x)
MlmeGet._ confirm (success,x,v)

Process MIB

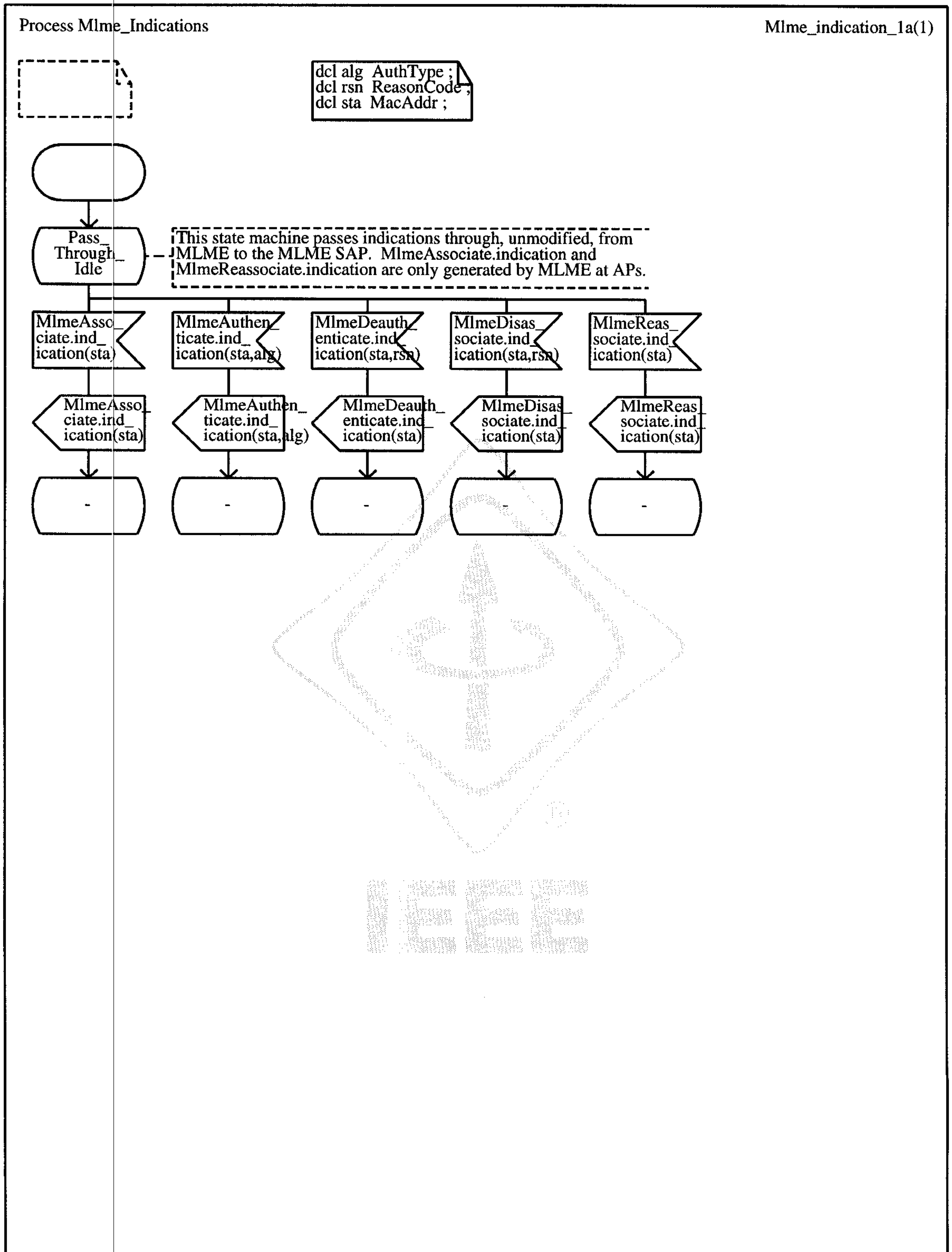
Mib_import_export_2a(2)

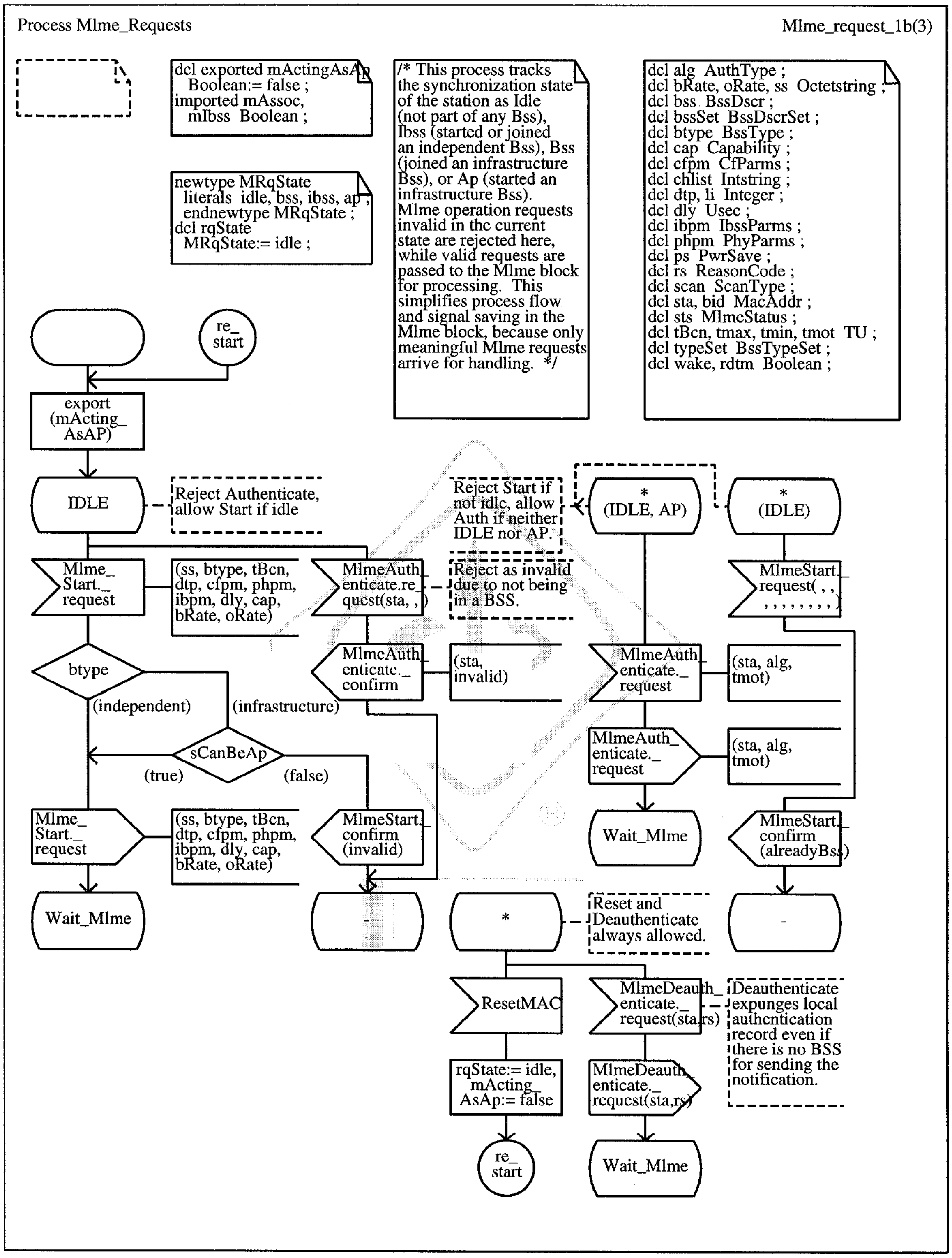
```
/* Import of {Read-Only} MIB counter
  values exported from other processes */
imported
 aAckFailureCount,
 aFailedCount,
 aFcsErrorCount,
 aFrameDuplicateCount,
 aMulticastReceivedFrameCount,
 aMulticastTransmittedFrameCount,
 aMultipleRetryCount,
 aReceivedFragmentCount,
 aRetryCount,
 aRtsFailureCount,
 aRtsSuccessCount,
 aTransmittedFragmentCount,
 aWepExcludedCount,
 aWepIcvErrorCount,
 aWepUndecryptableCount  Counter32 ;
```

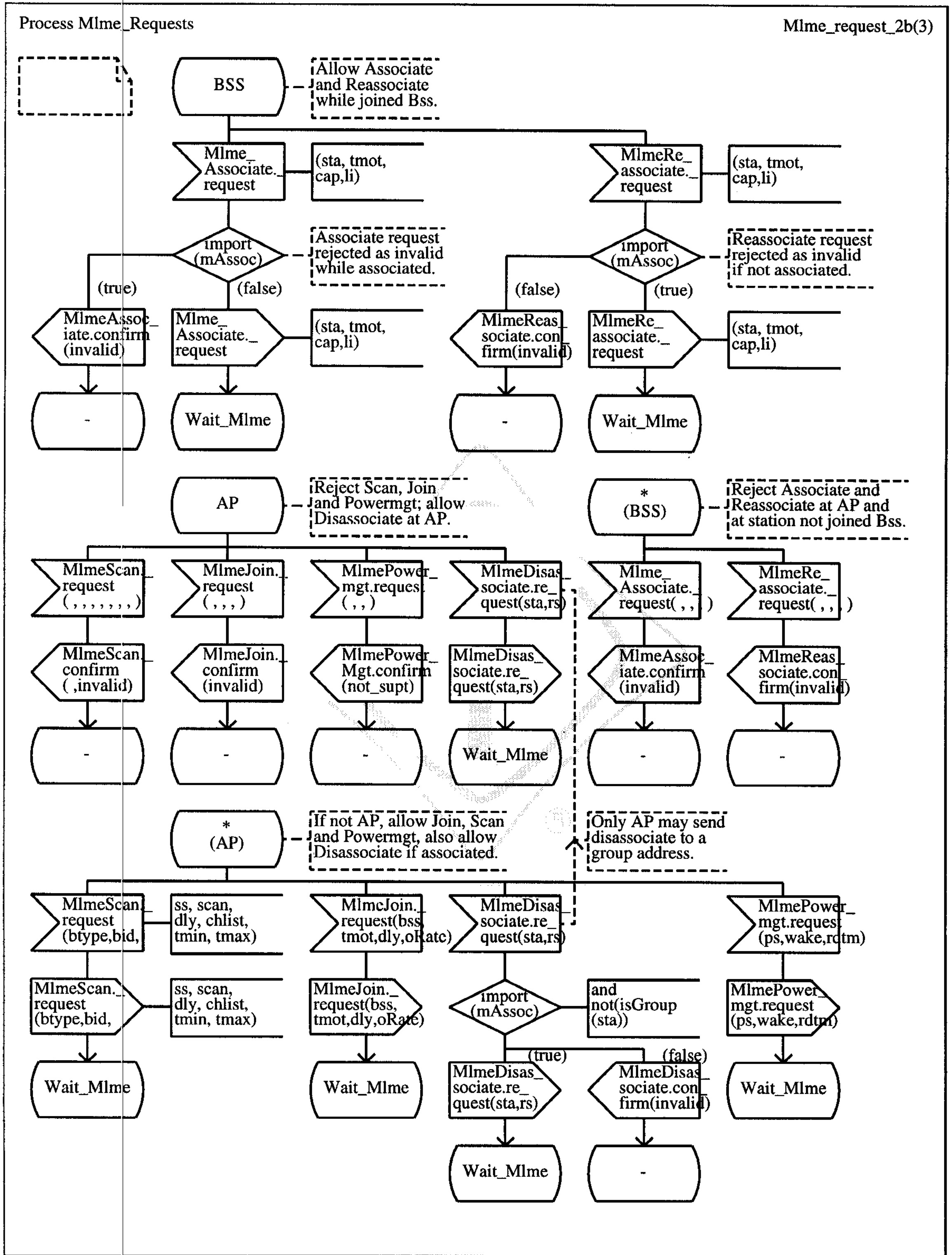
```
/* Declarations of MIB attributes exported from this process */

     /* Read-Write attributes */
dcl exported
 aAuthenticationType  AuthTypeSet:=
    incl(open_system, shared_key),
 aExcludeUnencrypted  Boolean:= false,
 aFragmentationThreshold  Integer:= 2346,
 aGroupAddresses  MacAddrSet:= empty,
 aLongRetryLimit  Integer:= 4,
 aMaxReceiveLifetime  TU:= 512,
 aMaxTransmitMsduLifetime  TU:= 512,
 aMediumOccupancyLimit  TU:= 100,
 aPrivacyInvoked  Boolean:= false,
 aReceiveDTIMs  Boolean:= true,
 aCfpPeriod  Integer:= 1,
 aCfpMaxDuration  TU:= 200,
 aAuthenticationResponseTimeout  TU:= 512,
 aRtsThreshold  Integer:= 3000,
 aShortRetryLimit  Integer:= 7,
 aWepDefaultKeyId  KeyIndex:= 0,
 aCurrentChannelNumber  Integer:= 0,
 aCurrentSet  Integer:= 0,
 aCurrentPattern  Integer:= 0,
 aCurrentIndex  Integer:= 0 ;

     /* Write-Only attributes */
dcl exported
 aWepDefaultKeys  KeyVector:= nullKey,
 aWepKeyMappings
    KeyMapArray:= (. nullAddr, false, nullKey .) ;
```

```
/* The followin[illegible] the
  MAC MIB ar[illegible] med
  constants) rat[illegible]
  because they [illegible]
  station which are static, at least during
  any single instance of MAC operation:
    aAuthenticationAlgorithms  AuthTypeSet,
    aCfPollable  Boolean,
    aMacAddress  MacAddr,
    aManufacturerID  Octetstring,
    aPrivacyOptionImplemented  Boolean,
    aProductID  Octetstring,
    aStationID  MacAddr,
    aWepKeyMappingLength  Integer ;

  In addition, all Read-Only attributes in the
  PHY MIB which are accessed by the MAC
  are defined as synonyms.
*/
```

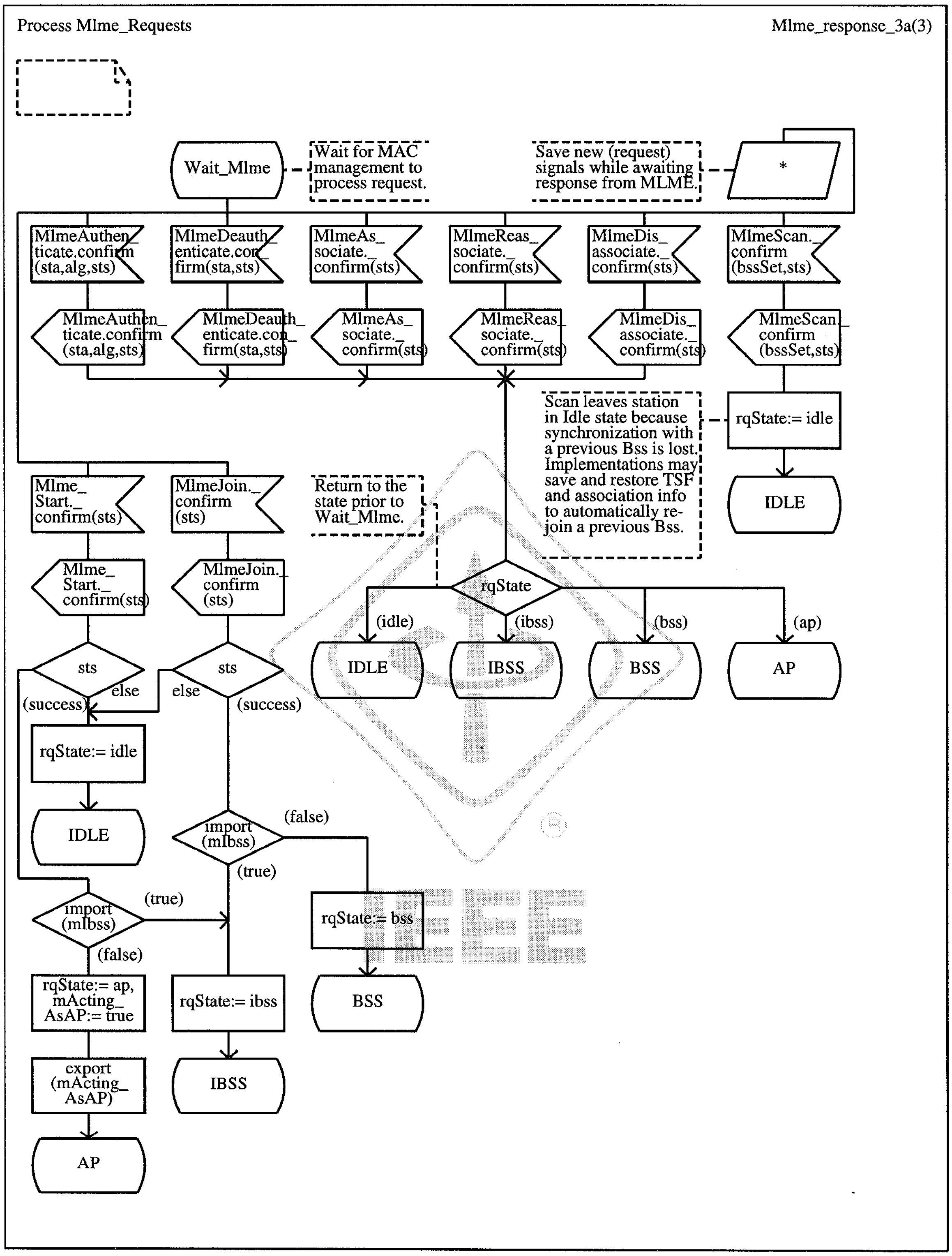
```
/* NOTE:
  The values listed for MAC MIB attributes are the
  specified default values for those attributes.
  The values listed for PHY MIB attributes are either
  the default values for the FH PHY, or arbitrary
  values within the specified range.  The specific
  values for PHY attributes in this SDL description
  of the MAC do not have normative significance.
*/
```

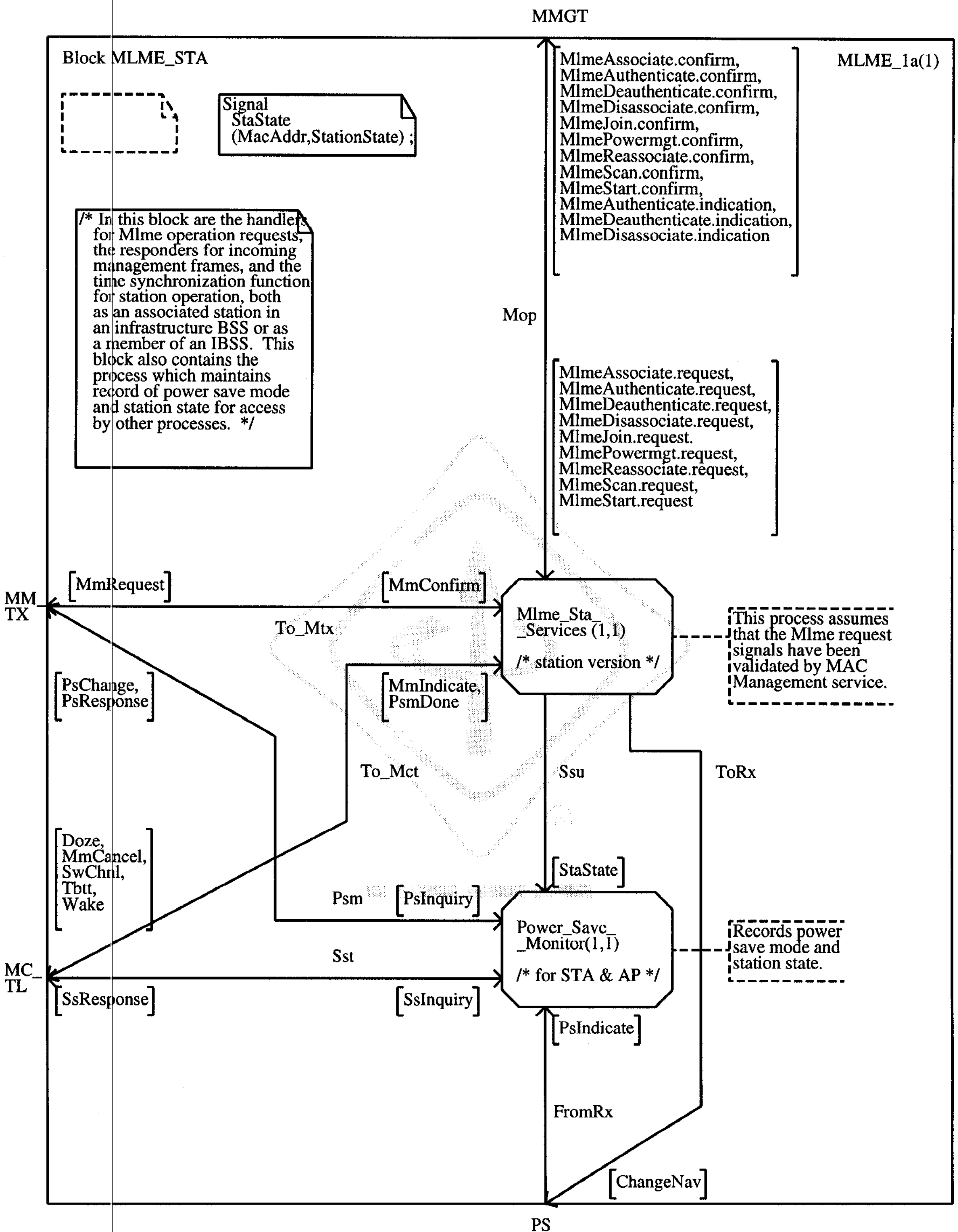

Process Mlme_Indications
Mlme_indication_1a(1)
dcl alg AuthType ;
dcl rsn ReasonCode ;
dcl sta MacAddr ;
Pass_ Through_ Idle
This state machine passes indications through, unmodified, from MLME to the MLME SAP. MlmeAssociate.indication and MlmeReassociate.indication are only generated by MLME at APs.
MlmeAsso_ ciate.ind_ ication(sta)
MlmeAuthen_ ticate.ind_ ication(sta,alg)
MlmeDeauth_ enticate.ind_ ication(sta,rsn)
MlmeDisas_ sociate.ind_ ication(sta,rsn)
MlmeReas_ sociate.ind_ ication(sta)
MlmeAsso_ ciate.ind_ ication(sta)
MlmeAuthen_ ticate.ind_ ication(sta,alg)
MlmeDeauth_ enticate.ind_ ication(sta)
MlmeDisas_ sociate.ind_ ication(sta)
MlmeReas_ sociate.ind_ ication(sta)
-
-
-
-
-

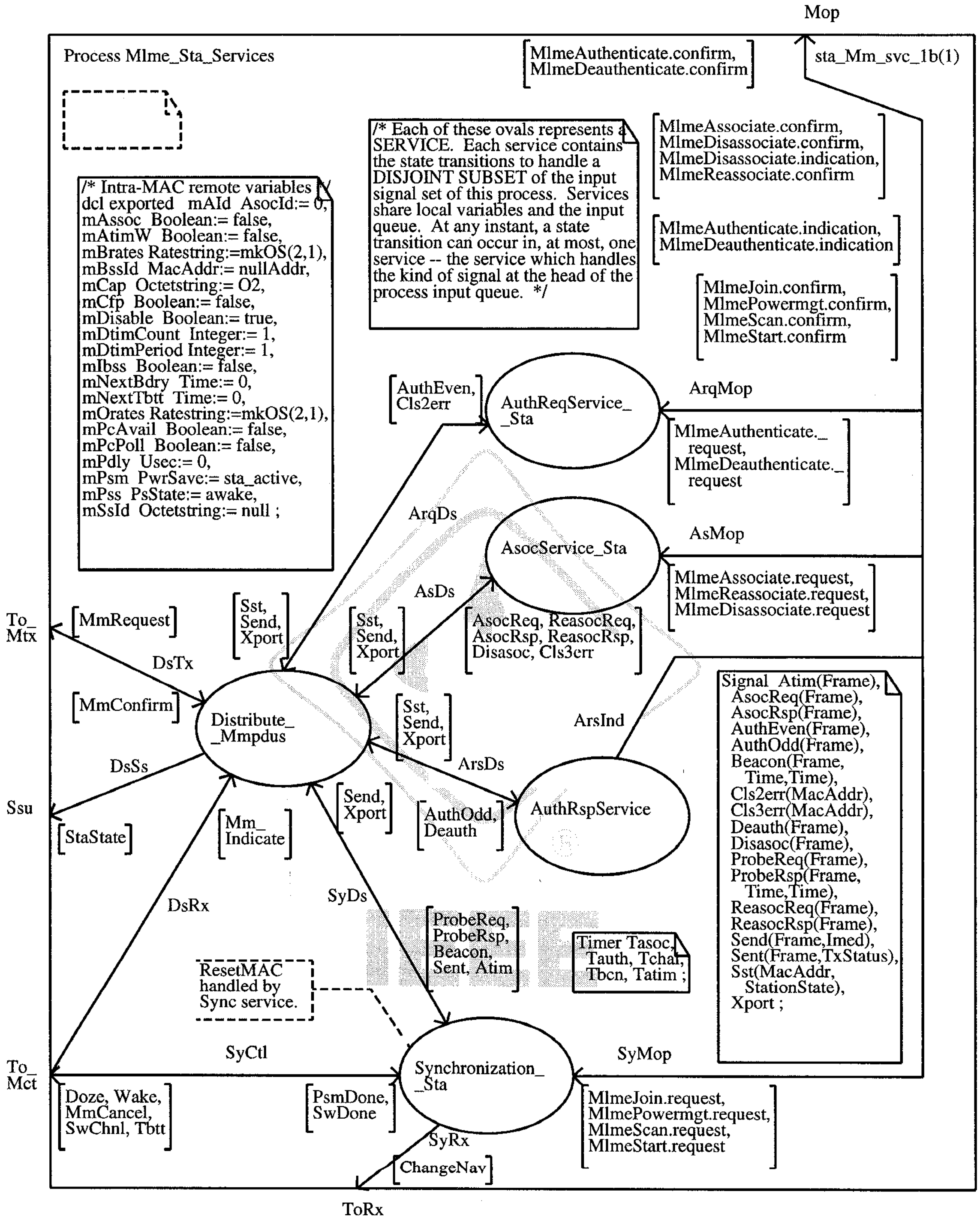

Process Mlme_Requests
Mlme_request_1b(3)
dcl exported mActingAsAp Boolean:= false ; imported mAssoc, mIbss Boolean ;
newtype MRqState literals idle, bss, ibss, ap ; endnewtype MRqState ; dcl rqState MRqState:= idle ;
/* This process tracks the synchronization state of the station as Idle (not part of any Bss), Ibss (started or joined an independent Bss), Bss (joined an infrastructure Bss), or Ap (started an infrastructure Bss). Mlme operation requests invalid in the current state are rejected here, while valid requests are passed to the Mlme block for processing. This simplifies process flow and signal saving in the Mlme block, because only meaningful Mlme requests arrive for handling. */
dcl alg AuthType ; dcl bRate, oRate, ss Octetstring ; dcl bss BssDscr ; dcl bssSet BssDscrSet ; dcl btype BssType ; dcl cap Capability ; dcl cfpm CfParms ; dcl chlist Intstring ; dcl dtp, li Integer ; dcl dly Usec ; dcl ibpm IbssParms ; dcl phpm PhyParms ; dcl ps PwrSave ; dcl rs ReasonCode ; dcl scan ScanType ; dcl sta, bid MacAddr ; dcl sts MlmeStatus ; dcl tBcn, tmax, tmin, tmot TU ; dcl typeSet BssTypeSet ; dcl wake, rdtm Boolean ;
re_ start
export (mActing_ AsAP)
IDLE
Reject Authenticate, allow Start if idle
Mlme_ Start._ request
(ss, btype, tBcn, dtp, cfpm, phpm, ibpm, dly, cap, bRate, oRate)
MlmeAuth_ enticate.re_ quest(sta, ,)
Reject as invalid due to not being in a BSS.
MlmeAuth_ enticate._ confirm
(sta, invalid)
btype
(independent)
(infrastructure)
sCanBeAp
(true)
(false)
Mlme_ Start._ request
(ss, btype, tBcn, dtp, cfpm, phpm, ibpm, dly, cap, bRate, oRate)
MlmeStart_ confirm (invalid)
Wait_Mlme
-
Reject Start if not idle, allow Auth if neither IDLE nor AP.
* (IDLE, AP)
* (IDLE)
MlmeStart_ request(, , , , , , , , , ,)
MlmeAuth_ enticate._ request
(sta, alg, tmot)
MlmeAuth_ enticate._ request
(sta, alg, tmot)
Wait_Mlme
MlmeStart_ confirm (alreadyBss)
-
*
Reset and Deauthenticate always allowed.
ResetMAC
MlmeDeauth_ enticate._ request(sta,rs)
Deauthenticate expunges local authentication record even if there is no BSS for sending the notification.
rqState:= idle, mActing_ AsAp:= false
MlmeDeauth_ enticate._ request(sta,rs)
re_ start
Wait_Mlme

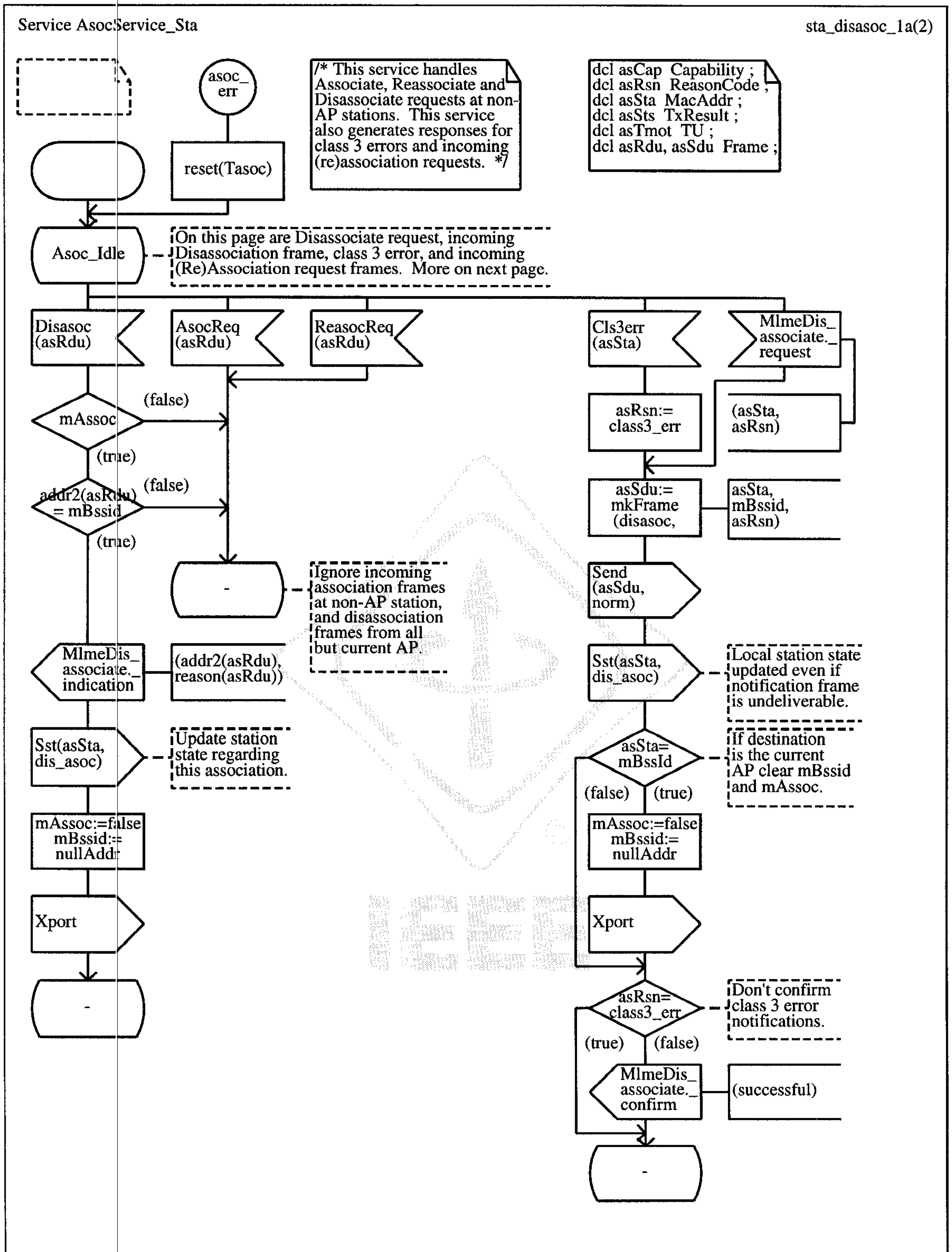


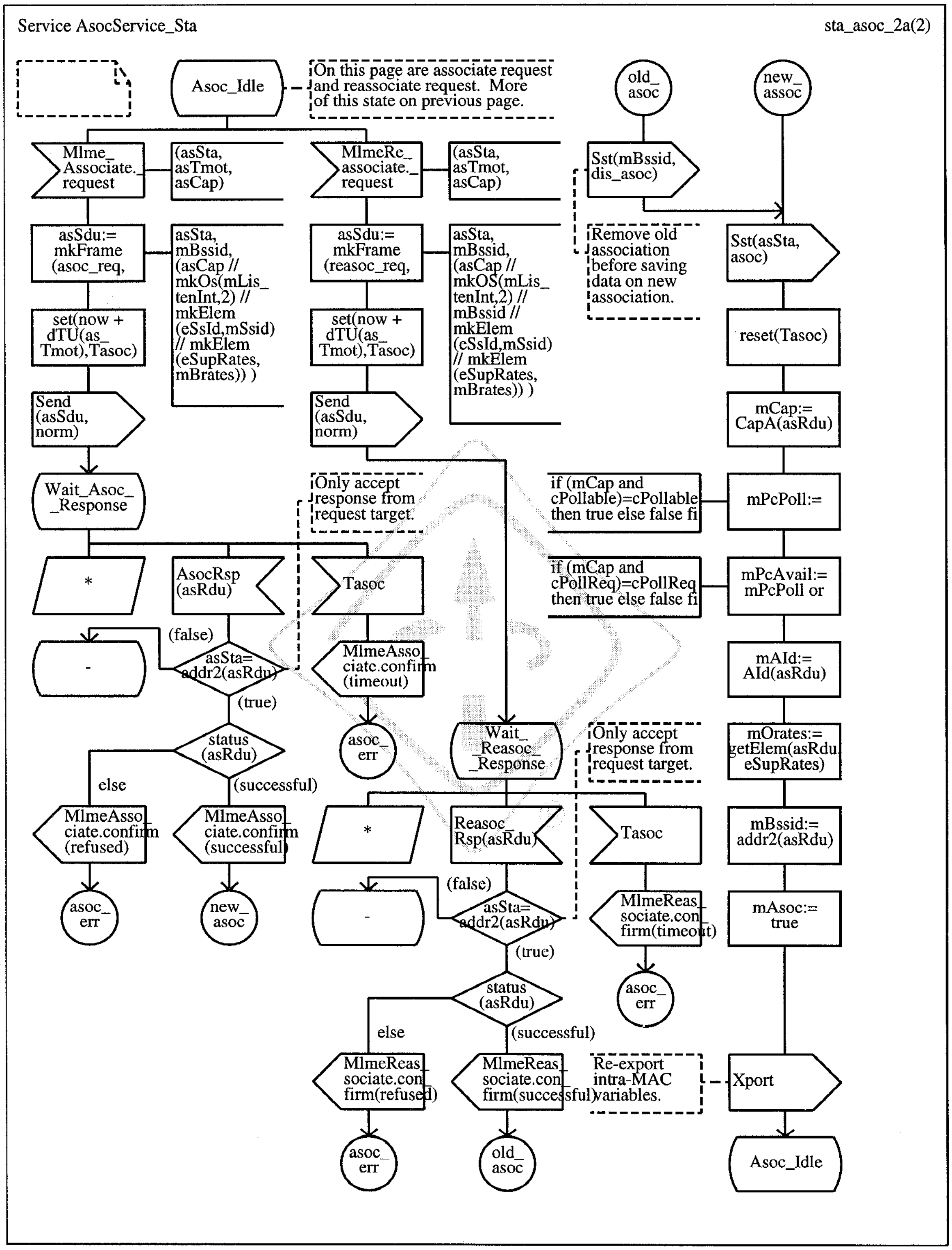
Process Mlme_Requests
Mlme_request_2b(3)
BSS
Allow Associate and Reassociate while joined Bss.
Mlme_ Associate._ request
(sta, tmot, cap,li)
MlmeRe_ associate._ request
(sta, tmot, cap,li)
import (mAssoc)
Associate request rejected as invalid while associated.
(true)
(false)
import (mAssoc)
Reassociate request rejected as invalid if not associated.
(false)
(true)
MlmeAssoc_ iate.confirm (invalid)
Mlme_ Associate._ request
(sta, tmot, cap,li)
MlmeReas_ sociate.con_ firm(invalid)
MlmeRe_ associate._ request
(sta, tmot, cap,li)
-
Wait_Mlme
-
Wait_Mlme
AP
Reject Scan, Join and Powermgt; allow Disassociate at AP.
* (BSS)
Reject Associate and Reassociate at AP and at station not joined Bss.
MlmeScan_ request (, , , , , , ,)
MlmeJoin._ request (, , ,)
MlmePowdr_ mgt.request (, ,)
MlmeDisas_ sociate.re_ quest(sta,rs)
Mlme_ Associate._ request(, , ,)
MlmeRe_ associate._ request(, , ,)
MlmeScan_ confirm (,invalid)
MlmeJoin._ confirm (invalid)
MlmePowdr_ Mgt.confirm (not_supt)
MlmeDisas_ sociate.re_ quest(sta,rs)
MlmeAssoc_ iate.confirm (invalid)
MlmeReas_ sociate.con_ firm(invalid)
-
-
-
Wait_Mlme
-
-
* (AP)
If not AP, allow Join, Scan and Powermgt, also allow Disassociate if associated.
Only AP may send disassociate to a group address.
MlmeScan_ request (btype,bid,
ss, scan, dly, chlist, tmin, tmax)
MlmeJoin._ request(bss, tmot,dly,oRate)
MlmeDisas_ sociate.re_ quest(sta,rs)
MlmePowdr_ mgt.request (ps,wake,rdtm)
MlmeScan._ request (btype,bid,
ss, scan, dly, chlist, tmin, tmax)
MlmeJoin._ request(bss, tmot,dly,oRate)
import (mAssoc)
and not(isGroup (sta))
MlmePower_ mgt.request (ps,wake,rdtm)
(true)
(false)
Wait_Mlme
Wait_Mlme
MlmeDisas_ sociate.re_ quest(sta,rs)
MlmeDisas_ sociate.con_ firm(invalid)
Wait_Mlme
Wait_Mlme
-

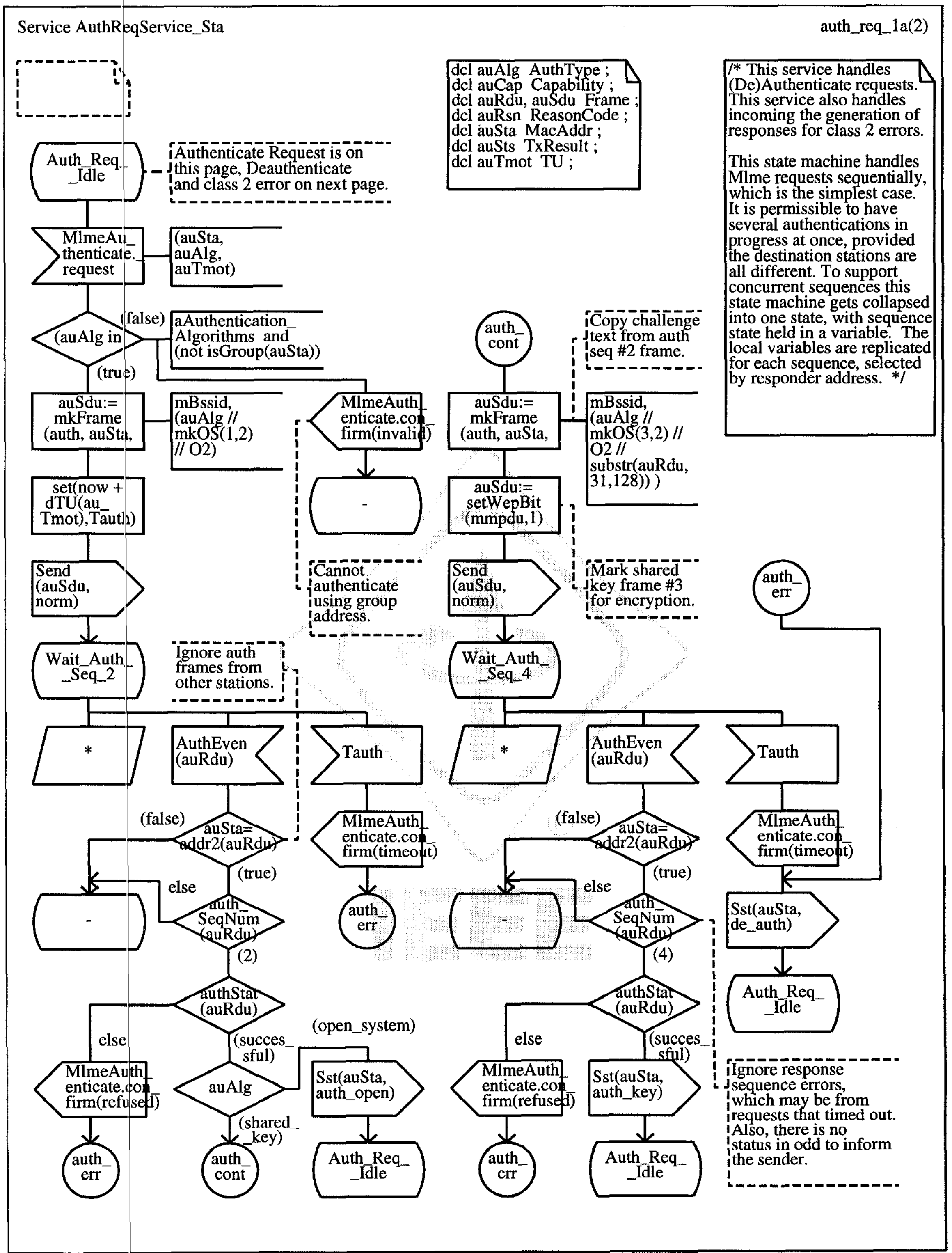

Process Mlme_Requests
Mlme_response_3a(3)
Wait_Mlme
Wait for MAC management to process request.
Save new (request) signals while awaiting response from MLME.
*
MlmeAuthenticate.confirm (sta,alg,sts)
MlmeDeauthenticate.confirm(sta,sts)
MlmeAssociate._confirm(sts)
MlmeReassociate._confirm(sts)
MlmeDisassociate._confirm(sts)
MlmeScan._confirm (bssSet,sts)
Scan leaves station in Idle state because synchronization with a previous Bss is lost. Implementations may save and restore TSF and association info to automatically re-join a previous Bss.
rqState:= idle
IDLE
Mlme_Start._confirm(sts)
MlmeJoin._confirm (sts)
Return to the state prior to Wait_Mlme.
rqState
(idle)
(ibss)
(bss)
(ap)
IDLE
IBSS
BSS
AP
sts
(success)
else
rqState:= idle
IDLE
import (mIbss)
(false)
(true)
rqState:= bss
BSS
rqState:= ap, mActing_AsAP:= true
rqState:= ibss
IBSS
export (mActing_AsAP)
AP

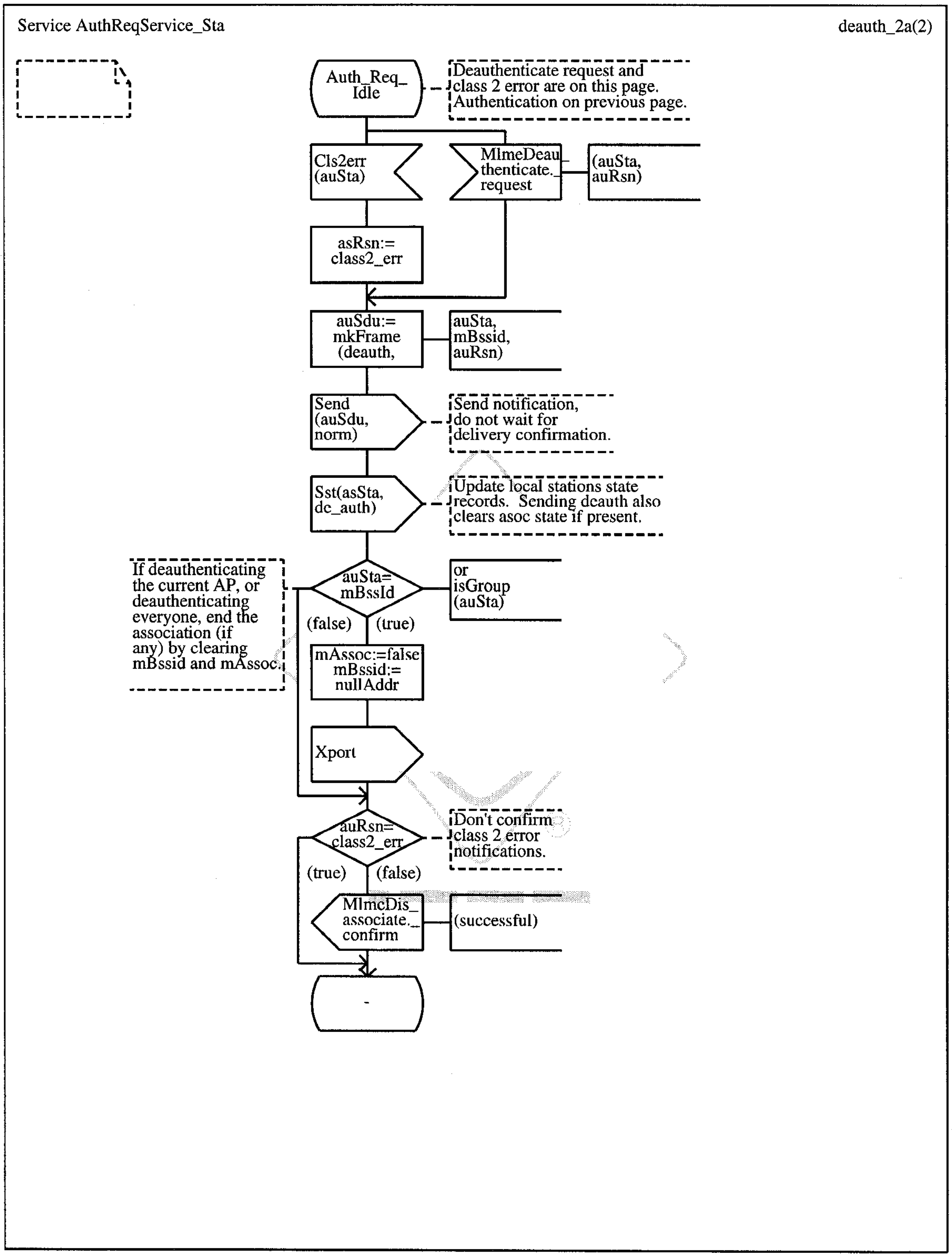

MMGT
Block MLME_STA
MLME_1a(1)
Signal StaState (MacAddr,StationState) ;
/* In this block are the handlers for Mlme operation requests, the responders for incoming management frames, and the time synchronization function for station operation, both as an associated station in an infrastructure BSS or as a member of an IBSS. This block also contains the process which maintains record of power save mode and station state for access by other processes. */
MlmeAssociate.confirm, MlmeAuthenticate.confirm, MlmeDeauthenticate.confirm, MlmeDisassociate.confirm, MlmeJoin.confirm, MlmePowermgt.confirm, MlmeReassociate.confirm, MlmeScan.confirm, MlmeStart.confirm, MlmeAuthenticate.indication, MlmeDeauthenticate.indication, MlmeDisassociate.indication
Mop
MlmeAssociate.request, MlmeAuthenticate.request, MlmeDeauthenticate.request, MlmeDisassociate.request, MlmeJoin.request. MlmePowermgt.request, MlmeReassociate.request, MlmeScan.request, MlmeStart.request
MM_TX
MmRequest
MmConfirm
To_Mtx
Mlme_Sta_ _Services (1,1) /* station version */
This process assumes that the Mlme request signals have been validated by MAC Management service.
PsChange, PsResponse
MmIndicate, PsmDone
To_Mct
Ssu
ToRx
Doze, MmCancel, SwChnl, Tbtt, Wake
StaState
Psm
PsInquiry
Power_Save_ _Monitor(1,1) /* for STA & AP */
Records power save mode and station state.
Sst
MC_TL
SsResponse
SsInquiry
PsIndicate
FromRx
ChangeNav
PS

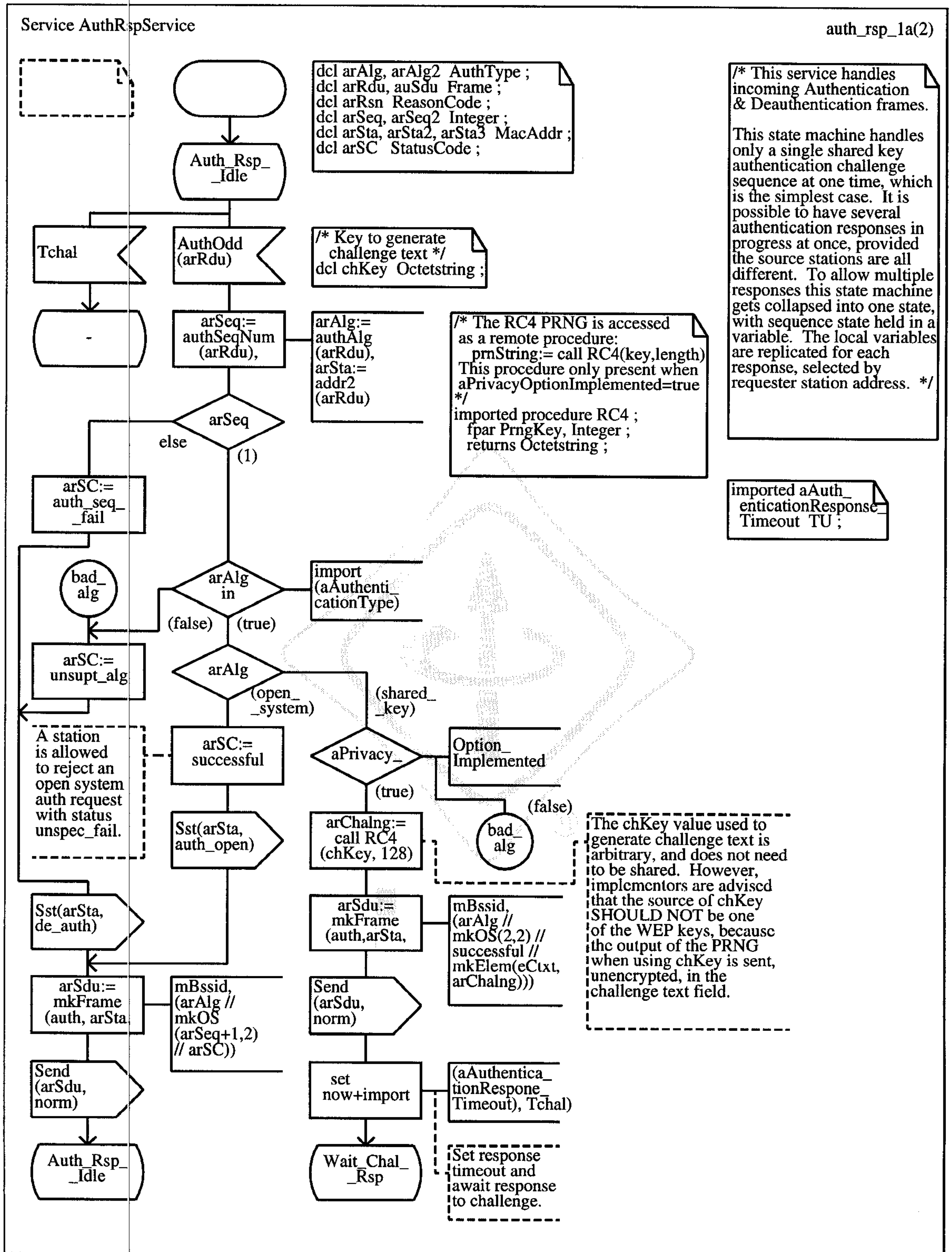

Mop
Process Mlme_Sta_Services
MlmeAuthenticate.confirm, MlmeDeauthenticate.confirm
sta_Mm_svc_1b(1)
/* Each of these ovals represents a SERVICE. Each service contains the state transitions to handle a DISJOINT SUBSET of the input signal set of this process. Services share local variables and the input queue. At any instant, a state transition can occur in, at most, one service -- the service which handles the kind of signal at the head of the process input queue. */
MlmeAssociate.confirm, MlmeDisassociate.confirm, MlmeDisassociate.indication, MlmeReassociate.confirm
MlmeAuthenticate.indication, MlmeDeauthenticate.indication
MlmeJoin.confirm, MlmePowermgt.confirm, MlmeScan.confirm, MlmeStart.confirm
/* Intra-MAC remote variables */
dcl exported mAId AsocId:= 0,
mAssoc Boolean:= false,
mAtimW Boolean:= false,
mBrates Ratestring:=mkOS(2,1),
mBssId MacAddr:= nullAddr,
mCap Octetstring:= O2,
mCfp Boolean:= false,
mDisable Boolean:= true,
mDtimCount Integer:= 1,
mDtimPeriod Integer:= 1,
mIbss Boolean:= false,
mNextBdry Time:= 0,
mNextTbtt Time:= 0,
mOrates Ratestring:=mkOS(2,1),
mPcAvail Boolean:= false,
mPcPoll Boolean:= false,
mPdly Usec:= 0,
mPsm PwrSave:= sta_active,
mPss PsState:= awake,
mSsId Octetstring:= null ;
AuthEven, Cls2err
AuthReqService_ _Sta
ArqMop
MlmeAuthenticate._ request, MlmeDeauthenticate._ request
ArqDs
AsocService_Sta
AsMop
MlmeAssociate.request, MlmeReassociate.request, MlmeDisassociate.request
AsDs
To_ Mtx
MmRequest
Sst, Send, Xport
Sst, Send, Xport
AsocReq, ReasocReq, AsocRsp, ReasocRsp, Disasoc, Cls3err
DsTx
MmConfirm
Distribute_ _Mmpdus
Sst, Send, Xport
ArsInd
Signal Atim(Frame), AsocReq(Frame), AsocRsp(Frame), AuthEven(Frame), AuthOdd(Frame), Beacon(Frame, Time,Time), Cls2err(MacAddr), Cls3err(MacAddr), Deauth(Frame), Disasoc(Frame), ProbeReq(Frame), ProbeRsp(Frame, Time,Time), ReasocReq(Frame), ReasocRsp(Frame), Send(Frame,Imed), Sent(Frame,TxStatus), Sst(MacAddr, StationState), Xport ;
DsSs
ArsDs
Ssu
Send, Xport
AuthOdd, Deauth
AuthRspService
StaState
Mm_ Indicate
DsRx
SyDs
ProbeReq, ProbeRsp, Beacon, Sent, Atim
Timer Tasoc, Tauth, Tchal, Tbcn, Tatim ;
ResetMAC handled by Sync service.
SyCtl
SyMop
To_ Mct
Synchronization_ _Sta
Doze, Wake, MmCancel, SwChnl, Tbtt
PsmDone, SwDone
MlmeJoin.request, MlmePowermgt.request, MlmeScan.request, MlmeStart.request
SyRx
ChangeNav
ToRx

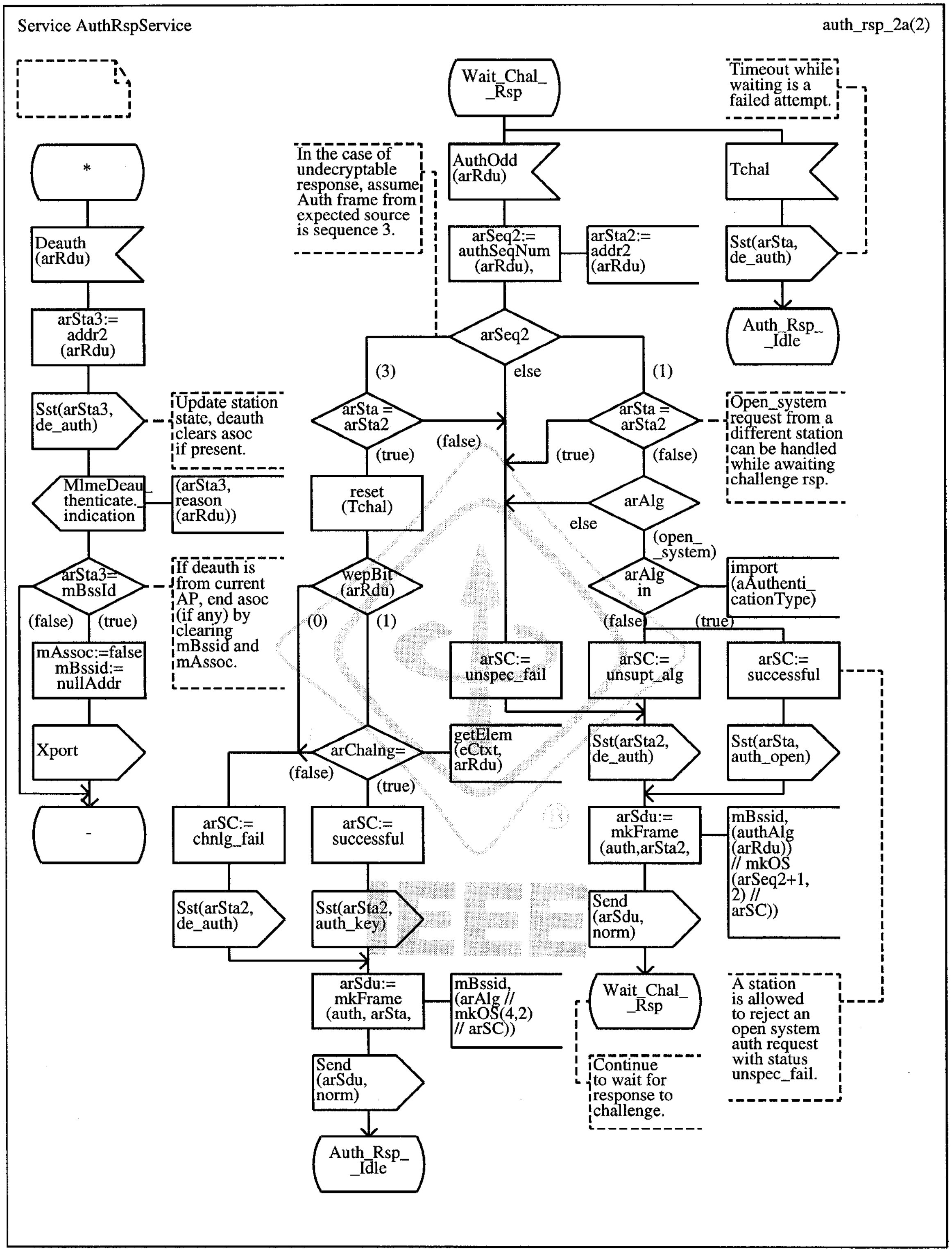

Service AsocService_Sta
sta_disasoc_1a(2)
asoc_ err
reset(Tasoc)
/* This service handles Associate, Reassociate and Disassociate requests at non-AP stations. This service also generates responses for class 3 errors and incoming (re)association requests. */
dcl asCap Capability ;
dcl asRsn ReasonCode ;
dcl asSta MacAddr ;
dcl asSts TxResult ;
dcl asTmot TU ;
dcl asRdu, asSdu Frame ;
Asoc_Idle
On this page are Disassociate request, incoming Disassociation frame, class 3 error, and incoming (Re)Association request frames. More on next page.
Disasoc (asRdu)
AsocReq (asRdu)
ReasocReq (asRdu)
Cls3err (asSta)
MlmeDis_ associate._ request
mAssoc
(false)
(true)
addr2(asRdu) = mBssid
(false)
(true)
asRsn:= class3_err
(asSta, asRsn)
asSdu:= mkFrame (disasoc,
asSta, mBssid, asRsn)
-
Ignore incoming association frames at non-AP station, and disassociation frames from all but current AP.
Send (asSdu, norm)
MlmeDis_ associate._ indication
(addr2(asRdu), reason(asRdu))
Sst(asSta, dis_asoc)
Local station state updated even if notification frame is undeliverable.
Sst(asSta, dis_asoc)
Update station state regarding this association.
asSta= mBssId
If destination is the current AP clear mBssid and mAssoc.
(false)
(true)
mAssoc:=false mBssid:= nullAddr
mAssoc:=false mBssid:= nullAddr
Xport
Xport
-
asRsn= class3_err
Don't confirm class 3 error notifications.
(true)
(false)
MlmeDis_ associate._ confirm
(successful)
-

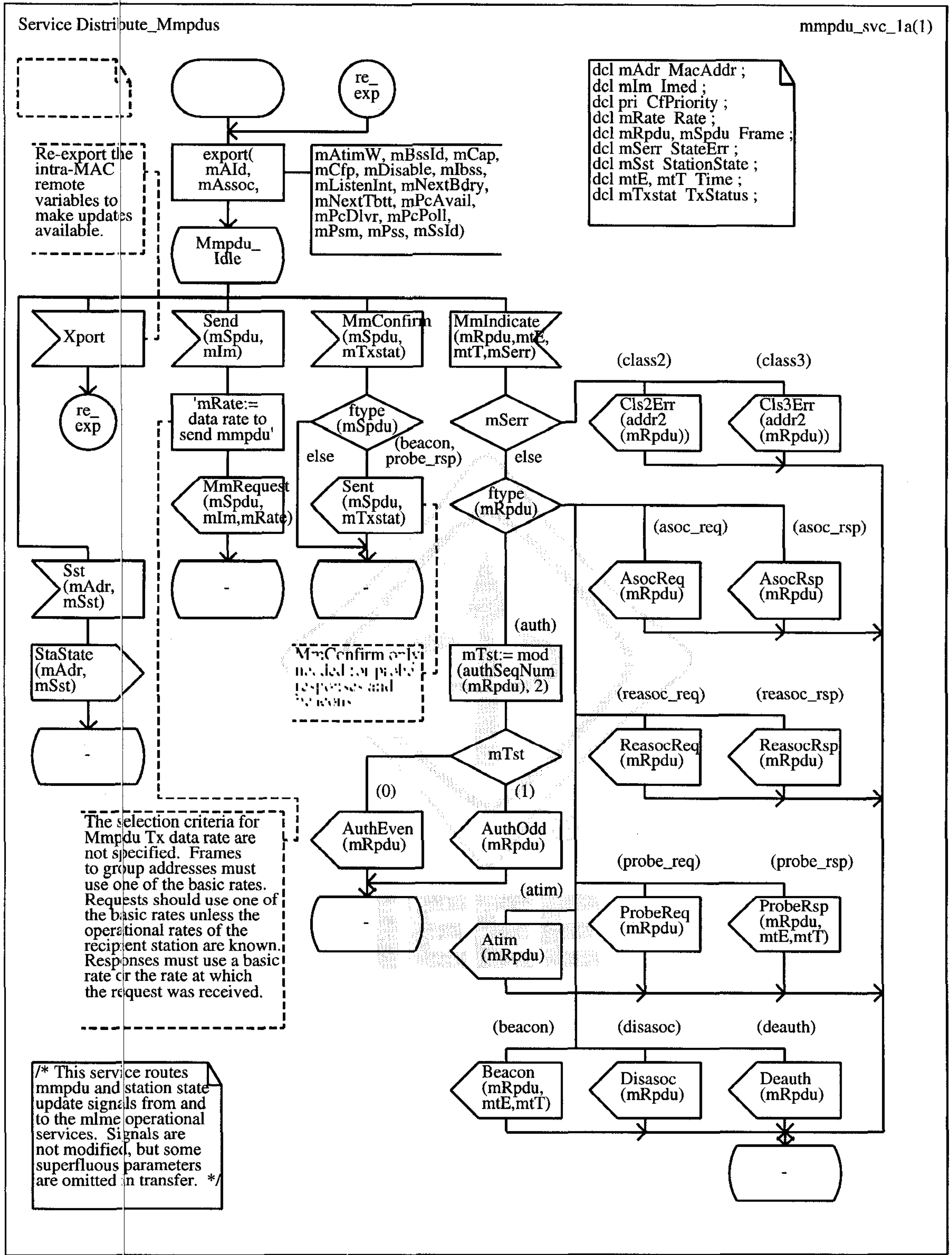

Service AsocService_Sta
sta_asoc_2a(2)
Asoc_Idle
On this page are associate request and reassociate request. More of this state on previous page.
old_ asoc
new_ asoc
Mlme_ Associate._ request
(asSta, asTmot, asCap)
MlmeRe_ associate._ request
(asSta, asTmot, asCap)
Sst(mBssid, dis_asoc)
Remove old association before saving data on new association.
asSdu:= mkFrame (asoc_req,
asSta, mBssid, (asCap // mkOs(mLis_ tenInt,2) // mkElem (eSsId,mSsid) // mkElem (eSupRates, mBrates)))
asSdu:= mkFrame (reasoc_req,
asSta, mBssid, (asCap // mkOS(mLis_ tenInt,2) // mBssid // mkElem (eSsId,mSsid) // mkElem (eSupRates, mBrates)))
Sst(asSta, asoc)
set(now + dTU(as_ Tmot),Tasoc)
set(now + dTU(as_ Tmot),Tasoc)
reset(Tasoc)
Send (asSdu, norm)
Send (asSdu, norm)
mCap:= CapA(asRdu)
Wait_Asoc_ _Response
Only accept response from request target.
if (mCap and cPollable)=cPollable then true else false fi
mPcPoll:=
*
AsocRsp (asRdu)
Tasoc
if (mCap and cPollReq)=cPollReq then true else false fi
mPcAvail:= mPcPoll or
(false)
-
asSta= addr2(asRdu)
MlmeAsso ciate.confirm (timeout)
mAId:= AId(asRdu)
(true)
status (asRdu)
asoc_ err
Wait_ Reasoc_ _Response
Only accept response from request target.
mOrates:= getElem(asRdu, eSupRates)
else
(successful)
MlmeAsso ciate.confirm (refused)
MlmeAsso ciate.confirm (successful)
*
Reasoc_ Rsp(asRdu)
Tasoc
mBssid:= addr2(asRdu)
asoc_ err
new_ asoc
(false)
-
asSta= addr2(asRdu)
MlmeReas sociate.con firm(timeout)
mAsoc:= true
(true)
asoc_ err
status (asRdu)
else
(successful)
MlmeReas sociate.con firm(refused)
MlmeReas sociate.con firm(successful)
Re-export intra-MAC variables.
Xport
asoc_ err
old_ asoc
Asoc_Idle

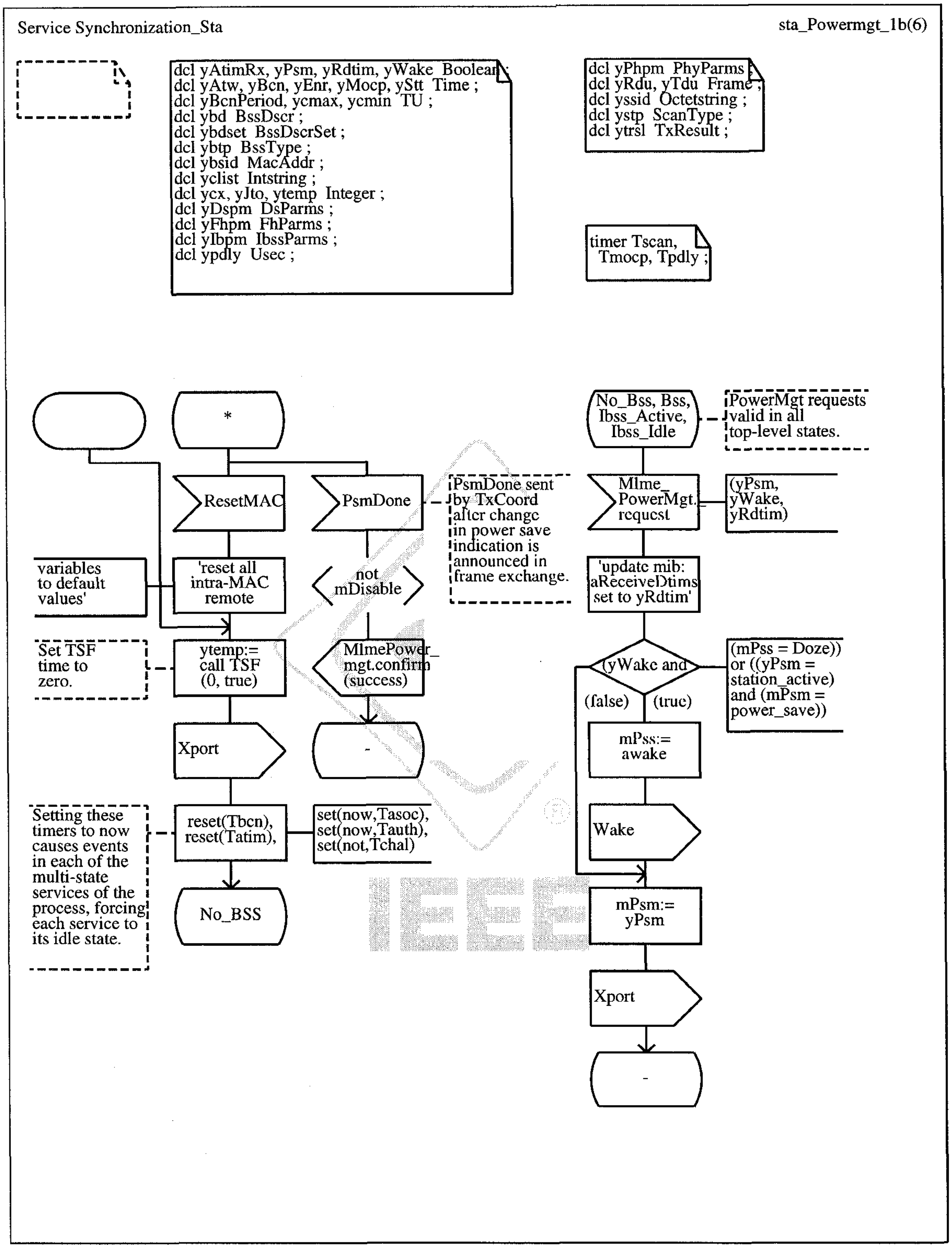

Service AuthReqService_Sta
auth_req_1a(2)
dcl auAlg AuthType ;
dcl auCap Capability ;
dcl auRdu, auSdu Frame ;
dcl auRsn ReasonCode ;
dcl auSta MacAddr ;
dcl auSts TxResult ;
dcl auTmot TU ;
/* This service handles (De)Authenticate requests. This service also handles incoming the generation of responses for class 2 errors.
This state machine handles Mlme requests sequentially, which is the simplest case. It is permissible to have several authentications in progress at once, provided the destination stations are all different. To support concurrent sequences this state machine gets collapsed into one state, with sequence state held in a variable. The local variables are replicated for each sequence, selected by responder address. */
Auth_Req_ _Idle
Authenticate Request is on this page, Deauthenticate and class 2 error on next page.
MlmeAuthenticate_request
(auSta, auAlg, auTmot)
(auAlg in
(false)
(true)
aAuthentication_Algorithms and (not isGroup(auSta))
auSdu:= mkFrame (auth, auSta,
mBssid, (auAlg // mkOS(1,2) // O2)
set(now + dTU(au_Tmot),Tauth)
Send (auSdu, norm)
Wait_Auth_ _Seq_2
Ignore auth frames from other stations.
MlmeAuthenticate.confirm(invalid)
-
Cannot authenticate using group address.
auth_cont
Copy challenge text from auth seq #2 frame.
auSdu:= mkFrame (auth, auSta,
mBssid, (auAlg // mkOS(3,2) // O2 // substr(auRdu, 31,128)))
auSdu:= setWepBit (mmpdu,1)
Mark shared key frame #3 for encryption.
Send (auSdu, norm)
Wait_Auth_ _Seq_4
auth_err
*
AuthEven (auRdu)
Tauth
auSta= addr2(auRdu)
(false)
(true)
else
auth_SeqNum (auRdu)
(2)
(4)
authStat (auRdu)
(succes_sful)
auAlg
(open_system)
(shared_key)
Sst(auSta, auth_open)
Sst(auSta, auth_key)
Sst(auSta, de_auth)
MlmeAuthenticate.confirm(timeout)
MlmeAuthenticate.confirm(refused)
auth_err
auth_cont
Auth_Req_ _Idle
Ignore response sequence errors, which may be from requests that timed out. Also, there is no status in odd to inform the sender.

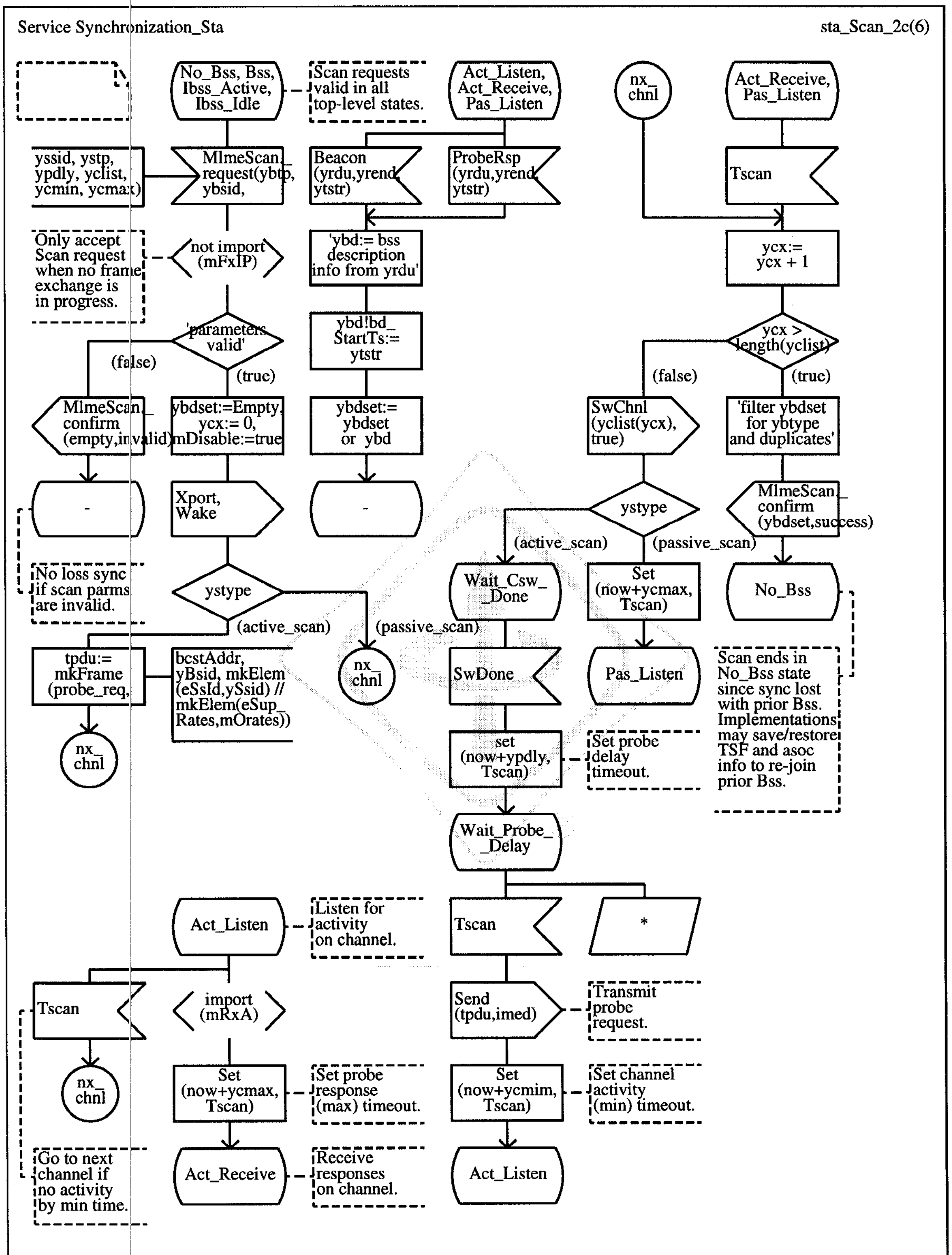

Service AuthReqService_Sta
deauth_2a(2)
Auth_Req_ Idle
Deauthenticate request and class 2 error are on this page. Authentication on previous page.
Cls2err (auSta)
MlmeDeauthenticate._request
(auSta, auRsn)
asRsn:= class2_err
auSdu:= mkFrame (deauth,
auSta, mBssid, auRsn)
Send (auSdu, norm)
Send notification, do not wait for delivery confirmation.
Sst(asSta, dc_auth)
Update local stations state records. Sending deauth also clears asoc state if present.
If deauthenticating the current AP, or deauthenticating everyone, end the association (if any) by clearing mBssid and mAssoc.
auSta= mBssId
or isGroup (auSta)
(false)
(true)
mAssoc:=false mBssid:= nullAddr
Xport
auRsn= class2_err
Don't confirm class 2 error notifications.
(true)
(false)
MlmeDis_associate._confirm
(successful)
-

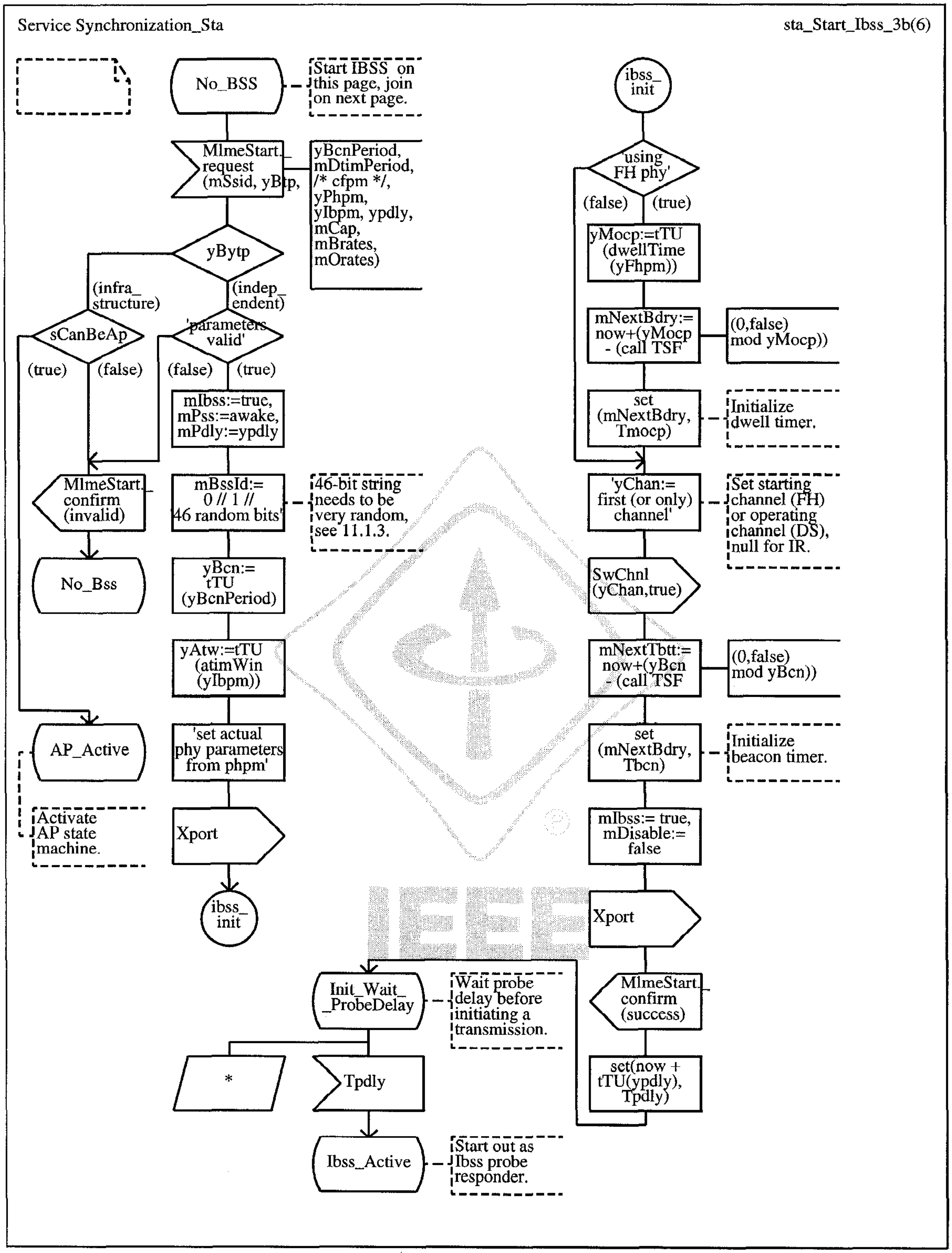

Service AuthRspService
auth_rsp_1a(2)
dcl arAlg, arAlg2 AuthType ;
dcl arRdu, auSdu Frame ;
dcl arRsn ReasonCode ;
dcl arSeq, arSeq2 Integer ;
dcl arSta, arSta2, arSta3 MacAddr ;
dcl arSC StatusCode ;
/* This service handles incoming Authentication & Deauthentication frames.
This state machine handles only a single shared key authentication challenge sequence at one time, which is the simplest case. It is possible to have several authentication responses in progress at once, provided the source stations are all different. To allow multiple responses this state machine gets collapsed into one state, with sequence state held in a variable. The local variables are replicated for each response, selected by requester station address. */
Auth_Rsp_ _Idle
Tchal
AuthOdd (arRdu)
/* Key to generate challenge text */
dcl chKey Octetstring ;
-
arSeq:= authSeqNum (arRdu),
arAlg:= authAlg (arRdu), arSta:= addr2 (arRdu)
/* The RC4 PRNG is accessed as a remote procedure:
prnString:= call RC4(key,length)
This procedure only present when aPrivacyOptionImplemented=true */
imported procedure RC4 ;
fpar PrngKey, Integer ;
returns Octetstring ;
arSeq
else
(1)
arSC:= auth_seq_ _fail
imported aAuth_ enticationResponse_ Timeout TU ;
bad_ alg
arAlg in
import (aAuthenti_ cationType)
(false)
(true)
arSC:= unsupt_alg
arAlg
(open_ _system)
(shared_ _key)
A station is allowed to reject an open system auth request with status unspec_fail.
arSC:= successful
aPrivacy_
Option_ Implemented
(true)
(false)
Sst(arSta, auth_open)
arChalng:= call RC4 (chKey, 128)
bad_ alg
The chKey value used to generate challenge text is arbitrary, and does not need to be shared. However, implementors are advised that the source of chKey SHOULD NOT be one of the WEP keys, because the output of the PRNG when using chKey is sent, unencrypted, in the challenge text field.
Sst(arSta, de_auth)
arSdu:= mkFrame (auth,arSta,
mBssid, (arAlg // mkOS(2,2) // successful // mkElem(eCtxt, arChalng)))
arSdu:= mkFrame (auth, arSta,
mBssid, (arAlg // mkOS (arSeq+1,2) // arSC))
Send (arSdu, norm)
Send (arSdu, norm)
set now+import
(aAuthentica_ tionRespone_ Timeout), Tchal)
Auth_Rsp_ _Idle
Wait_Chal_ _Rsp
Set response timeout and await response to challenge.

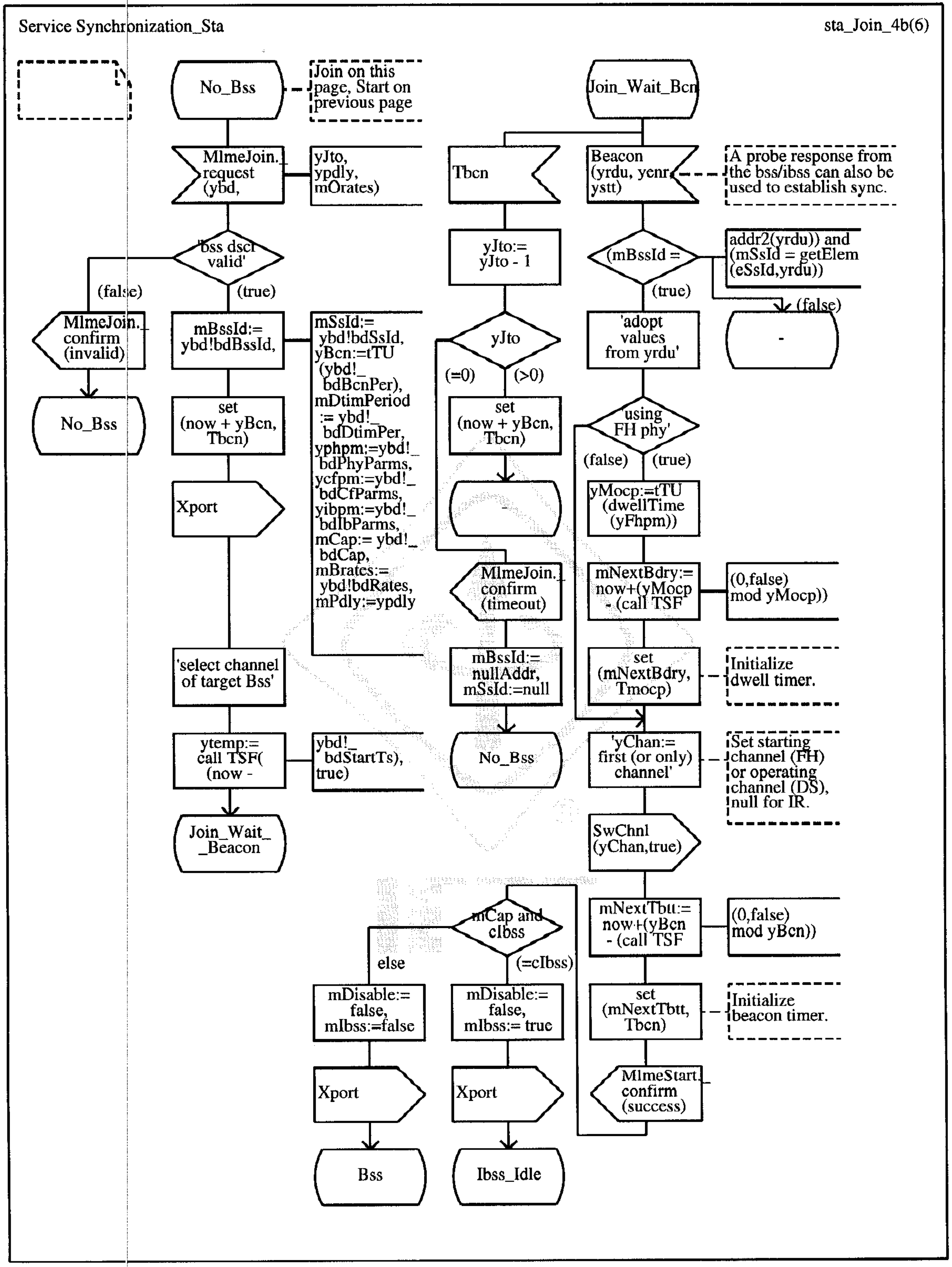

Service AuthRspService
auth_rsp_2a(2)
*
Deauth (arRdu)
arSta3:= addr2 (arRdu)
Sst(arSta3, de_auth)
Update station state, deauth clears asoc if present.
MlmeDeauthenticate_indication
(arSta3, reason (arRdu))
arSta3= mBssId
If deauth is from current AP, end asoc (if any) by clearing mBssid and mAssoc.
(false)
(true)
mAssoc:=false mBssid:= nullAddr
Xport
-
Wait_Chal_ _Rsp
Timeout while waiting is a failed attempt.
In the case of undecryptable response, assume Auth frame from expected source is sequence 3.
AuthOdd (arRdu)
Tchal
arSeq2:= authSeqNum (arRdu),
arSta2:= addr2 (arRdu)
Sst(arSta, de_auth)
Auth_Rsp_ _Idle
arSeq2
(3)
else
(1)
arSta = arSta2
(false)
(true)
Open_system request from a different station can be handled while awaiting challenge rsp.
reset (Tchal)
arAlg
(open_ _system)
wepBit (arRdu)
(0)
(1)
arAlg in
import (aAuthenticationType)
arSC:= unspec_fail
arSC:= unsupt_alg
arSC:= successful
getElem (eCtxt, arRdu)
arChalng=
Sst(arSta2, de_auth)
Sst(arSta, auth_open)
arSC:= chnlg_fail
arSC:= successful
arSdu:= mkFrame (auth,arSta2,
mBssid, (authAlg (arRdu)) // mkOS (arSeq2+1, 2) // arSC))
Sst(arSta2, de_auth)
Sst(arSta2, auth_key)
Send (arSdu, norm)
arSdu:= mkFrame (auth, arSta,
mBssid, (arAlg // mkOS(4,2) // arSC))
Wait_Chal_ _Rsp
A station is allowed to reject an open system auth request with status unspec_fail.
Continue to wait for response to challenge.
Send (arSdu, norm)
Auth_Rsp_ _Idle

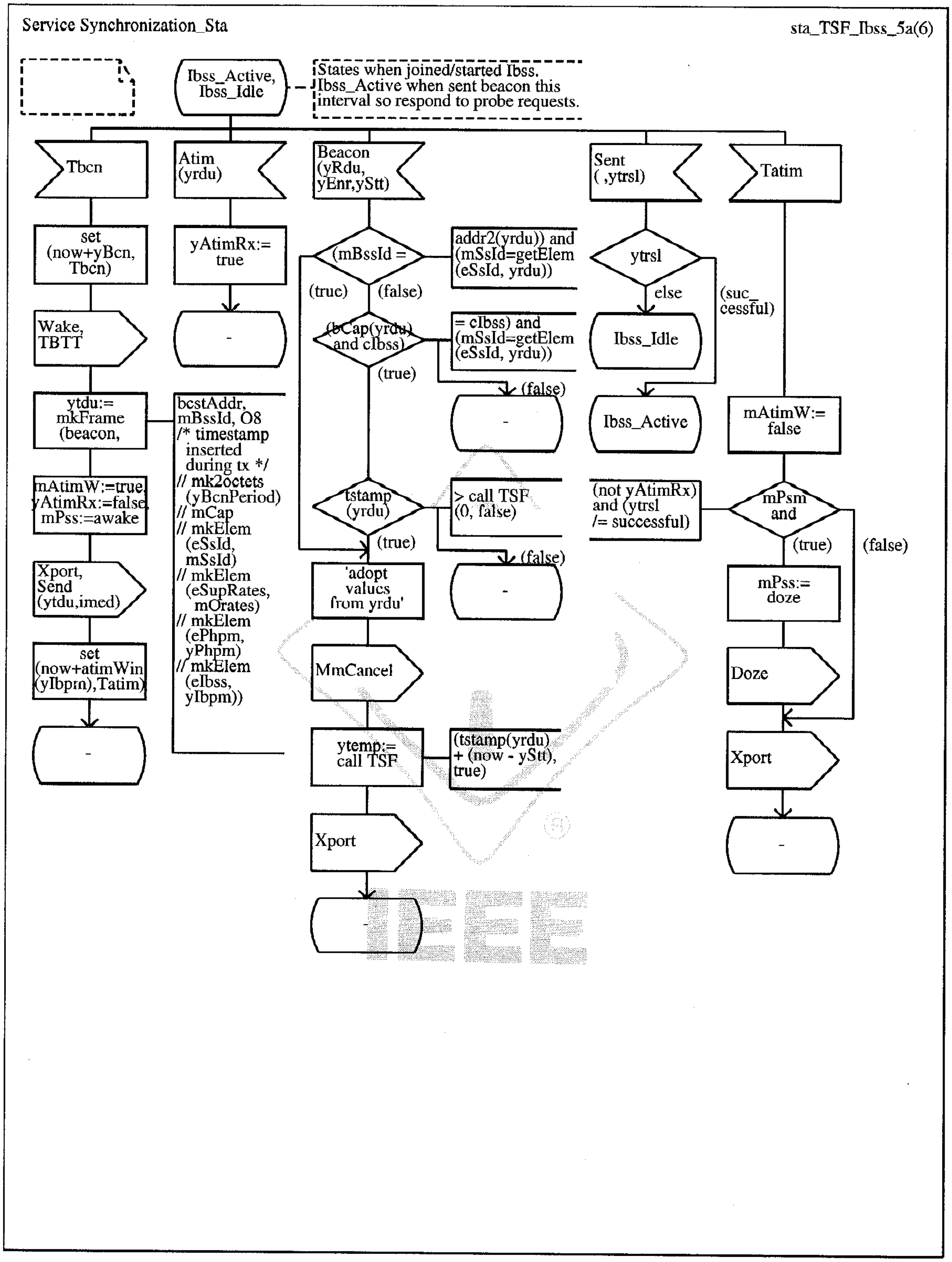

Service Distribute_Mmpdus
mmpdu_svc_1a(1)
re_ exp
dcl mAdr MacAddr ;
dcl mIm Imed ;
dcl pri CfPriority ;
dcl mRate Rate ;
dcl mRpdu, mSpdu Frame ;
dcl mSerr StateErr ;
dcl mSst StationState ;
dcl mtE, mtT Time ;
dcl mTxstat TxStatus ;
Re-export the intra-MAC remote variables to make updates available.
export(mAId, mAssoc,
mAtimW, mBssId, mCap, mCfp, mDisable, mIbss, mListenInt, mNextBdry, mNextTbtt, mPcAvail, mPcDlvr, mPcPoll, mPsm, mPss, mSsId)
Mmpdu_ Idle
Xport
Send (mSpdu, mIm)
MmConfirm (mSpdu, mTxstat)
MmIndicate (mRpdu,mtE, mtT,mSerr)
(class2)
(class3)
re_ exp
'mRate:= data rate to send mmpdu'
ftype (mSpdu)
(beacon, probe_rsp)
else
mSerr
else
Cls2Err (addr2 (mRpdu))
Cls3Err (addr2 (mRpdu))
MmRequest (mSpdu, mIm,mRate)
Sent (mSpdu, mTxstat)
ftype (mRpdu)
(asoc_req)
(asoc_rsp)
Sst (mAdr, mSst)
-
-
AsocReq (mRpdu)
AsocRsp (mRpdu)
(auth)
StaState (mAdr, mSst)
MmConfirm only needed for probe responses and beacons
mTst:= mod (authSeqNum (mRpdu), 2)
(reasoc_req)
(reasoc_rsp)
-
ReasocReq (mRpdu)
ReasocRsp (mRpdu)
mTst
(0)
(1)
The selection criteria for Mmpdu Tx data rate are not specified. Frames to group addresses must use one of the basic rates. Requests should use one of the basic rates unless the operational rates of the recipient station are known. Responses must use a basic rate or the rate at which the request was received.
AuthEven (mRpdu)
AuthOdd (mRpdu)
(probe_req)
(probe_rsp)
(atim)
-
Atim (mRpdu)
ProbeReq (mRpdu)
ProbeRsp (mRpdu, mtE,mtT)
(beacon)
(disasoc)
(deauth)
/* This service routes mmpdu and station state update signals from and to the mlme operational services. Signals are not modified, but some superfluous parameters are omitted in transfer. */
Beacon (mRpdu, mtE,mtT)
Disasoc (mRpdu)
Deauth (mRpdu)
-

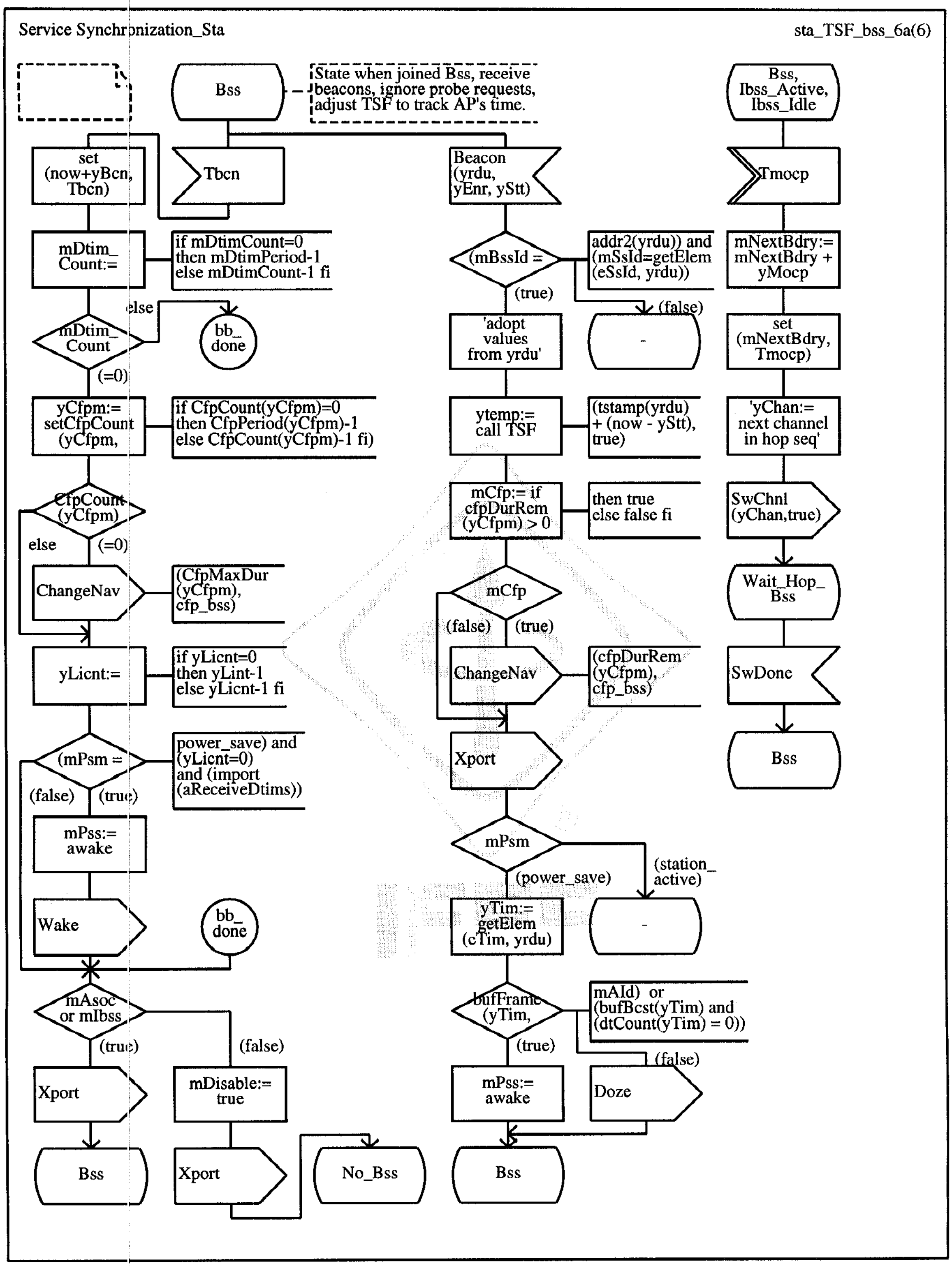

Service Synchronization_Sta
sta_Powermgt_1b(6)
dcl yAtimRx, yPsm, yRdtim, yWake Boolean;
dcl yAtw, yBcn, yEnr, yMocp, yStt Time ;
dcl yBcnPeriod, ycmax, ycmin TU ;
dcl ybd BssDscr ;
dcl ybdset BssDscrSet ;
dcl ybtp BssType ;
dcl ybsid MacAddr ;
dcl yclist Intstring ;
dcl ycx, yJto, ytemp Integer ;
dcl yDspm DsParms ;
dcl yFhpm FhParms ;
dcl yIbpm IbssParms ;
dcl ypdly Usec ;
dcl yPhpm PhyParms ;
dcl yRdu, yTdu Frame ;
dcl yssid Octetstring ;
dcl ystp ScanType ;
dcl ytrsl TxResult ;
timer Tscan, Tmocp, Tpdly ;
*
ResetMAC
'reset all intra-MAC remote
variables to default values'
ytemp:= call TSF (0, true)
Set TSF time to zero.
Xport
reset(Tbcn), reset(Tatim),
set(now,Tasoc), set(now,Tauth), set(not,Tchal)
Setting these timers to now causes events in each of the multi-state services of the process, forcing each service to its idle state.
No_BSS
PsmDone
PsmDone sent by TxCoord after change in power save indication is announced in frame exchange.
not mDisable
MlmePower_ mgt.confirm (success)
-
No_Bss, Bss, Ibss_Active, Ibss_Idle
PowerMgt requests valid in all top-level states.
Mlme_ PowerMgt. request
(yPsm, yWake, yRdtim)
'update mib: aReceiveDtims set to yRdtim'
yWake and
(mPss = Doze)) or ((yPsm = station_active) and (mPsm = power_save))
(false)
(true)
mPss:= awake
Wake
mPsm:= yPsm
Xport
-

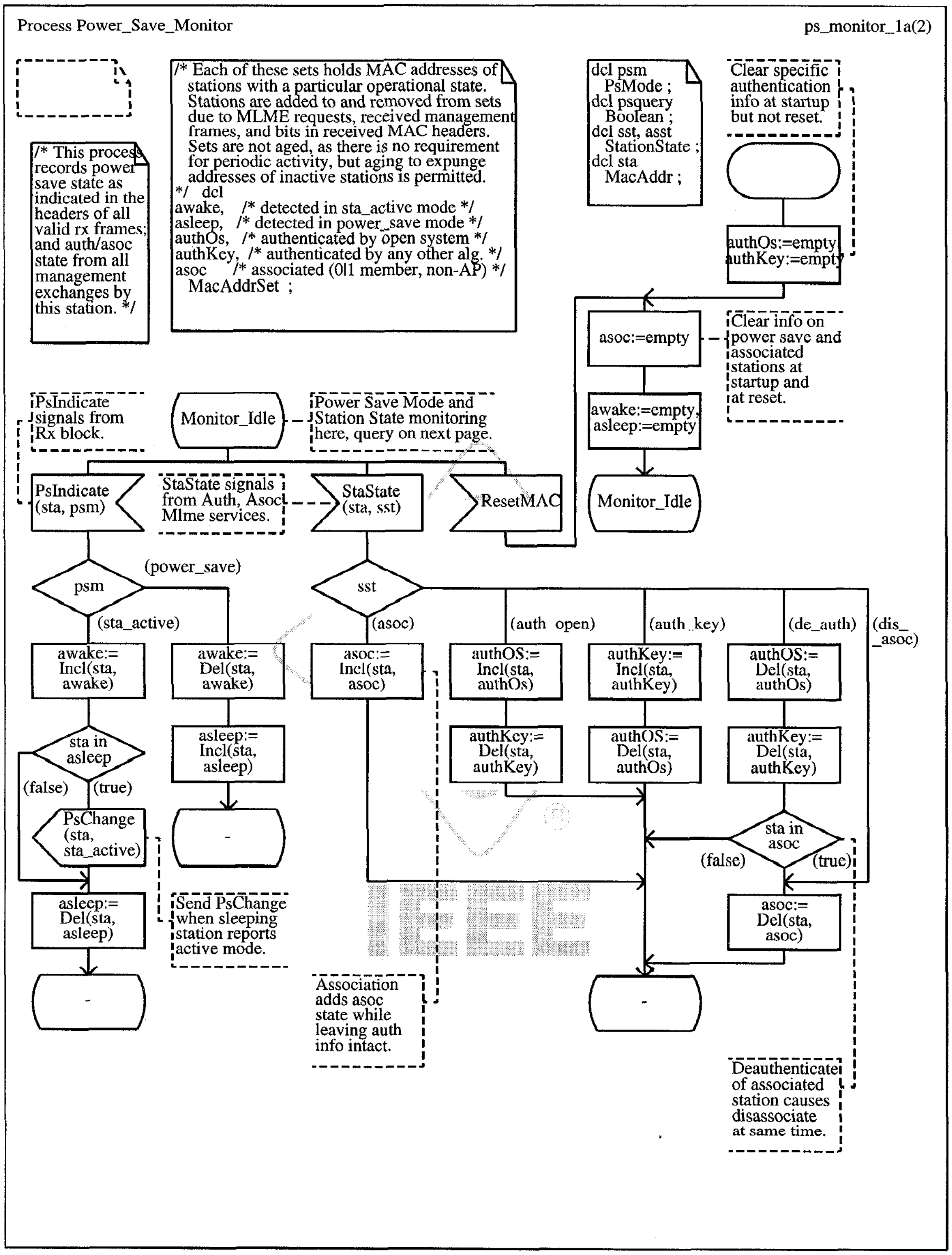

Service Synchronization_Sta
sta_Scan_2c(6)
No_Bss, Bss, Ibss_Active, Ibss_Idle
Scan requests valid in all top-level states.
Act_Listen, Act_Receive, Pas_Listen
nx_ chnl
Act_Receive, Pas_Listen
yssid, ystp, ypdly, yclist, ycmin, ycmax)
MlmeScan_ request(ybtp, ybsid,
Beacon (yrdu,yrend, ytstr)
ProbeRsp (yrdu,yrend, ytstr)
Tscan
Only accept Scan request when no frame exchange is in progress.
not import (mFxIP)
'ybd:= bss description info from yrdu'
ycx:= ycx + 1
'parameters valid'
(false)
(true)
ybd!bd_ StartTs:= ytstr
ycx > length(yclist)
MlmeScan_ confirm (empty,invalid)
ybdset:=Empty, ycx:= 0, mDisable:=true
ybdset:= ybdset or ybd
SwChnl (yclist(ycx), true)
'filter ybdset for ybtype and duplicates'
-
Xport, Wake
ystype
MlmeScan_ confirm (ybdset,success)
(active_scan)
(passive_scan)
No loss sync if scan parms are invalid.
Wait_Csw_ _Done
Set (now+ycmax, Tscan)
No_Bss
tpdu:= mkFrame (probe_req,
bcstAddr, yBsid, mkElem (eSsId,ySsid) // mkElem(eSup_ Rates,mOrates))
SwDone
Pas_Listen
Scan ends in No_Bss state since sync lost with prior Bss. Implementations may save/restore TSF and asoc info to re-join prior Bss.
set (now+ypdly, Tscan)
Set probe delay timeout.
Wait_Probe_ _Delay
Act_Listen
Listen for activity on channel.
*
import (mRxA)
Send (tpdu,imed)
Transmit probe request.
Set (now+ycmax, Tscan)
Set probe response (max) timeout.
Set (now+ycmim, Tscan)
Set channel activity (min) timeout.
Go to next channel if no activity by min time.
Act_Receive
Receive responses on channel.

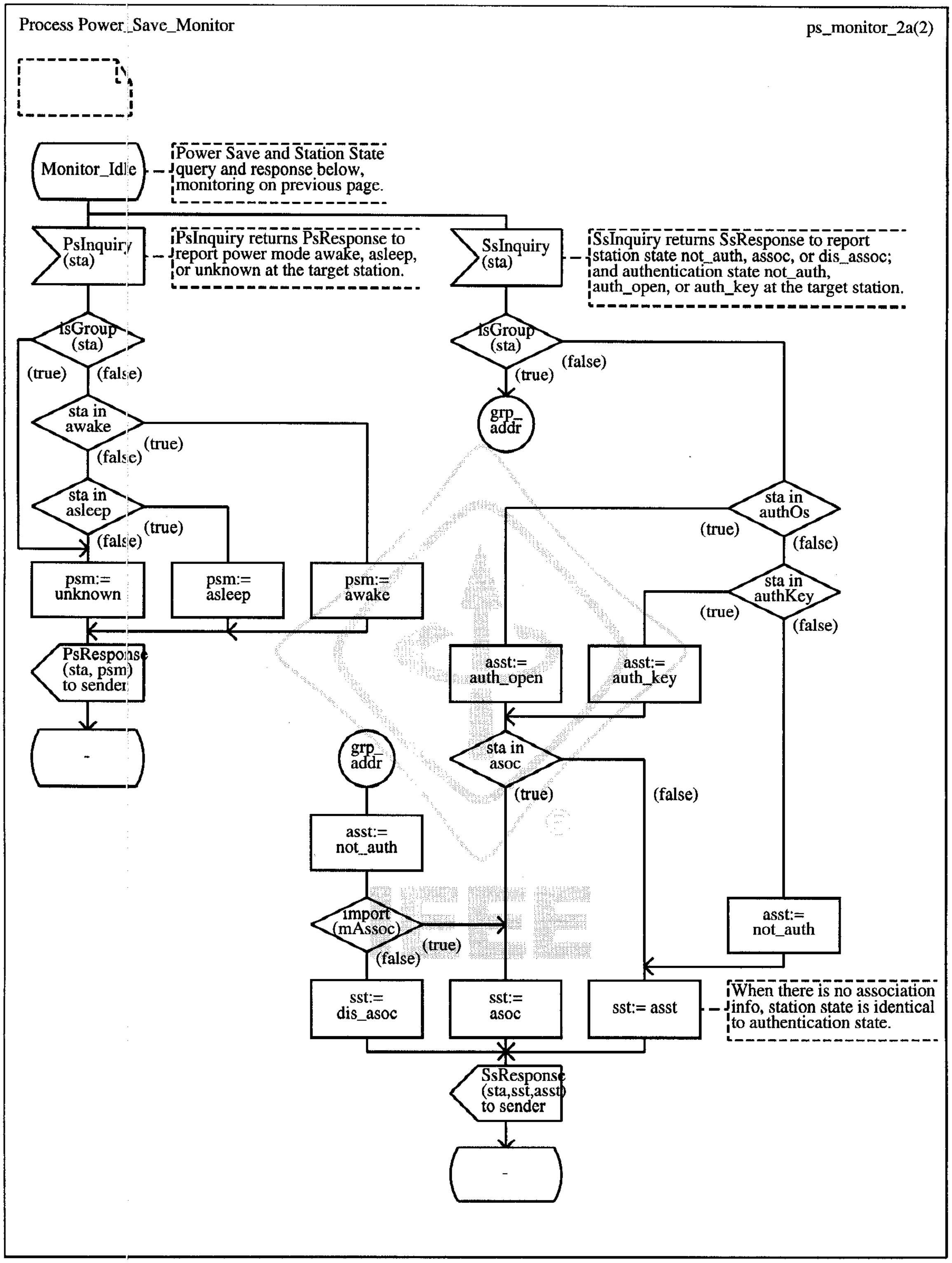

Service Synchronization_Sta
sta_Start_Ibss_3b(6)
No_BSS
Start IBSS on this page, join on next page.
MlmeStart_ request (mSsid, yBtp,
yBcnPeriod, mDtimPeriod, /* cfpm */, yPhpm, yIbpm, ypdly, mCap, mBrates, mOrates)
yBytp
(infra_ structure)
(indep_ endent)
sCanBeAp
'parameters valid'
(true)
(false)
(false)
(true)
mIbss:=true, mPss:=awake, mPdly:=ypdly
MlmeStart_ confirm (invalid)
mBssId:= 0 // 1 // 46 random bits'
46-bit string needs to be very random, see 11.1.3.
No_Bss
yBcn:= tTU (yBcnPeriod)
yAtw:=tTU (atimWin (yIbpm))
AP_Active
'set actual phy parameters from phpm'
Activate AP state machine.
Xport
ibss_ init
ibss_ init
'using FH phy'
(false)
(true)
yMocp:=tTU (dwellTime (yFhpm))
mNextBdry:= now+(yMocp - (call TSF
(0,false) mod yMocp))
set (mNextBdry, Tmocp)
Initialize dwell timer.
'yChan:= first (or only) channel'
Set starting channel (FH) or operating channel (DS), null for IR.
SwChnl (yChan,true)
mNextTbtt:= now+(yBcn - (call TSF
(0,false) mod yBcn))
set (mNextBdry, Tbcn)
Initialize beacon timer.
mIbss:= true, mDisable:= false
Xport
MlmeStart_ confirm (success)
set(now + tTU(ypdly), Tpdly)
Init_Wait_ _ProbeDelay
Wait probe delay before initiating a transmission.
*
Tpdly
Ibss_Active
Start out as Ibss probe responder.

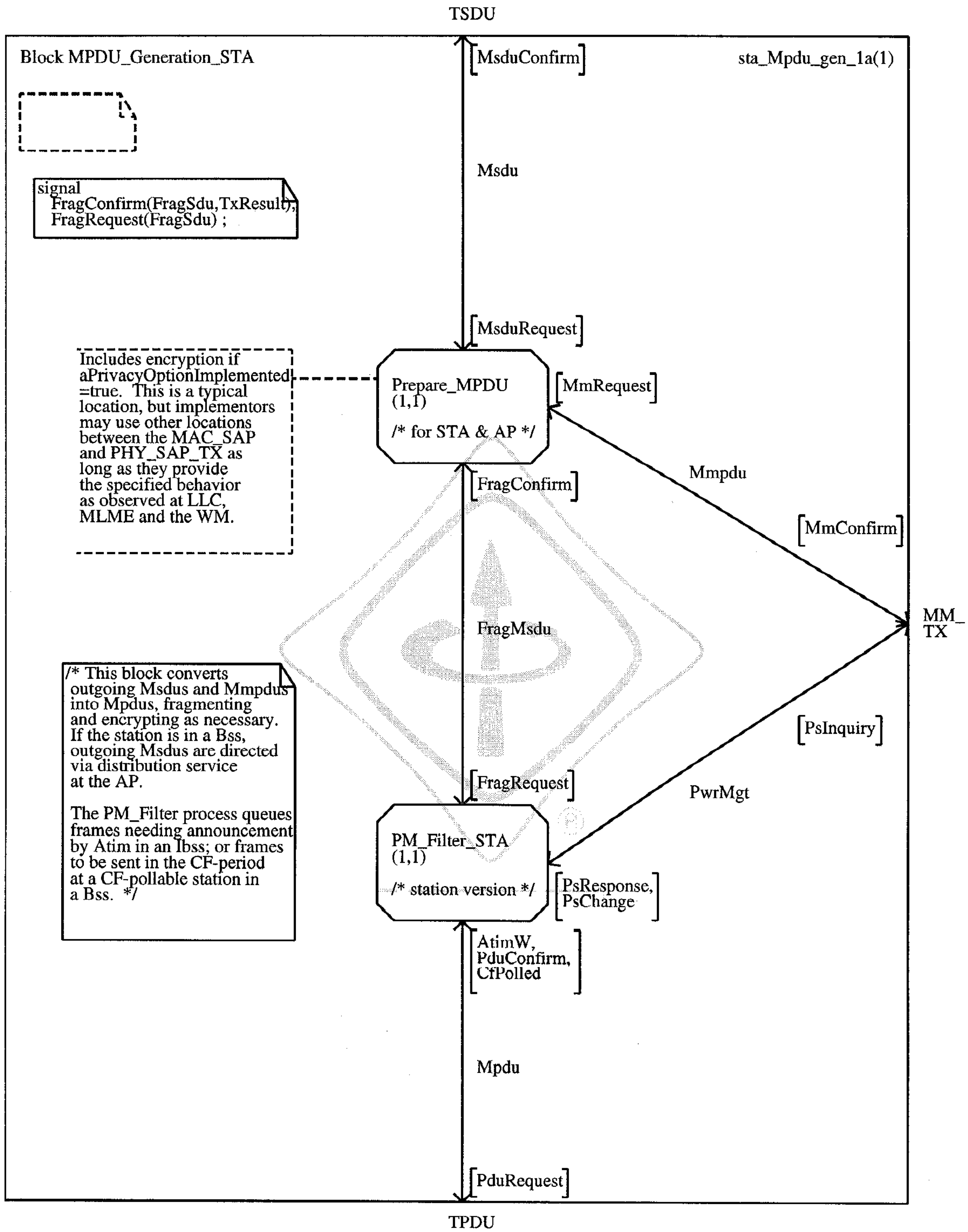

Service Synchronization_Sta
sta_Join_4b(6)
No_Bss
Join on this page, Start on previous page
MlmeJoin._ request (ybd,
yJto, ypdly, mOrates)
'bss dscr valid'
(false)
(true)
MlmeJoin._ confirm (invalid)
No_Bss
mBssId:= ybd!bdBssId,
mSsId:= ybd!bdSsId, yBcn:=tTU (ybd!_ bdBcnPer), mDtimPeriod := ybd!_ bdDtimPer, yphpm:=ybd!_ bdPhyParms, ycfpm:=ybd!_ bdCfParms, yibpm:=ybd!_ bdIbParms, mCap:= ybd!_ bdCap, mBrates:= ybd!bdRates, mPdly:=ypdly
set (now + yBcn, Tbcn)
Xport
'select channel of target Bss'
ytemp:= call TSF((now -
ybd!_ bdStartTs), true)
Join_Wait_ _Beacon
Join_Wait_Bcn
Tbcn
yJto:= yJto - 1
yJto
(=0)
(>0)
set (now + yBcn, Tbcn)
-
MlmeJoin._ confirm (timeout)
mBssId:= nullAddr, mSsId:=null
No_Bss
Beacon (yrdu, yenr, ystt)
A probe response from the bss/ibss can also be used to establish sync.
(mBssId =
addr2(yrdu)) and (mSsId = getElem (eSsId,yrdu))
(true)
(false)
-
'adopt values from yrdu'
'using FH phy'
(false)
(true)
yMocp:=tTU (dwellTime (yFhpm))
mNextBdry:= now+(yMocp - (call TSF
(0,false) mod yMocp))
set (mNextBdry, Tmocp)
Initialize dwell timer.
'yChan:= first (or only) channel'
Set starting channel (FH) or operating channel (DS), null for IR.
SwChnl (yChan,true)
mNextTbtt:= now+(yBcn - (call TSF
(0,false) mod yBcn))
set (mNextTbtt, Tbcn)
Initialize beacon timer.
MlmeStart._ confirm (success)
mCap and cIbss
else
(=cIbss)
mDisable:= false, mIbss:=false
mDisable:= false, mIbss:= true
Xport
Xport
Bss
Ibss_Idle

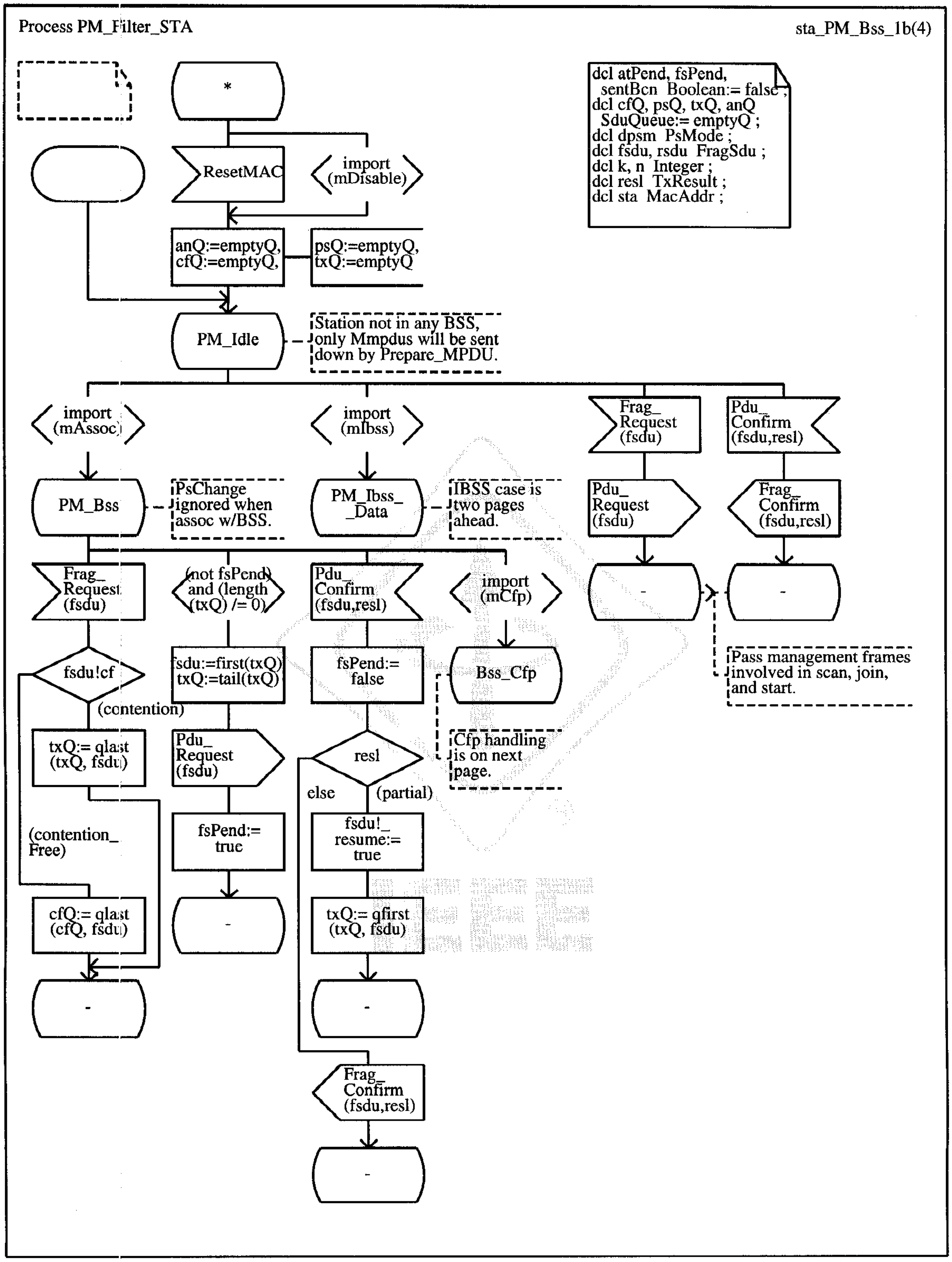

Service Synchronization_Sta
sta_TSF_Ibss_5a(6)
Ibss_Active, Ibss_Idle
States when joined/started Ibss. Ibss_Active when sent beacon this interval so respond to probe requests.
Tbcn
Atim (yrdu)
Beacon (yRdu, yEnr,ySTt)
Sent (,ytrsl)
Tatim
set (now+yBcn, Tbcn)
yAtimRx:= true
(mBssId =
addr2(yrdu)) and (mSsId=getElem (eSsId, yrdu))
(true)
(false)
ytrsl
else
(suc_ cessful)
Wake, TBTT
-
(bCap(yrdu) and cIbss)
= cIbss) and (mSsId=getElem (eSsId, yrdu))
(true)
(false)
Ibss_Idle
Ibss_Active
ytdu:= mkFrame (beacon,
bcstAddr, mBssId, O8 /* timestamp inserted during tx */ // mk2octets (yBcnPeriod) // mCap // mkElem (eSsId, mSsId) // mkElem (eSupRates, mOrates) // mkElem (ePhpm, yPhpm) // mkElem (eIbss, yIbpm))
mAtimW:= false
mAtimW:=true, yAtimRx:=false, mPss:=awake
tstamp (yrdu)
> call TSF (0, false)
(not yAtimRx) and (ytrsl /= successful)
mPsm and
(true)
(false)
Xport, Send (ytdu,imed)
'adopt values from yrdu'
mPss:= doze
set (now+atimWin (yIbpm),Tatim)
MmCancel
Doze
ytemp:= call TSF
(tstamp(yrdu) + (now - ySTt), true)
Xport
Xport

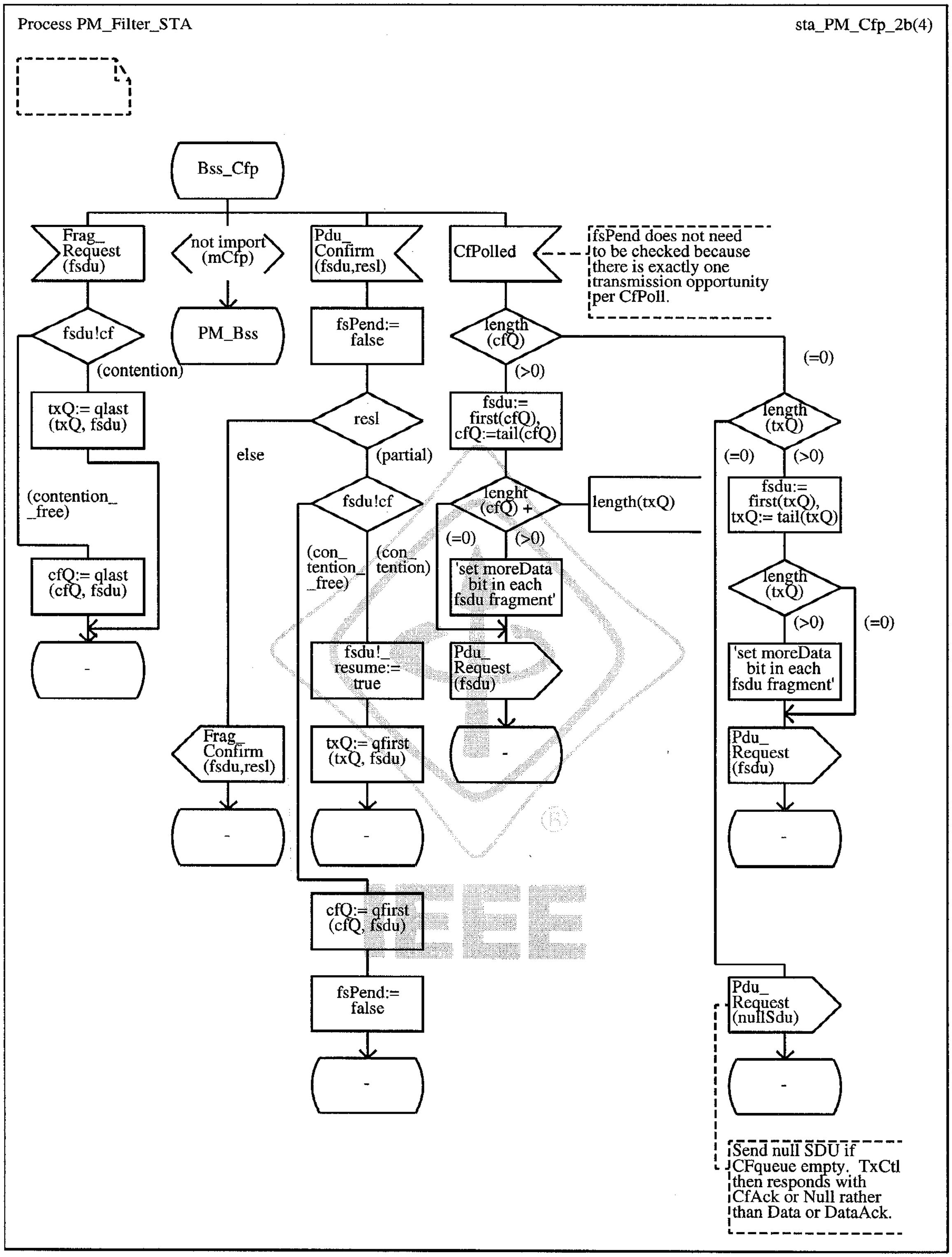

Service Synchronization_Sta
sta_TSF_bss_6a(6)
Bss
State when joined Bss, receive beacons, ignore probe requests, adjust TSF to track AP's time.
Bss, Ibss_Active, Ibss_Idle
set (now+yBcn, Tbcn)
Tbcn
Beacon (yrdu, yEnr, yStt)
Tmocp
mDtim_ Count:=
if mDtimCount=0 then mDtimPeriod-1 else mDtimCount-1 fi
(mBssId =
addr2(yrdu)) and (mSsId=getElem (eSsId, yrdu))
mNextBdry:= mNextBdry + yMocp
else
mDtim_ Count
bb_ done
(=0)
(true)
(false)
'adopt values from yrdu'
-
set (mNextBdry, Tmocp)
yCfpm:= setCfpCount (yCfpm,
if CfpCount(yCfpm)=0 then CfpPeriod(yCfpm)-1 else CfpCount(yCfpm)-1 fi)
ytemp:= call TSF
(tstamp(yrdu) + (now - yStt), true)
'yChan:= next channel in hop seq'
CfpCount (yCfpm)
else
(=0)
mCfp:= if cfpDurRem (yCfpm) > 0
then true else false fi
SwChnl (yChan,true)
ChangeNav
(CfpMaxDur (yCfpm), cfp_bss)
mCfp
(false)
(true)
Wait_Hop_ Bss
yLicnt:=
if yLicnt=0 then yLint-1 else yLicnt-1 fi
ChangeNav
(cfpDurRem (yCfpm), cfp_bss)
SwDone
(mPsm =
power_save) and (yLicnt=0) and (import (aReceiveDtims))
Xport
Bss
(false)
(true)
mPss:= awake
mPsm
(power_save)
(station_ active)
Wake
bb_ done
yTim:= getElem (cTim, yrdu)
-
mAsoc or mIbss
bufFrame (yTim,
mAId) or (bufBcst(yTim) and (dtCount(yTim) = 0))
(true)
(false)
(true)
(false)
Xport
mDisable:= true
mPss:= awake
Doze
Bss
Xport
No_Bss
Bss

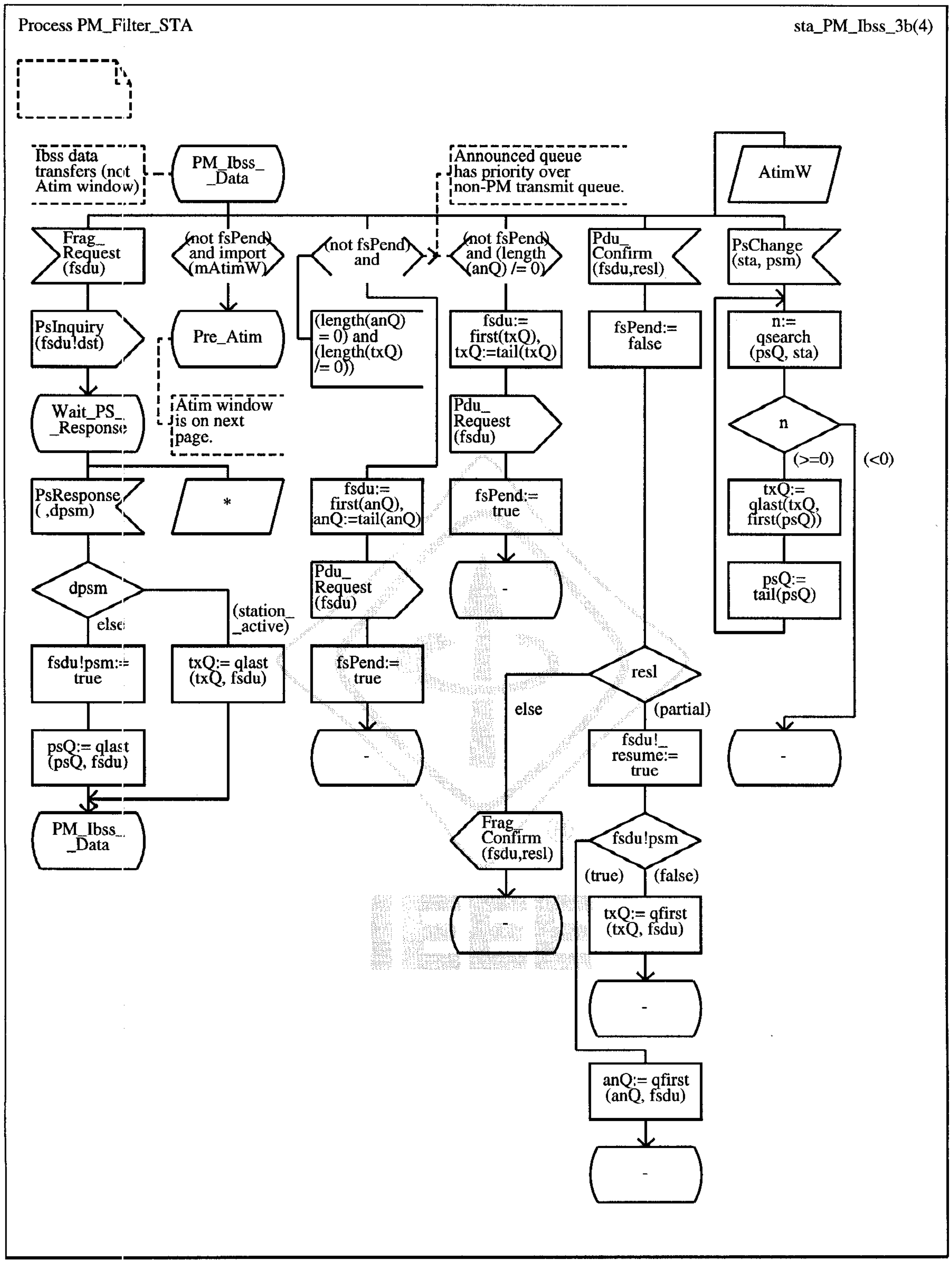

Process Power_Save_Monitor
ps_monitor_1a(2)
/* This process records power save state as indicated in the headers of all valid rx frames; and auth/asoc state from all management exchanges by this station. */
/* Each of these sets holds MAC addresses of stations with a particular operational state. Stations are added to and removed from sets due to MLME requests, received management frames, and bits in received MAC headers. Sets are not aged, as there is no requirement for periodic activity, but aging to expunge addresses of inactive stations is permitted.
*/ dcl
awake, /* detected in sta_active mode */
asleep, /* detected in power_save mode */
authOs, /* authenticated by open system */
authKey, /* authenticated by any other alg. */
asoc /* associated (0|1 member, non-AP) */
MacAddrSet ;
dcl psm PsMode ;
dcl psquery Boolean ;
dcl sst, asst StationState ;
dcl sta MacAddr ;
Clear specific authentication info at startup but not reset.
authOs:=empty
authKey:=empty
asoc:=empty
Clear info on power save and associated stations at startup and at reset.
awake:=empty, asleep:=empty
Monitor_Idle
PsIndicate signals from Rx block.
Power Save Mode and Station State monitoring here, query on next page.
PsIndicate (sta, psm)
StaState signals from Auth, Asoc Mlme services.
StaState (sta, sst)
ResetMAC
psm
(power_save)
(sta_active)
awake:= Incl(sta, awake)
awake:= Del(sta, awake)
sta in asleep
(false)
(true)
PsChange (sta, sta_active)
asleep:= Incl(sta, asleep)
asleep:= Del(sta, asleep)
Send PsChange when sleeping station reports active mode.
sst
(asoc)
(auth_open)
(auth_key)
(de_auth)
(dis_ _asoc)
asoc:= Incl(sta, asoc)
authOS:= Incl(sta, authOs)
authKey:= Incl(sta, authKey)
authOS:= Del(sta, authOs)
authKey:= Del(sta, authKey)
authOS:= Del(sta, authOs)
authKey:= Del(sta, authKey)
sta in asoc
(false)
(true)
asoc:= Del(sta, asoc)
Association adds asoc state while leaving auth info intact.
Deauthenticate of associated station causes disassociate at same time.

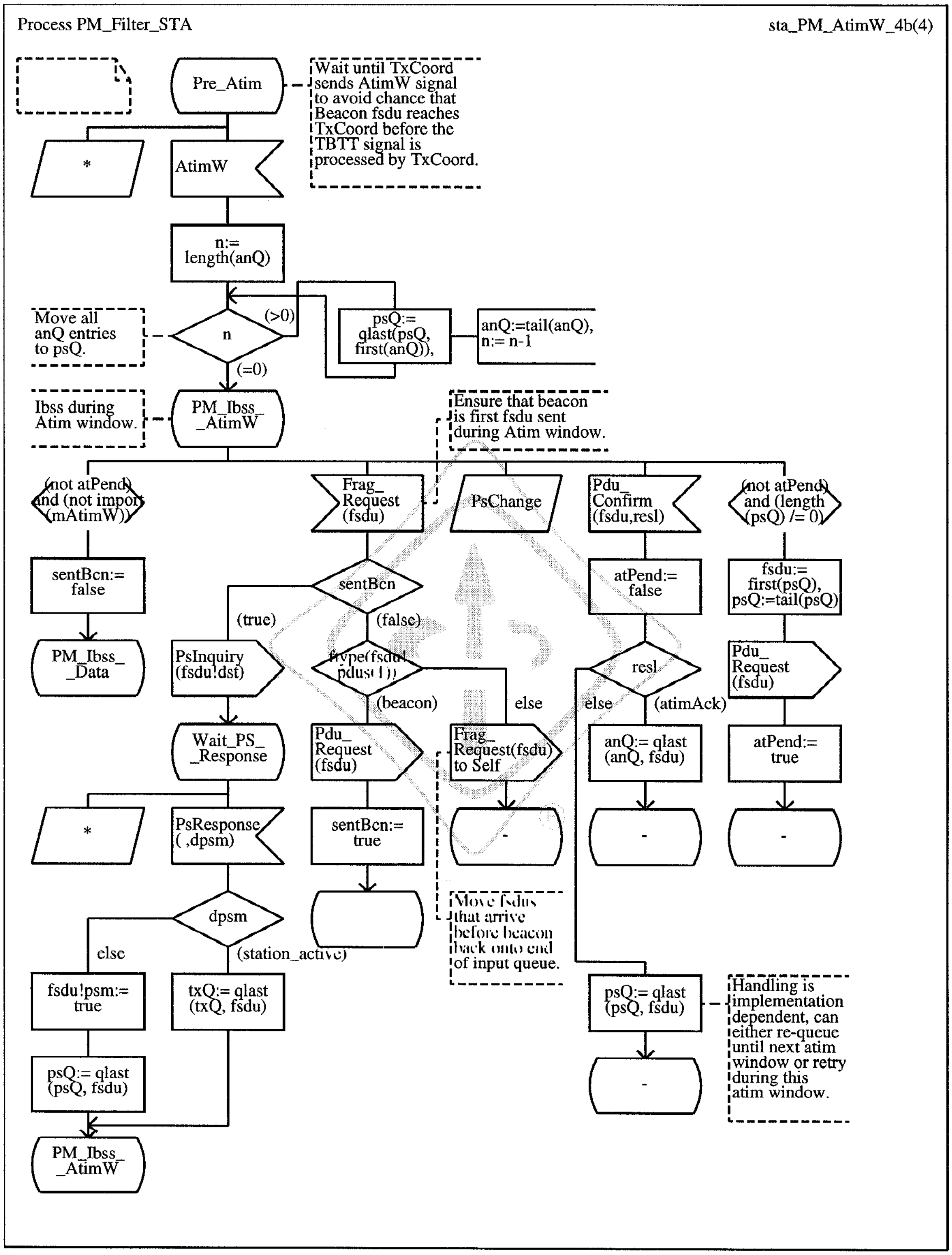

Process Power_Save_Monitor
ps_monitor_2a(2)
Monitor_Idle
Power Save and Station State query and response below, monitoring on previous page.
PsInquiry (sta)
PsInquiry returns PsResponse to report power mode awake, asleep, or unknown at the target station.
SsInquiry (sta)
SsInquiry returns SsResponse to report station state not_auth, assoc, or dis_assoc; and authentication state not_auth, auth_open, or auth_key at the target station.
IsGroup (sta)
(true)
(false)
sta in awake
sta in asleep
psm:= unknown
psm:= asleep
psm:= awake
PsResponse (sta, psm) to sender
-
grp_ addr
sta in authOs
sta in authKey
asst:= auth_open
asst:= auth_key
sta in asoc
asst:= not_auth
import (mAssoc)
sst:= dis_asoc
sst:= asoc
sst:= asst
When there is no association info, station state is identical to authentication state.
SsResponse (sta,sst,asst) to sender

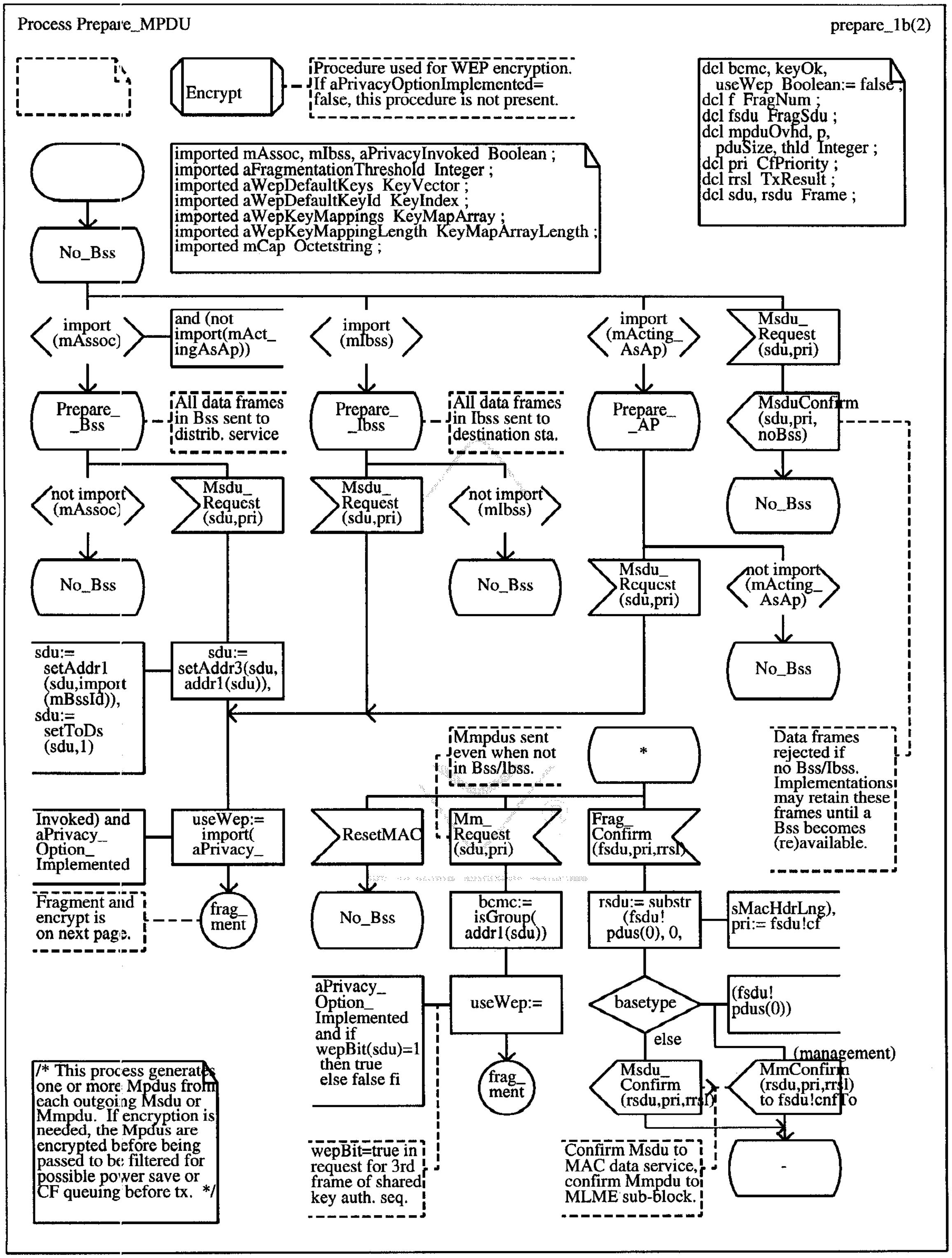

TSDU
Block MPDU_Generation_STA
sta_Mpdu_gen_1a(1)
MsduConfirm
Msdu
signal
FragConfirm(FragSdu,TxResult),
FragRequest(FragSdu) ;
MsduRequest
Includes encryption if aPrivacyOptionImplemented =true. This is a typical location, but implementors may use other locations between the MAC_SAP and PHY_SAP_TX as long as they provide the specified behavior as observed at LLC, MLME and the WM.
Prepare_MPDU
(1,1)
/* for STA & AP */
MmRequest
Mmpdu
FragConfirm
MmConfirm
FragMsdu
MM_
TX
/* This block converts outgoing Msdus and Mmpdus into Mpdus, fragmenting and encrypting as necessary. If the station is in a Bss, outgoing Msdus are directed via distribution service at the AP.
The PM_Filter process queues frames needing announcement by Atim in an Ibss; or frames to be sent in the CF-period at a CF-pollable station in a Bss. */
PsInquiry
FragRequest
PwrMgt
PM_Filter_STA
(1,1)
/* station version */
PsResponse,
PsChange
AtimW,
PduConfirm,
CfPolled
Mpdu
PduRequest
TPDU

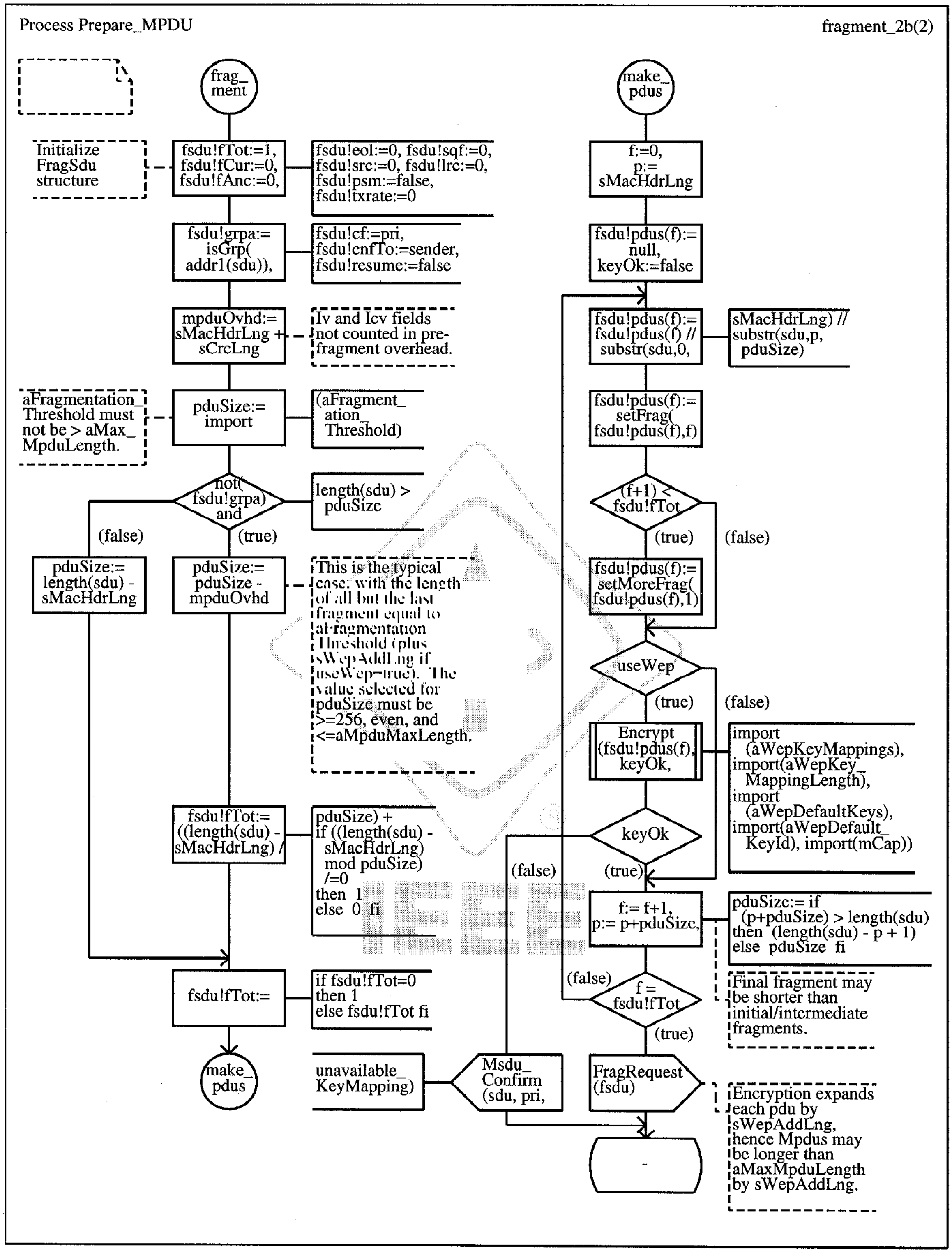

Process PM_Filter_STA
sta_PM_Bss_1b(4)
dcl atPend, fsPend, sentBcn Boolean:= false ;
dcl cfQ, psQ, txQ, anQ SduQueue:= emptyQ ;
dcl dpsm PsMode ;
dcl fsdu, rsdu FragSdu ;
dcl k, n Integer ;
dcl resl TxResult ;
dcl sta MacAddr ;
*
ResetMAC
import (mDisable)
anQ:=emptyQ, cfQ:=emptyQ,
psQ:=emptyQ, txQ:=emptyQ
PM_Idle
Station not in any BSS, only Mmpdus will be sent down by Prepare_MPDU.
import (mAssoc)
import (mIbss)
Frag_ Request (fsdu)
Pdu_ Confirm (fsdu,resl)
PM_Bss
PsChange ignored when assoc w/BSS.
PM_Ibss_ _Data
IBSS case is two pages ahead.
Pdu_ Request (fsdu)
Frag_ Confirm (fsdu,resl)
-
Pass management frames involved in scan, join, and start.
Frag_ Request (fsdu)
(not fsPend) and (length (txQ) /= 0)
Pdu_ Confirm (fsdu,resl)
import (mCfp)
fsdu!cf
(contention)
fsdu:=first(txQ) txQ:=tail(txQ)
fsPend:= false
Bss_Cfp
Cfp handling is on next page.
txQ:= qlast (txQ, fsdu)
Pdu_ Request (fsdu)
resl
else
(partial)
(contention_ Free)
fsPend:= true
fsdu!_ resume:= true
cfQ:= qlast (cfQ, fsdu)
txQ:= qfirst (txQ, fsdu)
Frag_ Confirm (fsdu,resl)

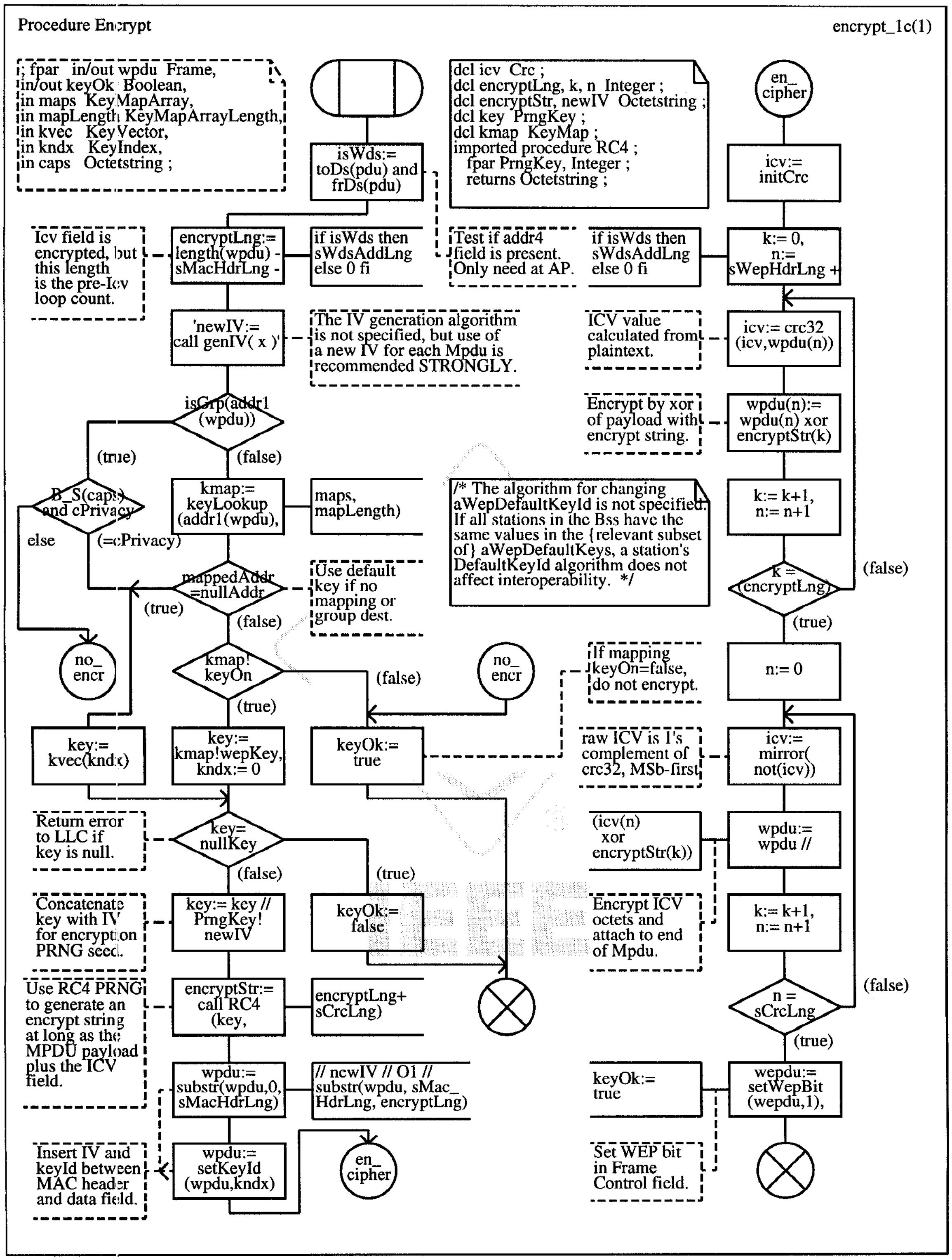

Process PM_Filter_STA
sta_PM_Cfp_2b(4)
Bss_Cfp
Frag_ Request (fsdu)
not import (mCfp)
Pdu_ Confirm (fsdu,resl)
CfPolled
fsPend does not need to be checked because there is exactly one transmission opportunity per CfPoll.
fsdu!cf
PM_Bss
fsPend:= false
length (cfQ)
(contention)
(=0)
(>0)
txQ:= qlast (txQ, fsdu)
resl
fsdu:= first(cfQ), cfQ:=tail(cfQ)
length (txQ)
else
(partial)
(=0)
(>0)
(contention_ _free)
fsdu!cf
lenght (cfQ) +
length(txQ)
fsdu:= first(txQ), txQ:= tail(txQ)
(con_ tention_ _free)
(con_ tention)
(=0)
(>0)
cfQ:= qlast (cfQ, fsdu)
'set moreData bit in each fsdu fragment'
length (txQ)
(>0)
(=0)
-
fsdu!_ resume:= true
Pdu_ Request (fsdu)
'set moreData bit in each fsdu fragment'
Frag_ Confirm (fsdu,resl)
txQ:= qfirst (txQ, fsdu)
-
Pdu_ Request (fsdu)
-
-
-
cfQ:= qfirst (cfQ, fsdu)
fsPend:= false
Pdu_ Request (nullSdu)
-
-
Send null SDU if CFqueue empty. TxCtl then responds with CfAck or Null rather than Data or DataAck.

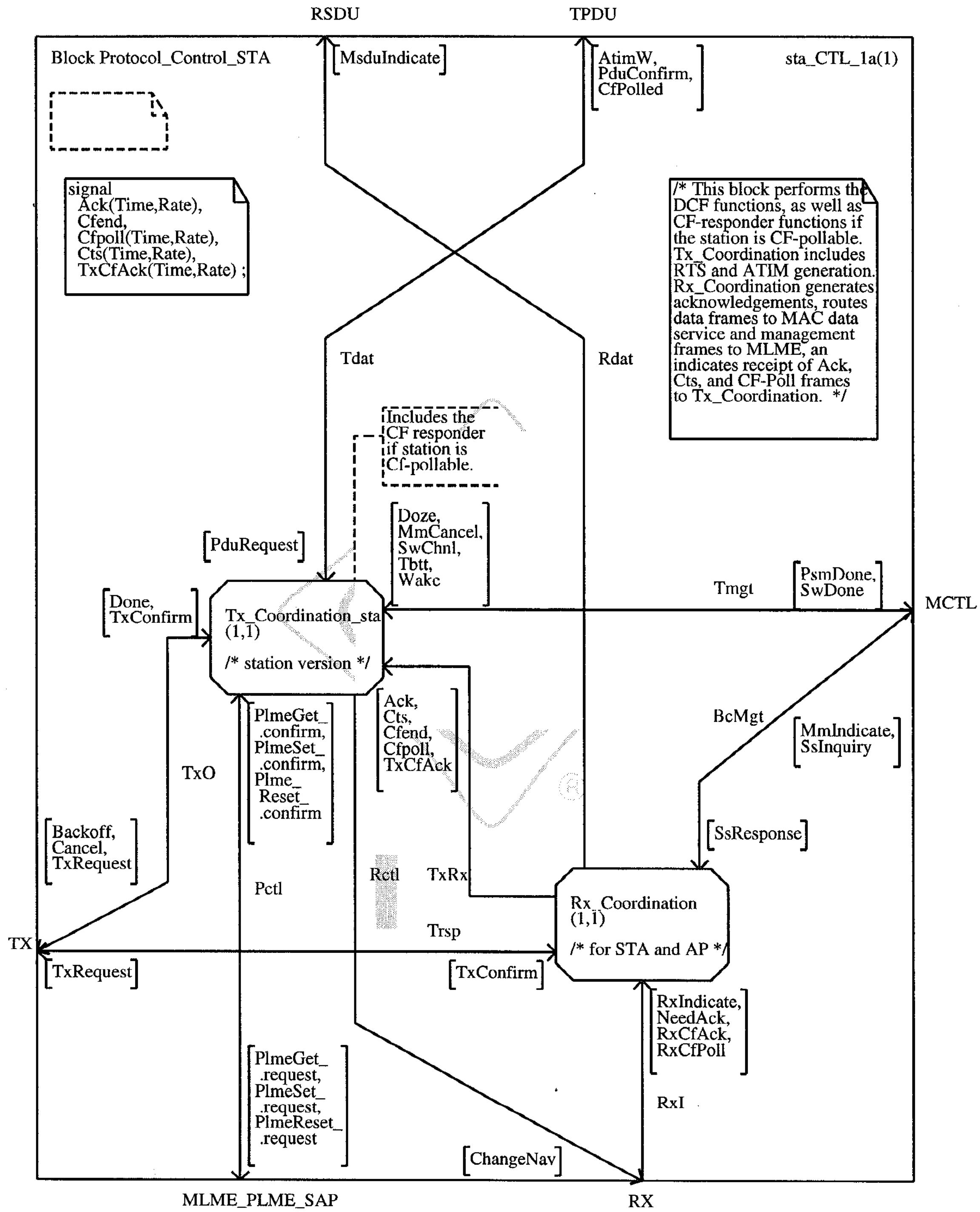

Process PM_Filter_STA
sta_PM_Ibss_3b(4)
Ibss data transfers (not Atim window)
PM_Ibss_ _Data
Announced queue has priority over non-PM transmit queue.
AtimW
Frag_ Request (fsdu)
(not fsPend) and import (mAtimW)
(not fsPend) and
(not fsPend) and (length (anQ) /= 0)
Pdu_ Confirm (fsdu,resl)
PsChange (sta, psm)
PsInquiry (fsdu!dst)
Pre_Atim
(length(anQ) = 0) and (length(txQ) /= 0))
fsdu:= first(txQ), txQ:=tail(txQ)
fsPend:= false
n:= qsearch (psQ, sta)
Wait_PS_ _Response
Atim window is on next page.
Pdu_ Request (fsdu)
n
(>=0)
(<0)
PsResponse (,dpsm)
*
fsdu:= first(anQ), anQ:=tail(anQ)
fsPend:= true
txQ:= qlast(txQ, first(psQ))
psQ:= tail(psQ)
dpsm
else
(station_ _active)
Pdu_ Request (fsdu)
-
fsdu!psm:= true
txQ:= qlast (txQ, fsdu)
fsPend:= true
resl
else
(partial)
psQ:= qlast (psQ, fsdu)
-
fsdu!_ resume:= true
-
PM_Ibss_ _Data
Frag_ Confirm (fsdu,resl)
fsdu!psm
(true)
(false)
-
txQ:= qfirst (txQ, fsdu)
-
anQ:= qfirst (anQ, fsdu)
-

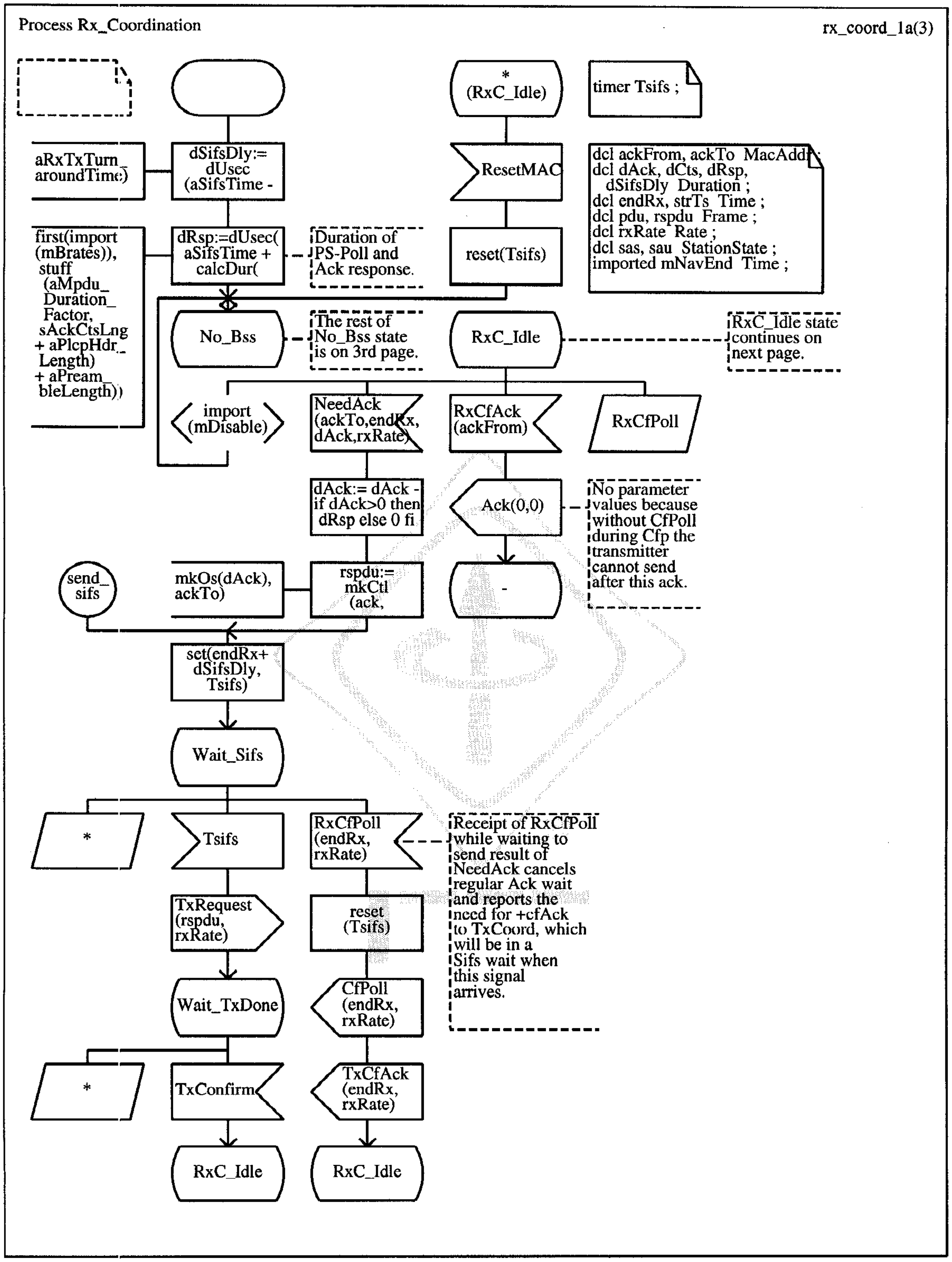

Process PM_Filter_STA
sta_PM_AtimW_4b(4)
Pre_Atim
Wait until TxCoord sends AtimW signal to avoid chance that Beacon fsdu reaches TxCoord before the TBTT signal is processed by TxCoord.
*
AtimW
n:= length(anQ)
Move all anQ entries to psQ.
n
(>0)
(=0)
psQ:= qlast(psQ, first(anQ)),
anQ:=tail(anQ), n:= n-1
Ibss during Atim window.
PM_Ibss_ _AtimW
Ensure that beacon is first fsdu sent during Atim window.
(not atPend) and (not import (mAtimW))
Frag_ Request (fsdu)
PsChange
Pdu_ Confirm (fsdu,resl)
(not atPend) and (length (psQ) /= 0)
sentBcn:= false
sentBcn
(true)
(false)
atPend:= false
fsdu:= first(psQ), psQ:=tail(psQ)
PM_Ibss_ _Data
PsInquiry (fsdu!dst)
ftype(fsdu! pdus(1))
(beacon)
else
resl
else
(atimAck)
Pdu_ Request (fsdu)
Wait_PS_ _Response
Pdu_ Request (fsdu)
Frag_ Request(fsdu) to Self
anQ:= qlast (anQ, fsdu)
atPend:= true
*
PsResponse (,dpsm)
sentBcn:= true
-
-
-
dpsm
else
(station_active)
Move fsdus that arrive before beacon back onto end of input queue.
fsdu!psm:= true
txQ:= qlast (txQ, fsdu)
psQ:= qlast (psQ, fsdu)
Handling is implementation dependent, can either re-queue until next atim window or retry during this atim window.
psQ:= qlast (psQ, fsdu)
-
PM_Ibss_ _AtimW

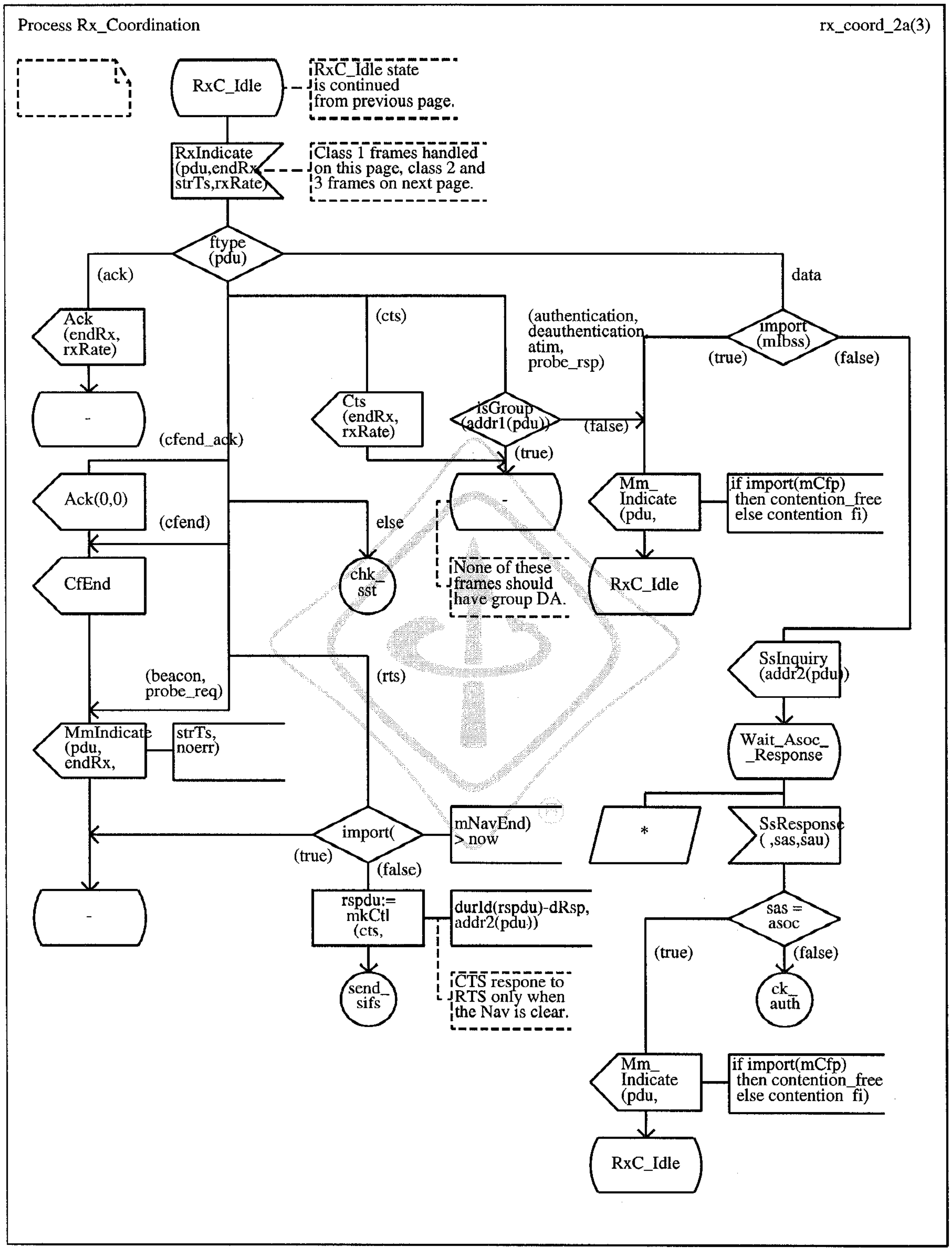

Process Prepare_MPDU
prepare_1b(2)
Encrypt
Procedure used for WEP encryption. If aPrivacyOptionImplemented= false, this procedure is not present.
dcl bcmc, keyOk, useWep Boolean:= false;
dcl f FragNum ;
dcl fsdu FragSdu ;
dcl mpduOvhd, p, pduSize, thld Integer ;
dcl pri CfPriority ;
dcl rrsl TxResult ;
dcl sdu, rsdu Frame ;
imported mAssoc, mIbss, aPrivacyInvoked Boolean ;
imported aFragmentationThreshold Integer ;
imported aWepDefaultKeys KeyVector ;
imported aWepDefaultKeyId KeyIndex ;
imported aWepKeyMappings KeyMapArray ;
imported aWepKeyMappingLength KeyMapArrayLength ;
imported mCap Octetstring ;
No_Bss
import (mAssoc)
and (not import(mActingAsAp))
import (mIbss)
import (mActing_AsAp)
Msdu_Request (sdu,pri)
Prepare__Bss
All data frames in Bss sent to distrib. service
Prepare__Ibss
All data frames in Ibss sent to destination sta.
Prepare__AP
MsduConfirm (sdu,pri, noBss)
not import (mAssoc)
Msdu_Request (sdu,pri)
Msdu_Request (sdu,pri)
not import (mIbss)
No_Bss
No_Bss
No_Bss
Msdu_Request (sdu,pri)
not import (mActing_AsAp)
No_Bss
sdu:= setAddr1 (sdu,import (mBssId)), sdu:= setToDs (sdu,1)
sdu:= setAddr3(sdu, addr1(sdu)),
Mmpdus sent even when not in Bss/Ibss.
*
Data frames rejected if no Bss/Ibss. Implementations may retain these frames until a Bss becomes (re)available.
Invoked) and aPrivacy_Option_Implemented
useWep:= import(aPrivacy_
ResetMAC
Mm_Request (sdu,pri)
Frag_Confirm (fsdu,pri,rrsl)
Fragment and encrypt is on next page.
frag_ment
No_Bss
bcmc:= isGroup(addr1(sdu))
rsdu:= substr (fsdu! pdus(0), 0,
sMacHdrLng), pri:= fsdu!cf
aPrivacy_Option_Implemented and if wepBit(sdu)=1 then true else false fi
useWep:=
basetype
(fsdu! pdus(0))
else
(management)
frag_ment
Msdu_Confirm (rsdu,pri,rrsl)
MmConfirm (rsdu,pri,rrsl) to fsdu!cnfTo
/* This process generates one or more Mpdus from each outgoing Msdu or Mmpdu. If encryption is needed, the Mpdus are encrypted before being passed to be filtered for possible power save or CF queuing before tx. */
wepBit=true in request for 3rd frame of shared key auth. seq.
Confirm Msdu to MAC data service, confirm Mmpdu to MLME sub-block.
-

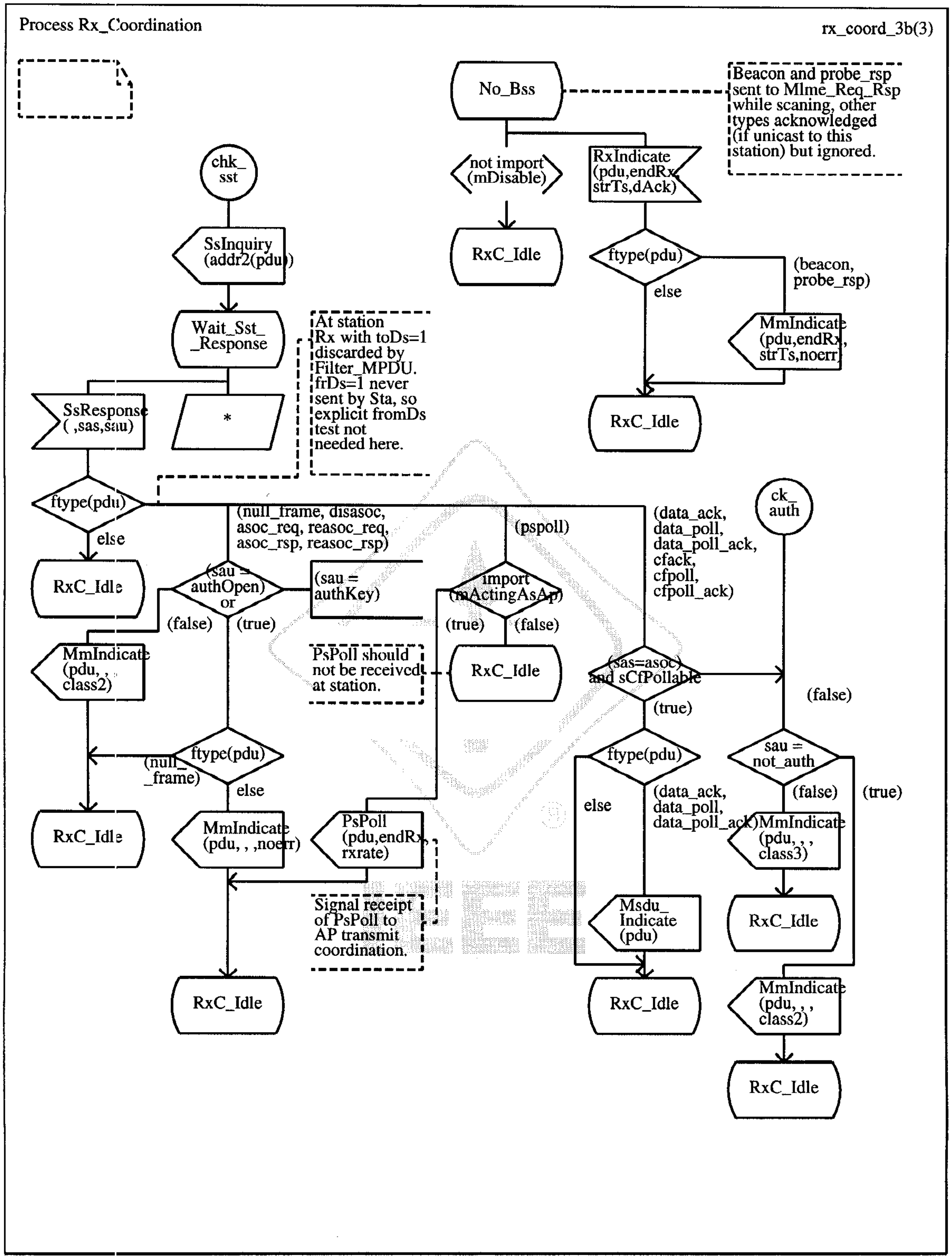

Process Prepare_MPDU
fragment_2b(2)
frag_ ment
Initialize FragSdu structure
fsdu!fTot:=1, fsdu!fCur:=0, fsdu!fAnc:=0,
fsdu!eol:=0, fsdu!sqf:=0, fsdu!src:=0, fsdu!lrc:=0, fsdu!psm:=false, fsdu!txrate:=0
fsdu!grpa:= isGrp(addr1(sdu)),
fsdu!cf:=pri, fsdu!cnfTo:=sender, fsdu!resume:=false
mpduOvhd:= sMacHdrLng + sCrcLng
Iv and Icv fields not counted in pre-fragment overhead.
aFragmentation_ Threshold must not be > aMax_ MpduLength.
pduSize:= import
(aFragment_ ation_ Threshold)
not(fsdu!grpa) and
length(sdu) > pduSize
(false)
(true)
pduSize:= length(sdu) - sMacHdrLng
pduSize:= pduSize - mpduOvhd
This is the typical case, with the length of all but the last fragment equal to aFragmentation Threshold (plus sWepAddLng if useWep=true). The value selected for pduSize must be >=256, even, and <=aMpduMaxLength.
fsdu!fTot:= ((length(sdu) - sMacHdrLng) /
pduSize) + if ((length(sdu) - sMacHdrLng) mod pduSize) /=0 then 1 else 0 fi
fsdu!fTot:=
if fsdu!fTot=0 then 1 else fsdu!fTot fi
make_ pdus
unavailable_ KeyMapping)
Msdu_ Confirm (sdu, pri,
make_ pdus
f:=0, p:= sMacHdrLng
fsdu!pdus(f):= null, keyOk:=false
fsdu!pdus(f):= fsdu!pdus(f) // substr(sdu,0,
sMacHdrLng) // substr(sdu,p, pduSize)
fsdu!pdus(f):= setFrag(fsdu!pdus(f),f)
(f+1) < fsdu!fTot
(true)
(false)
fsdu!pdus(f):= setMoreFrag(fsdu!pdus(f),1)
useWep
(true)
(false)
Encrypt (fsdu!pdus(f), keyOk,
import (aWepKeyMappings), import(aWepKey_ MappingLength), import (aWepDefaultKeys), import(aWepDefault_ KeyId), import(mCap))
keyOk
(false)
(true)
f:= f+1, p:= p+pduSize,
pduSize:= if (p+pduSize) > length(sdu) then (length(sdu) - p + 1) else pduSize fi
Final fragment may be shorter than initial/intermediate fragments.
(false)
f = fsdu!fTot
(true)
FragRequest (fsdu)
Encryption expands each pdu by sWepAddLng, hence Mpdus may be longer than aMaxMpduLength by sWepAddLng.
-

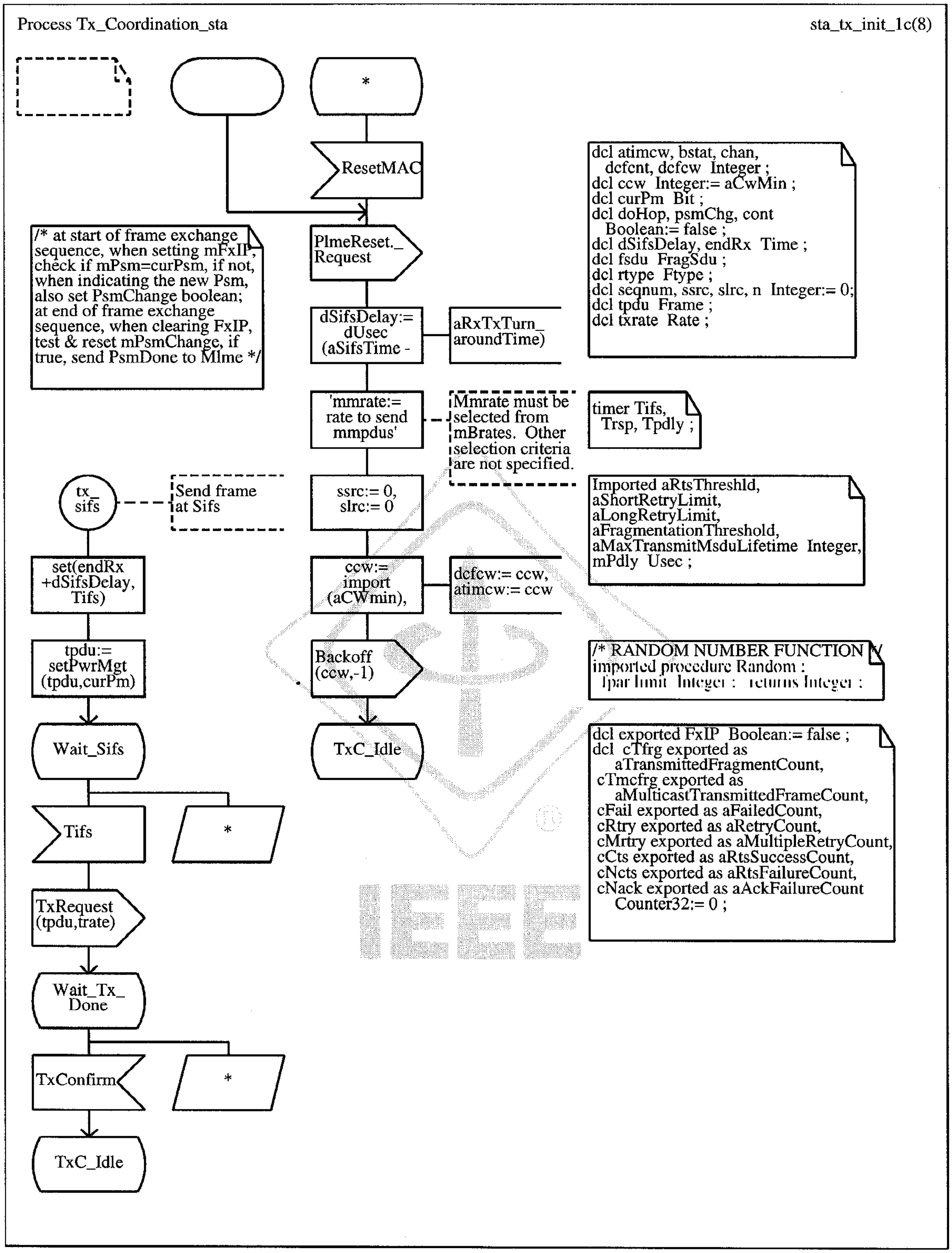

Procedure Encrypt
encrypt_1c(1)
; fpar in/out wpdu Frame, in/out keyOk Boolean, in maps KeyMapArray, in mapLength KeyMapArrayLength, in kvec KeyVector, in kndx KeyIndex, in caps Octetstring ;
isWds:= toDs(pdu) and frDs(pdu)
dcl icv Crc ; dcl encryptLng, k, n Integer ; dcl encryptStr, newIV Octetstring ; dcl key PrngKey ; dcl kmap KeyMap ; imported procedure RC4 ; fpar PrngKey, Integer ; returns Octetstring ;
en_ cipher
icv:= initCrc
Icv field is encrypted, but this length is the pre-Icv loop count.
encryptLng:= length(wpdu) - sMacHdrLng -
if isWds then sWdsAddLng else 0 fi
Test if addr4 field is present. Only need at AP.
if isWds then sWdsAddLng else 0 fi
k:= 0, n:= sWepHdrLng +
'newIV:= call genIV(x)'
The IV generation algorithm is not specified, but use of a new IV for each Mpdu is recommended STRONGLY.
ICV value calculated from plaintext.
icv:= crc32 (icv,wpdu(n))
isGrp(addr1 (wpdu))
Encrypt by xor of payload with encrypt string.
wpdu(n):= wpdu(n) xor encryptStr(k)
(true)
(false)
B_S(caps) and cPrivacy
else
(=cPrivacy)
kmap:= keyLookup (addr1(wpdu),
maps, mapLength)
/* The algorithm for changing aWepDefaultKeyId is not specified. If all stations in the Bss have the same values in the {relevant subset of} aWepDefaultKeys, a station's DefaultKeyId algorithm does not affect interoperability. */
k:= k+1, n:= n+1
k = encryptLng
(false)
(true)
mappedAddr =nullAddr
Use default key if no mapping or group dest.
no_ encr
kmap! keyOn
no_ encr
If mapping keyOn=false, do not encrypt.
n:= 0
key:= kvec(kndx)
key:= kmap!wepKey, kndx:= 0
keyOk:= true
raw ICV is 1's complement of crc32, MSb-first
icv:= mirror(not(icv))
Return error to LLC if key is null.
key= nullKey
(icv(n) xor encryptStr(k))
wpdu:= wpdu //
Concatenate key with IV for encryption PRNG seed.
key:= key // PrngKey! newIV
keyOk:= false
Encrypt ICV octets and attach to end of Mpdu.
Use RC4 PRNG to generate an encrypt string at long as the MPDU payload plus the ICV field.
encryptStr:= call RC4 (key,
encryptLng+ sCrcLng)
n = sCrcLng
wpdu:= substr(wpdu,0, sMacHdrLng)
// newIV // 01 // substr(wpdu, sMac_ HdrLng, encryptLng)
keyOk:= true
wepdu:= setWepBit (wepdu,1),
Insert IV and keyId between MAC header and data field.
wpdu:= setKeyId (wpdu,kndx)
en_ cipher
Set WEP bit in Frame Control field.

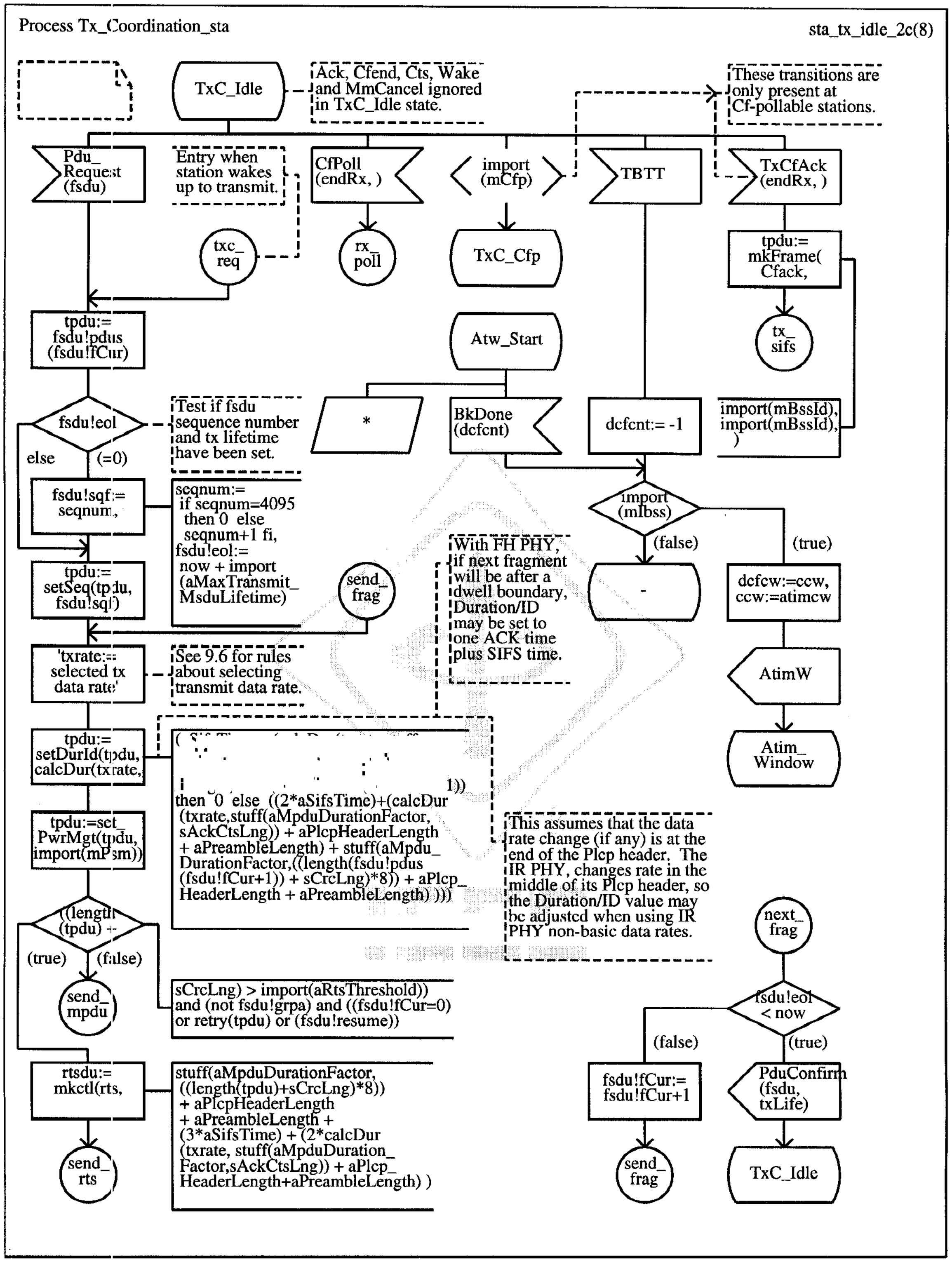

RSDU
TPDU
Block Protocol_Control_STA
MsduIndicate
AtimW, PduConfirm, CfPolled
sta_CTL_1a(1)
signal Ack(Time,Rate), Cfend, Cfpoll(Time,Rate), Cts(Time,Rate), TxCfAck(Time,Rate) ;
/* This block performs the DCF functions, as well as CF-responder functions if the station is CF-pollable. Tx_Coordination includes RTS and ATIM generation. Rx_Coordination generates acknowledgements, routes data frames to MAC data service and management frames to MLME, an indicates receipt of Ack, Cts, and CF-Poll frames to Tx_Coordination. */
Tdat
Rdat
Includes the CF responder if station is Cf-pollable.
PduRequest
Doze, MmCancel, SwChnl, Tbtt, Wake
Tmgt
PsmDone, SwDone
MCTL
Done, TxConfirm
Tx_Coordination_sta (1,1)
/* station version */
PlmeGet_ .confirm, PlmeSet_ .confirm, Plme_ Reset_ .confirm
Ack, Cts, Cfend, Cfpoll, TxCfAck
BcMgt
MmIndicate, SsInquiry
TxO
SsResponse
Backoff, Cancel, TxRequest
Pctl
Rctl
TxRx
Rx_Coordination (1,1)
/* for STA and AP */
TX
Trsp
TxRequest
TxConfirm
RxIndicate, NeedAck, RxCfAck, RxCfPoll
RxI
PlmeGet_ .request, PlmeSet_ .request, PlmeReset_ .request
ChangeNav
MLME_PLME_SAP
RX

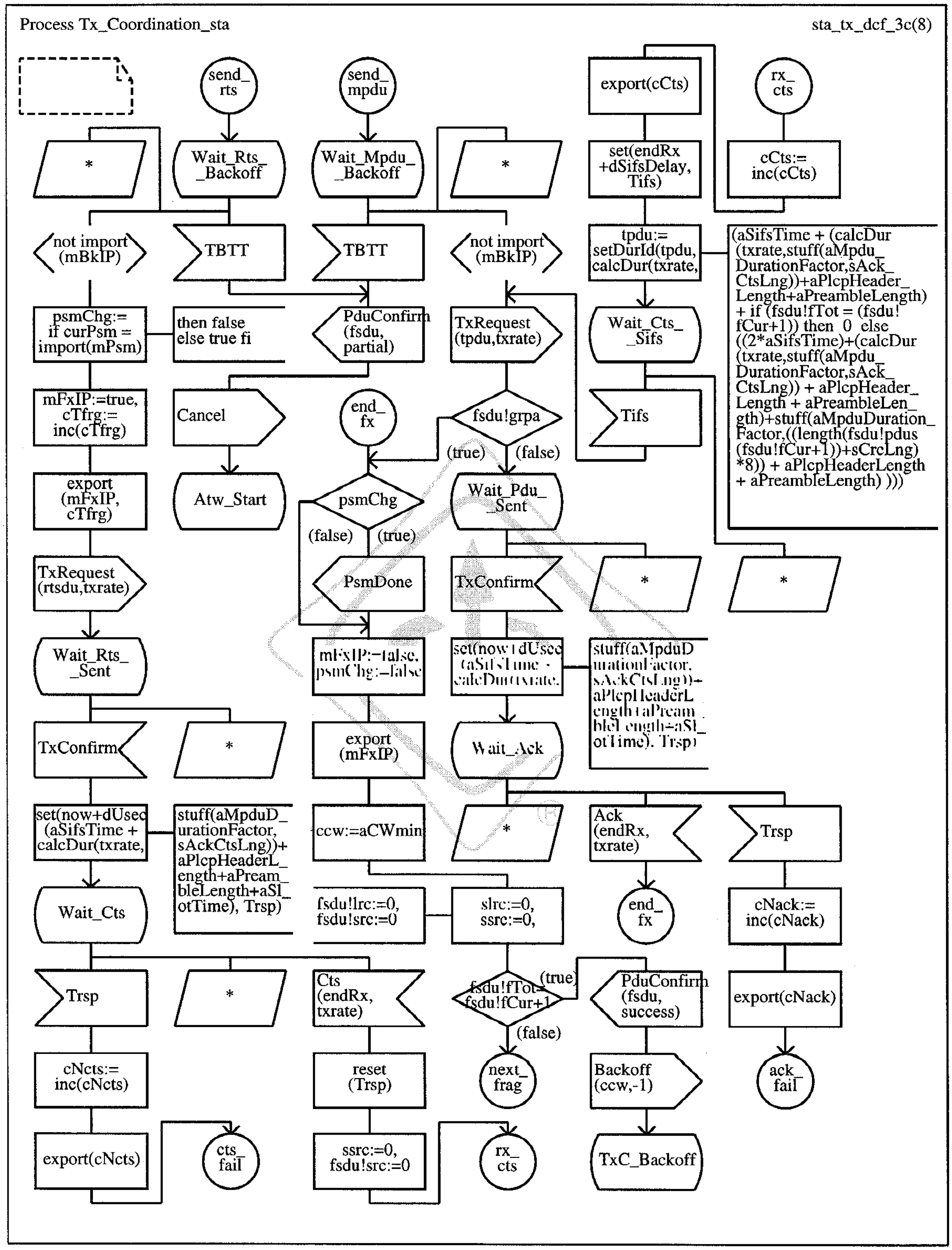

Process Rx_Coordination
rx_coord_1a(3)
*
(RxC_Idle)
timer Tsifs ;
aRxTxTurn_
aroundTime)
dSifsDly:=
dUsec
(aSifsTime -
ResetMAC
dcl ackFrom, ackTo MacAddr;
dcl dAck, dCts, dRsp,
dSifsDly Duration ;
dcl endRx, strTs Time ;
dcl pdu, rspdu Frame ;
dcl rxRate Rate ;
dcl sas, sau StationState ;
imported mNavEnd Time ;
first(import
(mBrates)),
stuff
(aMpdu_
Duration_
Factor,
sAckCtsLng
+ aPlcpHdr_
Length)
+ aPream_
bleLength))
dRsp:=dUsec(
aSifsTime +
calcDur(
Duration of
PS-Poll and
Ack response.
reset(Tsifs)
No_Bss
The rest of
No_Bss state
is on 3rd page.
RxC_Idle
RxC_Idle state
continues on
next page.
import
(mDisable)
NeedAck
(ackTo,endRx,
dAck,rxRate)
RxCfAck
(ackFrom)
RxCfPoll
dAck:= dAck -
if dAck>0 then
dRsp else 0 fi
Ack(0,0)
No parameter
values because
without CfPoll
during Cfp the
transmitter
cannot send
after this ack.
send_
sifs
mkOs(dAck),
ackTo)
rspdu:=
mkCtl
(ack,
-
set(endRx+
dSifsDly,
Tsifs)
Wait_Sifs
*
Tsifs
RxCfPoll
(endRx,
rxRate)
Receipt of RxCfPoll
while waiting to
send result of
NeedAck cancels
regular Ack wait
and reports the
need for +cfAck
to TxCoord, which
will be in a
Sifs wait when
this signal
arrives.
TxRequest
(rspdu,
rxRate)
reset
(Tsifs)
Wait_TxDone
CfPoll
(endRx,
rxRate)
*
TxConfirm
TxCfAck
(endRx,
rxRate)
RxC_Idle
RxC_Idle

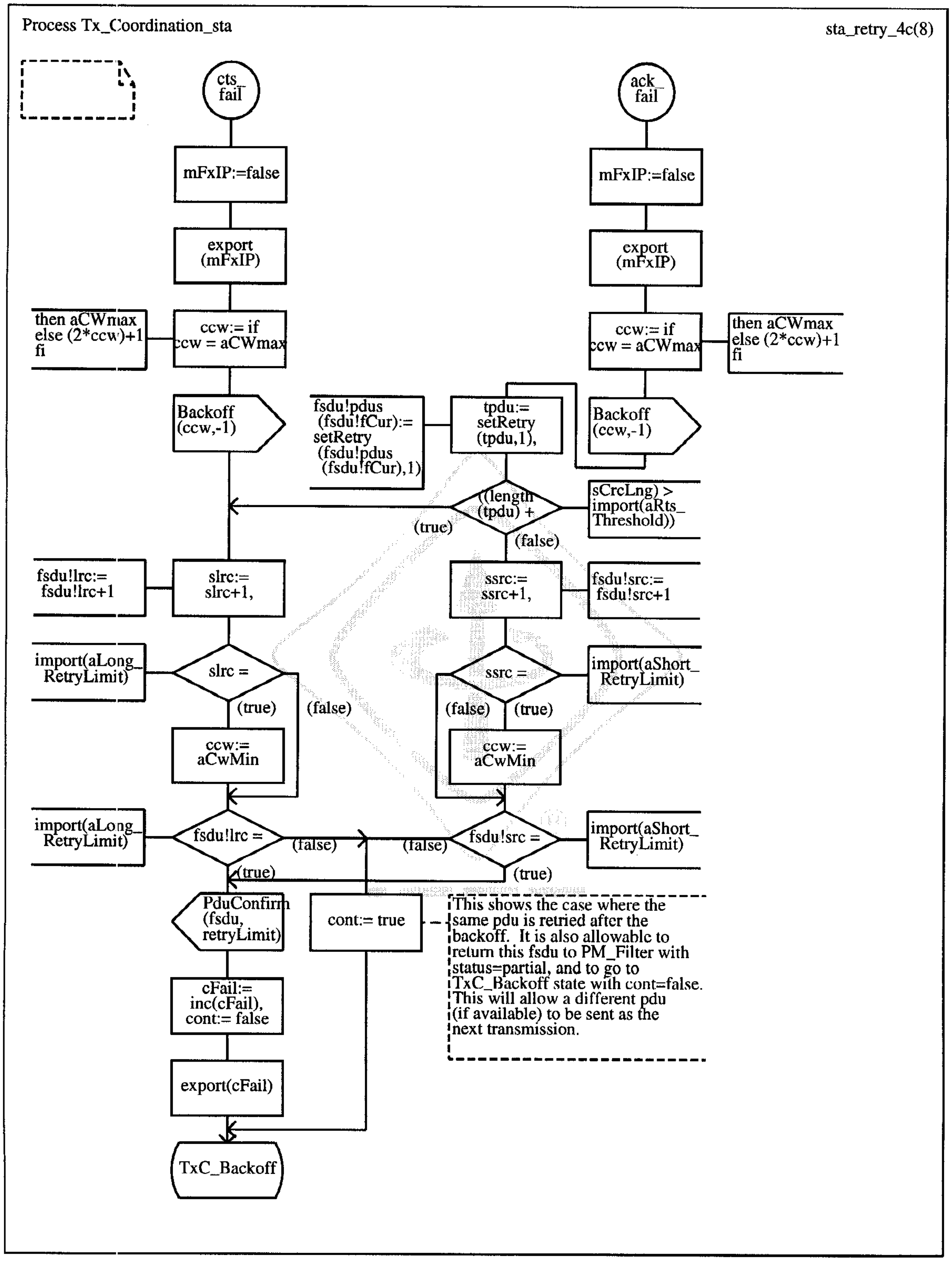

Process Rx_Coordination
rx_coord_2a(3)
RxC_Idle
RxC_Idle state is continued from previous page.
RxIndicate (pdu,endRx, strTs,rxRate)
Class 1 frames handled on this page, class 2 and 3 frames on next page.
ftype (pdu)
(ack)
data
(cts)
(authentication, deauthentication, atim, probe_rsp)
import (mIbss)
(true)
(false)
Ack (endRx, rxRate)
-
Cts (endRx, rxRate)
isGroup (addr1(pdu))
(false)
(true)
(cfend_ack)
Ack(0,0)
-
Mm_ Indicate (pdu,
if import(mCfp) then contention_free else contention fi)
(cfend)
else
CfEnd
chk_ sst
None of these frames should have group DA.
RxC_Idle
SsInquiry (addr2(pdu))
(beacon, probe_req)
(rts)
Wait_Asoc_ _Response
MmIndicate (pdu, endRx,
strTs, noerr)
import(
mNavEnd) > now
*
SsResponse (,sas,sau)
(true)
(false)
-
rspdu:= mkCtl (cts,
durId(rspdu)-dRsp, addr2(pdu))
sas = asoc
(true)
(false)
send_ sifs
CTS respone to RTS only when the Nav is clear.
ck_ auth
Mm_ Indicate (pdu,
if import(mCfp) then contention_free else contention fi)
RxC_Idle

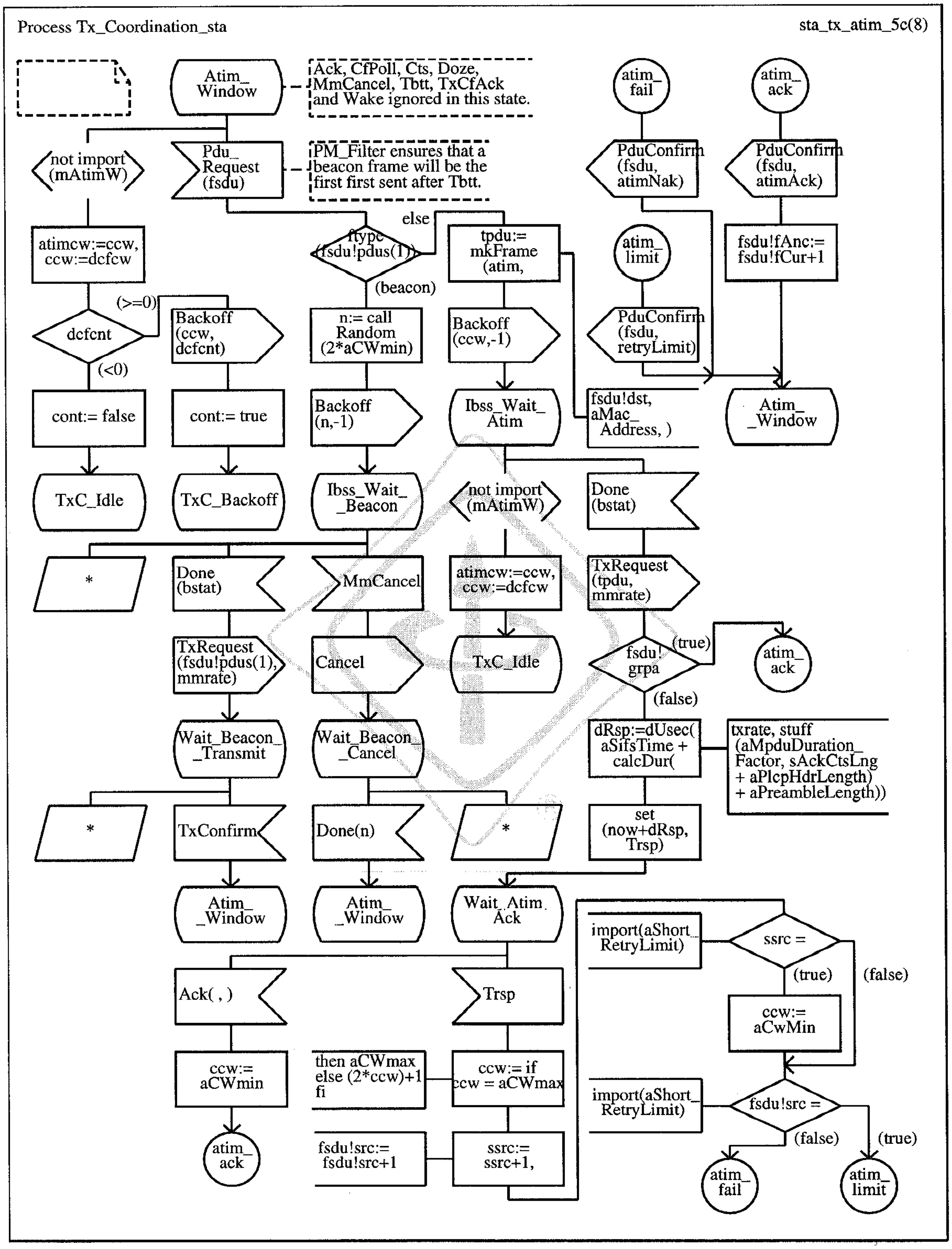

Process Rx_Coordination
rx_coord_3b(3)
No_Bss
Beacon and probe_rsp sent to Mlme_Req_Rsp while scaning, other types acknowledged (if unicast to this station) but ignored.
not import (mDisable)
RxIndicate (pdu,endRx, strTs,dAck)
RxC_Idle
ftype(pdu)
else
(beacon, probe_rsp)
MmIndicate (pdu,endRx, strTs,noerr)
RxC_Idle
chk_ sst
SsInquiry (addr2(pdu))
Wait_Sst_ _Response
At station Rx with toDs=1 discarded by Filter_MPDU. frDs=1 never sent by Sta, so explicit fromDs test not needed here.
SsResponse (,sas,sau)
*
ftype(pdu)
else
(null_frame, disasoc, asoc_req, reasoc_req, asoc_rsp, reasoc_rsp)
(pspoll)
(data_ack, data_poll, data_poll_ack, cfack, cfpoll, cfpoll_ack)
ck_ auth
RxC_Idle
(sau = authOpen) or
(sau = authKey)
(false)
(true)
import (mActingAsAp)
(true)
(false)
MmIndicate (pdu, , class2)
PsPoll should not be received at station.
RxC_Idle
(sas=asoc) and sCfPollable
(true)
(false)
ftype(pdu)
(null_ _frame)
else
ftype(pdu)
else
(data_ack, data_poll, data_poll_ack)
sau = not_auth
(false)
(true)
RxC_Idle
MmIndicate (pdu, , ,noerr)
PsPoll (pdu,endRx, rxrate)
MmIndicate (pdu, , , class3)
Signal receipt of PsPoll to AP transmit coordination.
Msdu_ Indicate (pdu)
RxC_Idle
RxC_Idle
RxC_Idle
MmIndicate (pdu, , , class2)
RxC_Idle

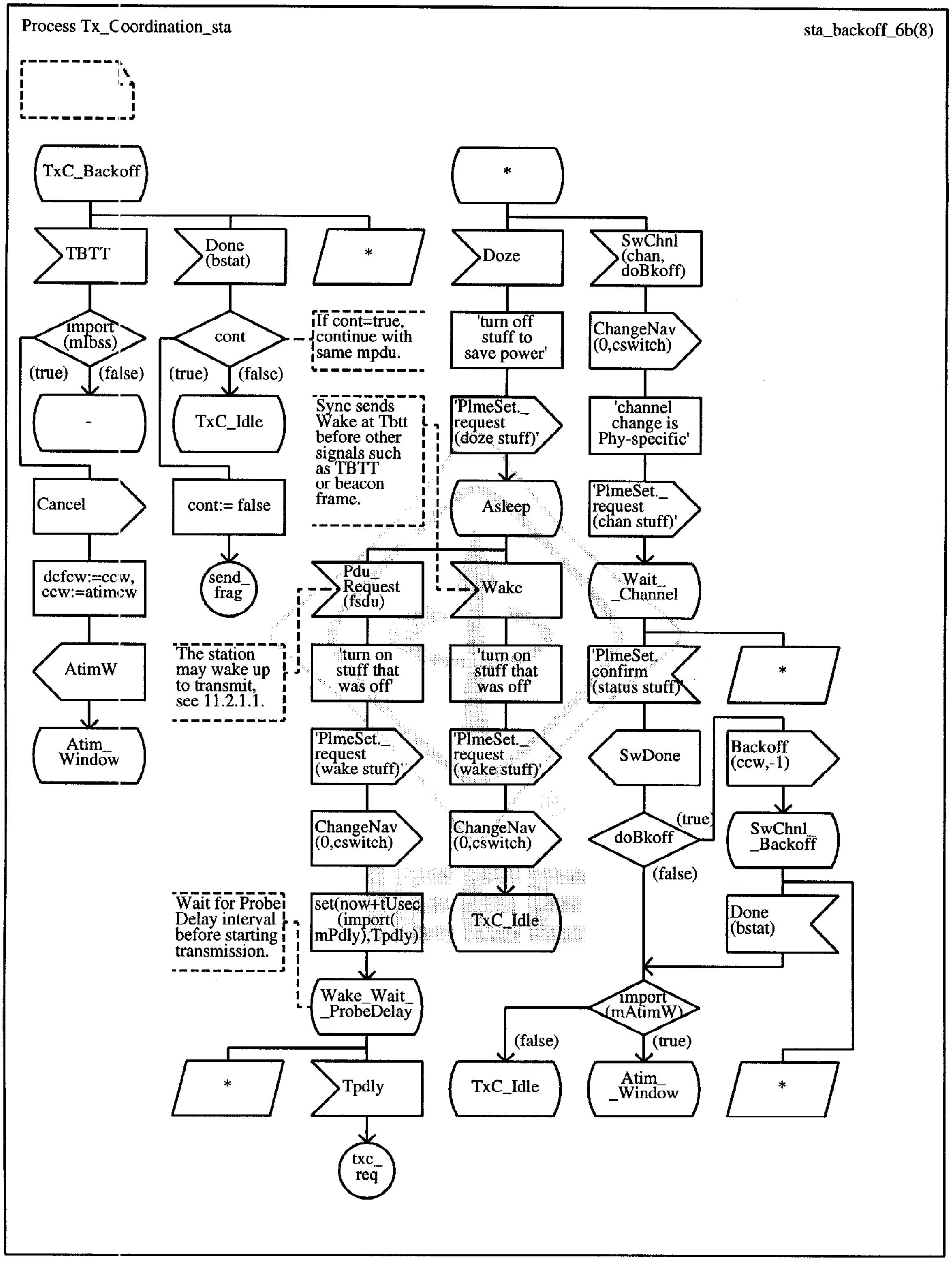

Process Tx_Coordination_sta
sta_tx_init_1c(8)
*
ResetMAC
PlmeReset._Request
dSifsDelay:= dUsec (aSifsTime -
aRxTxTurn_aroundTime)
'mmrate:= rate to send mmpdus'
Mmrate must be selected from mBrates. Other selection criteria are not specified.
ssrc:= 0, slrc:= 0
ccw:= import (aCWmin),
dcfcw:= ccw, atimcw:= ccw
Backoff (ccw,-1)
TxC_Idle
/* at start of frame exchange sequence, when setting mFxIP, check if mPsm=curPsm, if not, when indicating the new Psm, also set PsmChange boolean; at end of frame exchange sequence, when clearing FxIP, test & reset mPsmChange, if true, send PsmDone to Mlme */
tx_sifs
Send frame at Sifs
set(endRx +dSifsDelay, Tifs)
tpdu:= setPwrMgt (tpdu,curPm)
Wait_Sifs
Tifs
*
TxRequest (tpdu,trate)
Wait_Tx_Done
TxConfirm
*
TxC_Idle
dcl atimcw, bstat, chan, dcfcnt, dcfcw Integer ;
dcl ccw Integer:= aCwMin ;
dcl curPm Bit ;
dcl doHop, psmChg, cont Boolean:= false ;
dcl dSifsDelay, endRx Time ;
dcl fsdu FragSdu ;
dcl rtype Ftype ;
dcl seqnum, ssrc, slrc, n Integer:= 0;
dcl tpdu Frame ;
dcl txrate Rate ;
timer Tifs, Trsp, Tpdly ;
Imported aRtsThreshld, aShortRetryLimit, aLongRetryLimit, aFragmentationThreshold, aMaxTransmitMsduLifetime Integer, mPdly Usec ;
/* RANDOM NUMBER FUNCTION
imported procedure Random :
dcl exported FxIP Boolean:= false ;
dcl cTfrg exported as aTransmittedFragmentCount,
cTmcfrg exported as aMulticastTransmittedFrameCount,
cFail exported as aFailedCount,
cRtry exported as aRetryCount,
cMrtry exported as aMultipleRetryCount,
cCts exported as aRtsSuccessCount,
cNcts exported as aRtsFailureCount,
cNack exported as aAckFailureCount
Counter32:= 0 ;

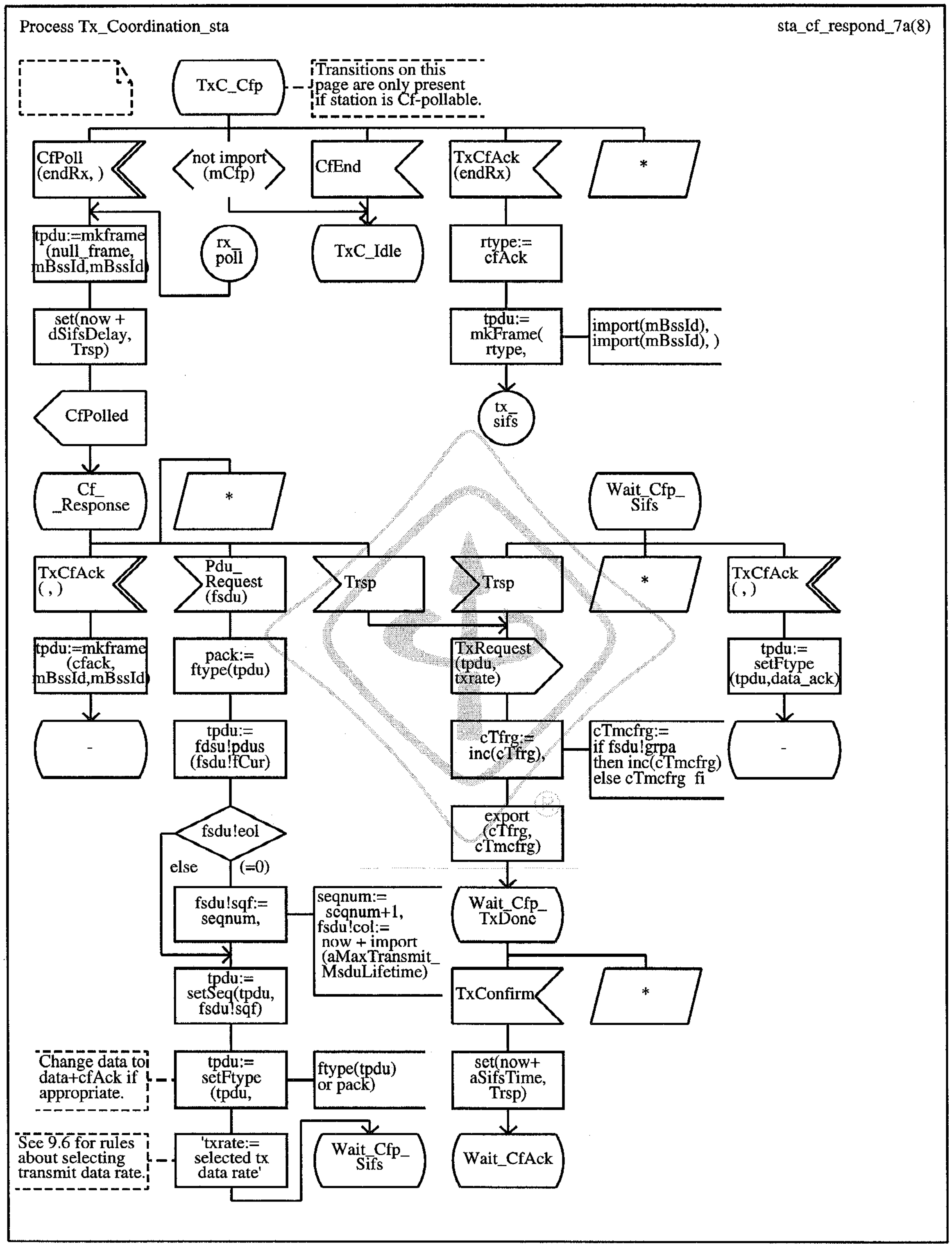

Process Tx_Coordination_sta
sta_tx_idle_2c(8)
TxC_Idle
Ack, Cfend, Cts, Wake and MmCancel ignored in TxC_Idle state.
These transitions are only present at Cf-pollable stations.
Pdu_ Request (fsdu)
Entry when station wakes up to transmit.
CfPoll (endRx,)
import (mCfp)
TBTT
TxCfAck (endRx,)
txc_ req
rx_ poll
TxC_Cfp
tpdu:= mkFrame(Cfack,
tx_ sifs
tpdu:= fsdu!pdus (fsdu!fCur)
Atw_Start
fsdu!eol
Test if fsdu sequence number and tx lifetime have been set.
*
BkDone (dcfcnt)
dcfcnt:= -1
import(mBssId), import(mBssId),)
else
(=0)
fsdu!sqf:= seqnum,
seqnum:= if seqnum=4095 then 0 else seqnum+1 fi, fsdu!eol:= now + import (aMaxTransmit_ MsduLifetime)
import (mIbss)
(false)
(true)
With FH PHY, if next fragment will be after a dwell boundary, Duration/ID may be set to one ACK time plus SIFS time.
tpdu:= setSeq(tpdu, fsdu!sqf)
send_ frag
-
dcfcw:=ccw, ccw:=atimcw
'txrate:= selected tx data rate'
See 9.6 for rules about selecting transmit data rate.
AtimW
tpdu:= setDurId(tpdu, calcDur(txrate,
then 0 else ((2*aSifsTime)+(calcDur (txrate,stuff(aMpduDurationFactor, sAckCtsLng)) + aPlcpHeaderLength + aPreambleLength) + stuff(aMpdu_ DurationFactor,((length(fsdu!pdus (fsdu!fCur+1)) + sCrcLng)*8)) + aPlcp_ HeaderLength + aPreambleLength))))
Atim_ Window
tpdu:=set_ PwrMgt(tpdu, import(mPsm))
This assumes that the data rate change (if any) is at the end of the Plcp header. The IR PHY, changes rate in the middle of its Plcp header, so the Duration/ID value may be adjusted when using IR PHY non-basic data rates.
((length (tpdu) +
(true)
(false)
next_ frag
send_ mpdu
sCrcLng) > import(aRtsThreshold)) and (not fsdu!grpa) and ((fsdu!fCur=0) or retry(tpdu) or (fsdu!resume))
fsdu!eol < now
(false)
(true)
rtsdu:= mkctl(rts,
stuff(aMpduDurationFactor, ((length(tpdu)+sCrcLng)*8)) + aPlcpHeaderLength + aPreambleLength + (3*aSifsTime) + (2*calcDur (txrate, stuff(aMpduDuration_ Factor,sAckCtsLng)) + aPlcp_ HeaderLength+aPreambleLength))
fsdu!fCur:= fsdu!fCur+1
PduConfirm (fsdu, txLife)
send_ rts
send_ frag
TxC_Idle

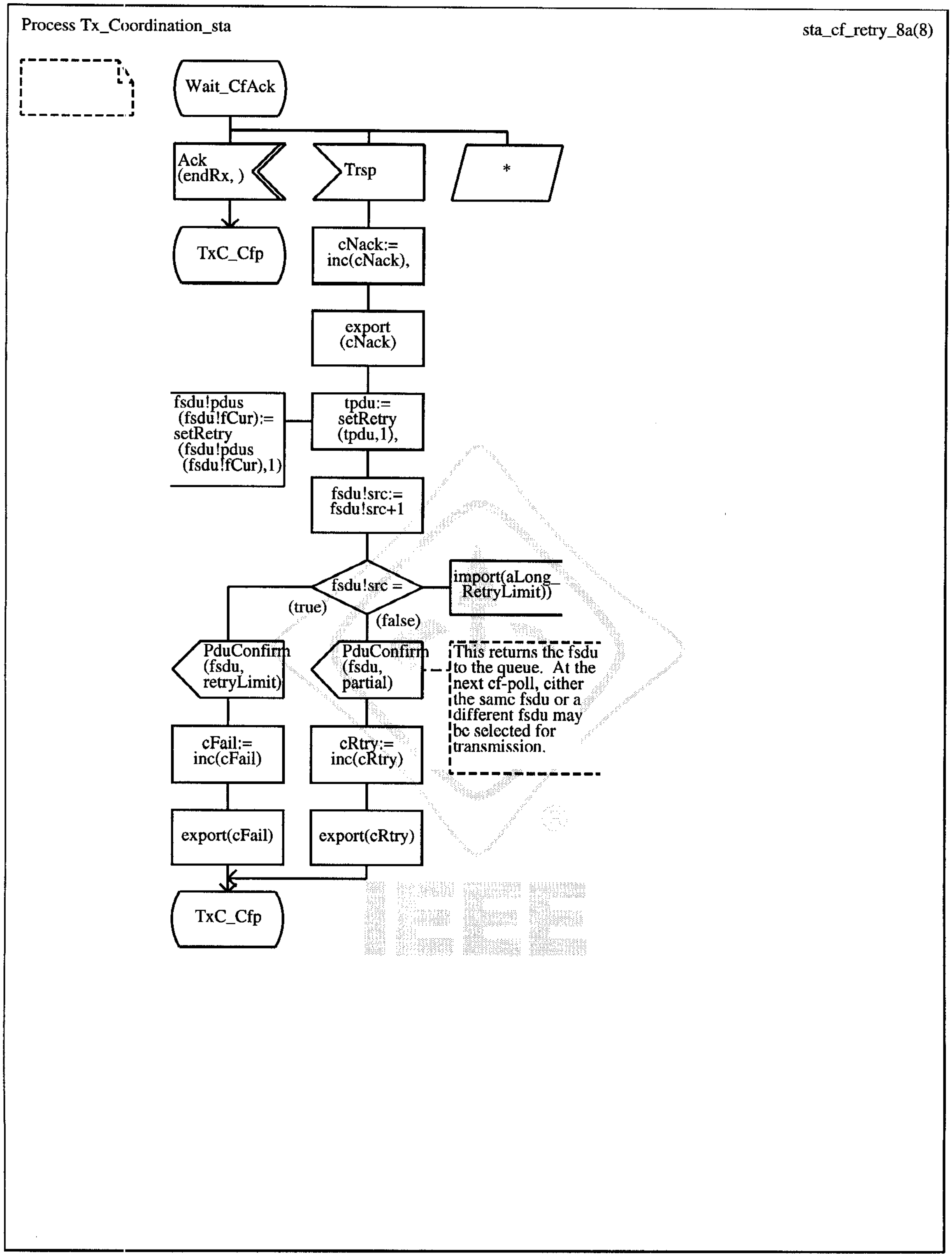

Process Tx_Coordination_sta
sta_tx_dcf_3c(8)
send_ rts
send_ mpdu
export(cCts)
rx_ cts
*
Wait_Rts_ _Backoff
Wait_Mpdu_ _Backoff
*
set(endRx +dSifsDelay, Tifs)
cCts:= inc(cCts)
not import (mBkIP)
TBTT
TBTT
not import (mBkIP)
tpdu:= setDurId(tpdu, calcDur(txrate,
(aSifsTime + (calcDur (txrate,stuff(aMpdu_ DurationFactor,sAck_ CtsLng))+aPlcpHeader_ Length+aPreambleLength) + if (fsdu!fTot = (fsdu! fCur+1)) then 0 else ((2*aSifsTime)+(calcDur (txrate,stuff(aMpdu_ DurationFactor,sAck_ CtsLng)) + aPlcpHeader_ Length + aPreambleLen_ gth)+stuff(aMpduDuration_ Factor,((length(fsdu!pdus (fsdu!fCur+1))+sCrcLng) *8)) + aPlcpHeaderLength + aPreambleLength))))
psmChg:= if curPsm = import(mPsm)
then false else true fi
PduConfirm (fsdu, partial)
TxRequest (tpdu,txrate)
Wait_Cts_ _Sifs
mFxIP:=true, cTfrg:= inc(cTfrg)
Cancel
end_ fx
fsdu!grpa
Tifs
(true)
(false)
export (mFxIP, cTfrg)
Atw_Start
psmChg
Wait_Pdu_ _Sent
(false)
(true)
TxRequest (rtsdu,txrate)
PsmDone
TxConfirm
*
*
Wait_Rts_ _Sent
mFxIP:=false, psmChg:=false
set(now+dUsec (aSifsTime + calcDur(txrate,
stuff(aMpduD urationFactor, sAckCtsLng))+ aPlcpHeaderL ength+aPream bleLength+aSl_ otTime), Trsp)
TxConfirm
*
export (mFxIP)
Wait_Ack
set(now+dUsec (aSifsTime + calcDur(txrate,
stuff(aMpduD_ urationFactor, sAckCtsLng))+ aPlcpHeaderL_ ength+aPream_ bleLength+aSl_ otTime), Trsp)
ccw:=aCWmin
*
Ack (endRx, txrate)
Trsp
Wait_Cts
fsdu!lrc:=0, fsdu!src:=0
slrc:=0, ssrc:=0,
end_ fx
cNack:= inc(cNack)
Trsp
*
Cts (endRx, txrate)
fsdu!fTot= fsdu!fCur+1
(true)
(false)
PduConfirm (fsdu, success)
export(cNack)
cNcts:= inc(cNcts)
reset (Trsp)
next_ frag
Backoff (ccw,-1)
ack_ fail
export(cNcts)
cts_ fail
ssrc:=0, fsdu!src:=0
rx_ cts
TxC_Backoff

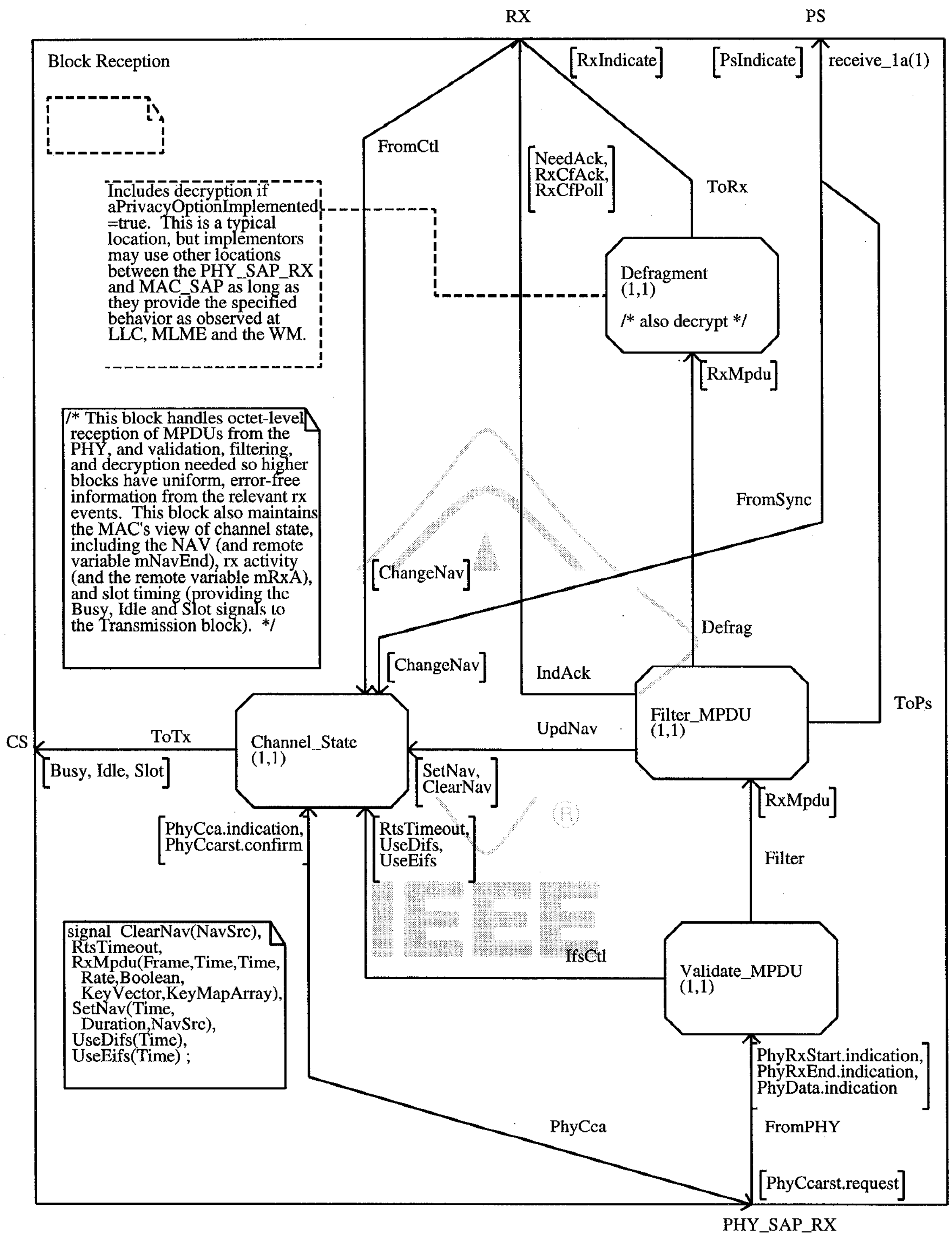

Process Tx_Coordination_sta
sta_retry_4c(8)
cts_fail
mFxIP:=false
export (mFxIP)
ccw:= if ccw = aCWmax
then aCWmax else (2*ccw)+1 fi
Backoff (ccw,-1)
ack_fail
mFxIP:=false
export (mFxIP)
ccw:= if ccw = aCWmax
then aCWmax else (2*ccw)+1 fi
Backoff (ccw,-1)
fsdu!pdus (fsdu!fCur):= setRetry (fsdu!pdus (fsdu!fCur),1)
tpdu:= setRetry (tpdu,1),
((length (tpdu) +
sCrcLng) > import(aRts_ Threshold))
(true)
(false)
slrc:= slrc+1,
fsdu!lrc:= fsdu!lrc+1
ssrc:= ssrc+1,
fsdu!src:= fsdu!src+1
slrc =
import(aLong_ RetryLimit)
(true)
(false)
ccw:= aCwMin
ssrc =
import(aShort_ RetryLimit)
(false)
(true)
ccw:= aCwMin
fsdu!lrc =
import(aLong_ RetryLimit)
(false)
(true)
fsdu!src =
import(aShort_ RetryLimit)
(false)
(true)
PduConfirm (fsdu, retryLimit)
cont:= true
This shows the case where the same pdu is retried after the backoff. It is also allowable to return this fsdu to PM_Filter with status=partial, and to go to TxC_Backoff state with cont=false. This will allow a different pdu (if available) to be sent as the next transmission.
cFail:= inc(cFail), cont:= false
export(cFail)
TxC_Backoff

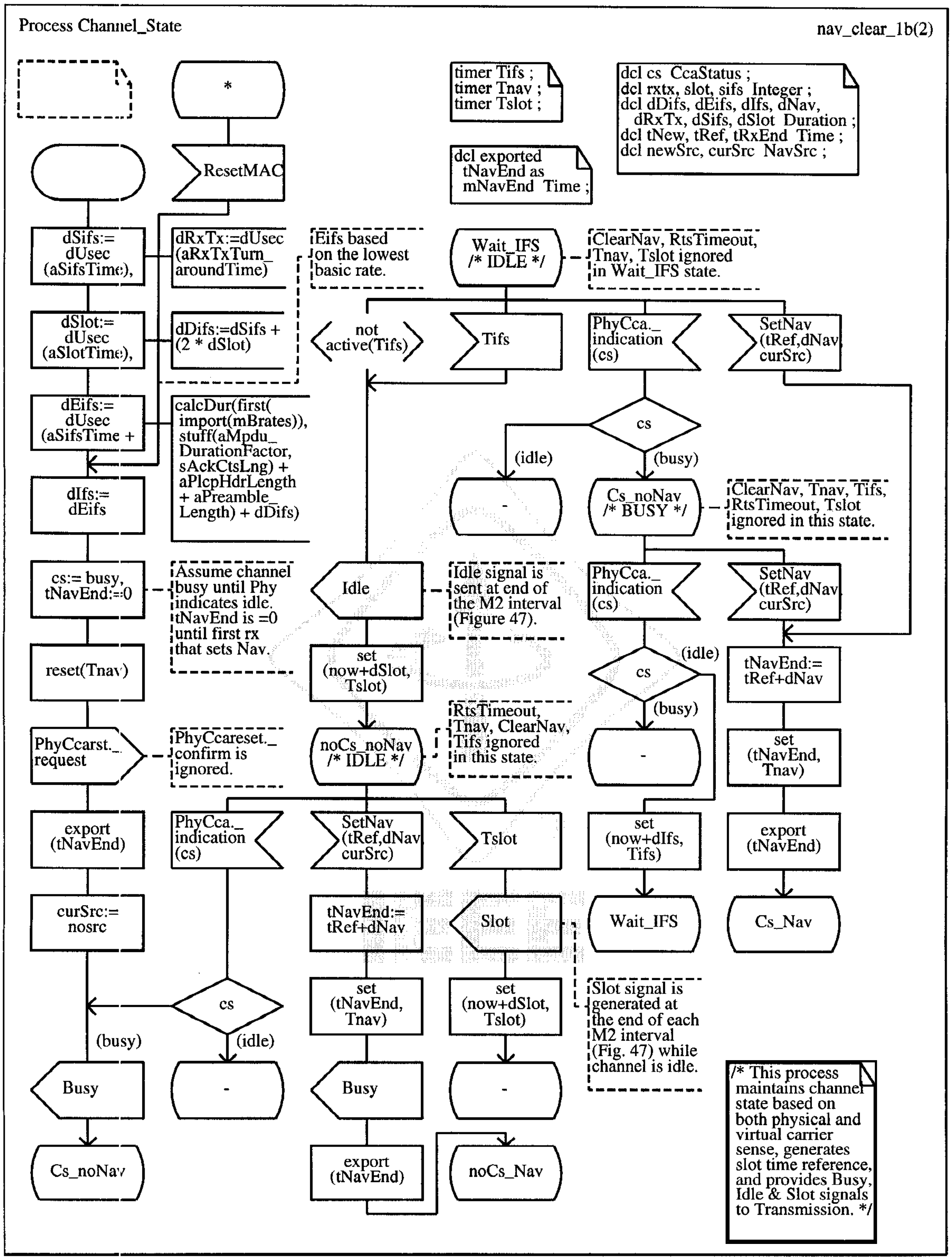

Process Tx_Coordination_sta
sta_tx_atim_5c(8)
Atim_ Window
Ack, CfPoll, Cts, Doze, MmCancel, Tbtt, TxCfAck and Wake ignored in this state.
not import (mAtimW)
Pdu_ Request (fsdu)
PM_Filter ensures that a beacon frame will be the first first sent after Tbtt.
atimcw:=ccw, ccw:=dcfcw
ftype (fsdu!pdus(1))
else
(beacon)
tpdu:= mkFrame (atim,
dcfcnt
(>=0)
(<0)
Backoff (ccw, dcfcnt)
n:= call Random (2*aCWmin)
Backoff (ccw,-1)
cont:= false
cont:= true
Backoff (n,-1)
Ibss_Wait_ Atim
fsdu!dst, aMac_ Address,)
TxC_Idle
TxC_Backoff
Ibss_Wait_ _Beacon
not import (mAtimW)
Done (bstat)
*
Done (bstat)
MmCancel
atimcw:=ccw, ccw:=dcfcw
TxRequest (tpdu, mmrate)
TxRequest (fsdu!pdus(1), mmrate)
Cancel
TxC_Idle
fsdu! grpa
(true)
(false)
atim_ ack
Wait_Beacon_ _Transmit
Wait_Beacon_ _Cancel
dRsp:=dUsec(aSifsTime + calcDur(
txrate, stuff (aMpduDuration_ Factor, sAckCtsLng + aPlcpHdrLength) + aPreambleLength))
*
TxConfirm
Done(n)
*
set (now+dRsp, Trsp)
Atim_ _Window
Atim_ _Window
Wait_Atim_ Ack
import(aShort_ RetryLimit)
ssrc =
(true)
(false)
Ack(,)
Trsp
ccw:= aCWmin
then aCWmax else (2*ccw)+1 fi
ccw:= if ccw = aCWmax
ccw:= aCwMin
import(aShort_ RetryLimit)
fsdu!src =
(false)
(true)
atim_ ack
fsdu!src:= fsdu!src+1
ssrc:= ssrc+1,
atim_ fail
atim_ limit
atim_ fail
atim_ ack
PduConfirm (fsdu, atimNak)
PduConfirm (fsdu, atimAck)
atim_ limit
fsdu!fAnc:= fsdu!fCur+1
PduConfirm (fsdu, retryLimit)
Atim_ _Window

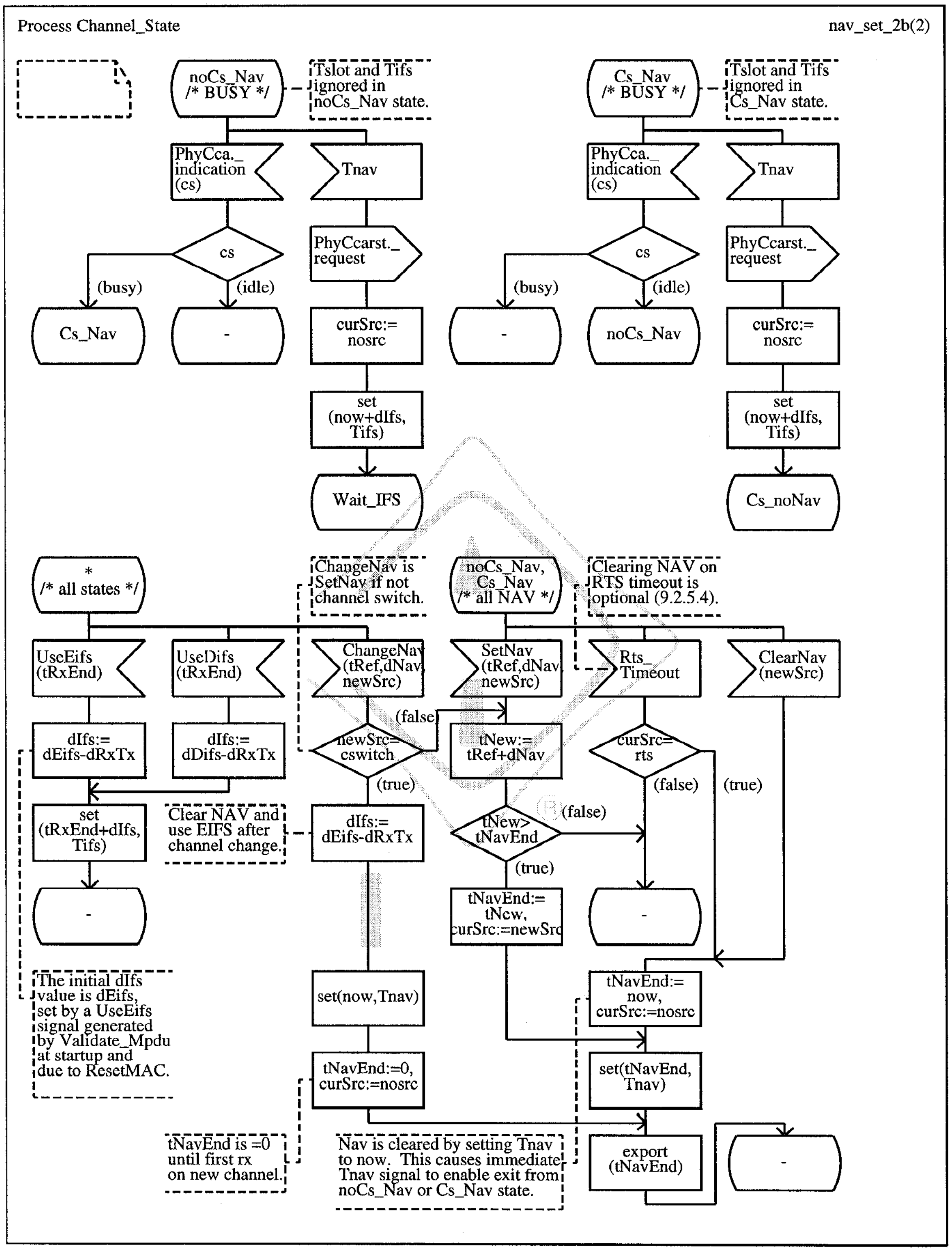

Process Tx_Coordination_sta
sta_backoff_6b(8)
TxC_Backoff
*
TBTT
Done (bstat)
*
Doze
SwChnl (chan, doBkoff)
import (mIbss)
(true)
(false)
cont
(true)
(false)
If cont=true, continue with same mpdu.
'turn off stuff to save power'
ChangeNav (0,cswitch)
-
TxC_Idle
Sync sends Wake at Tbtt before other signals such as TBTT or beacon frame.
'PlmeSet._request (doze stuff)'
'channel change is Phy-specific'
Cancel
cont:= false
Asleep
'PlmeSet._request (chan stuff)'
dcfcw:=ccw, ccw:=atimcw
send_frag
Pdu_Request (fsdu)
Wake
Wait__Channel
AtimW
The station may wake up to transmit, see 11.2.1.1.
'turn on stuff that was off'
'turn on stuff that was off'
'PlmeSet._confirm (status stuff)'
*
Atim_Window
'PlmeSet._request (wake stuff)'
'PlmeSet._request (wake stuff)'
SwDone
Backoff (ccw,-1)
ChangeNav (0,cswitch)
ChangeNav (0,cswitch)
doBkoff
(true)
(false)
SwChnl__Backoff
Wait for Probe Delay interval before starting transmission.
set(now+tUsec (import(mPdly),Tpdly)
TxC_Idle
Done (bstat)
Wake_Wait__ProbeDelay
import (mAtimW)
(false)
(true)
*
Tpdly
TxC_Idle
Atim__Window
*
txc_req

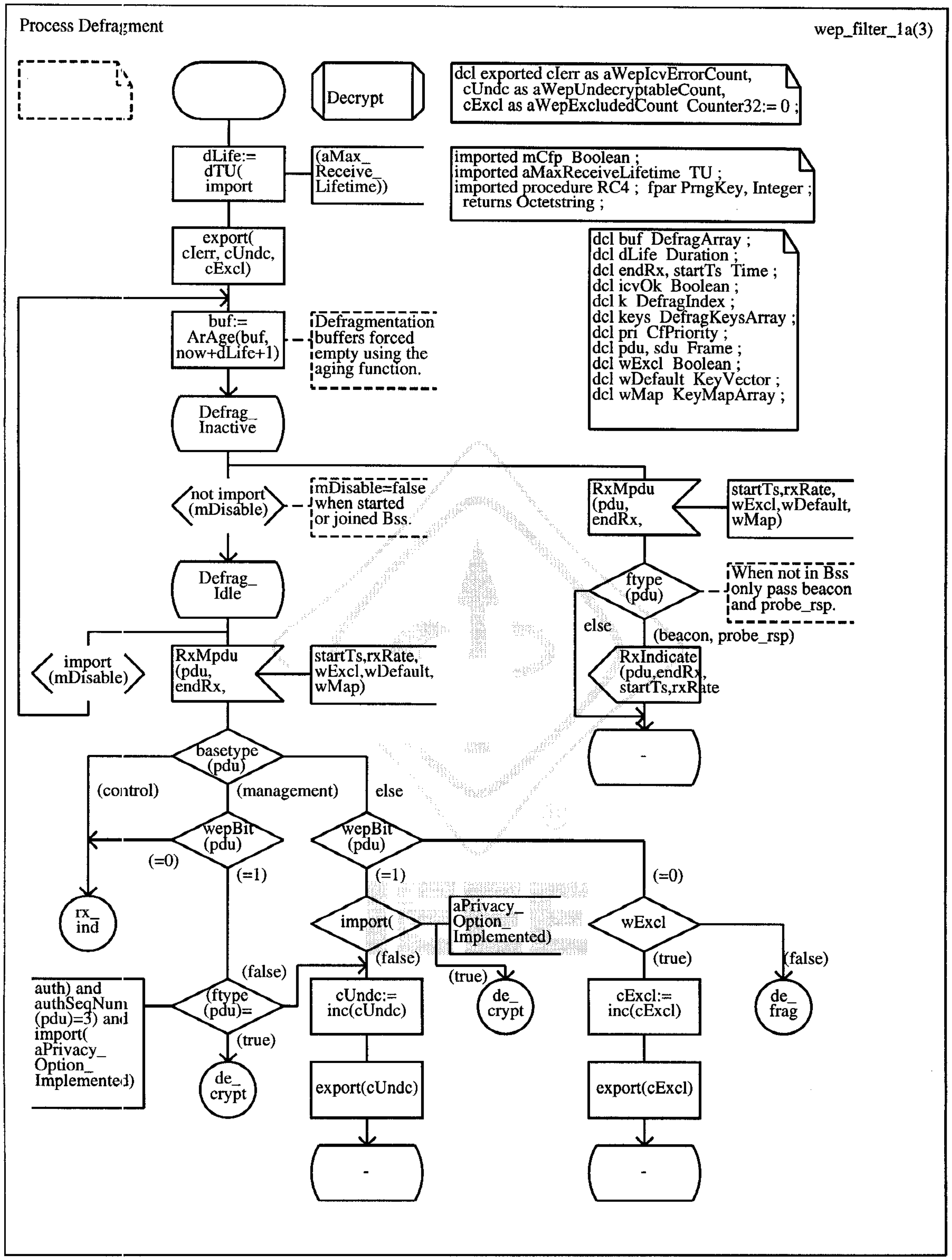

Process Tx_Coordination_sta
sta_cf_respond_7a(8)
TxC_Cfp
Transitions on this page are only present if station is Cf-pollable.
CfPoll (endRx,)
not import (mCfp)
CfEnd
TxCfAck (endRx)
*
tpdu:=mkframe (null_frame, mBssId,mBssId)
rx_ poll
TxC_Idle
rtype:= cfAck
set(now + dSifsDelay, Trsp)
tpdu:= mkFrame(rtype,
import(mBssId), import(mBssId),)
CfPolled
tx_ sifs
Cf_ _Response
*
Wait_Cfp_ Sifs
TxCfAck (,)
Pdu_ Request (fsdu)
Trsp
Trsp
*
TxCfAck (,)
tpdu:=mkframe (cfack, mBssId,mBssId)
pack:= ftype(tpdu)
TxRequest (tpdu, txrate)
tpdu:= setFtype (tpdu,data_ack)
-
tpdu:= fdsu!pdus (fsdu!fCur)
cTfrg:= inc(cTfrg),
cTmcfrg:= if fsdu!grpa then inc(cTmcfrg) else cTmcfrg fi
-
fsdu!eol
else
(=0)
export (cTfrg, cTmcfrg)
fsdu!sqf:= seqnum,
seqnum:= seqnum+1, fsdu!col:= now + import (aMaxTransmit_ MsduLifetime)
Wait_Cfp_ TxDone
tpdu:= setSeq(tpdu, fsdu!sqf)
TxConfirm
*
Change data to data+cfAck if appropriate.
tpdu:= setFtype (tpdu,
ftype(tpdu) or pack)
set(now+ aSifsTime, Trsp)
See 9.6 for rules about selecting transmit data rate.
'txrate:= selected tx data rate'
Wait_Cfp_ Sifs
Wait_CfAck

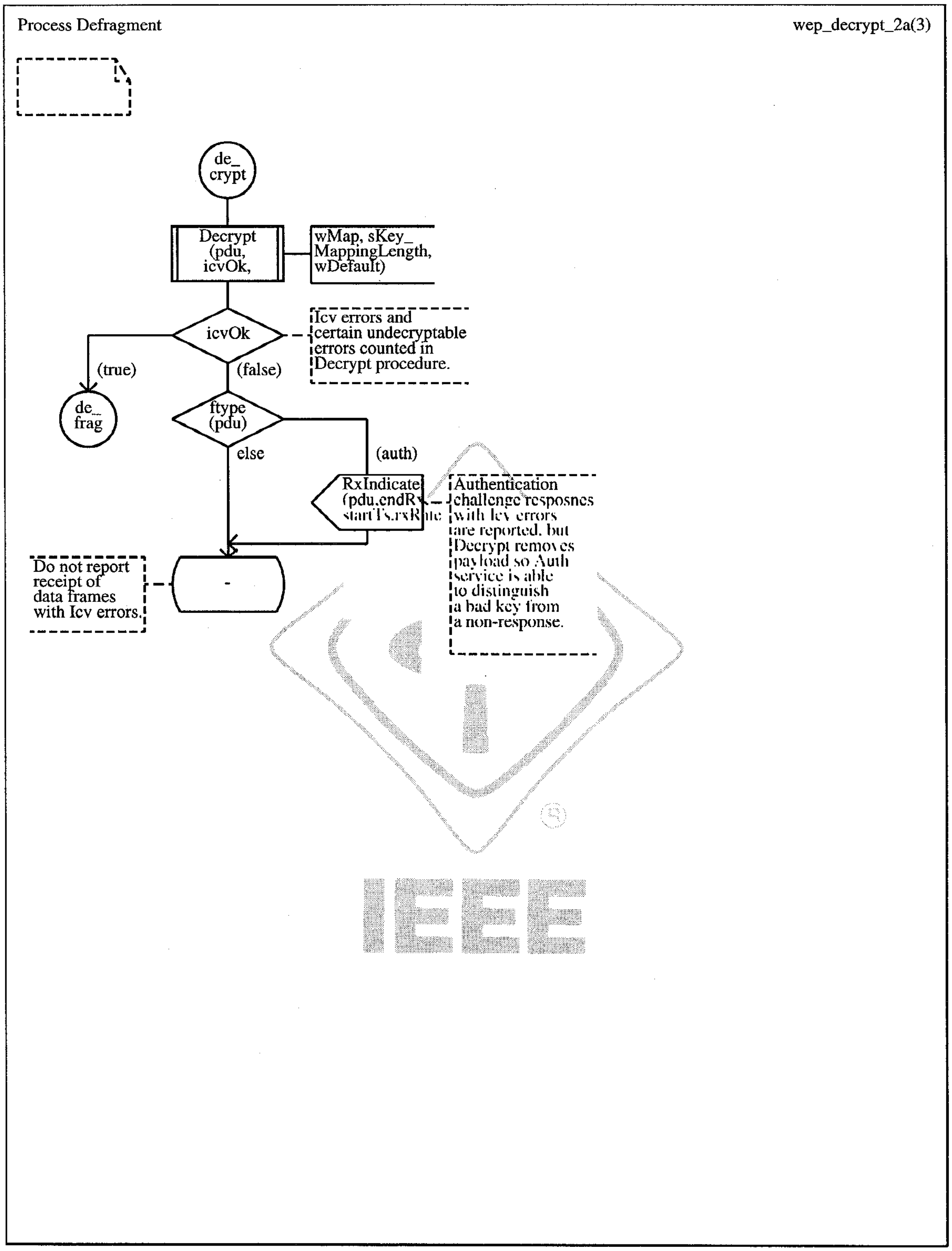

Process Tx_Coordination_sta
sta_cf_retry_8a(8)
Wait_CfAck
Ack
(endRx,)
Trsp
*
TxC_Cfp
cNack:=
inc(cNack),
export
(cNack)
fsdu!pdus
(fsdu!fCur):=
setRetry
(fsdu!pdus
(fsdu!fCur),1)
tpdu:=
setRetry
(tpdu,1),
fsdu!src:=
fsdu!src+1
fsdu!src =
import(aLong_
RetryLimit))
(true)
(false)
PduConfirm
(fsdu,
retryLimit)
PduConfirm
(fsdu,
partial)
This returns the fsdu
to the queue. At the
next cf-poll, either
the same fsdu or a
different fsdu may
be selected for
transmission.
cFail:=
inc(cFail)
cRtry:=
inc(cRtry)
export(cFail)
export(cRtry)
TxC_Cfp

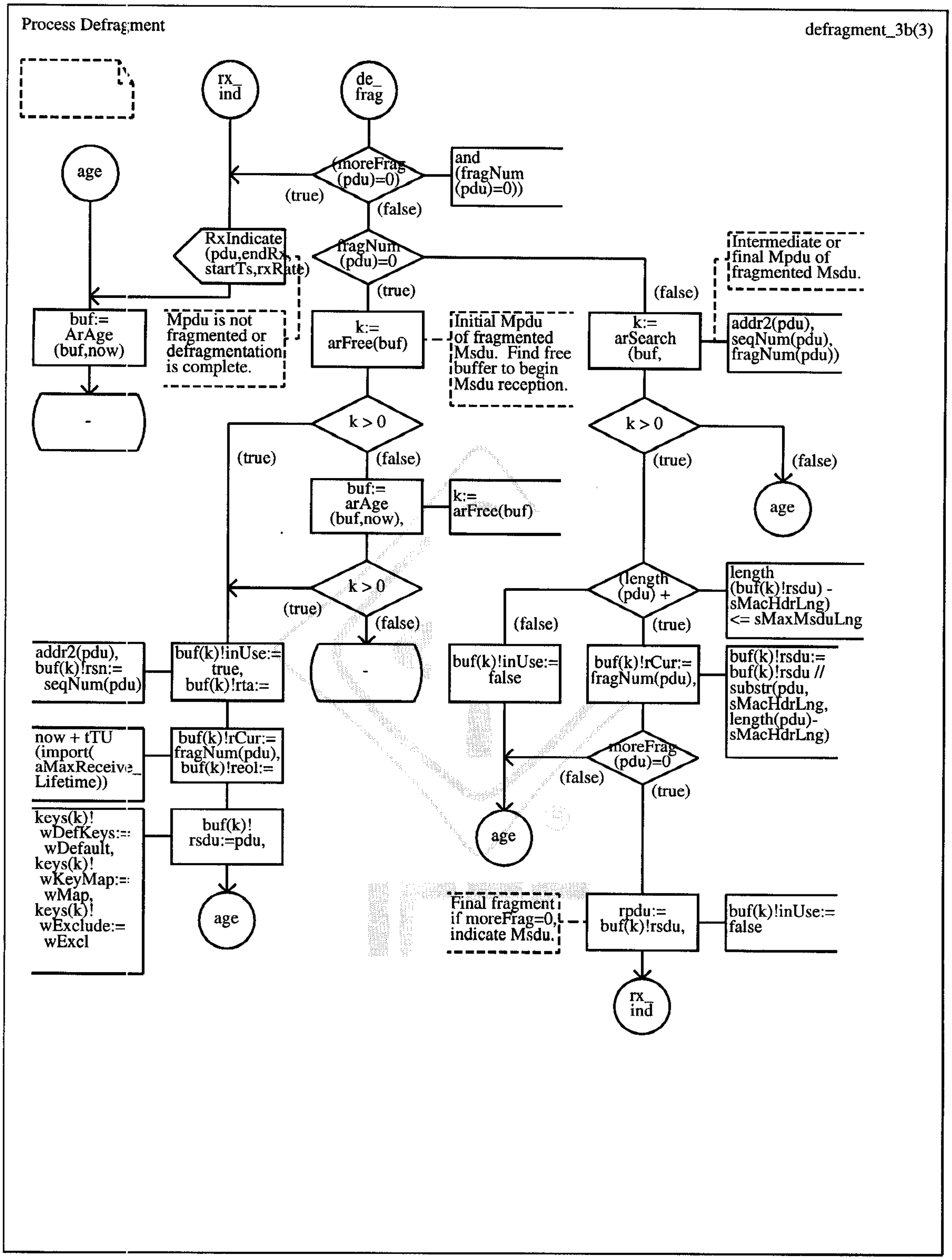

RX
PS
Block Reception
[RxIndicate]
[PsIndicate]
receive_1a(1)
FromCtl
[NeedAck, RxCfAck, RxCfPoll]
ToRx
Includes decryption if aPrivacyOptionImplemented =true. This is a typical location, but implementors may use other locations between the PHY_SAP_RX and MAC_SAP as long as they provide the specified behavior as observed at LLC, MLME and the WM.
Defragment (1,1)
/* also decrypt */
[RxMpdu]
/* This block handles octet-level reception of MPDUs from the PHY, and validation, filtering, and decryption needed so higher blocks have uniform, error-free information from the relevant rx events. This block also maintains the MAC's view of channel state, including the NAV (and remote variable mNavEnd), rx activity (and the remote variable mRxA), and slot timing (providing the Busy, Idle and Slot signals to the Transmission block). */
FromSync
[ChangeNav]
Defrag
[ChangeNav]
IndAck
ToPs
Filter_MPDU (1,1)
UpdNav
CS
ToTx
Channel_State (1,1)
[Busy, Idle, Slot]
[SetNav, ClearNav]
[RxMpdu]
[PhyCca.indication, PhyCcarst.confirm]
[RtsTimeout, UseDifs, UseEifs]
Filter
signal ClearNav(NavSrc), RtsTimeout, RxMpdu(Frame,Time,Time, Rate,Boolean, KeyVector,KeyMapArray), SetNav(Time, Duration,NavSrc), UseDifs(Time), UseEifs(Time) ;
IfsCtl
Validate_MPDU (1,1)
[PhyRxStart.indication, PhyRxEnd.indication, PhyData.indication]
FromPHY
PhyCca
[PhyCcarst.request]
PHY_SAP_RX

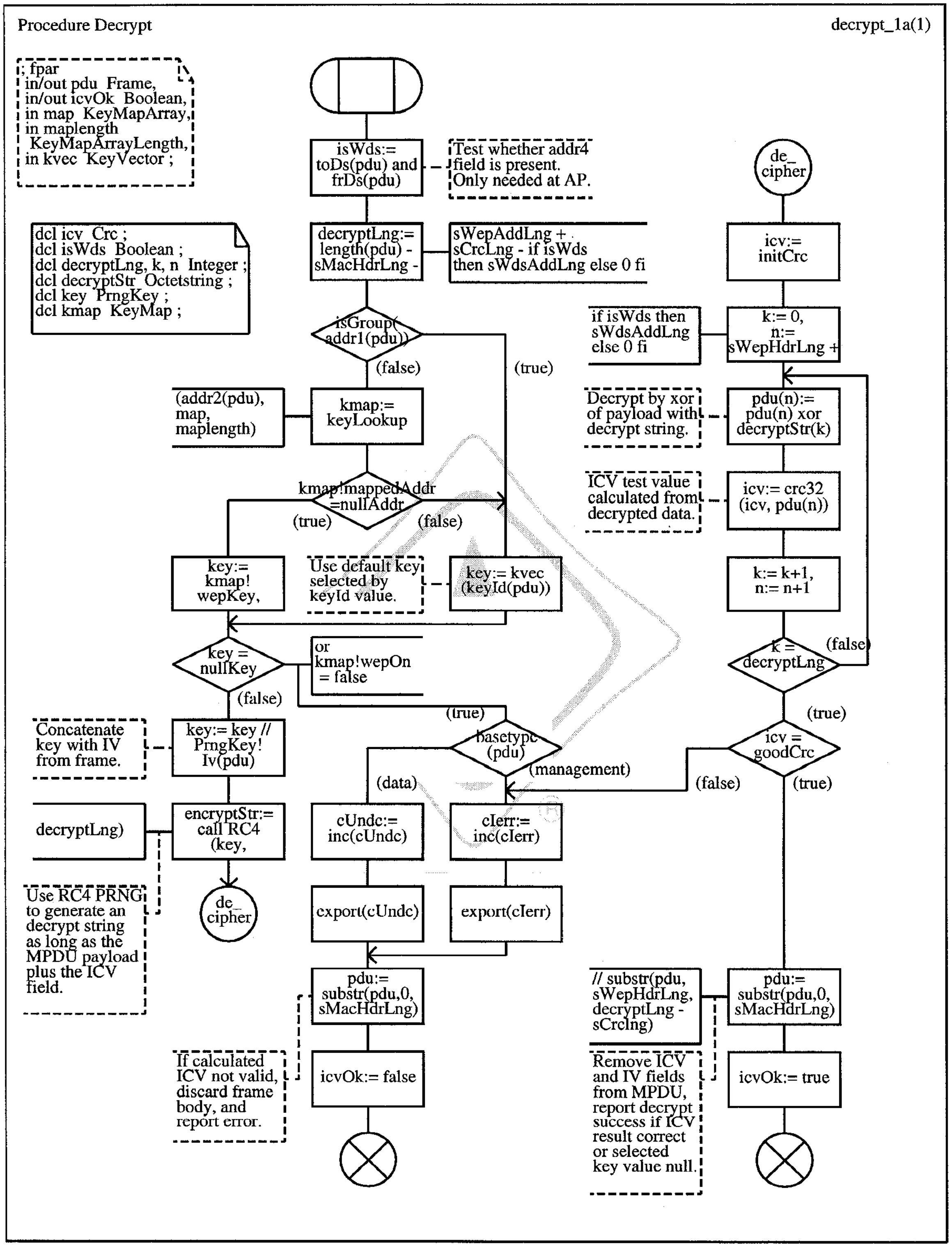

Process Channel_State
nav_clear_1b(2)
*
ResetMAC
timer Tifs ;
timer Tnav ;
timer Tslot ;
dcl cs CcaStatus ;
dcl rxtx, slot, sifs Integer ;
dcl dDifs, dEifs, dIfs, dNav,
dRxTx, dSifs, dSlot Duration ;
dcl tNew, tRef, tRxEnd Time ;
dcl newSrc, curSrc NavSrc ;
dcl exported
tNavEnd as
mNavEnd Time ;
dSifs:=
dUsec
(aSifsTime),
dRxTx:=dUsec
(aRxTxTurn_
aroundTime)
Eifs based
on the lowest
basic rate.
Wait_IFS
/* IDLE */
ClearNav, RtsTimeout,
Tnav, Tslot ignored
in Wait_IFS state.
dSlot:=
dUsec
(aSlotTime),
dDifs:=dSifs +
(2 * dSlot)
not
active(Tifs)
Tifs
PhyCca._
indication
(cs)
SetNav
(tRef,dNav
curSrc)
dEifs:=
dUsec
(aSifsTime +
calcDur(first(
import(mBrates)),
stuff(aMpdu_
DurationFactor,
sAckCtsLng) +
aPlcpHdrLength
+ aPreamble_
Length) + dDifs)
cs
(idle)
(busy)
dIfs:=
dEifs
-
Cs_noNav
/* BUSY */
ClearNav, Tnav, Tifs,
RtsTimeout, Tslot
ignored in this state.
cs:= busy,
tNavEnd:=0
Assume channel
busy until Phy
indicates idle.
tNavEnd is =0
until first rx
that sets Nav.
Idle
Idle signal is
sent at end of
the M2 interval
(Figure 47).
PhyCca._
indication
(cs)
SetNav
(tRef,dNav
curSrc)
reset(Tnav)
set
(now+dSlot,
Tslot)
cs
(idle)
(busy)
tNavEnd:=
tRef+dNav
PhyCcarst._
request
PhyCcareset._
confirm is
ignored.
noCs_noNav
/* IDLE */
RtsTimeout,
Tnav, ClearNav,
Tifs ignored
in this state.
-
set
(tNavEnd,
Tnav)
export
(tNavEnd)
PhyCca._
indication
(cs)
SetNav
(tRef,dNav
curSrc)
Tslot
set
(now+dIfs,
Tifs)
export
(tNavEnd)
curSrc:=
nosrc
tNavEnd:=
tRef+dNav
Slot
Wait_IFS
Cs_Nav
cs
(busy)
(idle)
set
(tNavEnd,
Tnav)
set
(now+dSlot,
Tslot)
Slot signal is
generated at
the end of each
M2 interval
(Fig. 47) while
channel is idle.
Busy
-
Busy
-
/* This process
maintains channel
state based on
both physical and
virtual carrier
sense, generates
slot time reference,
and provides Busy,
Idle & Slot signals
to Transmission. */
Cs_noNav
export
(tNavEnd)
noCs_Nav

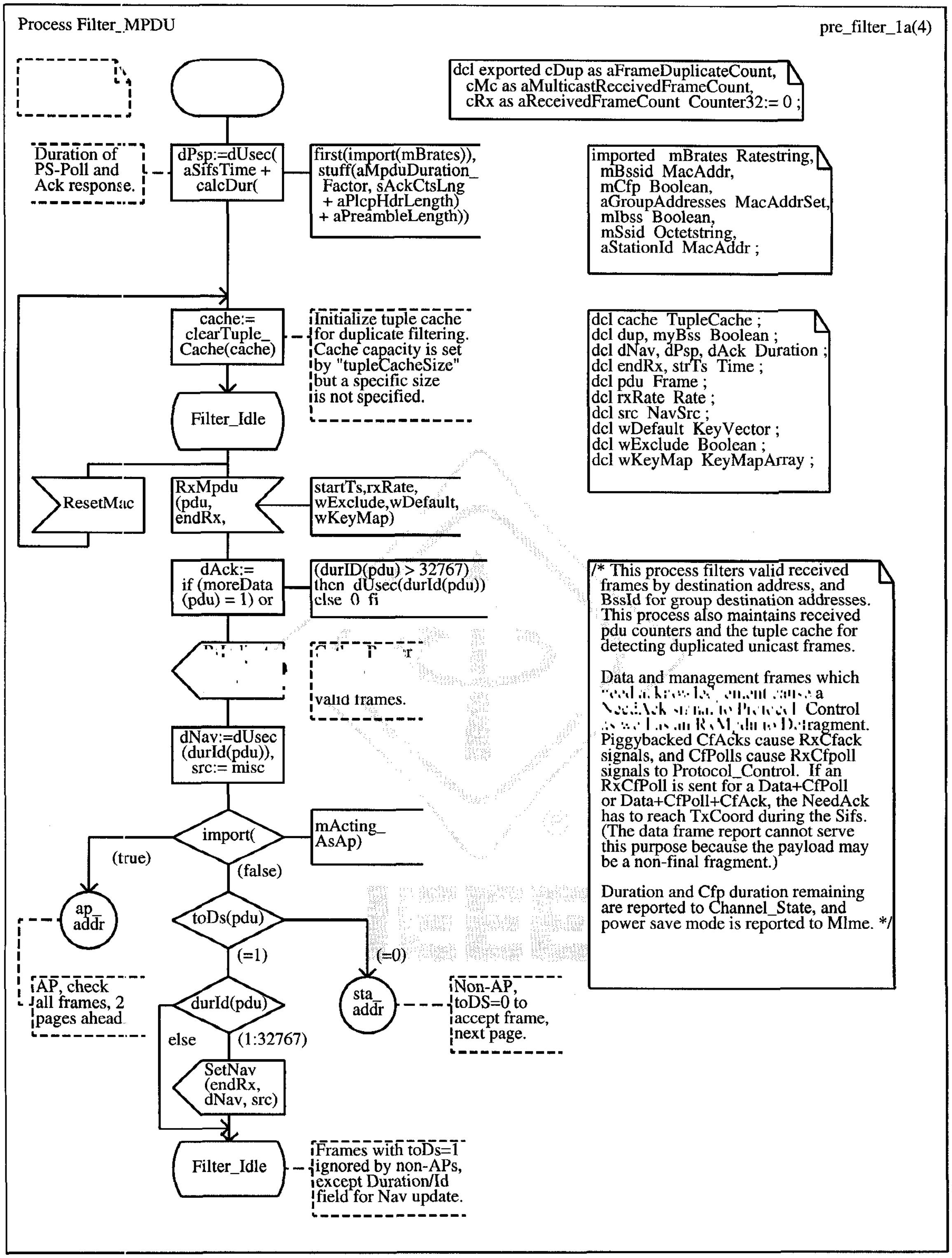

Process Channel_State
nav_set_2b(2)
noCs_Nav /* BUSY */
Tslot and Tifs ignored in noCs_Nav state.
Cs_Nav /* BUSY */
Tslot and Tifs ignored in Cs_Nav state.
PhyCca._ indication (cs)
Tnav
cs
(busy)
(idle)
Cs_Nav
-
PhyCcarst._ request
curSrc:= nosrc
set (now+dIfs, Tifs)
Wait_IFS
noCs_Nav
Cs_noNav
* /* all states */
ChangeNav is SetNav if not channel switch.
noCs_Nav, Cs_Nav /* all NAV */
Clearing NAV on RTS timeout is optional (9.2.5.4).
UseEifs (tRxEnd)
UseDifs (tRxEnd)
ChangeNav (tRef,dNav, newSrc)
SetNav (tRef,dNav, newSrc)
Rts_ Timeout
ClearNav (newSrc)
dIfs:= dEifs-dRxTx
dIfs:= dDifs-dRxTx
newSrc= cswitch
(false)
(true)
tNew:= tRef+dNav
curSrc= rts
set (tRxEnd+dIfs, Tifs)
Clear NAV and use EIFS after channel change.
dIfs:= dEifs-dRxTx
tNew> tNavEnd
tNavEnd:= tNew, curSrc:=newSrc
The initial dIfs value is dEifs, set by a UseEifs signal generated by Validate_Mpdu at startup and due to ResetMAC.
set(now,Tnav)
tNavEnd:= now, curSrc:=nosrc
tNavEnd:=0, curSrc:=nosrc
set(tNavEnd, Tnav)
tNavEnd is =0 until first rx on new channel.
Nav is cleared by setting Tnav to now. This causes immediate Tnav signal to enable exit from noCs_Nav or Cs_Nav state.
export (tNavEnd)

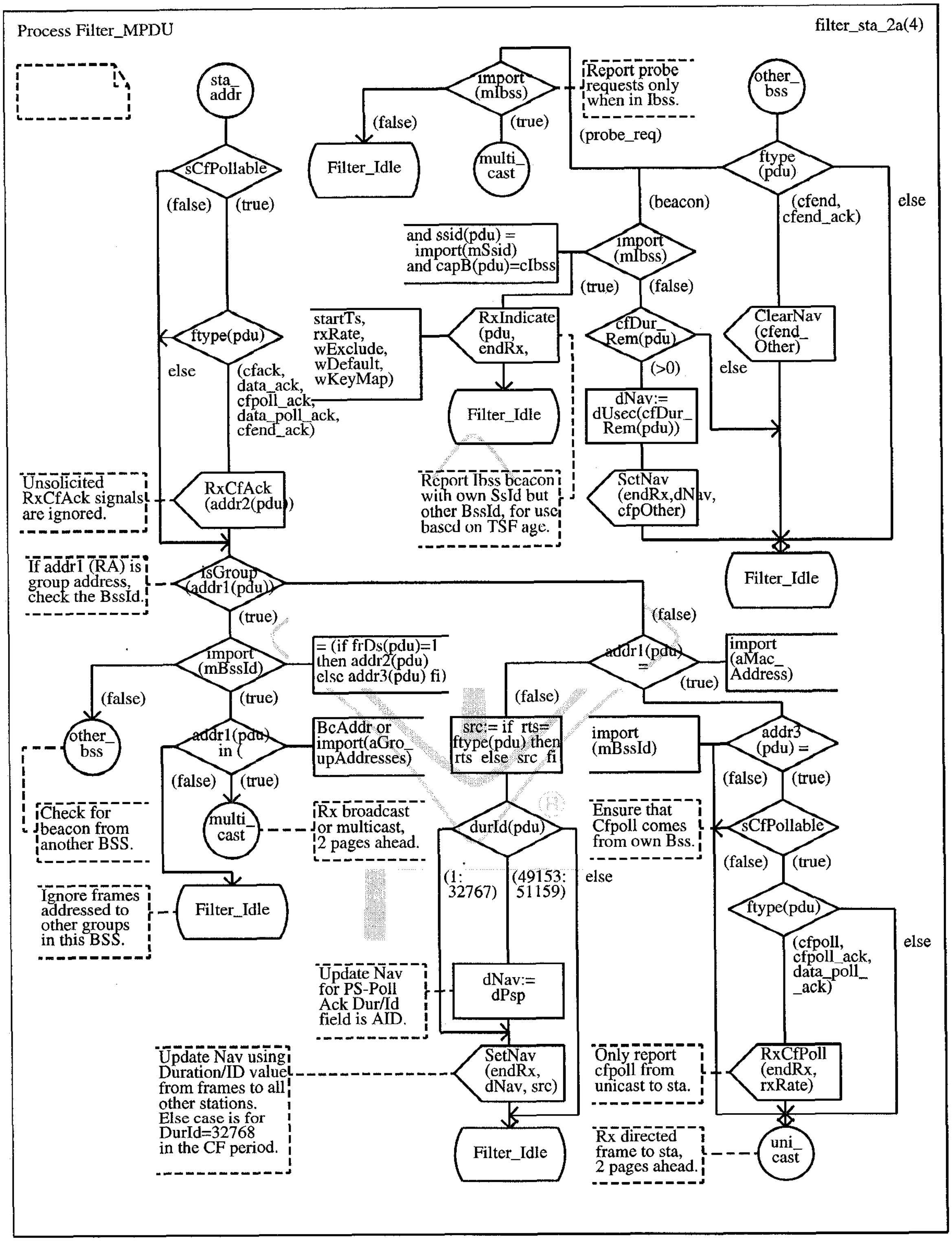

Process Defragment
wep_filter_1a(3)
Decrypt
dcl exported cIerr as aWepIcvErrorCount, cUndc as aWepUndecryptableCount, cExcl as aWepExcludedCount Counter32:= 0 ;
dLife:= dTU(import
(aMax_ Receive_ Lifetime))
imported mCfp Boolean ;
imported aMaxReceiveLifetime TU ;
imported procedure RC4 ; fpar PrngKey, Integer ; returns Octetstring ;
export(cIerr, cUndc, cExcl)
dcl buf DefragArray ;
dcl dLife Duration ;
dcl endRx, startTs Time ;
dcl icvOk Boolean ;
dcl k DefragIndex ;
dcl keys DefragKeysArray ;
dcl pri CfPriority ;
dcl pdu, sdu Frame ;
dcl wExcl Boolean ;
dcl wDefault KeyVector ;
dcl wMap KeyMapArray ;
buf:= ArAge(buf, now+dLife+1)
Defragmentation buffers forced empty using the aging function.
Defrag_ Inactive
not import (mDisable)
mDisable=false when started or joined Bss.
RxMpdu (pdu, endRx,
startTs,rxRate, wExcl,wDefault, wMap)
ftype (pdu)
When not in Bss only pass beacon and probe_rsp.
else
(beacon, probe_rsp)
RxIndicate (pdu,endRx, startTs,rxRate
-
Defrag_ Idle
import (mDisable)
RxMpdu (pdu, endRx,
startTs,rxRate, wExcl,wDefault, wMap)
basetype (pdu)
(control)
(management)
else
wepBit (pdu)
(=0)
(=1)
rx_ ind
wepBit (pdu)
(=1)
(=0)
import(
aPrivacy_ Option_ Implemented)
wExcl
(false)
(true)
(true)
(false)
(false)
auth) and authSeqNum (pdu)=3) and import(aPrivacy_ Option_ Implemented)
(ftype (pdu)=
(true)
de_ crypt
cUndc:= inc(cUndc)
de_ crypt
cExcl:= inc(cExcl)
de_ frag
export(cUndc)
export(cExcl)
-
-

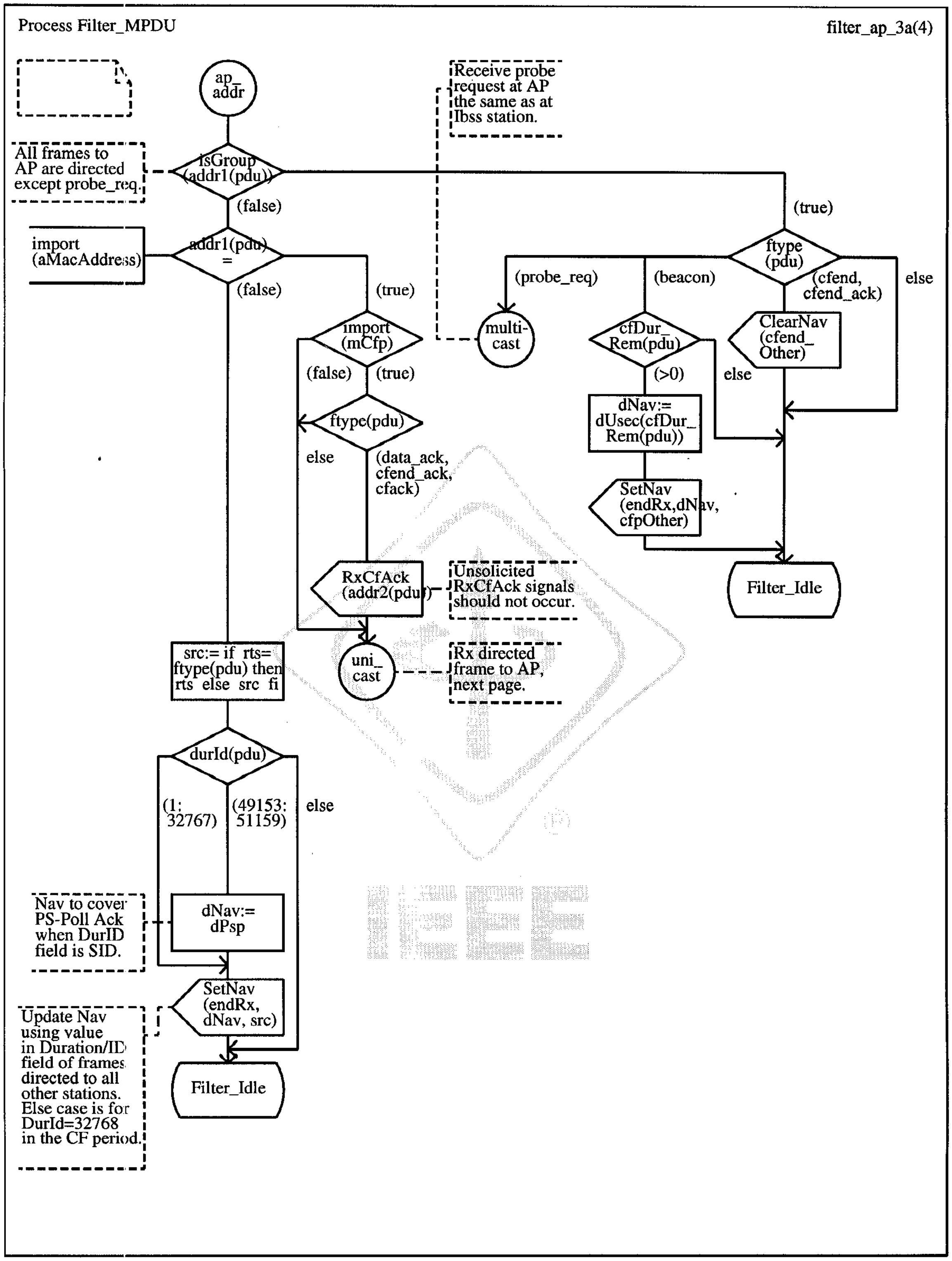

Process Defragment
wep_decrypt_2a(3)
de_
crypt
Decrypt
(pdu,
icvOk,
wMap, sKey_
MappingLength,
wDefault)
icvOk
Icv errors and
certain undecryptable
errors counted in
Decrypt procedure.
(true)
(false)
de_
frag
ftype
(pdu)
else
(auth)
RxIndicate
(pdu,endR
Authentication
challenge resposnes
with Icv errors
are reported, but
Decrypt removes
payload so Auth
service is able
to distinguish
a bad key from
a non-response.
Do not report
receipt of
data frames
with Icv errors.
-

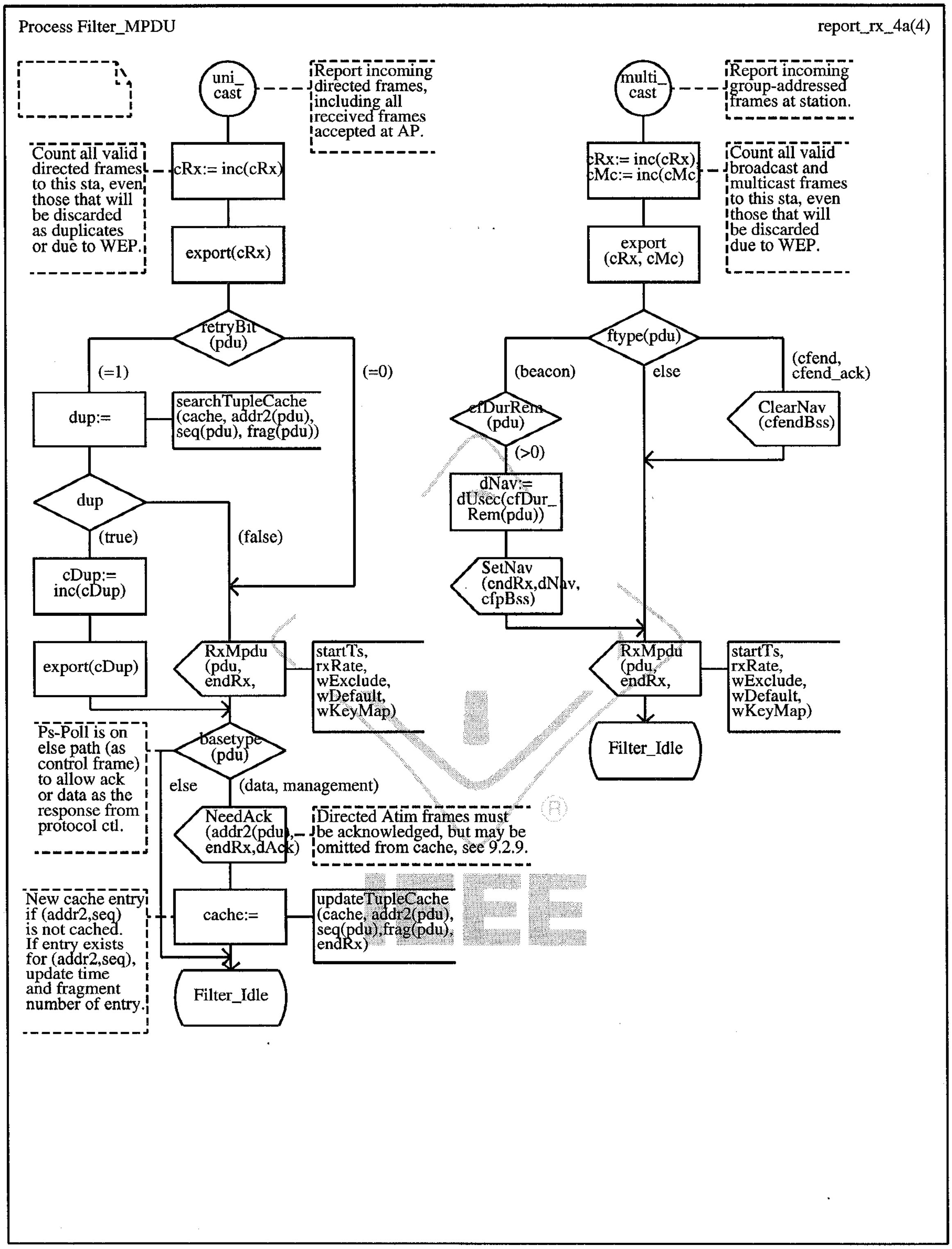

Process Defragment
defragment_3b(3)
rx_ ind
de_ frag
age
(moreFrag (pdu)=0)
and (fragNum (pdu)=0))
(true)
(false)
RxIndicate (pdu,endRx, startTs,rxRate)
fragNum (pdu)=0
(true)
(false)
Intermediate or final Mpdu of fragmented Msdu.
buf:= ArAge (buf,now)
Mpdu is not fragmented or defragmentation is complete.
k:= arFree(buf)
Initial Mpdu of fragmented Msdu. Find free buffer to begin Msdu reception.
k:= arSearch (buf,
addr2(pdu), seqNum(pdu), fragNum(pdu))
-
k > 0
(true)
(false)
k > 0
(true)
(false)
buf:= arAge (buf,now),
k:= arFree(buf)
age
k > 0
(true)
(false)
(length (pdu) +
length (buf(k)!rsdu) - sMacHdrLng) <= sMaxMsduLng
(false)
(true)
addr2(pdu), buf(k)!rsn:= seqNum(pdu)
buf(k)!inUse:= true, buf(k)!rta:=
-
buf(k)!inUse:= false
buf(k)!rCur:= fragNum(pdu),
buf(k)!rsdu:= buf(k)!rsdu // substr(pdu, sMacHdrLng, length(pdu)- sMacHdrLng)
now + tTU (import(aMaxReceive_ Lifetime))
buf(k)!rCur:= fragNum(pdu), buf(k)!reol:=
moreFrag (pdu)=0
(false)
(true)
keys(k)! wDefKeys:= wDefault, keys(k)! wKeyMap:= wMap, keys(k)! wExclude:= wExcl
buf(k)! rsdu:=pdu,
age
age
Final fragment if moreFrag=0, indicate Msdu.
rpdu:= buf(k)!rsdu,
buf(k)!inUse:= false
rx_ ind

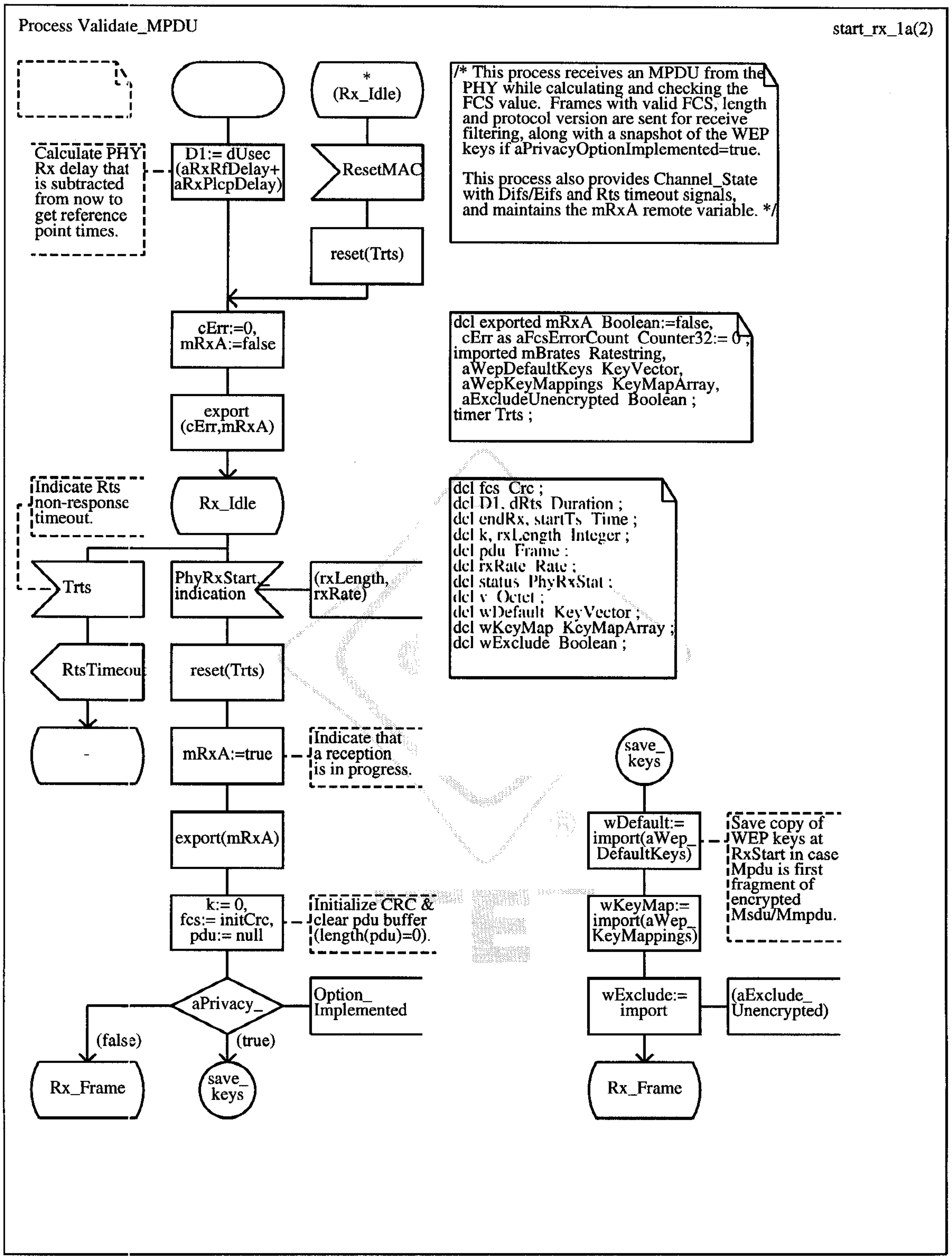

Procedure Decrypt
decrypt_1a(1)
; fpar
in/out pdu Frame,
in/out icvOk Boolean,
in map KeyMapArray,
in maplength
KeyMapArrayLength,
in kvec KeyVector ;
dcl icv Crc ;
dcl isWds Boolean ;
dcl decryptLng, k, n Integer ;
dcl decryptStr Octetstring ;
dcl key PrngKey ;
dcl kmap KeyMap ;
isWds:=
toDs(pdu) and
frDs(pdu)
Test whether addr4
field is present.
Only needed at AP.
decryptLng:=
length(pdu) -
sMacHdrLng -
sWepAddLng +
sCrcLng - if isWds
then sWdsAddLng else 0 fi
isGroup(
addr1(pdu))
(false)
(true)
(addr2(pdu),
map,
maplength)
kmap:=
keyLookup
kmap!mappedAddr
=nullAddr
(true)
(false)
key:=
kmap!
wepKey,
Use default key
selected by
keyId value.
key:= kvec
(keyId(pdu))
key =
nullKey
or
kmap!wepOn
= false
(false)
(true)
Concatenate
key with IV
from frame.
key:= key //
PrngKey!
Iv(pdu)
decryptLng)
encryptStr:=
call RC4
(key,
Use RC4 PRNG
to generate an
decrypt string
as long as the
MPDU payload
plus the ICV
field.
de_
cipher
basetype
(pdu)
(data)
(management)
cUndc:=
inc(cUndc)
cIerr:=
inc(cIerr)
export(cUndc)
export(cIerr)
pdu:=
substr(pdu,0,
sMacHdrLng)
If calculated
ICV not valid,
discard frame
body, and
report error.
icvOk:= false
de_
cipher
icv:=
initCrc
if isWds then
sWdsAddLng
else 0 fi
k:= 0,
n:=
sWepHdrLng +
Decrypt by xor
of payload with
decrypt string.
pdu(n):=
pdu(n) xor
decryptStr(k)
ICV test value
calculated from
decrypted data.
icv:= crc32
(icv, pdu(n))
k:= k+1,
n:= n+1
k =
decryptLng
(false)
(true)
icv =
goodCrc
(false)
(true)
// substr(pdu,
sWepHdrLng,
decryptLng -
sCrclng)
pdu:=
substr(pdu,0,
sMacHdrLng)
Remove ICV
and IV fields
from MPDU,
report decrypt
success if ICV
result correct
or selected
key value null.
icvOk:= true

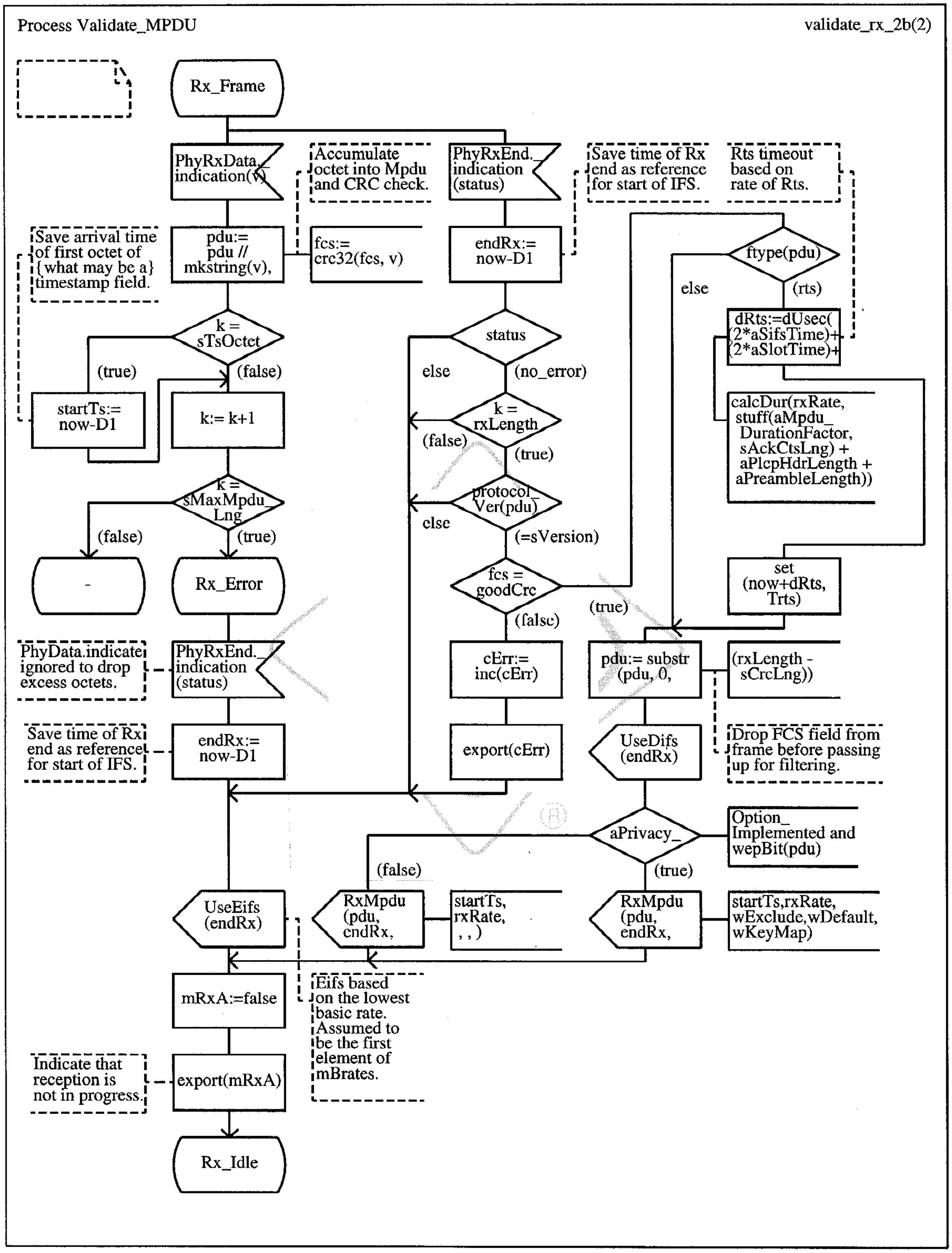

Process Filter_MPDU
pre_filter_1a(4)
dcl exported cDup as aFrameDuplicateCount, cMc as aMulticastReceivedFrameCount, cRx as aReceivedFrameCount Counter32:= 0 ;
Duration of PS-Poll and Ack response.
dPsp:=dUsec(aSifsTime + calcDur(
first(import(mBrates)), stuff(aMpduDuration_ Factor, sAckCtsLng + aPlcpHdrLength) + aPreambleLength))
imported mBrates Ratestring, mBssid MacAddr, mCfp Boolean, aGroupAddresses MacAddrSet, mIbss Boolean, mSsid Octetstring, aStationId MacAddr ;
cache:= clearTuple_ Cache(cache)
Initialize tuple cache for duplicate filtering. Cache capacity is set by "tupleCacheSize" but a specific size is not specified.
dcl cache TupleCache ;
dcl dup, myBss Boolean ;
dcl dNav, dPsp, dAck Duration ;
dcl endRx, strTs Time ;
dcl pdu Frame ;
dcl rxRate Rate ;
dcl src NavSrc ;
dcl wDefault KeyVector ;
dcl wExclude Boolean ;
dcl wKeyMap KeyMapArray ;
Filter_Idle
ResetMac
RxMpdu (pdu, endRx,
startTs,rxRate, wExclude,wDefault, wKeyMap)
dAck:= if (moreData (pdu) = 1) or
(durID(pdu) > 32767) then dUsec(durId(pdu)) else 0 fi
valid frames.
/* This process filters valid received frames by destination address, and BssId for group destination addresses. This process also maintains received pdu counters and the tuple cache for detecting duplicated unicast frames.
Data and management frames which
Control
Piggybacked CfAcks cause RxCfack signals, and CfPolls cause RxCfpoll signals to Protocol_Control. If an RxCfPoll is sent for a Data+CfPoll or Data+CfPoll+CfAck, the NeedAck has to reach TxCoord during the Sifs. (The data frame report cannot serve this purpose because the payload may be a non-final fragment.)
Duration and Cfp duration remaining are reported to Channel_State, and power save mode is reported to Mlme. */
dNav:=dUsec (durId(pdu)), src:= misc
import(
mActing_ AsAp)
(true)
(false)
ap_ addr
toDs(pdu)
(=1)
(=0)
AP, check all frames, 2 pages ahead
durId(pdu)
sta_ addr
Non-AP, toDS=0 to accept frame, next page.
else
(1:32767)
SetNav (endRx, dNav, src)
Filter_Idle
Frames with toDs=1 ignored by non-APs, except Duration/Id field for Nav update.

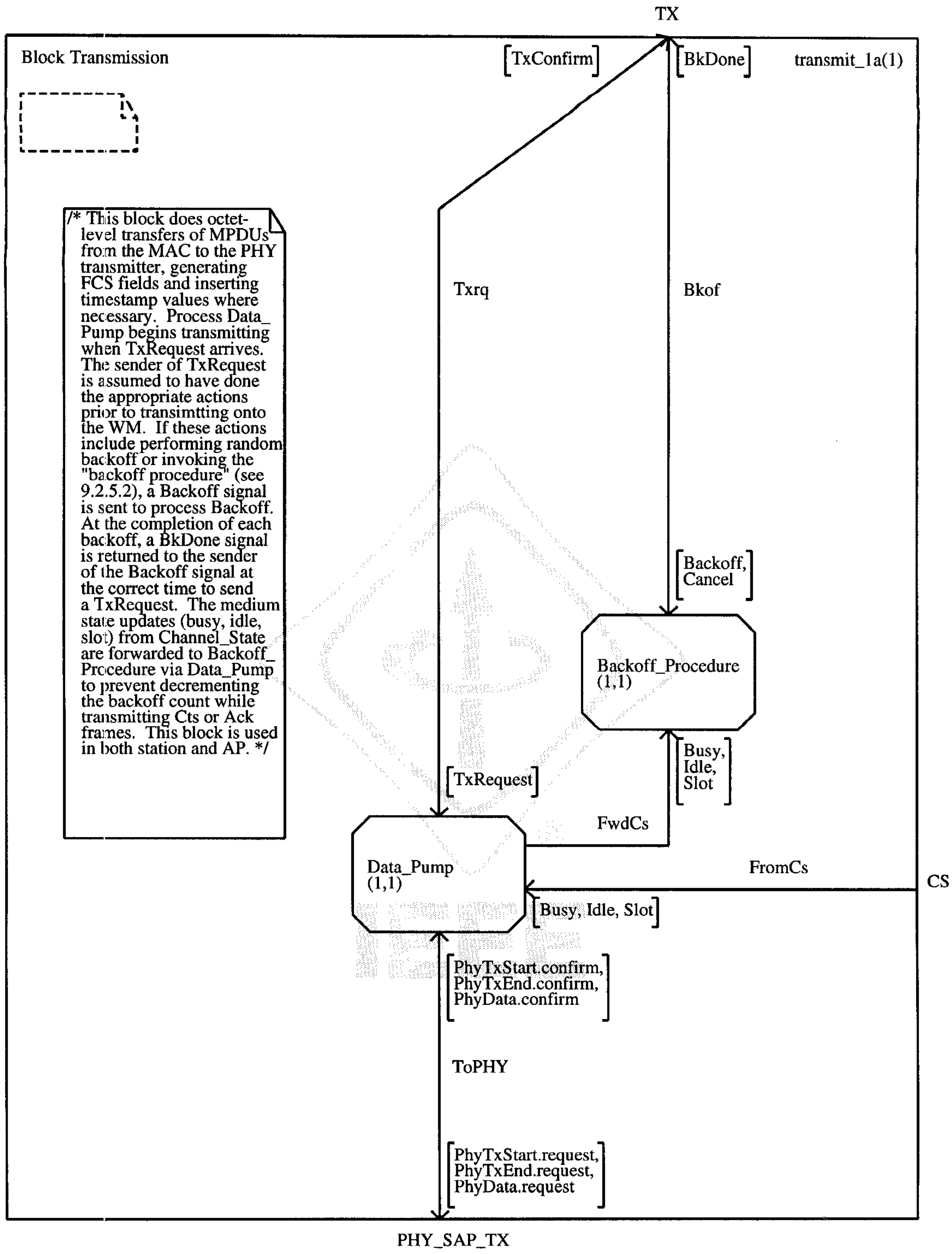

Process Filter_MPDU
filter_sta_2a(4)
sta_ addr
sCfPollable
(false)
(true)
ftype(pdu)
else
(cfack, data_ack, cfpoll_ack, data_poll_ack, cfend_ack)
Unsolicited RxCfAck signals are ignored.
RxCfAck (addr2(pdu))
import (mIbss)
(false)
(true)
Filter_Idle
multi_ cast
Report probe requests only when in Ibss.
(probe_req)
other_ bss
ftype (pdu)
(beacon)
(cfend, cfend_ack)
else
and ssid(pdu) = import(mSsid) and capB(pdu)=cIbss
import (mIbss)
(true)
(false)
startTs, rxRate, wExclude, wDefault, wKeyMap)
RxIndicate (pdu, endRx,
Filter_Idle
cfDur_ Rem(pdu)
(>0)
else
dNav:= dUsec(cfDur_ Rem(pdu))
ClearNav (cfend_ Other)
Report Ibss beacon with own SsId but other BssId, for use based on TSF age.
SetNav (endRx,dNav, cfpOther)
Filter_Idle
If addr1 (RA) is group address, check the BssId.
isGroup (addr1(pdu))
(true)
(false)
import (mBssId)
= (if frDs(pdu)=1 then addr2(pdu) else addr3(pdu) fi)
addr1(pdu) =
import (aMac_ Address)
(true)
(false)
other_ bss
Check for beacon from another BSS.
addr1(pdu) in (
BcAddr or import(aGroupAddresses)
(false)
(true)
multi_ cast
Rx broadcast or multicast, 2 pages ahead.
Ignore frames addressed to other groups in this BSS.
Filter_Idle
src:= if rts= ftype(pdu) then rts else src fi
import (mBssId)
addr3 (pdu) =
(false)
(true)
durId(pdu)
(1: 32767)
(49153: 51159)
else
Ensure that Cfpoll comes from own Bss.
sCfPollable
(false)
(true)
ftype(pdu)
(cfpoll, cfpoll_ack, data_poll_ _ack)
else
Update Nav for PS-Poll Ack Dur/Id field is AID.
dNav:= dPsp
Update Nav using Duration/ID value from frames to all other stations. Else case is for DurId=32768 in the CF period.
SetNav (endRx, dNav, src)
Filter_Idle
Only report cfpoll from unicast to sta.
RxCfPoll (endRx, rxRate)
Rx directed frame to sta, 2 pages ahead.
uni_ cast

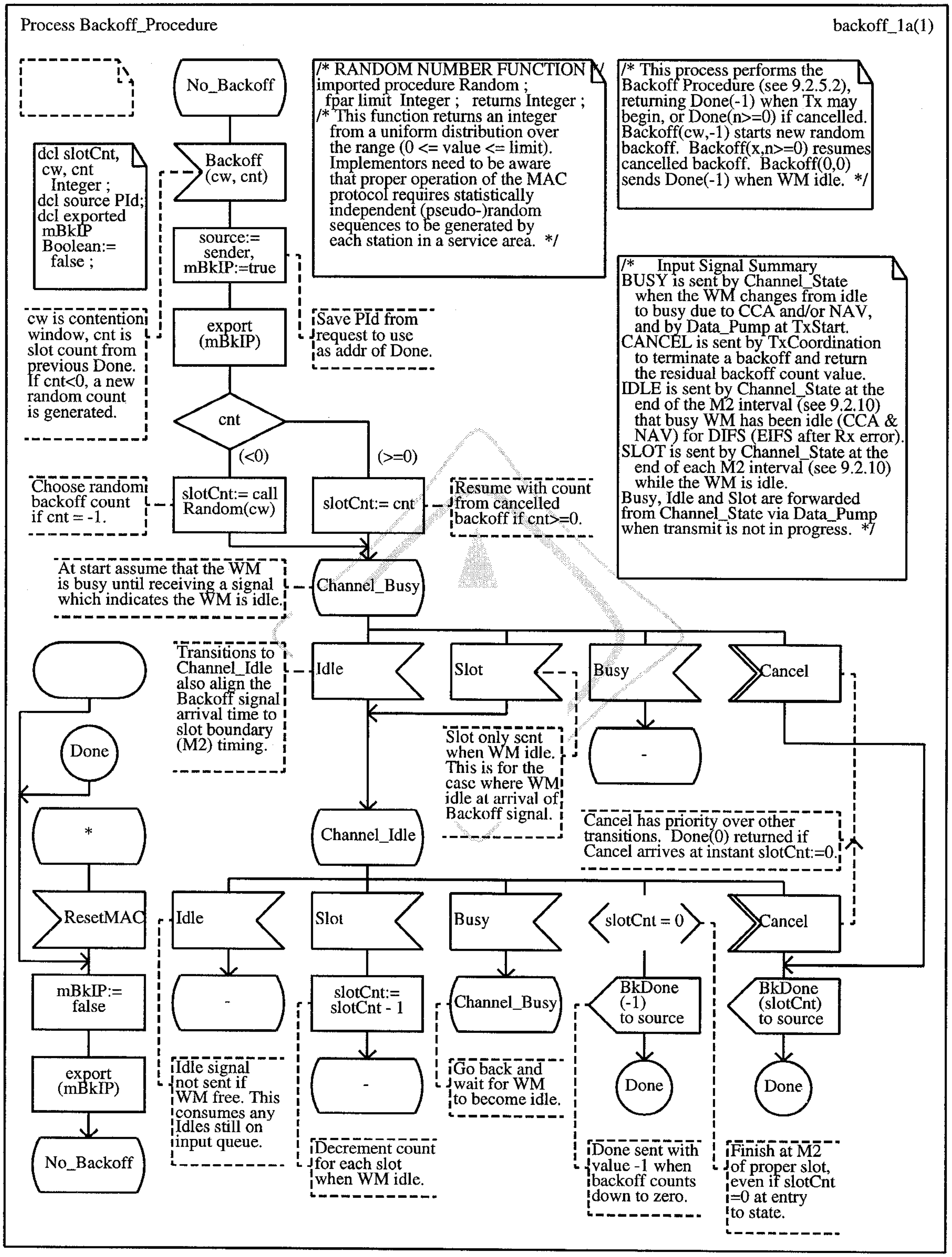

Process Filter_MPDU
filter_ap_3a(4)
ap_
addr
Receive probe
request at AP
the same as at
Ibss station.
All frames to
AP are directed
except probe_req.
isGroup
(addr1(pdu))
(false)
(true)
import
(aMacAddress)
addr1(pdu)
=
(false)
(true)
ftype
(pdu)
(probe_req)
(beacon)
(cfend,
cfend_ack)
else
import
(mCfp)
multi-
cast
cfDur_
Rem(pdu)
ClearNav
(cfend_
Other)
(false)
(true)
(>0)
else
ftype(pdu)
dNav:=
dUsec(cfDur_
Rem(pdu))
else
(data_ack,
cfend_ack,
cfack)
SetNav
(endRx,dNav,
cfpOther)
RxCfAck
(addr2(pdu))
Unsolicited
RxCfAck signals
should not occur.
Filter_Idle
src:= if rts=
ftype(pdu) then
rts else src fi
uni_
cast
Rx directed
frame to AP,
next page.
durId(pdu)
(1:
32767)
(49153:
51159)
else
Nav to cover
PS-Poll Ack
when DurID
field is SID.
dNav:=
dPsp
SetNav
(endRx,
dNav, src)
Update Nav
using value
in Duration/ID
field of frames
directed to all
other stations.
Else case is for
DurId=32768
in the CF period.
Filter_Idle

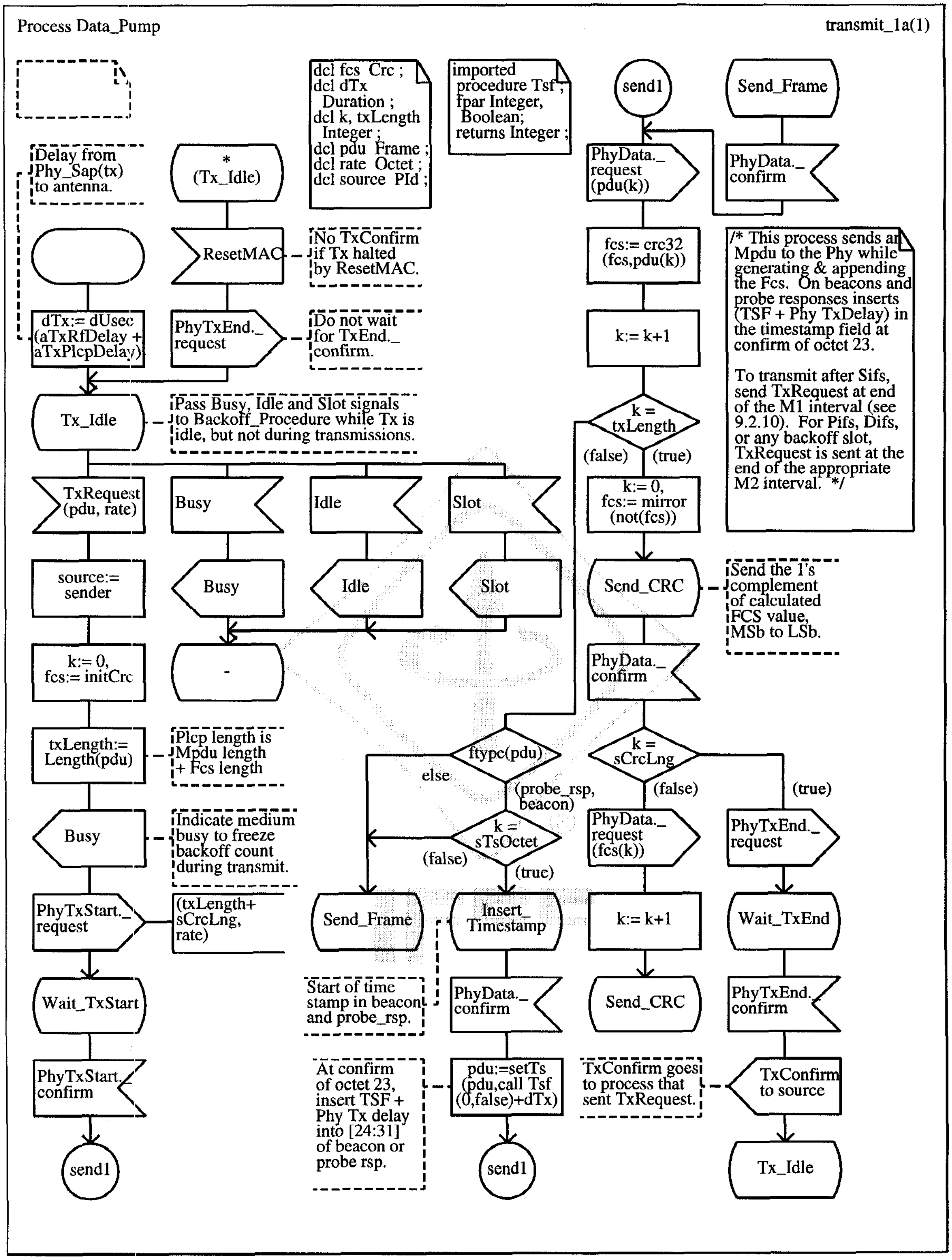

Process Filter_MPDU
report_rx_4a(4)
uni_ cast
Report incoming directed frames, including all received frames accepted at AP.
Count all valid directed frames to this sta, even those that will be discarded as duplicates or due to WEP.
cRx:= inc(cRx)
export(cRx)
retryBit (pdu)
(=1)
(=0)
dup:=
searchTupleCache (cache, addr2(pdu), seq(pdu), frag(pdu))
dup
(true)
(false)
cDup:= inc(cDup)
export(cDup)
RxMpdu (pdu, endRx,
startTs, rxRate, wExclude, wDefault, wKeyMap)
Ps-Poll is on else path (as control frame) to allow ack or data as the response from protocol ctl.
basetype (pdu)
else
(data, management)
NeedAck (addr2(pdu), endRx,dAck)
Directed Atim frames must be acknowledged, but may be omitted from cache, see 9.2.9.
New cache entry if (addr2,seq) is not cached. If entry exists for (addr2,seq), update time and fragment number of entry.
cache:=
updateTupleCache (cache, addr2(pdu), seq(pdu),frag(pdu), endRx)
Filter_Idle
multi_ cast
Report incoming group-addressed frames at station.
cRx:= inc(cRx); cMc:= inc(cMc)
Count all valid broadcast and multicast frames to this sta, even those that will be discarded due to WEP.
export (cRx, cMc)
ftype(pdu)
(beacon)
else
(cfend, cfend_ack)
cfDurRem (pdu)
(>0)
ClearNav (cfendBss)
dNav:= dUsec(cfDur_ Rem(pdu))
SetNav (endRx,dNav, cfpBss)
RxMpdu (pdu, endRx,
startTs, rxRate, wExclude, wDefault, wKeyMap)
Filter_Idle


Process Validate_MPDU
start_rx_1a(2)
* (Rx_Idle)
ResetMAC
reset(Trts)
/* This process receives an MPDU from the PHY while calculating and checking the FCS value. Frames with valid FCS, length and protocol version are sent for receive filtering, along with a snapshot of the WEP keys if aPrivacyOptionImplemented=true.
This process also provides Channel_State with Difs/Eifs and Rts timeout signals, and maintains the mRxA remote variable. */
Calculate PHY Rx delay that is subtracted from now to get reference point times.
D1:= dUsec (aRxRfDelay+ aRxPlcpDelay)
cErr:=0, mRxA:=false
export (cErr,mRxA)
dcl exported mRxA Boolean:=false, cErr as aFcsErrorCount Counter32:= 0, imported mBrates Ratestring, aWepDefaultKeys KeyVector, aWepKeyMappings KeyMapArray, aExcludeUnencrypted Boolean ; timer Trts ;
Indicate Rts non-response timeout.
Rx_Idle
dcl fcs Crc ; dcl D1, dRts Duration ; dcl endRx, startTs Time ; dcl k, rxLength Integer ; dcl pdu Frame ; dcl rxRate Rate ; dcl status PhyRxStat ; dcl v Octet ; dcl wDefault KeyVector ; dcl wKeyMap KeyMapArray ; dcl wExclude Boolean ;
Trts
PhyRxStart. indication
(rxLength, rxRate)
RtsTimeou
reset(Trts)
-
mRxA:=true
Indicate that a reception is in progress.
save_ keys
export(mRxA)
wDefault:= import(aWep_ DefaultKeys)
Save copy of WEP keys at RxStart in case Mpdu is first fragment of encrypted Msdu/Mmpdu.
k:= 0, fcs:= initCrc, pdu:= null
Initialize CRC & clear pdu buffer (length(pdu)=0).
wKeyMap:= import(aWep_ KeyMappings)
aPrivacy_
Option_ Implemented
wExclude:= import
(aExclude_ Unencrypted)
(false)
(true)
Rx_Frame
save_ keys
Rx_Frame


Process Validate_MPDU
validate_rx_2b(2)
Rx_Frame
PhyRxData_ indication(v)
Accumulate octet into Mpdu and CRC check.
PhyRxEnd_ indication (status)
Save time of Rx end as reference for start of IFS.
Rts timeout based on rate of Rts.
Save arrival time of first octet of {what may be a} timestamp field.
pdu:= pdu // mkstring(v),
fcs:= crc32(fcs, v)
endRx:= now-D1
ftype(pdu)
else
(rts)
dRts:=dUsec((2*aSifsTime)+ (2*aSlotTime)+
calcDur(rxRate, stuff(aMpdu_ DurationFactor, sAckCtsLng) + aPlcpHdrLength + aPreambleLength))
k = sTsOctet
(true)
(false)
startTs:= now-D1
k:= k+1
status
else
(no_error)
k = rxLength
(false)
(true)
protocol_ Ver(pdu)
else
(=sVersion)
k = sMaxMpdu_ Lng
(false)
(true)
-
Rx_Error
fcs = goodCrc
(true)
(false)
set (now+dRts, Trts)
PhyData.indicate ignored to drop excess octets.
PhyRxEnd_ indication (status)
cErr:= inc(cErr)
pdu:= substr (pdu, 0,
(rxLength - sCrcLng))
Save time of Rx end as reference for start of IFS.
endRx:= now-D1
export(cErr)
UseDifs (endRx)
Drop FCS field from frame before passing up for filtering.
aPrivacy_
Option_ Implemented and wepBit(pdu)
(false)
(true)
UseEifs (endRx)
RxMpdu (pdu, endRx,
startTs, rxRate, , ,)
RxMpdu (pdu, endRx,
startTs,rxRate, wExclude,wDefault, wKeyMap)
mRxA:=false
Eifs based on the lowest basic rate. Assumed to be the first element of mBrates.
Indicate that reception is not in progress.
export(mRxA)
Rx_Idle


TX
Block Transmission
TxConfirm
BkDone
transmit_1a(1)
/* This block does octet-level transfers of MPDUs from the MAC to the PHY transmitter, generating FCS fields and inserting timestamp values where necessary. Process Data_Pump begins transmitting when TxRequest arrives. The sender of TxRequest is assumed to have done the appropriate actions prior to transimtting onto the WM. If these actions include performing random backoff or invoking the "backoff procedure" (see 9.2.5.2), a Backoff signal is sent to process Backoff. At the completion of each backoff, a BkDone signal is returned to the sender of the Backoff signal at the correct time to send a TxRequest. The medium state updates (busy, idle, slot) from Channel_State are forwarded to Backoff_Procedure via Data_Pump to prevent decrementing the backoff count while transmitting Cts or Ack frames. This block is used in both station and AP. */
Txrq
Bkof
Backoff, Cancel
Backoff_Procedure (1,1)
Busy, Idle, Slot
TxRequest
FwdCs
Data_Pump (1,1)
FromCs
CS
Busy, Idle, Slot
PhyTxStart.confirm, PhyTxEnd.confirm, PhyData.confirm
ToPHY
PhyTxStart.request, PhyTxEnd.request, PhyData.request
PHY_SAP_TX

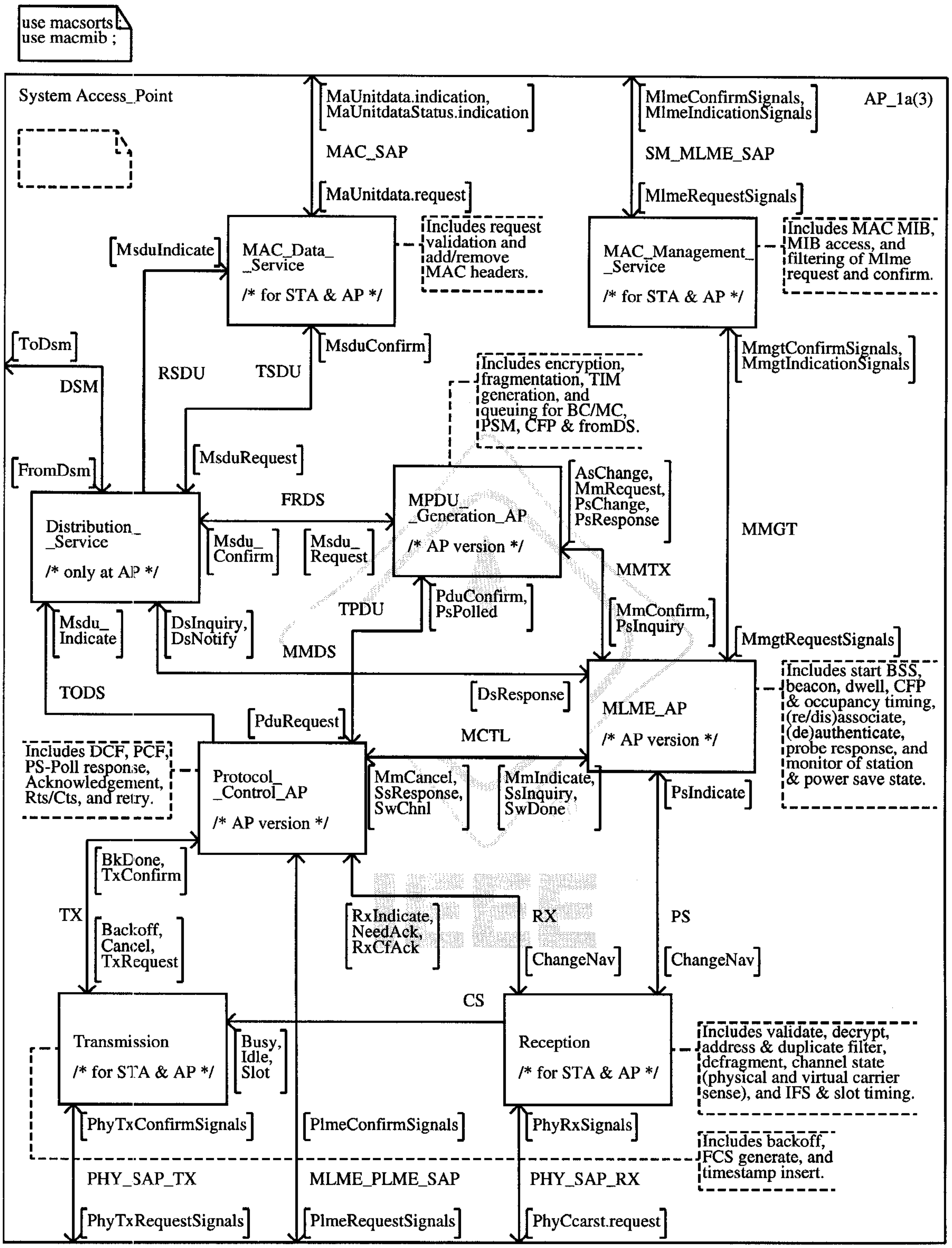

Process Backoff_Procedure
backoff_1a(1)
No_Backoff
/* RANDOM NUMBER FUNCTION */
imported procedure Random ;
fpar limit Integer ; returns Integer ;
/* This function returns an integer from a uniform distribution over the range (0 <= value <= limit). Implementors need to be aware that proper operation of the MAC protocol requires statistically independent (pseudo-)random sequences to be generated by each station in a service area. */
/* This process performs the Backoff Procedure (see 9.2.5.2), returning Done(-1) when Tx may begin, or Done(n>=0) if cancelled. Backoff(cw,-1) starts new random backoff. Backoff(x,n>=0) resumes cancelled backoff. Backoff(0,0) sends Done(-1) when WM idle. */
dcl slotCnt, cw, cnt Integer ; dcl source PId; dcl exported mBkIP Boolean:= false ;
Backoff (cw, cnt)
source:= sender, mBkIP:=true
export (mBkIP)
Save PId from request to use as addr of Done.
cw is contention window, cnt is slot count from previous Done. If cnt<0, a new random count is generated.
cnt
(<0)
(>=0)
Choose random backoff count if cnt = -1.
slotCnt:= call Random(cw)
slotCnt:= cnt
Resume with count from cancelled backoff if cnt>=0.
/* Input Signal Summary
BUSY is sent by Channel_State when the WM changes from idle to busy due to CCA and/or NAV, and by Data_Pump at TxStart.
CANCEL is sent by TxCoordination to terminate a backoff and return the residual backoff count value.
IDLE is sent by Channel_State at the end of the M2 interval (see 9.2.10) that busy WM has been idle (CCA & NAV) for DIFS (EIFS after Rx error).
SLOT is sent by Channel_State at the end of each M2 interval (see 9.2.10) while the WM is idle.
Busy, Idle and Slot are forwarded from Channel_State via Data_Pump when transmit is not in progress. */
At start assume that the WM is busy until receiving a signal which indicates the WM is idle.
Channel_Busy
Transitions to Channel_Idle also align the Backoff signal arrival time to slot boundary (M2) timing.
Idle
Slot
Busy
Cancel
-
Done
Slot only sent when WM idle. This is for the case where WM idle at arrival of Backoff signal.
*
Channel_Idle
Cancel has priority over other transitions. Done(0) returned if Cancel arrives at instant slotCnt:=0.
ResetMAC
Idle
Slot
Busy
slotCnt = 0
Cancel
mBkIP:= false
-
slotCnt:= slotCnt - 1
Channel_Busy
BkDone (-1) to source
BkDone (slotCnt) to source
export (mBkIP)
-
Go back and wait for WM to become idle.
Done
Done
No_Backoff
Idle signal not sent if WM free. This consumes any Idles still on input queue.
Decrement count for each slot when WM idle.
Done sent with value -1 when backoff counts down to zero.
Finish at M2 of proper slot, even if slotCnt =0 at entry to state.


Process Data_Pump
transmit_1a(1)
dcl fcs Crc ; dcl dTx Duration ; dcl k, txLength Integer ; dcl pdu Frame ; dcl rate Octet ; dcl source PId ;
imported procedure Tsf ; fpar Integer, Boolean; returns Integer ;
send1
Send_Frame
Delay from Phy_Sap(tx) to antenna.
*
(Tx_Idle)
ResetMAC
No TxConfirm if Tx halted by ResetMAC.
dTx:= dUsec (aTxRfDelay + aTxPlcpDelay)
PhyTxEnd._ request
Do not wait for TxEnd._ confirm.
Tx_Idle
Pass Busy, Idle and Slot signals to Backoff_Procedure while Tx is idle, but not during transmissions.
TxRequest (pdu, rate)
Busy
Idle
Slot
source:= sender
k:= 0, fcs:= initCrc
-
txLength:= Length(pdu)
Plcp length is Mpdu length + Fcs length
Indicate medium busy to freeze backoff count during transmit.
PhyTxStart._ request
(txLength+ sCrcLng, rate)
Wait_TxStart
PhyTxStart._ confirm
PhyData._ request (pdu(k))
PhyData._ confirm
fcs:= crc32 (fcs,pdu(k))
k:= k+1
/* This process sends an Mpdu to the Phy while generating & appending the Fcs. On beacons and probe responses inserts (TSF + Phy TxDelay) in the timestamp field at confirm of octet 23.
To transmit after Sifs, send TxRequest at end of the M1 interval (see 9.2.10). For Pifs, Difs, or any backoff slot, TxRequest is sent at the end of the appropriate M2 interval. */
k = txLength
(false)
(true)
k:= 0, fcs:= mirror (not(fcs))
Send_CRC
Send the 1's complement of calculated FCS value, MSb to LSb.
ftype(pdu)
else
(probe_rsp, beacon)
k = sTsOctet
k = sCrcLng
PhyData._ request (fcs(k))
PhyTxEnd._ request
Insert_ Timestamp
Wait_TxEnd
Start of time stamp in beacon and probe_rsp.
PhyTxEnd._ confirm
pdu:=setTs (pdu,call Tsf (0,false)+dTx)
At confirm of octet 23, insert TSF + Phy Tx delay into [24:31] of beacon or probe rsp.
TxConfirm goes to process that sent TxRequest.
TxConfirm to source

C.4 State machines for MAC access point

The following SDL-92 system specification defines operation of the MAC protocol at an IEEE 802.11 AP. Many aspects of AP operation are identical to STA operation. These are defined in blocks and processes referenced from both the STA and AP system specifications. Blocks and processes used in both STA and AP are identifiable by the SDL comment /* for STA & AP */ below the block or process name. The definitions of these blocks and processes appear in Clause C.3. Blocks and processes specific to AP operation are identifiable by the SDL comment /* AP version */ below the block or process name. The definitions of these blocks and processes appear in this subclause.

The remainder of Clause C.4 is the formal description, in SDL/GR, of an IEEE 802.11 AP.


use macsorts ;
use macmib ;
System Access_Point
AP_1a(3)
MaUnitdata.indication,
MaUnitdataStatus.indication
MAC_SAP
MaUnitdata.request
MlmeConfirmSignals,
MlmeIndicationSignals
SM_MLME_SAP
MlmeRequestSignals
MsduIndicate
MAC_Data_
_Service
/* for STA & AP */
Includes request validation and add/remove MAC headers.
MAC_Management_
_Service
/* for STA & AP */
Includes MAC MIB, MIB access, and filtering of Mlme request and confirm.
ToDsm
DSM
RSDU
TSDU
MsduConfirm
Includes encryption, fragmentation, TIM generation, and queuing for BC/MC, PSM, CFP & fromDS.
MmgtConfirmSignals,
MmgtIndicationSignals
MsduRequest
FromDsm
FRDS
Distribution_
_Service
/* only at AP */
MPDU_
_Generation_AP
/* AP version */
AsChange,
MmRequest,
PsChange,
PsResponse
MMGT
Msdu_
Confirm
Msdu_
Request
MMTX
TPDU
PduConfirm,
PsPolled
Msdu_
Indicate
DsInquiry,
DsNotify
MMDS
MmConfirm,
PsInquiry
MmgtRequestSignals
TODS
DsResponse
MLME_AP
/* AP version */
Includes start BSS, beacon, dwell, CFP & occupancy timing, (re/dis)associate, (de)authenticate, probe response, and monitor of station & power save state.
PduRequest
MCTL
Includes DCF, PCF, PS-Poll response, Acknowledgement, Rts/Cts, and retry.
Protocol_
_Control_AP
/* AP version */
MmCancel,
SsResponse,
SwChnl
MmIndicate,
SsInquiry,
SwDone
PsIndicate
BkDone,
TxConfirm
TX
Backoff,
Cancel,
TxRequest
RxIndicate,
NeedAck,
RxCfAck
RX
ChangeNav
PS
ChangeNav
CS
Transmission
/* for STA & AP */
Busy,
Idle,
Slot
Reception
/* for STA & AP */
Includes validate, decrypt, address & duplicate filter, defragment, channel state (physical and virtual carrier sense), and IFS & slot timing.
PhyTxConfirmSignals
PlmeConfirmSignals
PhyRxSignals
Includes backoff, FCS generate, and timestamp insert.
PHY_SAP_TX
MLME_PLME_SAP
PHY_SAP_RX
PhyTxRequestSignals
PlmeRequestSignals
PhyCcarst.request

use macsorts
use macmib ;

System Access_Point

AP_signals_2b(3)

newtype DsStatus literals
assoc, disassoc, reassoc, unknown
endnewtype DsStatus ;

signal
AsChange(Frame,DsStatus)
Backoff(Integer,Integer),
BkDone(Integer),
Busy,
Cancel,
[illegible] Src),
[illegible]
DsNotify(MacAddr,DsStatus)
DsResponse(MacAddr,MacAddr,DsStatus),
FromDsm(MacAddr,MacAddr,Octetstring),
Idle,
MaUnitdata.indication(MacAddr,MacAddr,
Routing,Octetstring,RxStatus,
CfPriority,ServiceClass),
MaUnitdata.request(MacAddr,MacAddr,
Routing,Octetstring,CfPriority,ServiceClass),
MaUnitdataStatus.indication(MacAddr,
[illegible]
MlmeGet.request(MibAtrib),
MlmeJoin.confirm(MlmeStatus),
[illegible]
MlmeReset.confirm(MlmeStatus),
MlmeReset.request,
[illegible]

signal
MmCancel,
MmConfirm(Frame,TxStatus),
MmIndicate(Frame,Time,Time,StateErr),
MmRequest(Frame,Imed,Rate),
MsduConfirm(Frame,CfPriority,TxStatus),
MsduIndicate(Frame,CfPriority),
MsduRequest(Frame,CfPriority),
NeedAck(MacAddr,Time,Duration,Rate),
PduConfirm(FragSdu,TxResult),
PduRequest(FragSdu),
PhyCca.indication(Ccastatus),
PhyCcarst.confirm,
PhyCcarst.request,
PhyData.confirm,
PhyData.indication(Octet),
PhyData.request(Octet),
PhyRxEnd.indication(PhyRxStat),
PhyRxStart.indication(Integer,Rate),
PhyTxEnd.confirm,
PhyTxEnd.request,
PhyTxStart.confirm,
PhyTxStart.request(Integer,Rate),
PlmeGet.confirm(MibStatus,
MibAtrib,MibValue),
PlmeGet.request(MibAtrib),
PlmeReset.confirm(Boolean),
PlmeReset request,
PlmeSet.confirm(MibStatus,MibAtrib),
PlmeSet.request(MibAtrib,MibValue),
PsmDone,
PsPolled(MacAddr,AsocId),
PsChange(MacAddr,PsMode),
PsIndicate(MacAddr,PsMode),
PsInquiry(MacAddr),
PsResponse(MacAddr,PsMode),
ResetMAC,
RxCfAck(MacAddr),
RxIndicate(Frame,Time,Time,Rate),
Slot,
SsInquiry(MacAddr),
SsResponse(MacAddr,
StationState,StationState),
SwChnl(Integer,Boolean),
SwDone,
ToDsm(MacAddr,MacAddr,Octetstring),
TxConfirm,
TxRequest(Frame,Rate) ;

```
use macsorts;
use macmib ;
```

System Access_Point

AP_signallists_3a(3)

```
signallist
MlmeRequestSignals=
  MlmeAssociate.request,
  MlmeAuthenticate.request,
  MlmeDeauthenticate.request,
  MlmeDisassociate.request,
  MlmeGet.request,
  MlmeJoin.request,
  MlmePowermgt.request,
  MlmeReassociate.request,
  MlmeReset.request,
  MlmeScan.request,
  MlmeSet.request,
  MlmeStart.request ;
```

```
signallist
MlmeConfirmSignals=
  MlmeAssociate.confirm,
  MlmeAuthenticate.confirm,
  MlmeDeauthenticate.confirm,
  MlmeDisassociate.confirm,
  MlmeGet.confirm,
  MlmeJoin.confirm,
  MlmePowermgt.confirm,
  MlmeReassociate.confirm,
  MlmeReset.confirm,
  MlmeScan.confirm,
  MlmeSet.confirm,
  MlmeStart.confirm ;
```

```
signallist
MlmeIndicationSignals=
  MlmeAuthenticate.indication,
  MlmeDeauthenticate.indication,
  MlmeDisassociate.indication,
  MlmeAssociate.indication,
  MlmeReassociate.indication ;
```

```
signallist
SmtRequestSignals=
  MlmeAssociate.request,
  MlmeAuthenticate.request,
  MlmeDeauthenticate.request,
  MlmeDisassociate.request,
  MlmeJoin.request,
  MlmeReassociate.request,
  MlmeScan.request,
  MlmeStart.request ;
```

```
signallist
SmtConfirmSignals=
  MlmeAssociate.confirm,
  MlmeAuthenticate.confirm,
  MlmeDeauthenticate.confirm,
  MlmeDisassociate.confirm,
  MlmeJoin.confirm,
  MlmeReassociate.confirm,
  MlmeScan.confirm,
  MlmeStart.confirm ;
```

```
signallist
SmtIndicationSignals=
  MlmeAuthenticate.indication,
  [illegible]
```

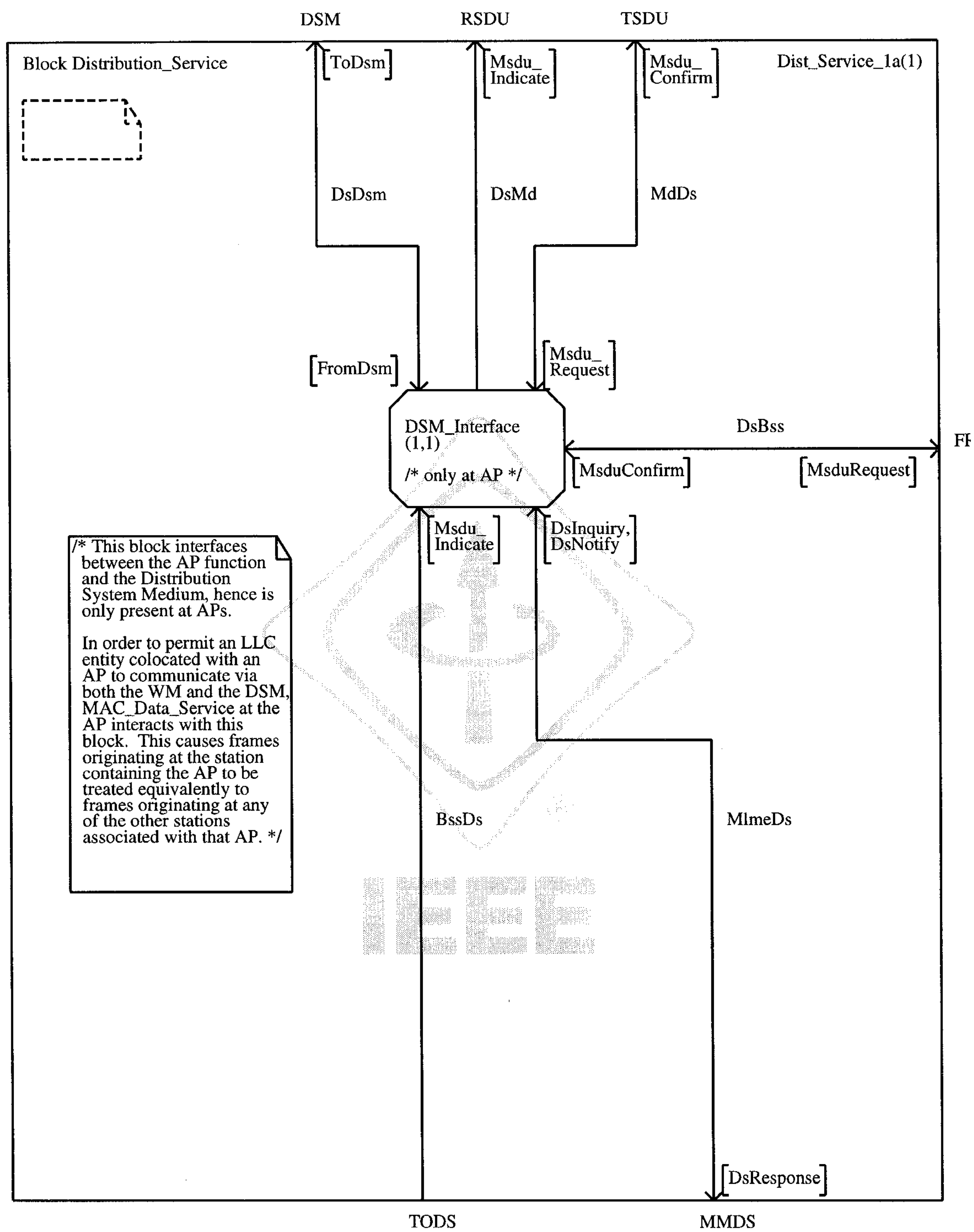
```
signallist
PhyTxRequestSignals=
  PhyTxStart.request,
  PhyTxEnd.request,
  PhyData.request ;
```

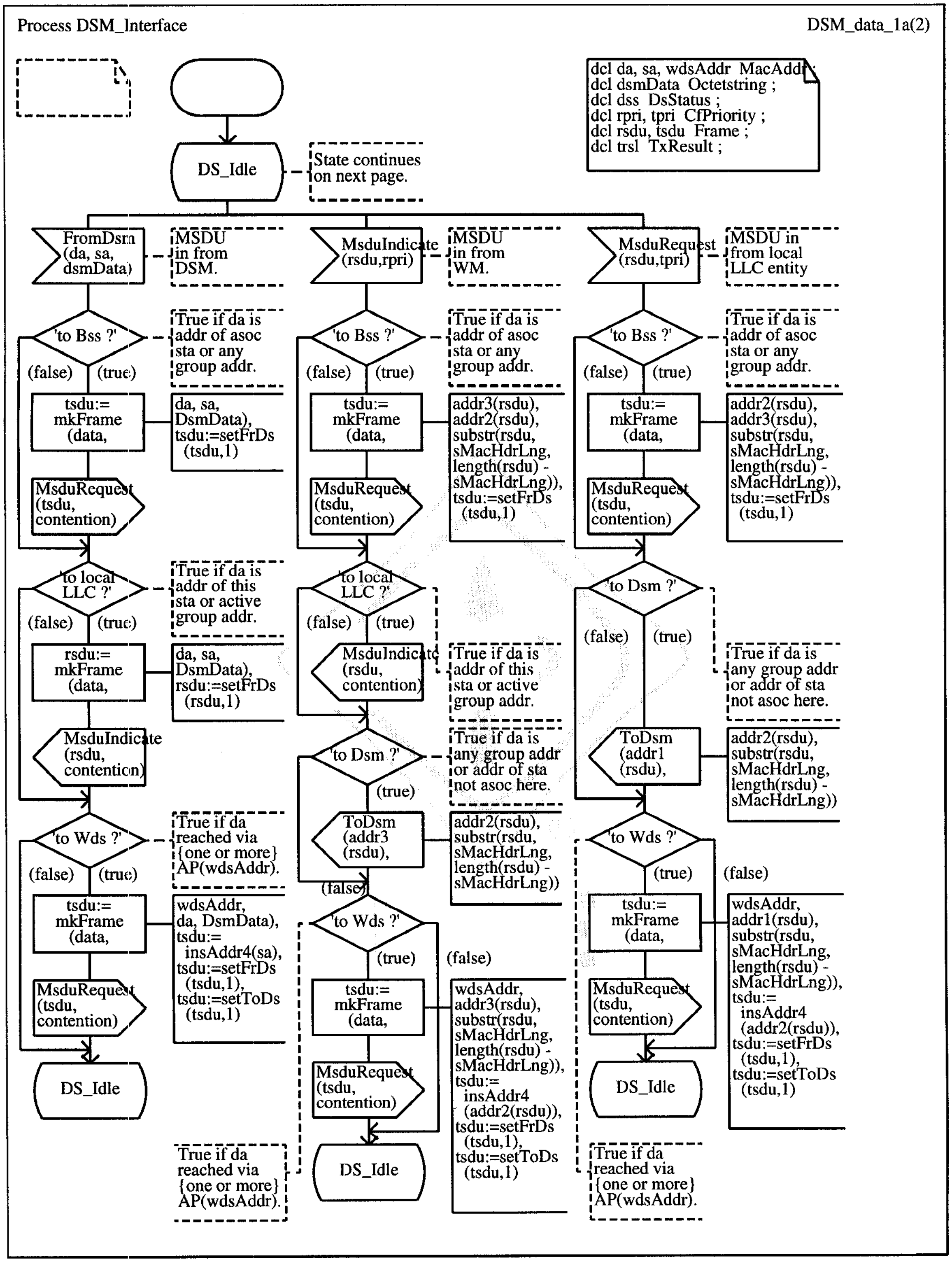
```
signallist
[illegible]
  [illegible]
  [illegible]
  PhyData.confirm ;
```

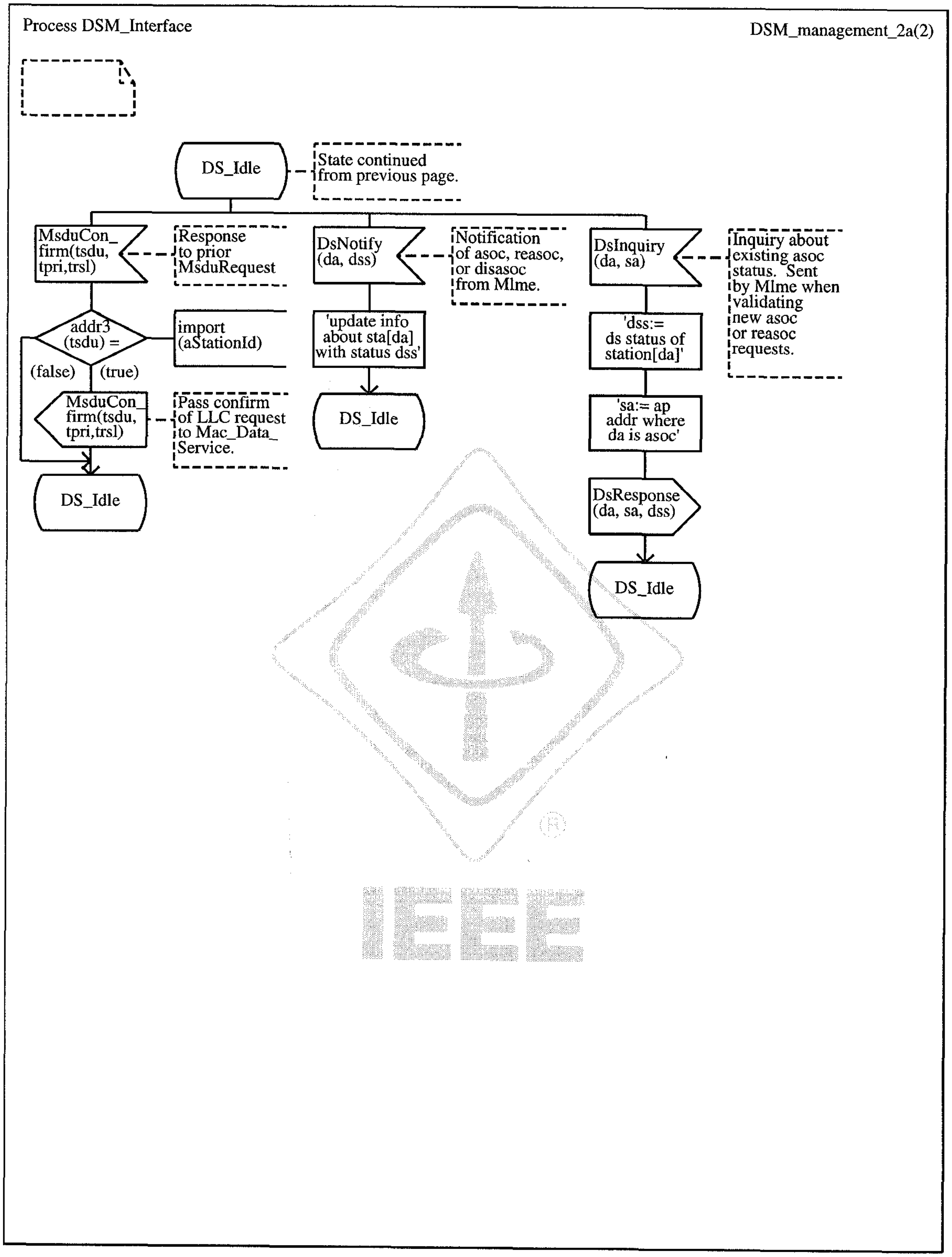
```
signallist
PhyRxSignals=
  PhyRxStart.indication,
  PhyRxEnd.indication,
  PhyData.indication
  PhyCca.indication,
  PhyCcarst.confirm ;
```

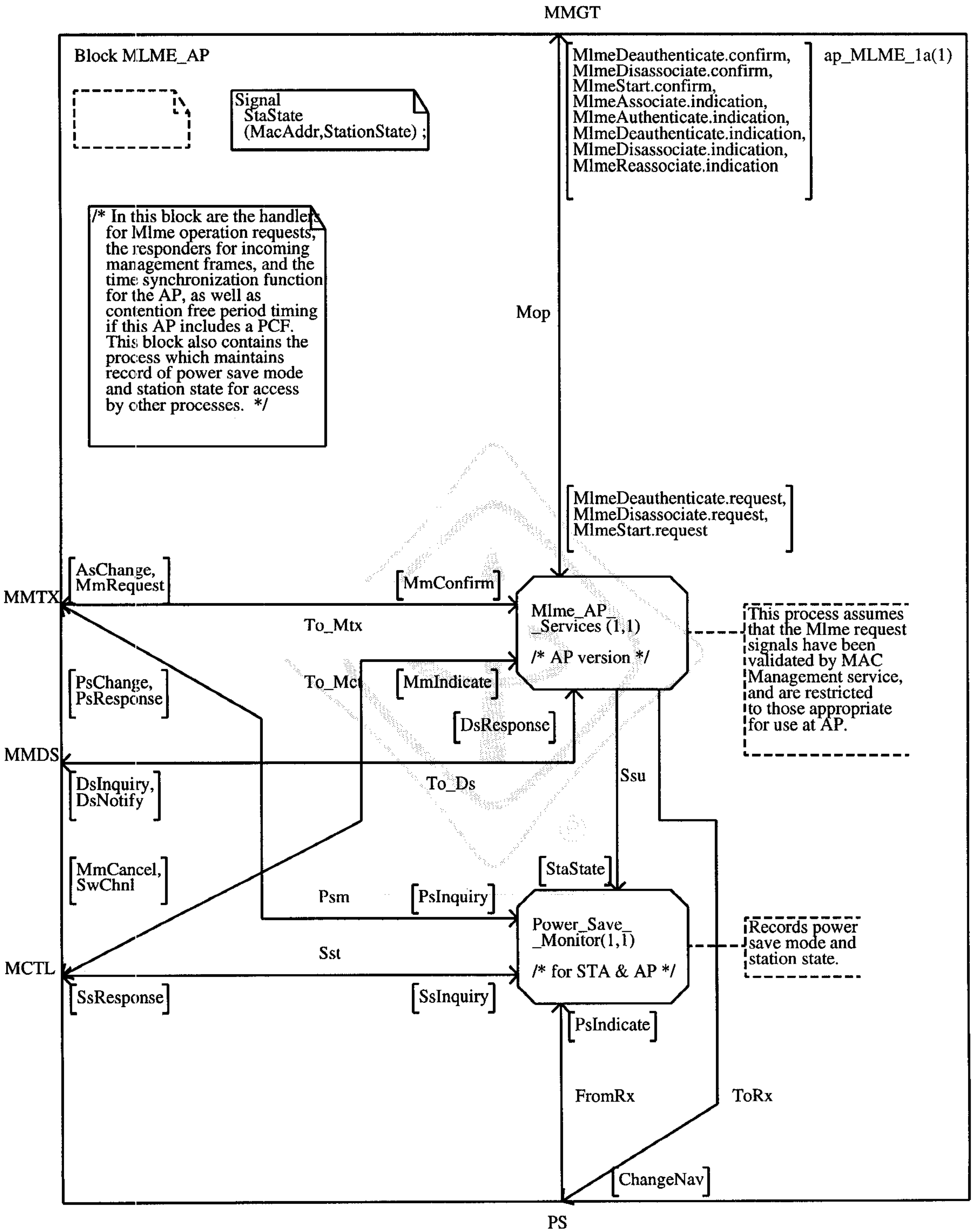
```
signallist
PlmeRequestSignals=
  PlmeGet.request,
  PlmeSet.request,
  PlmeReset.request ;
```

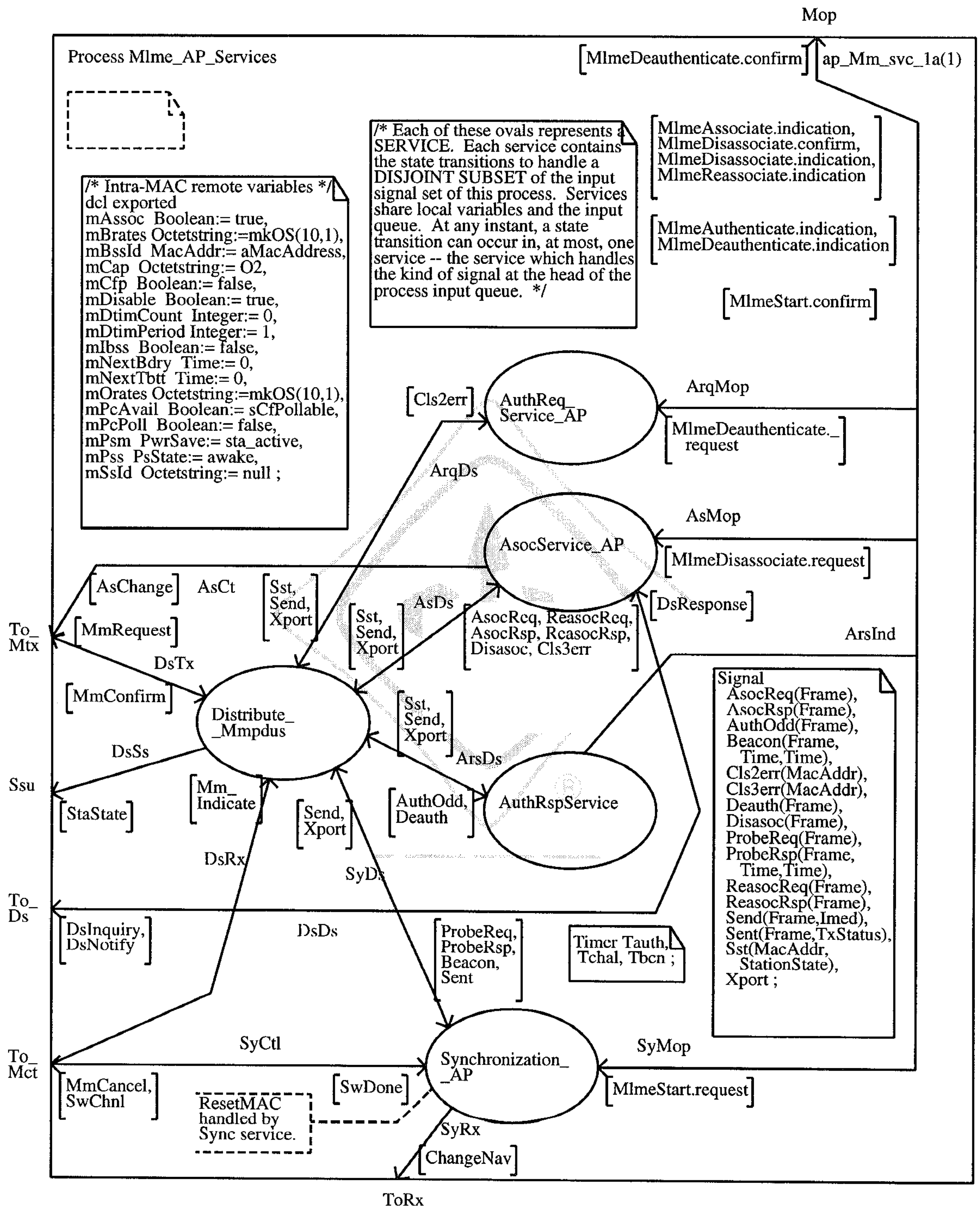
```
signallist
PlmeConfirmSignals=
  PlmeGet.confirm,
  PlmeSet.confirm,
  PlmeReset.confirm ;
```

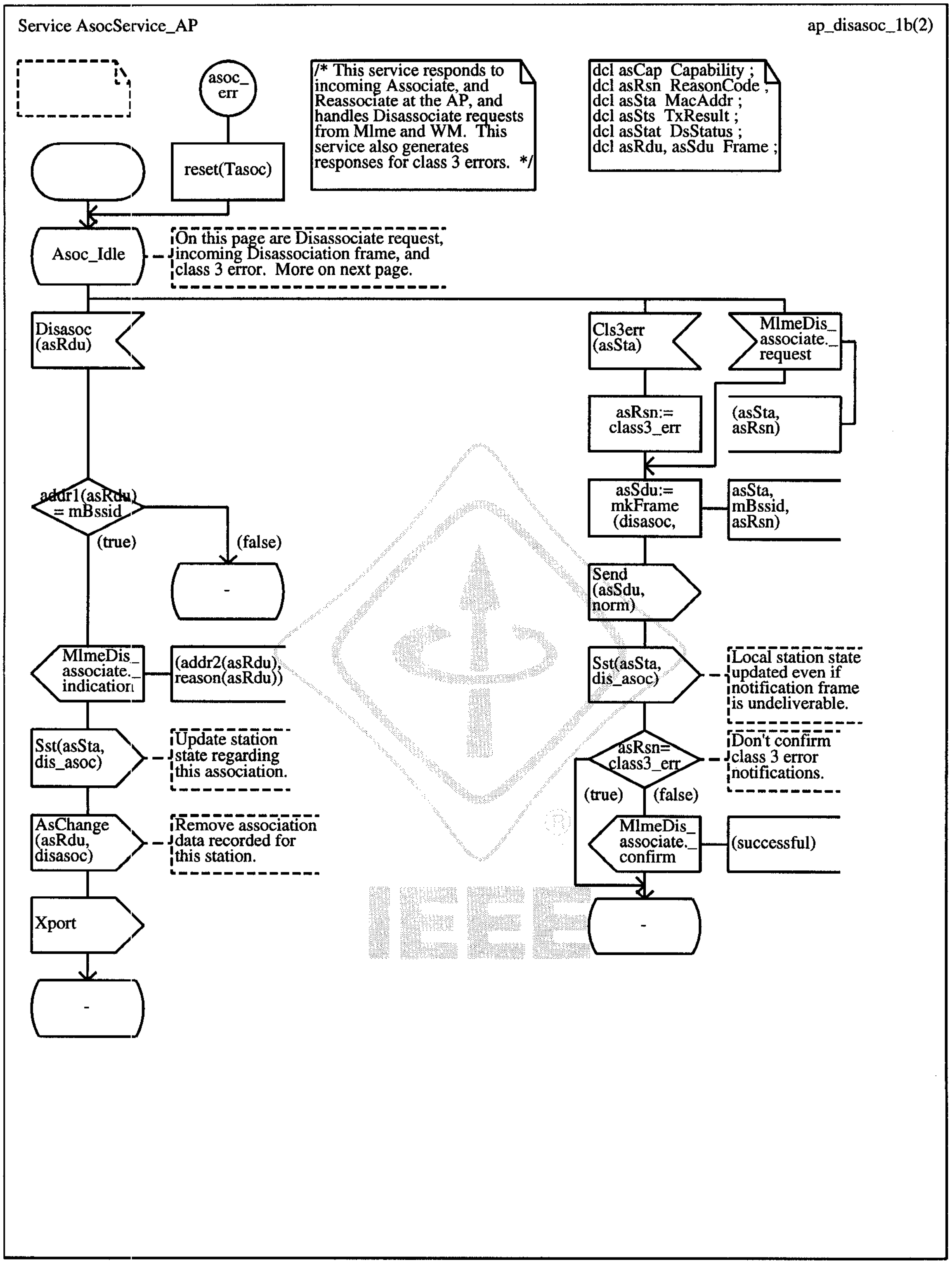

DSM
RSDU
TSDU
Block Distribution_Service
ToDsm
Msdu_ Indicate
Msdu_ Confirm
Dist_Service_1a(1)
DsDsm
DsMd
MdDs
FromDsm
Msdu_ Request
DSM_Interface (1,1)
/* only at AP */
DsBss
MsduConfirm
MsduRequest
Msdu_ Indicate
DsInquiry, DsNotify
/* This block interfaces between the AP function and the Distribution System Medium, hence is only present at APs.
In order to permit an LLC entity colocated with an AP to communicate via both the WM and the DSM, MAC_Data_Service at the AP interacts with this block. This causes frames originating at the station containing the AP to be treated equivalently to frames originating at any of the other stations associated with that AP. */
BssDs
MlmeDs
DsResponse
TODS
MMDS

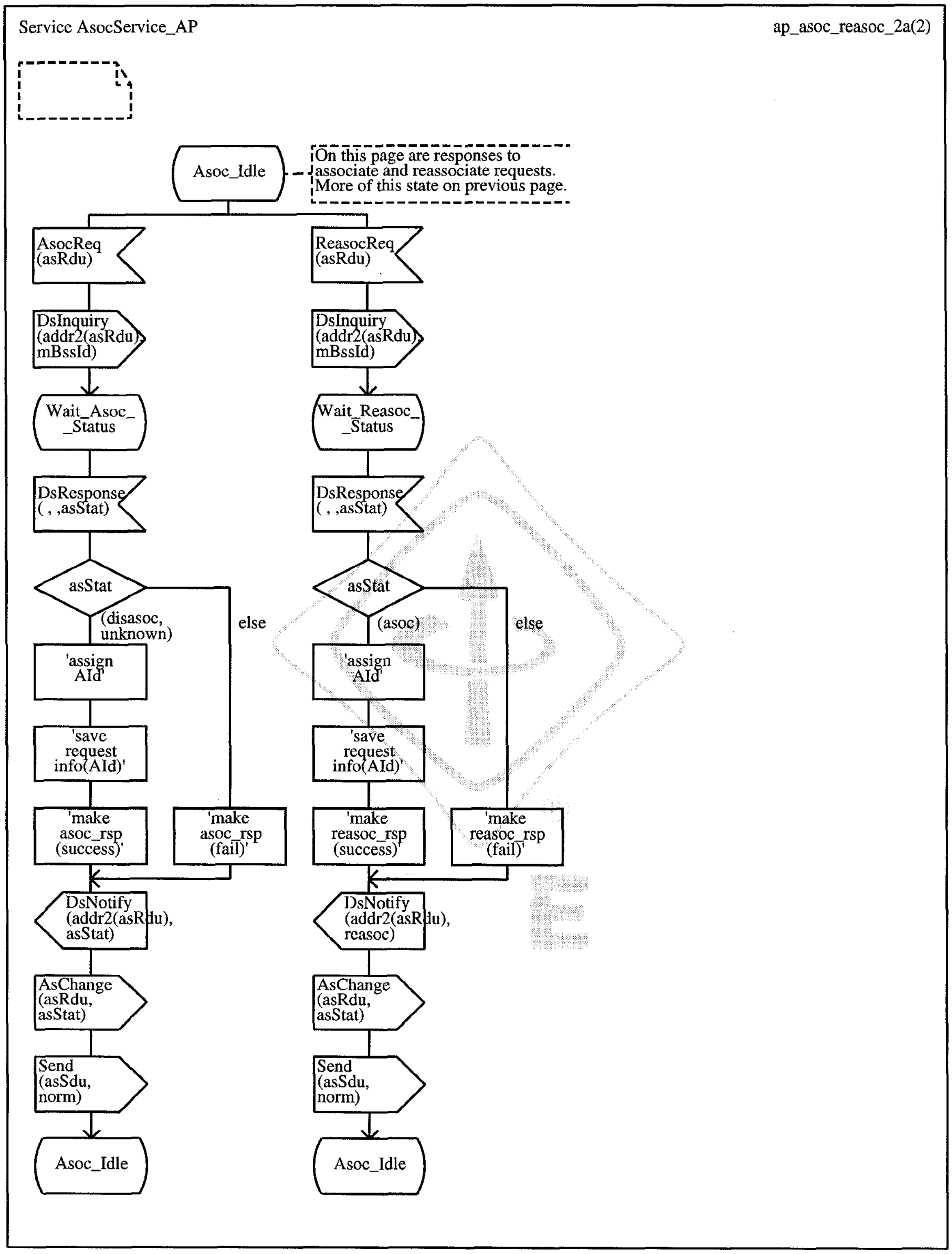

Process DSM_Interface
DSM_data_1a(2)
dcl da, sa, wdsAddr MacAddr;
dcl dsmData Octetstring ;
dcl dss DsStatus ;
dcl rpri, tpri CfPriority ;
dcl rsdu, tsdu Frame ;
dcl trsl TxResult ;
DS_Idle
State continues on next page.
FromDsm (da, sa, dsmData)
MSDU in from DSM.
MsduIndicate (rsdu,rpri)
MSDU in from WM.
MsduRequest (rsdu,tpri)
MSDU in from local LLC entity
'to Bss ?'
True if da is addr of asoc sta or any group addr.
(false)
(true)
tsdu:= mkFrame (data,
da, sa, DsmData), tsdu:=setFrDs (tsdu,1)
addr3(rsdu), addr2(rsdu), substr(rsdu, sMacHdrLng, length(rsdu) - sMacHdrLng)), tsdu:=setFrDs (tsdu,1)
addr2(rsdu), addr3(rsdu), substr(rsdu, sMacHdrLng, length(rsdu) - sMacHdrLng)), tsdu:=setFrDs (tsdu,1)
MsduRequest (tsdu, contention)
'to local LLC ?'
True if da is addr of this sta or active group addr.
rsdu:= mkFrame (data,
da, sa, DsmData), rsdu:=setFrDs (rsdu,1)
MsduIndicate (rsdu, contention)
'to Dsm ?'
True if da is any group addr or addr of sta not asoc here.
ToDsm (addr3 (rsdu),
ToDsm (addr1 (rsdu),
addr2(rsdu), substr(rsdu, sMacHdrLng, length(rsdu) - sMacHdrLng))
'to Wds ?'
True if da reached via {one or more} AP(wdsAddr).
wdsAddr, da, DsmData), tsdu:= insAddr4(sa), tsdu:=setFrDs (tsdu,1), tsdu:=setToDs (tsdu,1)
wdsAddr, addr3(rsdu), substr(rsdu, sMacHdrLng, length(rsdu) - sMacHdrLng)), tsdu:= insAddr4 (addr2(rsdu)), tsdu:=setFrDs (tsdu,1), tsdu:=setToDs (tsdu,1)
wdsAddr, addr1(rsdu), substr(rsdu, sMacHdrLng, length(rsdu) - sMacHdrLng)), tsdu:= insAddr4 (addr2(rsdu)), tsdu:=setFrDs (tsdu,1), tsdu:=setToDs (tsdu,1)

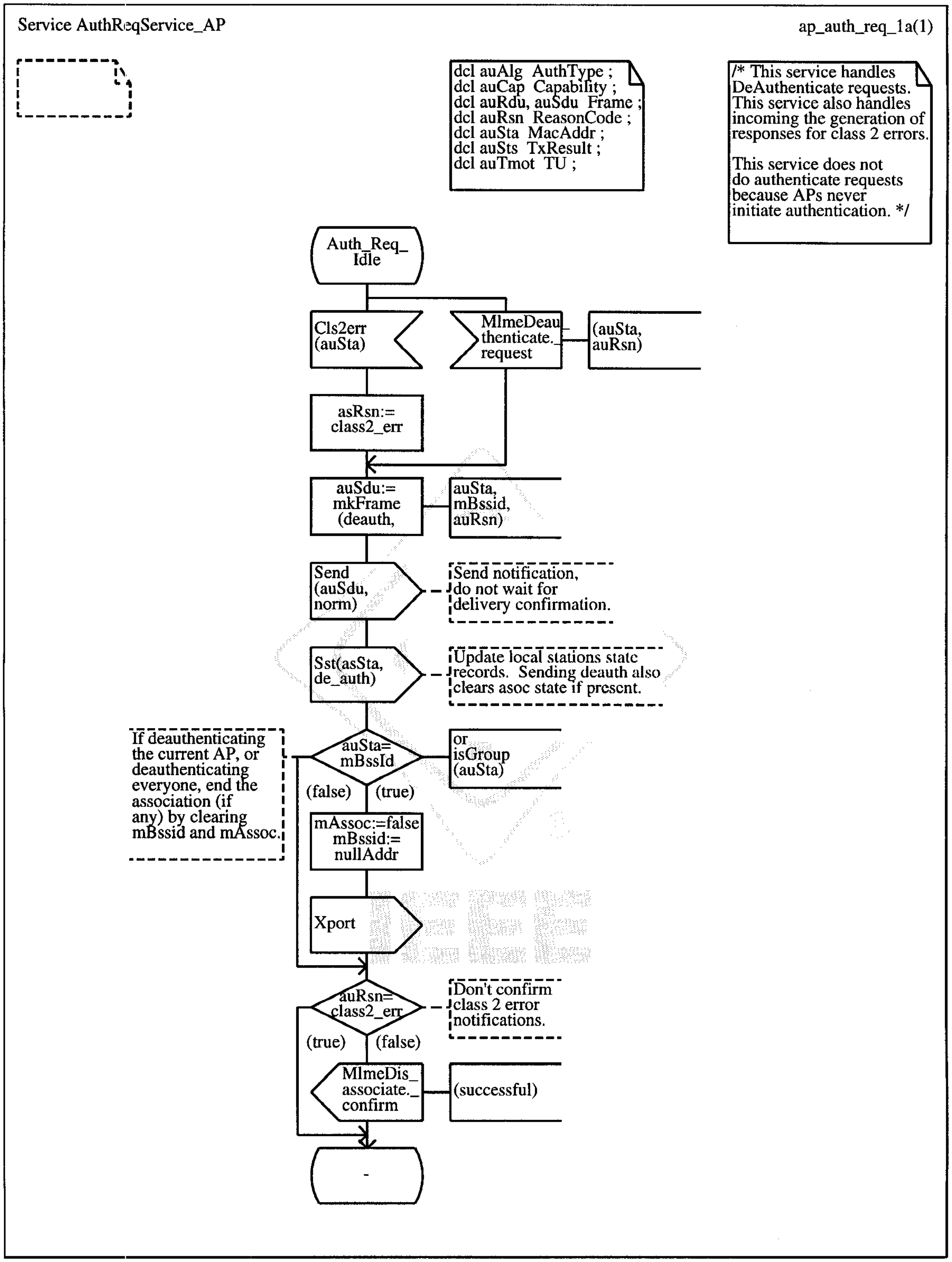

Process DSM_Interface
DSM_management_2a(2)
DS_Idle
State continued from previous page.
MsduConfirm(tsdu, tpri,trsl)
Response to prior MsduRequest
addr3 (tsdu) =
import (aStationId)
(false)
(true)
MsduConfirm(tsdu, tpri,trsl)
Pass confirm of LLC request to Mac_Data_Service.
DS_Idle
DsNotify (da, dss)
Notification of asoc, reasoc, or disasoc from Mlme.
'update info about sta[da] with status dss'
DS_Idle
DsInquiry (da, sa)
Inquiry about existing asoc status. Sent by Mlme when validating new asoc or reasoc requests.
'dss:= ds status of station[da]'
'sa:= ap addr where da is asoc'
DsResponse (da, sa, dss)
DS_Idle


MMGT
Block MLME_AP
ap_MLME_1a(1)
Signal
StaState
(MacAddr,StationState) ;
/* In this block are the handlers for Mlme operation requests, the responders for incoming management frames, and the time synchronization function for the AP, as well as contention free period timing if this AP includes a PCF. This block also contains the process which maintains record of power save mode and station state for access by other processes. */
MlmeDeauthenticate.confirm,
MlmeDisassociate.confirm,
MlmeStart.confirm,
MlmeAssociate.indication,
MlmeAuthenticate.indication,
MlmeDeauthenticate.indication,
MlmeDisassociate.indication,
MlmeReassociate.indication
Mop
MlmeDeauthenticate.request,
MlmeDisassociate.request,
MlmeStart.request
MMTX
AsChange,
MmRequest
MmConfirm
To_Mtx
Mlme_AP_
_Services (1,1)
/* AP version */
This process assumes that the Mlme request signals have been validated by MAC Management service, and are restricted to those appropriate for use at AP.
PsChange,
PsResponse
To_Mc
MmIndicate
DsResponse
MMDS
DsInquiry,
DsNotify
To_Ds
Ssu
MmCancel,
SwChnl
StaState
Psm
PsInquiry
Power_Save_
_Monitor(1,1)
/* for STA & AP */
Records power save mode and station state.
Sst
MCTL
SsResponse
SsInquiry
PsIndicate
FromRx
ToRx
ChangeNav
PS

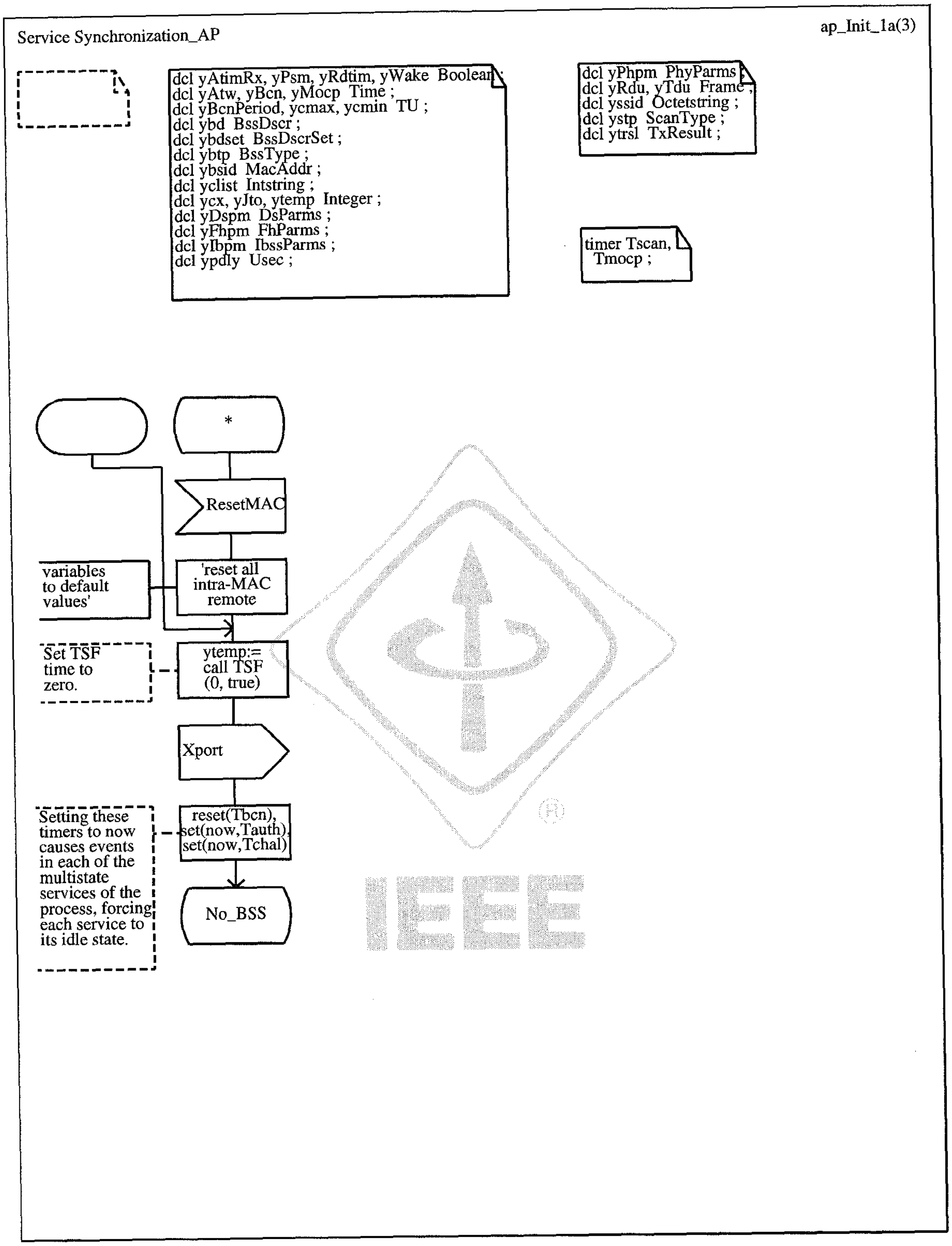

Mop
Process Mlme_AP_Services
MlmeDeauthenticate.confirm
ap_Mm_svc_1a(1)
/* Each of these ovals represents a SERVICE. Each service contains the state transitions to handle a DISJOINT SUBSET of the input signal set of this process. Services share local variables and the input queue. At any instant, a state transition can occur in, at most, one service -- the service which handles the kind of signal at the head of the process input queue. */
MlmeAssociate.indication, MlmeDisassociate.confirm, MlmeDisassociate.indication, MlmeReassociate.indication
MlmeAuthenticate.indication, MlmeDeauthenticate.indication
MlmeStart.confirm
/* Intra-MAC remote variables */
dcl exported
mAssoc Boolean:= true,
mBrates Octetstring:=mkOS(10,1),
mBssId MacAddr:= aMacAddress,
mCap Octetstring:= O2,
mCfp Boolean:= false,
mDisable Boolean:= true,
mDtimCount Integer:= 0,
mDtimPeriod Integer:= 1,
mIbss Boolean:= false,
mNextBdry Time:= 0,
mNextTbtt Time:= 0,
mOrates Octetstring:=mkOS(10,1),
mPcAvail Boolean:= sCfPollable,
mPcPoll Boolean:= false,
mPsm PwrSave:= sta_active,
mPss PsState:= awake,
mSsId Octetstring:= null ;
Cls2err
AuthReq_ Service_AP
ArqMop
MlmeDeauthenticate._ request
ArqDs
AsMop
AsocService_AP
MlmeDisassociate.request
AsChange
AsCt
Sst, Send, Xport
To_ Mtx
MmRequest
Sst, Send, Xport
AsDs
AsocReq, ReasocReq, AsocRsp, ReasocRsp, Disasoc, Cls3err
DsResponse
ArsInd
DsTx
MmConfirm
Distribute_ _Mmpdus
Sst, Send, Xport
ArsDs
Signal
AsocReq(Frame),
AsocRsp(Frame),
AuthOdd(Frame),
Beacon(Frame, Time,Time),
Cls2err(MacAddr),
Cls3err(MacAddr),
Deauth(Frame),
Disasoc(Frame),
ProbeReq(Frame),
ProbeRsp(Frame, Time,Time),
ReasocReq(Frame),
ReasocRsp(Frame),
Send(Frame,Imed),
Sent(Frame,TxStatus),
Sst(MacAddr, StationState),
Xport ;
DsSs
Ssu
StaState
Mm_ Indicate
Send, Xport
AuthOdd, Deauth
AuthRspService
DsRx
SyDs
To_ Ds
DsInquiry, DsNotify
DsDs
ProbeReq, ProbeRsp, Beacon, Sent
Timer Tauth, Tchal, Tbcn ;
SyCtl
SyMop
To_ Mct
Synchronization_ _AP
MlmeStart.request
MmCancel, SwChnl
ResetMAC handled by Sync service.
SwDone
SyRx
ChangeNav
ToRx

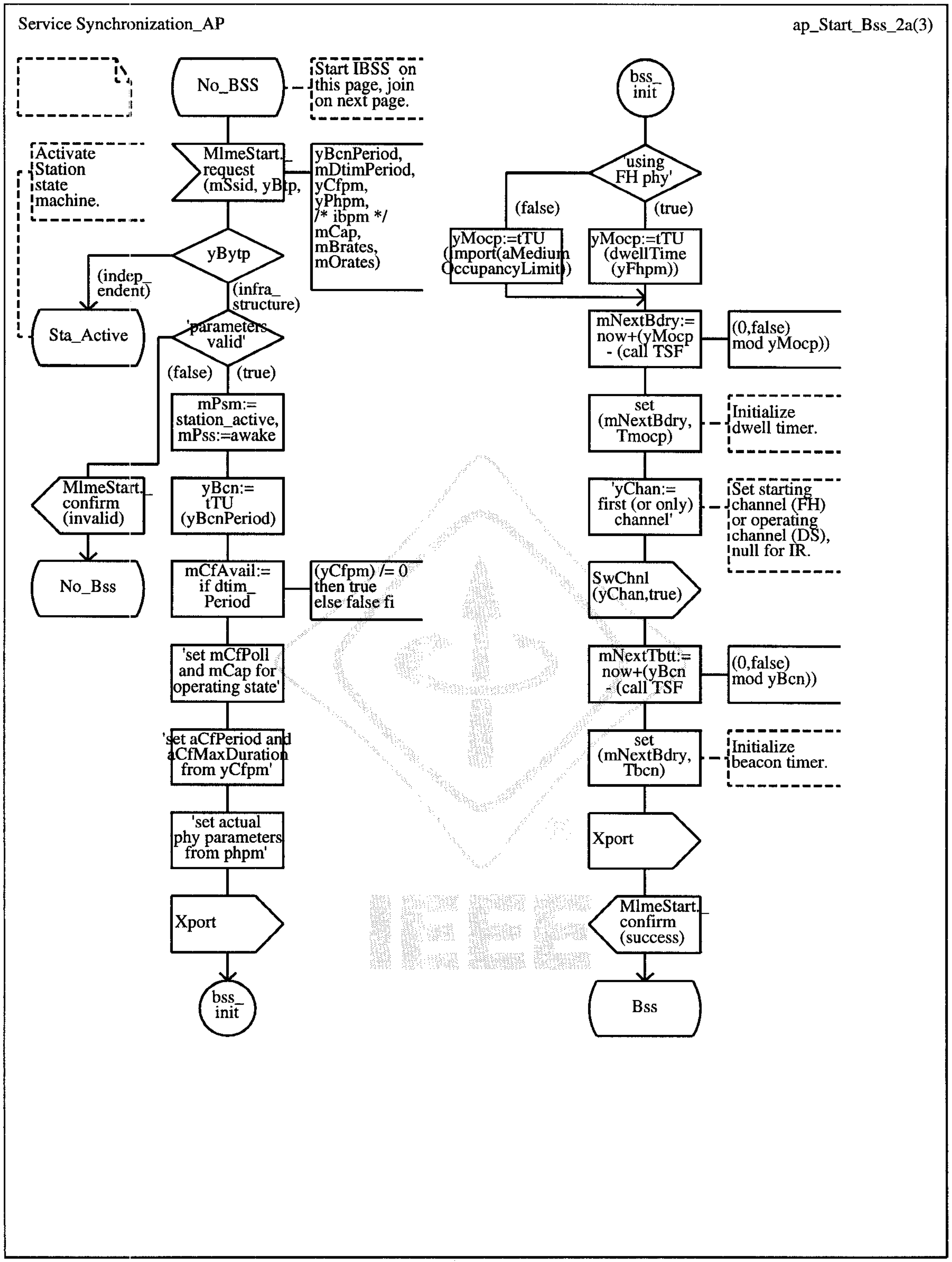

Service AsocService_AP
ap_disasoc_1b(2)
asoc_err
reset(Tasoc)
/* This service responds to incoming Associate, and Reassociate at the AP, and handles Disassociate requests from Mlme and WM. This service also generates responses for class 3 errors. */
dcl asCap Capability ;
dcl asRsn ReasonCode ;
dcl asSta MacAddr ;
dcl asSts TxResult ;
dcl asStat DsStatus ;
dcl asRdu, asSdu Frame ;
Asoc_Idle
On this page are Disassociate request, incoming Disassociation frame, and class 3 error. More on next page.
Disasoc (asRdu)
Cls3err (asSta)
MlmeDis_associate._request
asRsn:= class3_err
(asSta, asRsn)
addr1(asRdu) = mBssid
(true)
(false)
asSdu:= mkFrame (disasoc,
asSta, mBssid, asRsn)
Send (asSdu, norm)
MlmeDis_associate._indication
(addr2(asRdu), reason(asRdu))
Sst(asSta, dis_asoc)
Local station state updated even if notification frame is undeliverable.
Sst(asSta, dis_asoc)
Update station state regarding this association.
asRsn= class3_err
Don't confirm class 3 error notifications.
(true)
(false)
AsChange (asRdu, disasoc)
Remove association data recorded for this station.
MlmeDis_associate._confirm
(successful)
Xport
-

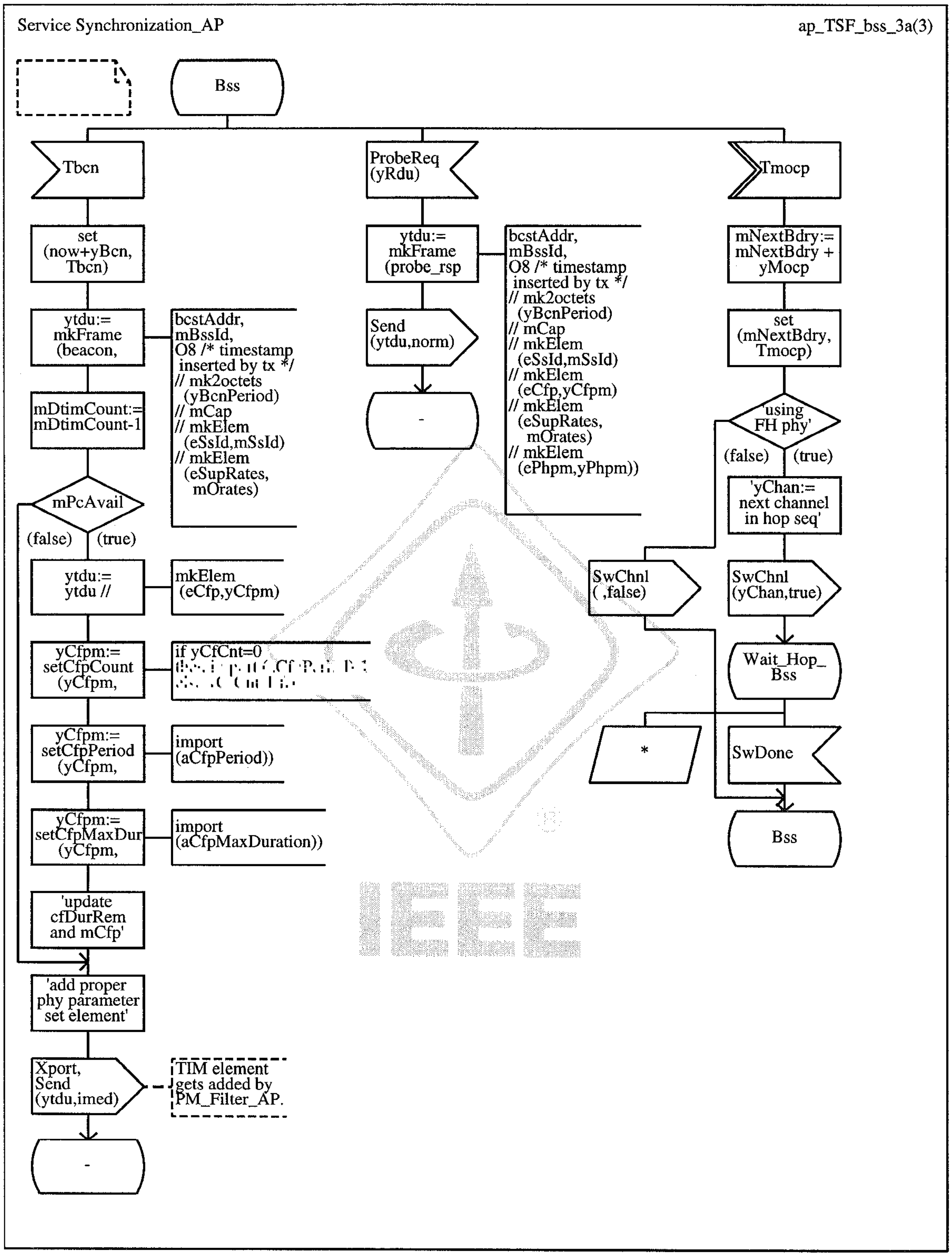

Service AsocService_AP
ap_asoc_reasoc_2a(2)
Asoc_Idle
On this page are responses to associate and reassociate requests. More of this state on previous page.
AsocReq (asRdu)
ReasocReq (asRdu)
DsInquiry (addr2(asRdu), mBssId)
DsInquiry (addr2(asRdu), mBssId)
Wait_Asoc_ _Status
Wait_Reasoc_ _Status
DsResponse (, ,asStat)
DsResponse (, ,asStat)
asStat
(disasoc, unknown)
else
asStat
(asoc)
else
'assign AId'
'assign AId'
'save request info(AId)'
'save request info(AId)'
'make asoc_rsp (success)'
'make asoc_rsp (fail)'
'make reasoc_rsp (success)'
'make reasoc_rsp (fail)'
DsNotify (addr2(asRdu), asStat)
DsNotify (addr2(asRdu), reasoc)
AsChange (asRdu, asStat)
AsChange (asRdu, asStat)
Send (asSdu, norm)
Send (asSdu, norm)
Asoc_Idle
Asoc_Idle

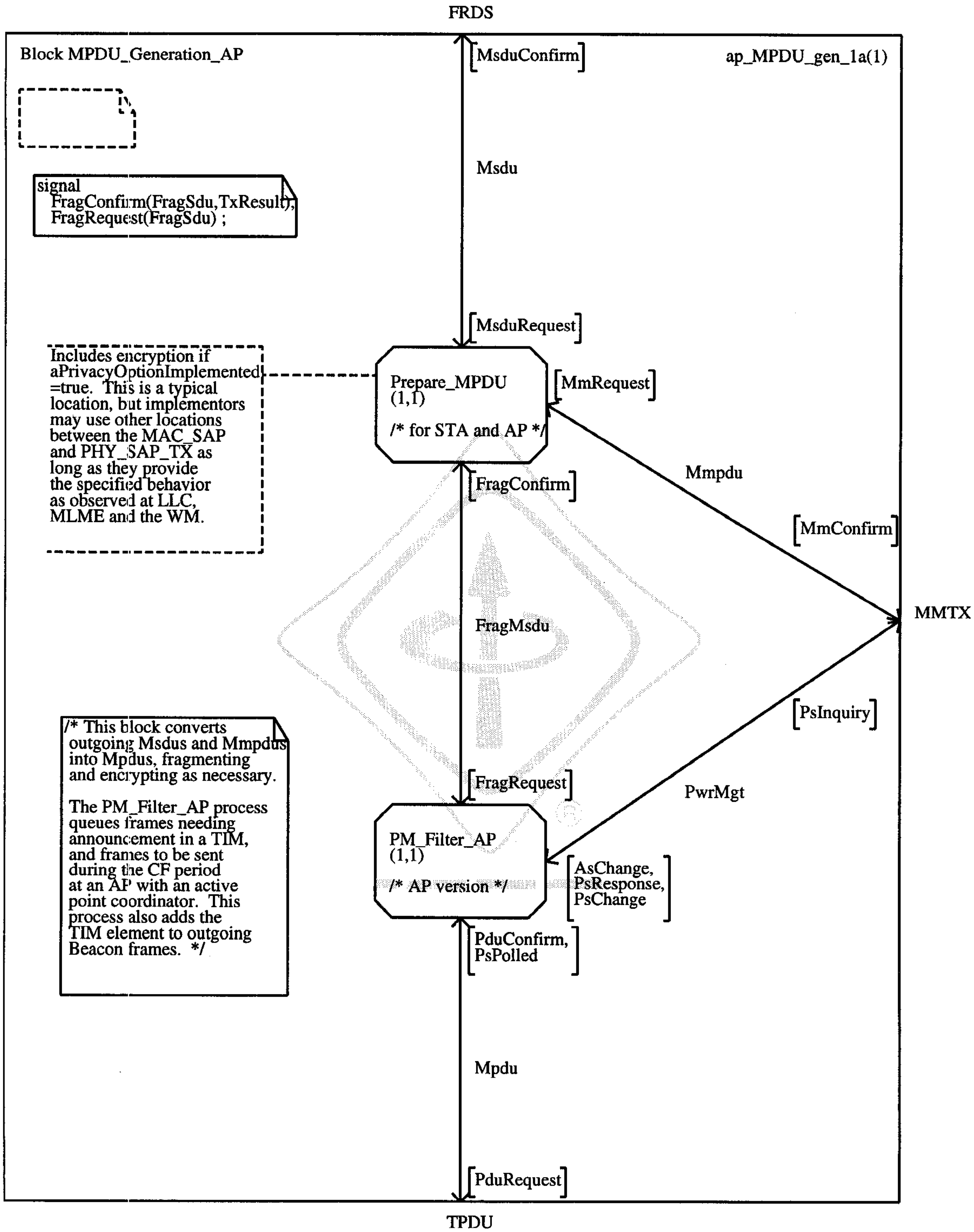


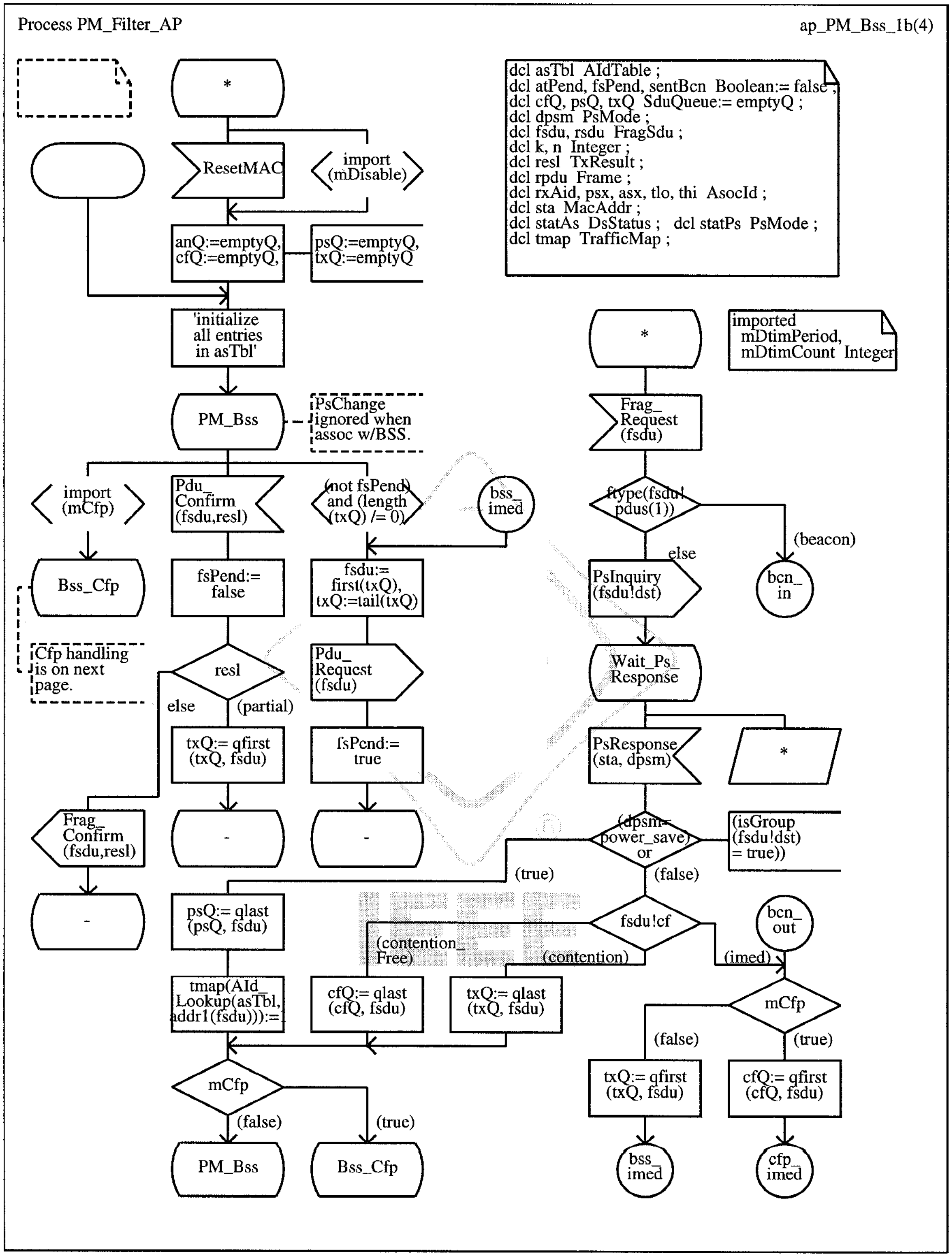
Service AuthReqService_AP
ap_auth_req_1a(1)
dcl auAlg AuthType ;
dcl auCap Capability ;
dcl auRdu, auSdu Frame ;
dcl auRsn ReasonCode ;
dcl auSta MacAddr ;
dcl auSts TxResult ;
dcl auTmot TU ;
/* This service handles DeAuthenticate requests. This service also handles incoming the generation of responses for class 2 errors.
This service does not do authenticate requests because APs never initiate authentication. */
Auth_Req_ Idle
Cls2err (auSta)
MlmeDeauthenticate._ request
(auSta, auRsn)
asRsn:= class2_err
auSdu:= mkFrame (deauth,
auSta, mBssid, auRsn)
Send (auSdu, norm)
Send notification, do not wait for delivery confirmation.
Sst(asSta, de_auth)
Update local stations state records. Sending deauth also clears asoc state if present.
If deauthenticating the current AP, or deauthenticating everyone, end the association (if any) by clearing mBssid and mAssoc.
auSta= mBssId
or isGroup (auSta)
(false)
(true)
mAssoc:=false mBssid:= nullAddr
Xport
auRsn= class2_err
Don't confirm class 2 error notifications.
(true)
(false)
MlmeDis_ associate._ confirm
(successful)
-

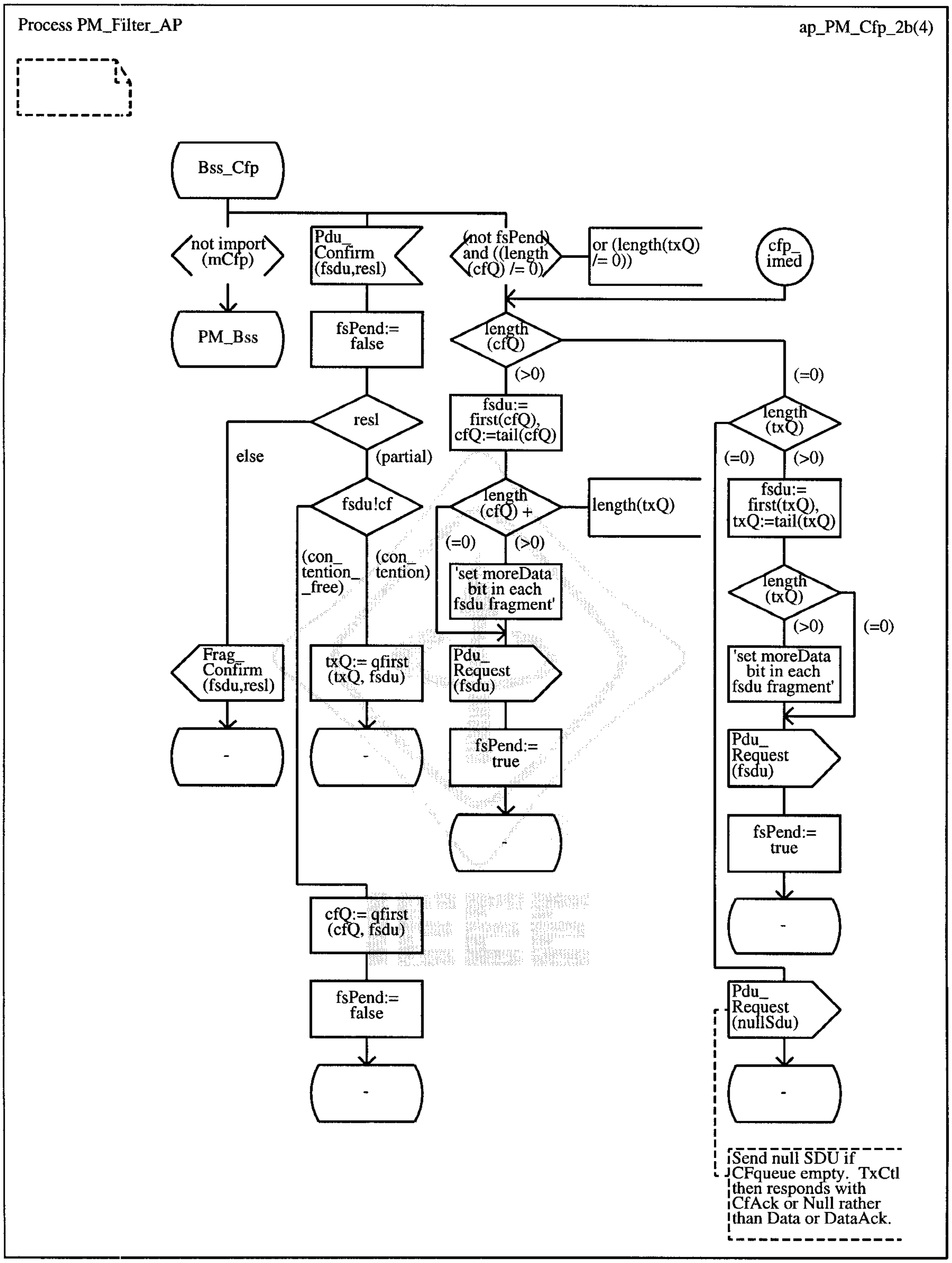

Service Synchronization_AP
ap_Init_1a(3)
dcl yAtimRx, yPsm, yRdtim, yWake Boolean;
dcl yAtw, yBcn, yMocp Time ;
dcl yBcnPeriod, ycmax, ycmin TU ;
dcl ybd BssDscr ;
dcl ybdset BssDscrSet ;
dcl ybtp BssType ;
dcl ybsid MacAddr ;
dcl yclist Intstring ;
dcl ycx, yJto, ytemp Integer ;
dcl yDspm DsParms ;
dcl yFhpm FhParms ;
dcl yIbpm IbssParms ;
dcl ypdly Usec ;
dcl yPhpm PhyParms ;
dcl yRdu, yTdu Frame ;
dcl yssid Octetstring ;
dcl ystp ScanType ;
dcl ytrsl TxResult ;
timer Tscan, Tmocp ;
*
ResetMAC
'reset all intra-MAC remote variables to default values'
Set TSF time to zero.
ytemp:= call TSF (0, true)
Xport
Setting these timers to now causes events in each of the multistate services of the process, forcing each service to its idle state.
reset(Tbcn), set(now,Tauth), set(now,Tchal)
No_BSS

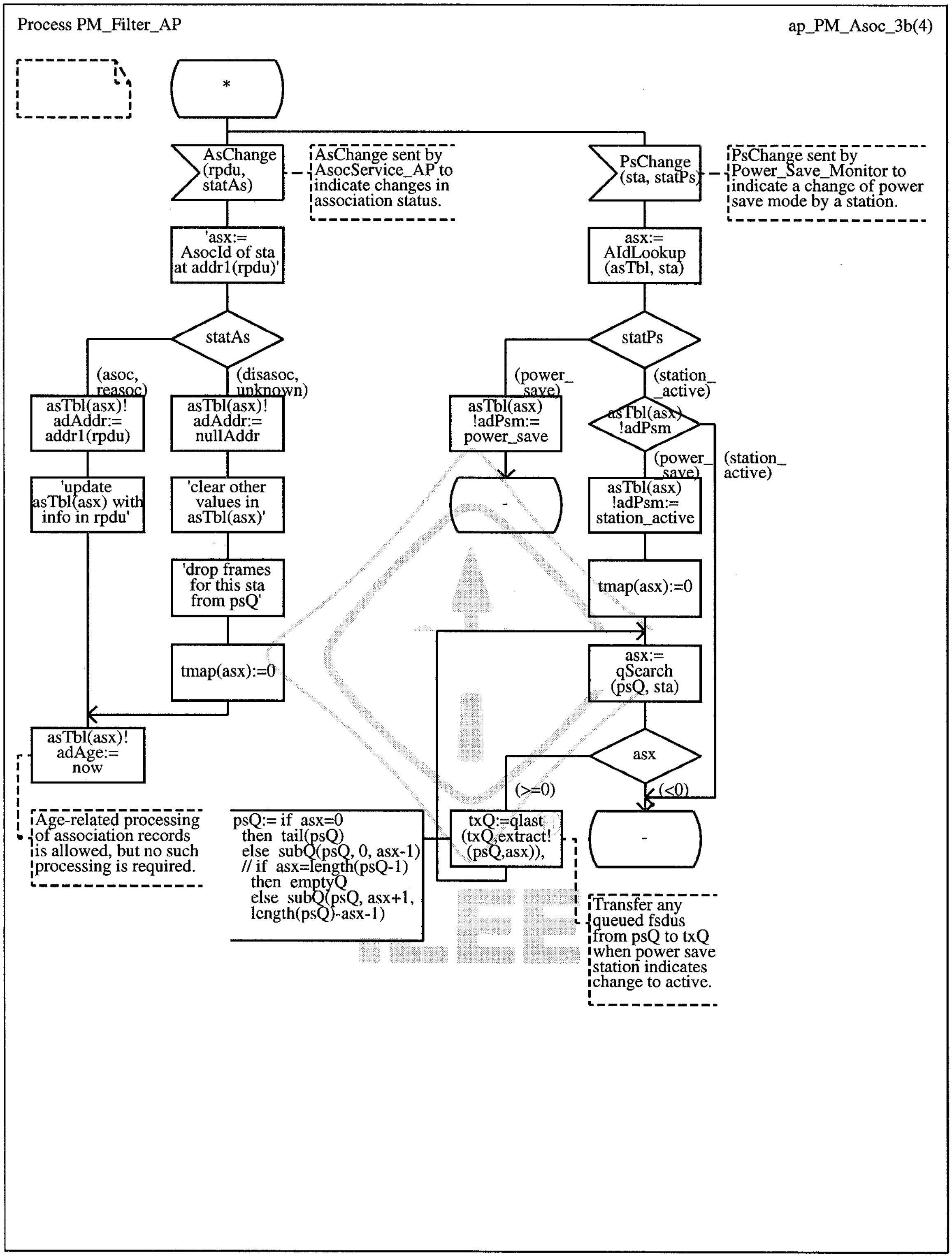

Service Synchronization_AP
ap_Start_Bss_2a(3)
No_BSS
Start IBSS on this page, join on next page.
Activate Station state machine.
MlmeStart_ request (mSsid, yBtp,
yBcnPeriod, mDtimPeriod, yCfpm, yPhpm, /* ibpm */ mCap, mBrates, mOrates)
yBytp
(indep_ endent)
(infra_ structure)
Sta_Active
'parameters valid'
(false)
(true)
mPsm:= station_active, mPss:=awake
MlmeStart_ confirm (invalid)
No_Bss
yBcn:= tTU (yBcnPeriod)
mCfAvail:= if dtim_ Period
(yCfpm) /= 0 then true else false fi
'set mCfPoll and mCap for operating state'
'set aCfPeriod and aCfMaxDuration from yCfpm'
'set actual phy parameters from phpm'
Xport
bss_ init
bss_ init
'using FH phy'
(false)
(true)
yMocp:=tTU (import(aMedium OccupancyLimit))
yMocp:=tTU (dwellTime (yFhpm))
mNextBdry:= now+(yMocp - (call TSF
(0,false) mod yMocp))
set (mNextBdry, Tmocp)
Initialize dwell timer.
'yChan:= first (or only) channel'
Set starting channel (FH) or operating channel (DS), null for IR.
SwChnl (yChan,true)
mNextTbtt:= now+(yBcn - (call TSF
(0,false) mod yBcn))
set (mNextBdry, Tbcn)
Initialize beacon timer.
Xport
MlmeStart_ confirm (success)
Bss

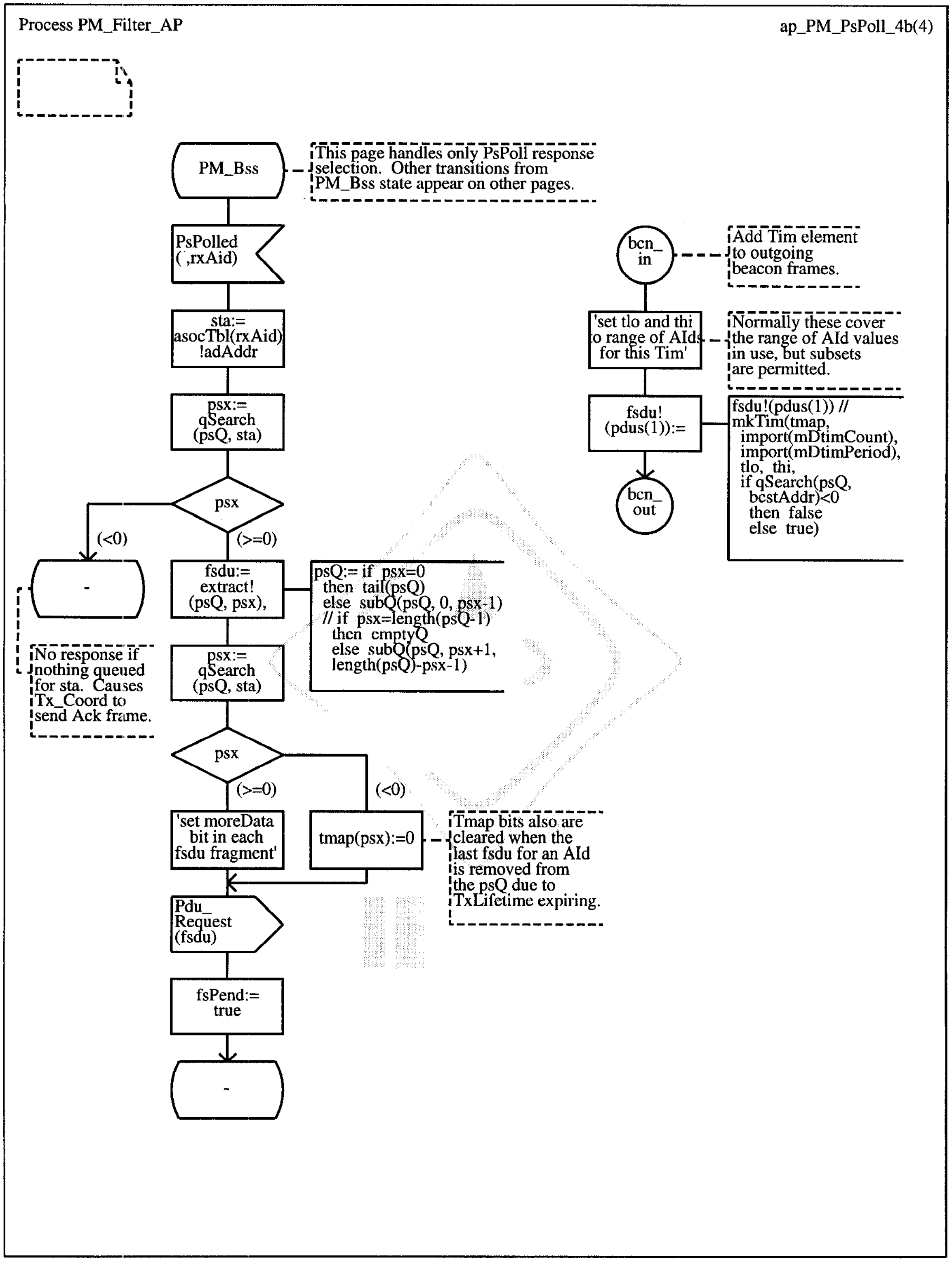

Service Synchronization_AP
ap_TSF_bss_3a(3)
Bss
Tbcn
set (now+yBcn, Tbcn)
ytdu:= mkFrame (beacon,
bcstAddr, mBssId, O8 /* timestamp inserted by tx */ // mk2octets (yBcnPeriod) // mCap // mkElem (eSsId,mSsId) // mkElem (eSupRates, mOrates)
mDtimCount:= mDtimCount-1
mPcAvail
(false)
(true)
ytdu:= ytdu //
mkElem (eCfp,yCfpm)
yCfpm:= setCfpCount (yCfpm,
if yCfCnt=0
yCfpm:= setCfpPeriod (yCfpm,
import (aCfpPeriod))
yCfpm:= setCfpMaxDur (yCfpm,
import (aCfpMaxDuration))
'update cfDurRem and mCfp'
'add proper phy parameter set element'
Xport, Send (ytdu,imed)
TIM element gets added by PM_Filter_AP.
-
ProbeReq (yRdu)
ytdu:= mkFrame (probe_rsp
bcstAddr, mBssId, O8 /* timestamp inserted by tx */ // mk2octets (yBcnPeriod) // mCap // mkElem (eSsId,mSsId) // mkElem (eCfp,yCfpm) // mkElem (eSupRates, mOrates) // mkElem (ePhpm,yPhpm))
Send (ytdu,norm)
-
Tmocp
mNextBdry:= mNextBdry + yMocp
set (mNextBdry, Tmocp)
'using FH phy'
(false)
(true)
'yChan:= next channel in hop seq'
SwChnl (,false)
SwChnl (yChan,true)
Wait_Hop_ Bss
*
SwDone
Bss

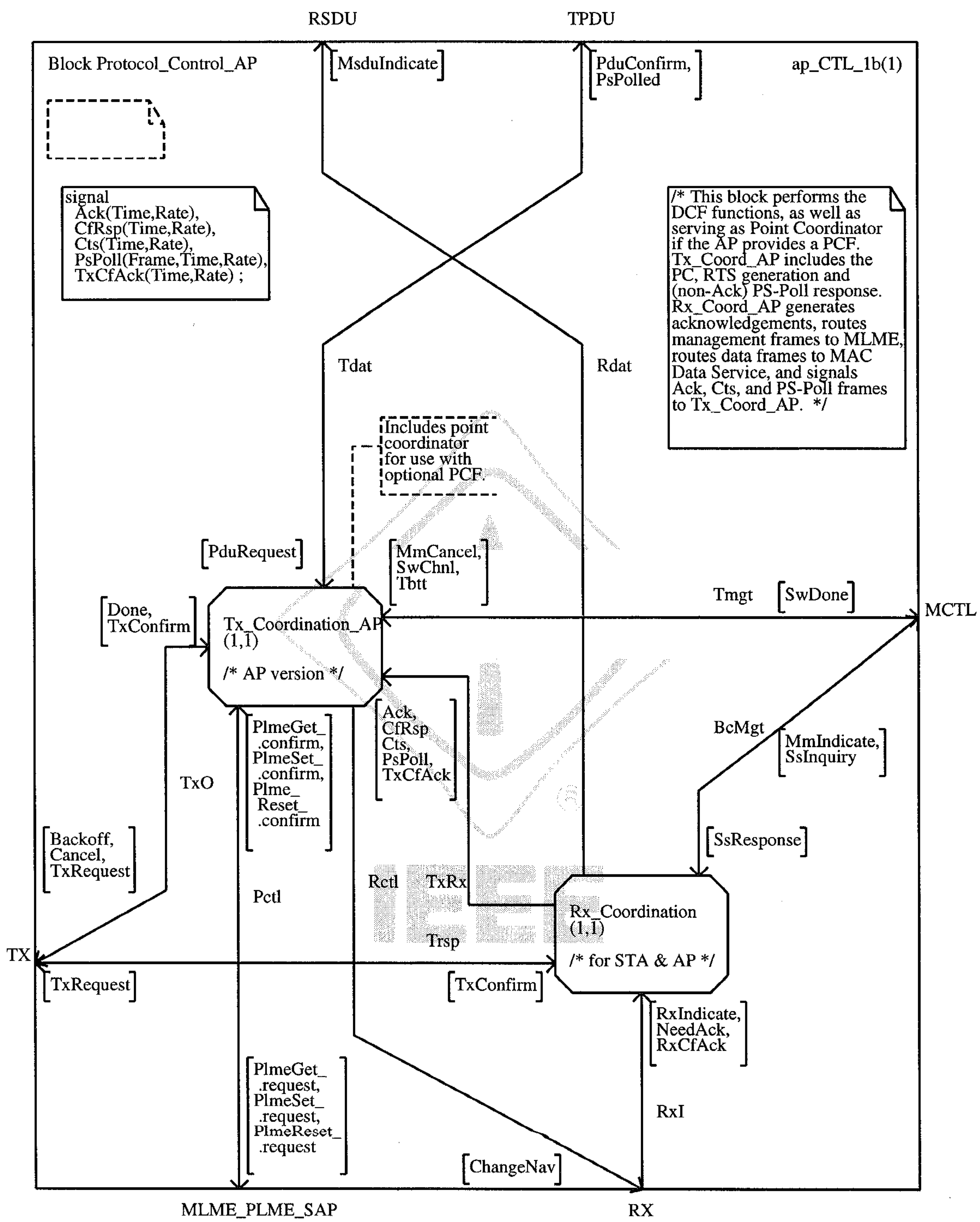

FRDS
Block MPDU_Generation_AP
ap_MPDU_gen_1a(1)
MsduConfirm
Msdu
signal
FragConfirm(FragSdu,TxResult),
FragRequest(FragSdu) ;
MsduRequest
Includes encryption if aPrivacyOptionImplemented =true. This is a typical location, but implementors may use other locations between the MAC_SAP and PHY_SAP_TX as long as they provide the specified behavior as observed at LLC, MLME and the WM.
Prepare_MPDU
(1,1)
/* for STA and AP */
MmRequest
Mmpdu
FragConfirm
MmConfirm
MMTX
FragMsdu
PsInquiry
/* This block converts outgoing Msdus and Mmpdus into Mpdus, fragmenting and encrypting as necessary.
The PM_Filter_AP process queues frames needing announcement in a TIM, and frames to be sent during the CF period at an AP with an active point coordinator. This process also adds the TIM element to outgoing Beacon frames. */
FragRequest
PwrMgt
PM_Filter_AP
(1,1)
/* AP version */
AsChange,
PsResponse,
PsChange
PduConfirm,
PsPolled
Mpdu
PduRequest
TPDU

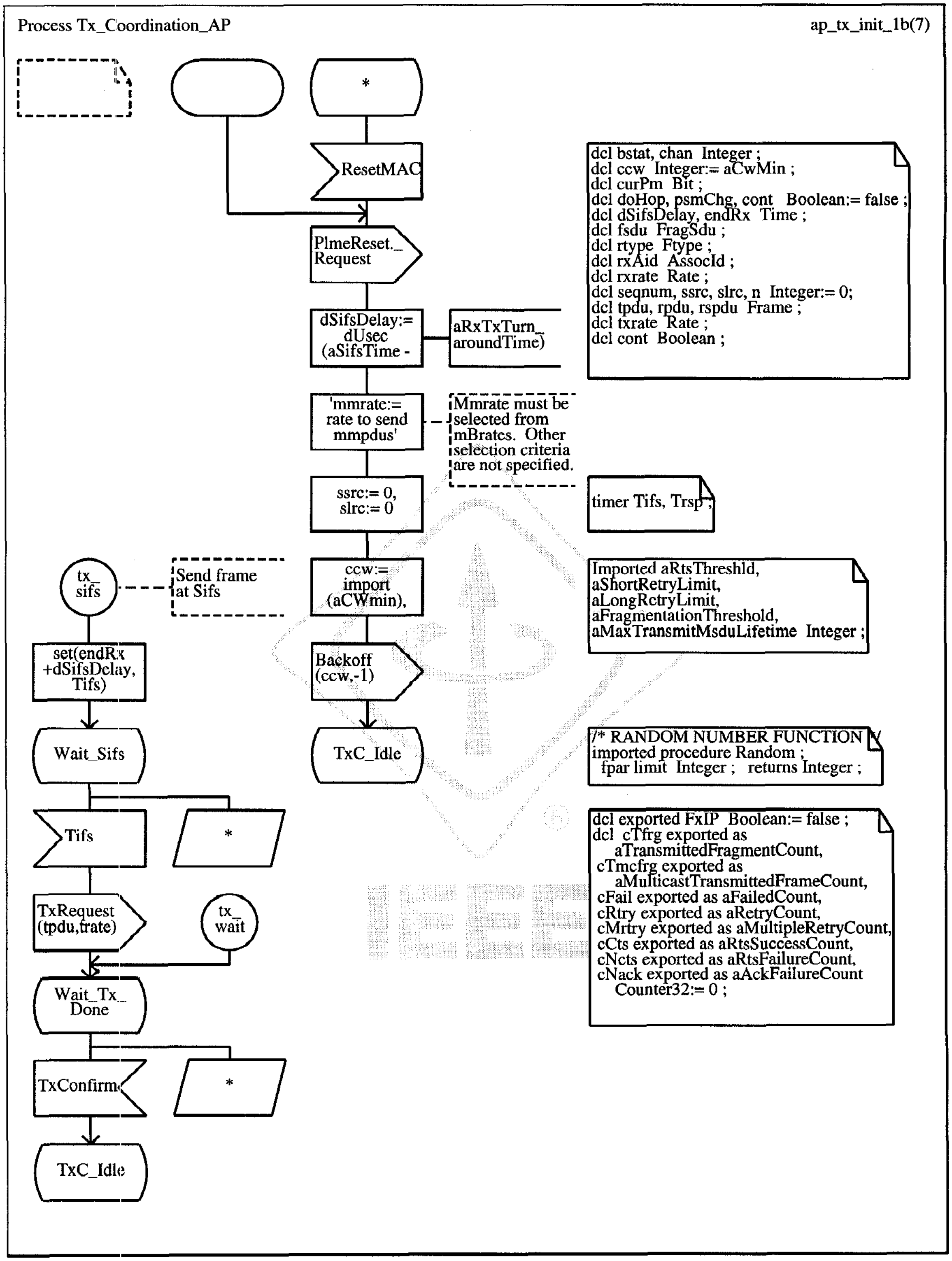

Process PM_Filter_AP
ap_PM_Bss_1b(4)
dcl asTbl AIdTable ;
dcl atPend, fsPend, sentBcn Boolean:= false ;
dcl cfQ, psQ, txQ SduQueue:= emptyQ ;
dcl dpsm PsMode ;
dcl fsdu, rsdu FragSdu ;
dcl k, n Integer ;
dcl resl TxResult ;
dcl rpdu Frame ;
dcl rxAid, psx, asx, tlo, thi AsocId ;
dcl sta MacAddr ;
dcl statAs DsStatus ; dcl statPs PsMode ;
dcl tmap TrafficMap ;
*
ResetMAC
import (mDisable)
anQ:=emptyQ, cfQ:=emptyQ,
psQ:=emptyQ, txQ:=emptyQ
'initialize all entries in asTbl'
PM_Bss
PsChange ignored when assoc w/BSS.
import (mCfp)
Pdu_ Confirm (fsdu,resl)
(not fsPend) and (length (txQ) /= 0)
bss_ imed
Bss_Cfp
Cfp handling is on next page.
fsPend:= false
fsdu:= first(txQ), txQ:=tail(txQ)
resl
else
(partial)
Pdu_ Request (fsdu)
txQ:= qfirst (txQ, fsdu)
fsPend:= true
Frag_ Confirm (fsdu,resl)
-
psQ:= qlast (psQ, fsdu)
tmap(AId_ Lookup(asTbl, addr1(fsdu))):=1
mCfp
(false)
(true)
PM_Bss
Bss_Cfp
imported mDtimPeriod, mDtimCount Integer
Frag_ Request (fsdu)
ftype(fsdu! pdus(1))
else
(beacon)
bcn_ in
PsInquiry (fsdu!dst)
Wait_Ps_ Response
PsResponse (sta, dpsm)
(dpsm= power_save) or
(isGroup (fsdu!dst) = true))
(true)
(false)
fsdu!cf
(contention_ Free)
(contention)
(imed)
bcn_ out
cfQ:= qlast (cfQ, fsdu)
txQ:= qlast (txQ, fsdu)
txQ:= qfirst (txQ, fsdu)
cfQ:= qfirst (cfQ, fsdu)
bss_ imed
cfp_ imed

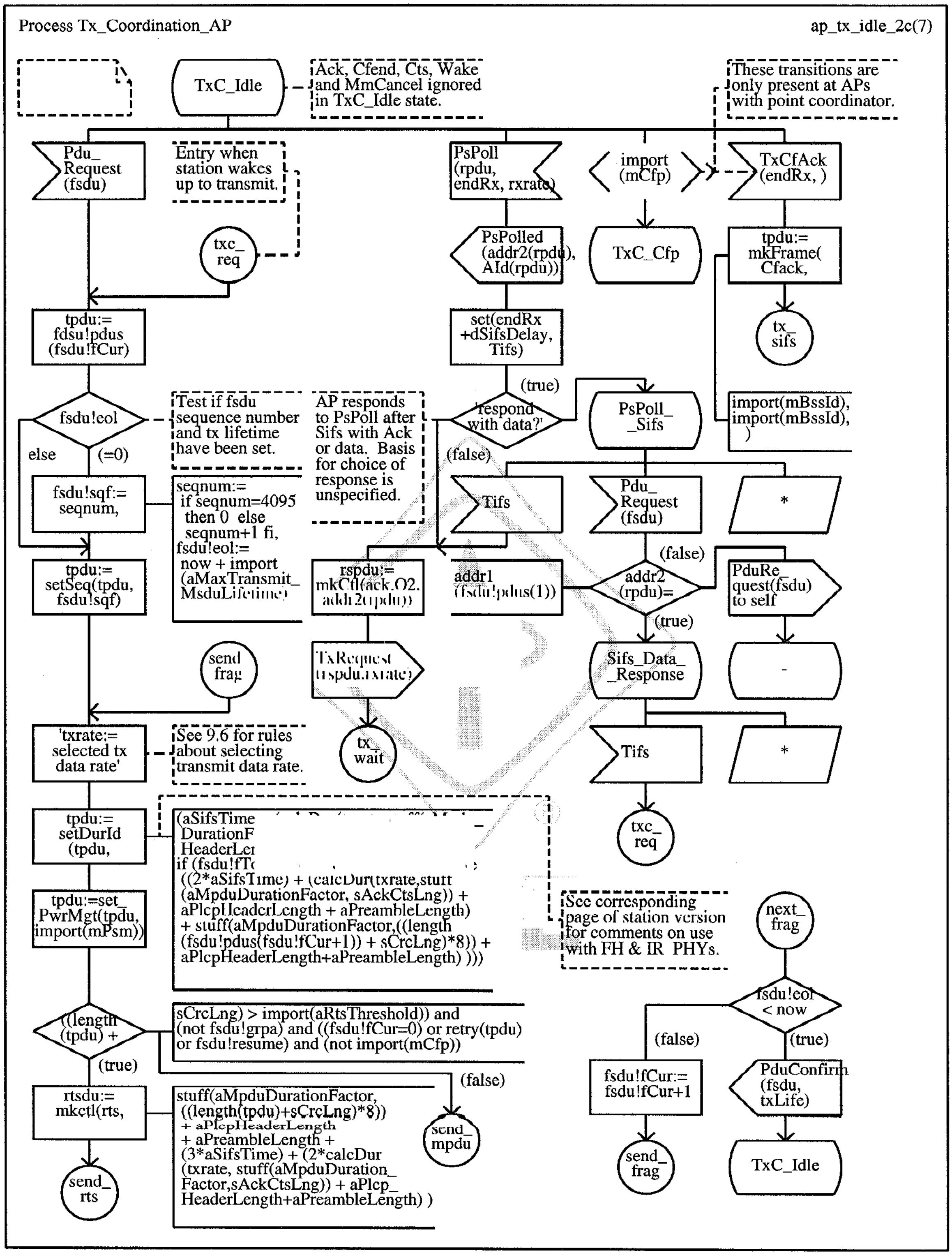

Process PM_Filter_AP
ap_PM_Cfp_2b(4)
Bss_Cfp
not import (mCfp)
PM_Bss
Pdu_ Confirm (fsdu,resl)
fsPend:= false
resl
else
(partial)
fsdu!cf
(con_ tention_ _free)
(con_ tention)
Frag_ Confirm (fsdu,resl)
-
txQ:= qfirst (txQ, fsdu)
-
cfQ:= qfirst (cfQ, fsdu)
fsPend:= false
-
(not fsPend) and ((length (cfQ) /= 0)
or (length(txQ) /= 0))
cfp_ imed
length (cfQ)
(>0)
(=0)
fsdu:= first(cfQ), cfQ:=tail(cfQ)
length (cfQ) +
length(txQ)
(=0)
(>0)
'set moreData bit in each fsdu fragment'
Pdu_ Request (fsdu)
fsPend:= true
-
length (txQ)
(=0)
(>0)
fsdu:= first(txQ), txQ:=tail(txQ)
length (txQ)
(>0)
(=0)
'set moreData bit in each fsdu fragment'
Pdu_ Request (fsdu)
fsPend:= true
-
Pdu_ Request (nullSdu)
-
Send null SDU if CFqueue empty. TxCtl then responds with CfAck or Null rather than Data or DataAck.

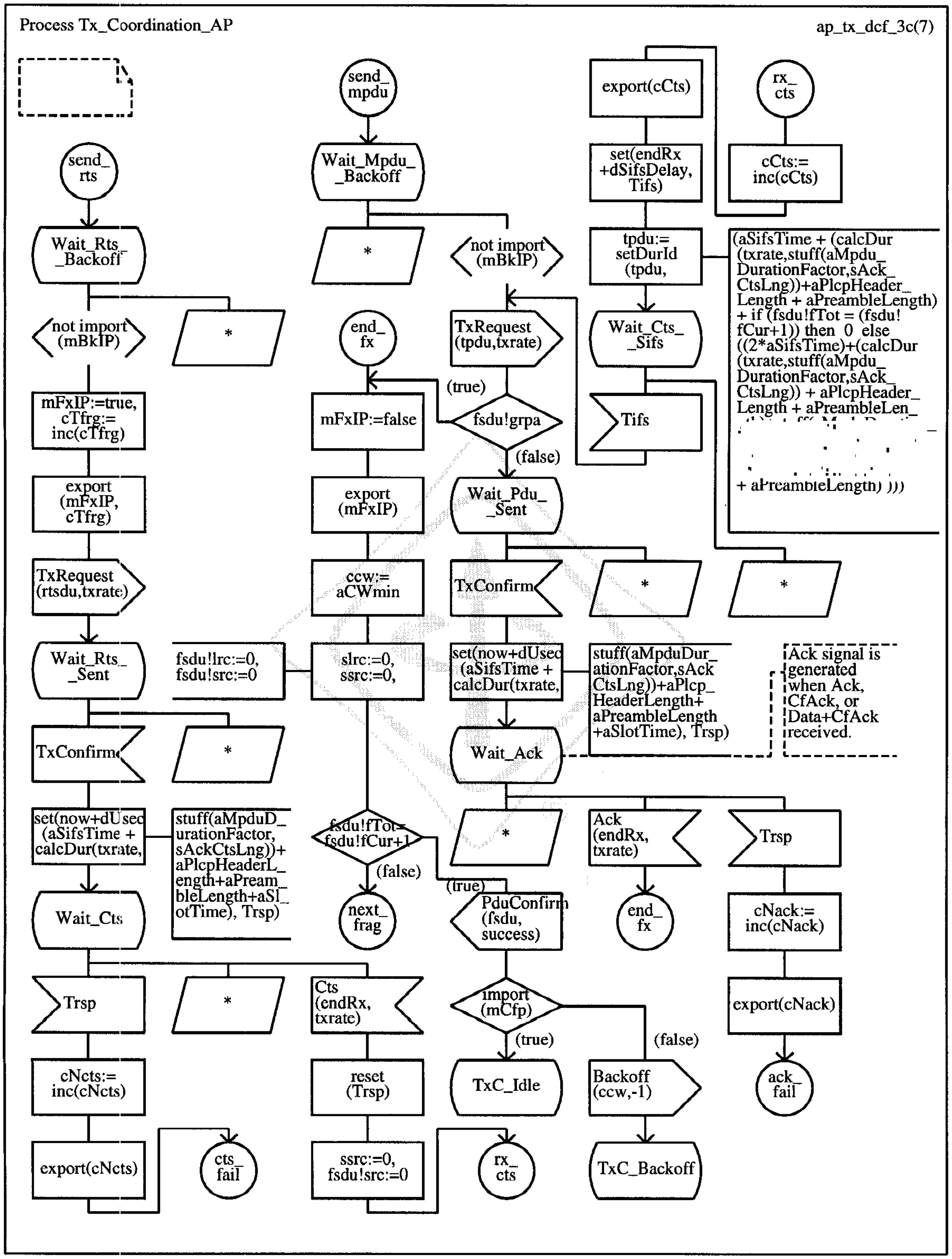

Process PM_Filter_AP
ap_PM_Asoc_3b(4)
*
AsChange (rpdu, statAs)
AsChange sent by AsocService_AP to indicate changes in association status.
PsChange (sta, statPs)
PsChange sent by Power_Save_Monitor to indicate a change of power save mode by a station.
'asx:= AsocId of sta at addr1(rpdu)'
asx:= AIdLookup (asTbl, sta)
statAs
(asoc, reasoc)
(disasoc, unknown)
asTbl(asx)! adAddr:= addr1(rpdu)
asTbl(asx)! adAddr:= nullAddr
'update asTbl(asx) with info in rpdu'
'clear other values in asTbl(asx)'
'drop frames for this sta from psQ'
tmap(asx):=0
asTbl(asx)! adAge:= now
Age-related processing of association records is allowed, but no such processing is required.
statPs
(power_save)
(station_active)
asTbl(asx) !adPsm:= power_save
asTbl(asx) !adPsm
(power_save)
(station_active)
asTbl(asx) !adPsm:= station_active
tmap(asx):=0
asx:= qSearch (psQ, sta)
asx
(>=0)
(<0)
txQ:=qlast (txQ,extract! (psQ,asx)),
psQ:= if asx=0 then tail(psQ) else subQ(psQ, 0, asx-1) // if asx=length(psQ-1) then emptyQ else subQ(psQ, asx+1, length(psQ)-asx-1)
Transfer any queued fsdus from psQ to txQ when power save station indicates change to active.

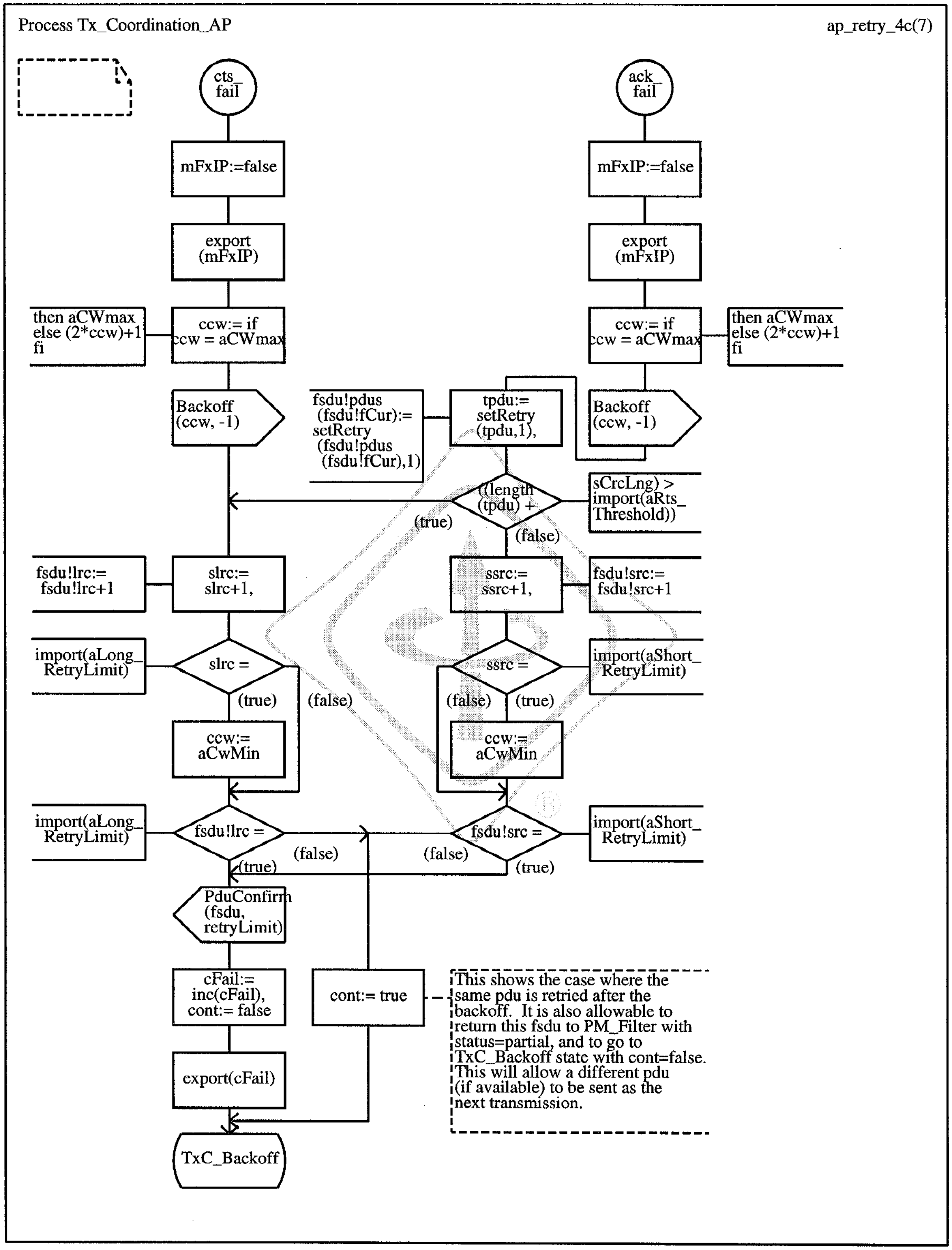

Process PM_Filter_AP
ap_PM_PsPoll_4b(4)
PM_Bss
This page handles only PsPoll response selection. Other transitions from PM_Bss state appear on other pages.
PsPolled (,rxAid)
sta:= asocTbl(rxAid) !adAddr
psx:= qSearch (psQ, sta)
psx
(<0)
(>=0)
-
No response if nothing queued for sta. Causes Tx_Coord to send Ack frame.
fsdu:= extract! (psQ, psx),
psQ:= if psx=0
then tail(psQ)
else subQ(psQ, 0, psx-1)
// if psx=length(psQ-1)
then emptyQ
else subQ(psQ, psx+1, length(psQ)-psx-1)
psx:= qSearch (psQ, sta)
psx
(>=0)
(<0)
'set moreData bit in each fsdu fragment'
tmap(psx):=0
Tmap bits also are cleared when the last fsdu for an AId is removed from the psQ due to TxLifetime expiring.
Pdu_ Request (fsdu)
fsPend:= true
-
bcn_ in
Add Tim element to outgoing beacon frames.
'set tlo and thi to range of AIds for this Tim'
Normally these cover the range of AId values in use, but subsets are permitted.
fsdu! (pdus(1)):=
fsdu!(pdus(1)) //
mkTim(tmap,
import(mDtimCount),
import(mDtimPeriod),
tlo, thi,
if qSearch(psQ, bcstAddr)<0
then false
else true)
bcn_ out

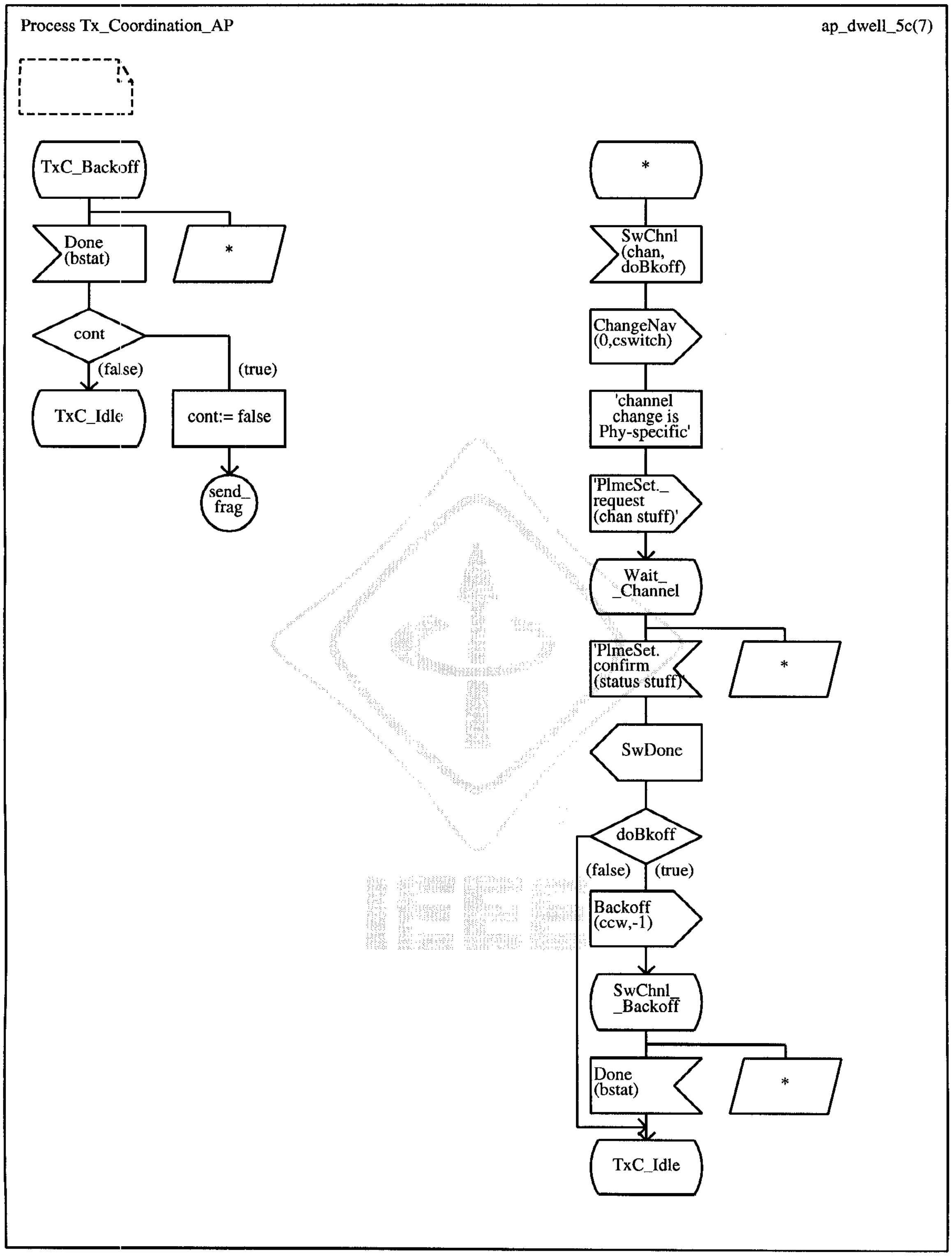

RSDU
TPDU
Block Protocol_Control_AP
MsduIndicate
PduConfirm, PsPolled
ap_CTL_1b(1)
signal
Ack(Time,Rate),
CfRsp(Time,Rate),
Cts(Time,Rate),
PsPoll(Frame,Time,Rate),
TxCfAck(Time,Rate) ;
/* This block performs the DCF functions, as well as serving as Point Coordinator if the AP provides a PCF. Tx_Coord_AP includes the PC, RTS generation and (non-Ack) PS-Poll response. Rx_Coord_AP generates acknowledgements, routes management frames to MLME, routes data frames to MAC Data Service, and signals Ack, Cts, and PS-Poll frames to Tx_Coord_AP. */
Tdat
Rdat
Includes point coordinator for use with optional PCF.
PduRequest
MmCancel, SwChnl, Tbtt
Tmgt
SwDone
MCTL
Done, TxConfirm
Tx_Coordination_AP (1,1)
/* AP version */
Ack, CfRsp Cts, PsPoll, TxCfAck
PlmeGet_ .confirm, PlmeSet_ .confirm, Plme_ Reset_ .confirm
BcMgt
MmIndicate, SsInquiry
TxO
SsResponse
Backoff, Cancel, TxRequest
Pctl
Rctl
TxRx
Rx_Coordination (1,1)
/* for STA & AP */
Trsp
TX
TxRequest
TxConfirm
RxIndicate, NeedAck, RxCfAck
PlmeGet_ .request, PlmeSet_ .request, PlmeReset_ .request
RxI
ChangeNav
MLME_PLME_SAP
RX

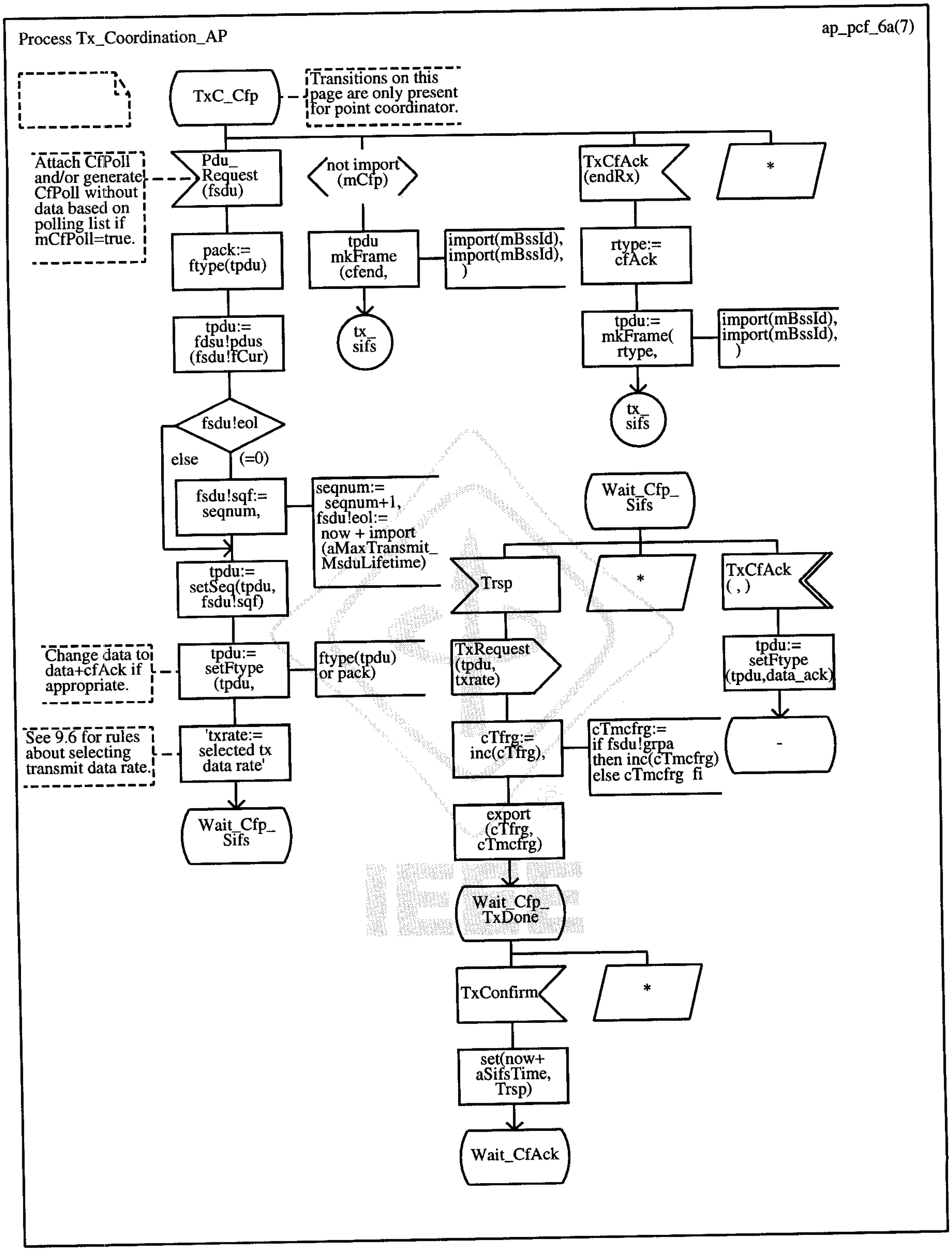

Process Tx_Coordination_AP
ap_tx_init_1b(7)
*
ResetMAC
PlmeReset._
Request
dSifsDelay:=
dUsec
(aSifsTime -
aRxTxTurn_
aroundTime)
'mmrate:=
rate to send
mmpdus'
Mmrate must be
selected from
mBrates. Other
selection criteria
are not specified.
ssrc:= 0,
slrc:= 0
ccw:=
import
(aCWmin),
Backoff
(ccw,-1)
TxC_Idle
dcl bstat, chan Integer ;
dcl ccw Integer:= aCwMin ;
dcl curPm Bit ;
dcl doHop, psmChg, cont Boolean:= false ;
dcl dSifsDelay, endRx Time ;
dcl fsdu FragSdu ;
dcl rtype Ftype ;
dcl rxAid AssocId ;
dcl rxrate Rate ;
dcl seqnum, ssrc, slrc, n Integer:= 0;
dcl tpdu, rpdu, rspdu Frame ;
dcl txrate Rate ;
dcl cont Boolean ;
timer Tifs, Trsp ;
Imported aRtsThreshld,
aShortRetryLimit,
aLongRetryLimit,
aFragmentationThreshold,
aMaxTransmitMsduLifetime Integer ;
/* RANDOM NUMBER FUNCTION */
imported procedure Random ;
fpar limit Integer ; returns Integer ;
dcl exported FxIP Boolean:= false ;
dcl cTfrg exported as
aTransmittedFragmentCount,
cTmcfrg exported as
aMulticastTransmittedFrameCount,
cFail exported as aFailedCount,
cRtry exported as aRetryCount,
cMrtry exported as aMultipleRetryCount,
cCts exported as aRtsSuccessCount,
cNcts exported as aRtsFailureCount,
cNack exported as aAckFailureCount
Counter32:= 0 ;
tx_
sifs
Send frame
at Sifs
set(endRx
+dSifsDelay,
Tifs)
Wait_Sifs
Tifs
*
TxRequest
(tpdu,trate)
tx_
wait
Wait_Tx_
Done
TxConfirm
*
TxC_Idle

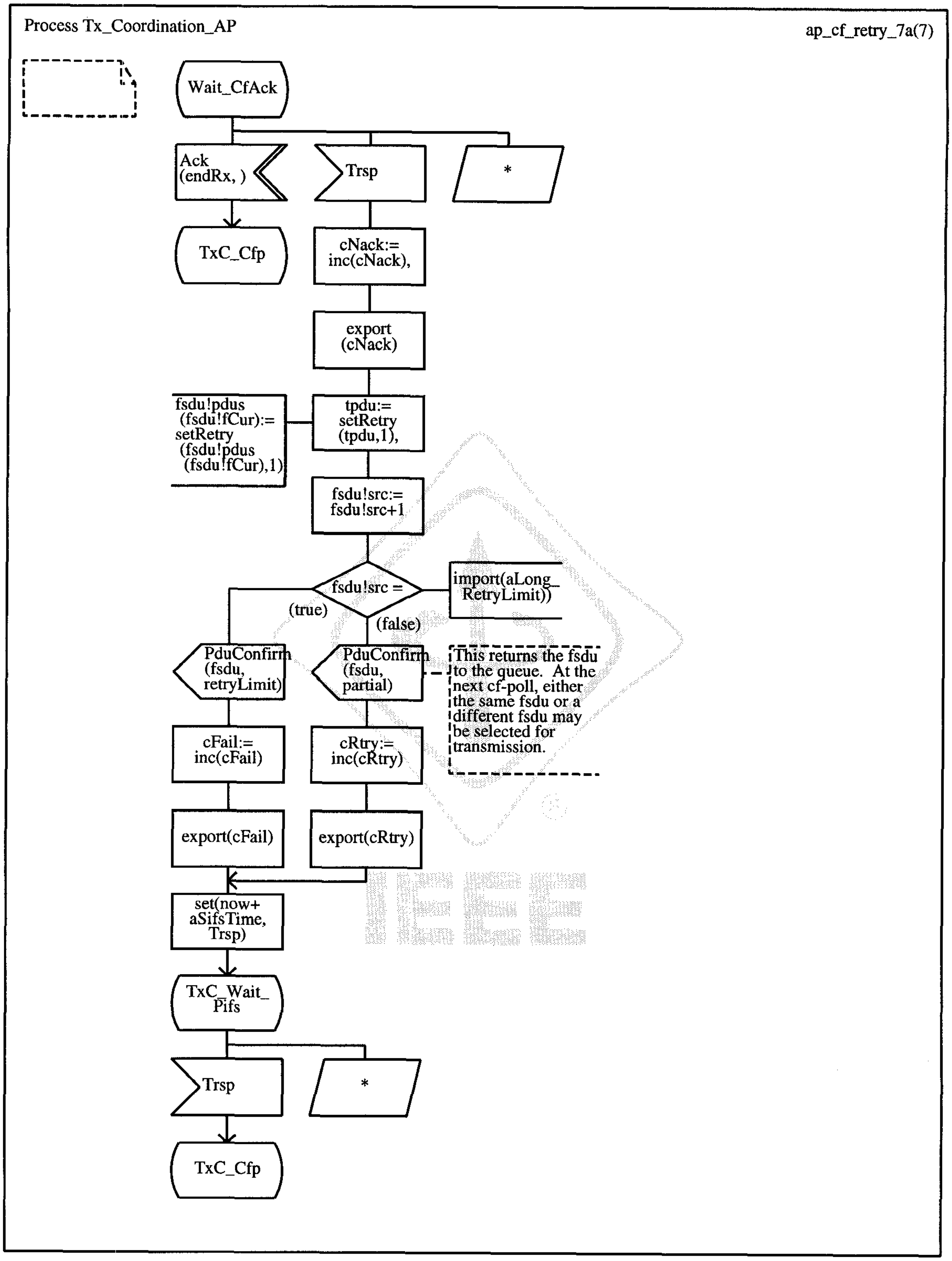

Process Tx_Coordination_AP
ap_tx_idle_2c(7)
TxC_Idle
Ack, Cfend, Cts, Wake and MmCancel ignored in TxC_Idle state.
These transitions are only present at APs with point coordinator.
Pdu_ Request (fsdu)
Entry when station wakes up to transmit.
PsPoll (rpdu, endRx, rxrate)
import (mCfp)
TxCfAck (endRx,)
txc_ req
PsPolled (addr2(rpdu), AId(rpdu))
TxC_Cfp
tpdu:= mkFrame(Cfack,
tpdu:= fdsu!pdus (fsdu!fCur)
set(endRx +dSifsDelay, Tifs)
tx_ sifs
(true)
fsdu!eol
Test if fsdu sequence number and tx lifetime have been set.
AP responds to PsPoll after Sifs with Ack or data. Basis for choice of response is unspecified.
'respond with data?'
PsPoll_ _Sifs
import(mBssId), import(mBssId),)
else
(=0)
(false)
fsdu!sqf:= seqnum,
seqnum:= if seqnum=4095 then 0 else seqnum+1 fi, fsdu!eol:= now + import (aMaxTransmit_ MsduLifetime)
Tifs
Pdu_ Request (fsdu)
*
tpdu:= setSeq(tpdu, fsdu!sqf)
rspdu:= mkCtl(ack,0,2, addr2(rpdu))
addr1 (fsdu!pdus(1))
addr2 (rpdu)=
(false)
PduRe_ quest(fsdu) to self
(true)
send frag
TxRequest (rspdu,rxrate)
Sifs_Data_ _Response
-
'txrate:= selected tx data rate'
See 9.6 for rules about selecting transmit data rate.
tx_ wait
Tifs
*
txc_ req
tpdu:= setDurId (tpdu,
(aSifsTime DurationF HeaderLe if (fsdu!fT ((2*aSifsTime) + (calcDur(txrate,stuff (aMpduDurationFactor, sAckCtsLng)) + aPlcpHeaderLength + aPreambleLength) + stuff(aMpduDurationFactor,((length (fsdu!pdus(fsdu!fCur+1)) + sCrcLng)*8)) + aPlcpHeaderLength+aPreambleLength))))
tpdu:=set_ PwrMgt(tpdu, import(mPsm))
See corresponding page of station version for comments on use with FH & IR PHYs.
next_ frag
((length (tpdu) +
sCrcLng) > import(aRtsThreshold)) and (not fsdu!grpa) and ((fsdu!fCur=0) or retry(tpdu) or fsdu!resume) and (not import(mCfp))
fsdu!eol < now
(false)
(true)
(true)
(false)
fsdu!fCur:= fsdu!fCur+1
PduConfirm (fsdu, txLife)
rtsdu:= mkctl(rts,
stuff(aMpduDurationFactor, ((length(tpdu)+sCrcLng)*8)) + aPlcpHeaderLength + aPreambleLength + (3*aSifsTime) + (2*calcDur (txrate, stuff(aMpduDuration_ Factor,sAckCtsLng)) + aPlcp_ HeaderLength+aPreambleLength))
send_ mpdu
send_ frag
TxC_Idle
send_ rts


Process Tx_Coordination_AP
ap_tx_dcf_3c(7)
send_ rts
Wait_Rts_ _Backoff
not import (mBkIP)
*
mFxIP:=true, cTfrg:= inc(cTfrg)
export (mFxIP, cTfrg)
TxRequest (rtsdu,txrate)
Wait_Rts_ _Sent
fsdu!lrc:=0, fsdu!src:=0
TxConfirm
*
set(now+dUsec (aSifsTime + calcDur(txrate,
stuff(aMpduD_ urationFactor, sAckCtsLng))+ aPlcpHeaderL_ ength+aPream_ bleLength+aSl_ otTime), Trsp)
Wait_Cts
Trsp
*
cNcts:= inc(cNcts)
export(cNcts)
cts_ fail
send_ mpdu
Wait_Mpdu_ _Backoff
*
not import (mBkIP)
end_ fx
TxRequest (tpdu,txrate)
(true)
mFxIP:=false
fsdu!grpa
(false)
export (mFxIP)
Wait_Pdu_ _Sent
ccw:= aCWmin
TxConfirm
slrc:=0, ssrc:=0,
set(now+dUsec (aSifsTime + calcDur(txrate,
stuff(aMpduDur_ ationFactor,sAck CtsLng))+aPlcp_ HeaderLength+ aPreambleLength +aSlotTime), Trsp)
Wait_Ack
fsdu!fTot= fsdu!fCur+1
(false)
(true)
next_ frag
PduConfirm (fsdu, success)
*
Cts (endRx, txrate)
import (mCfp)
(true)
(false)
reset (Trsp)
TxC_Idle
ssrc:=0, fsdu!src:=0
rx_ cts
export(cCts)
rx_ cts
set(endRx +dSifsDelay, Tifs)
cCts:= inc(cCts)
tpdu:= setDurId (tpdu,
(aSifsTime + (calcDur (txrate,stuff(aMpdu_ DurationFactor,sAck_ CtsLng))+aPlcpHeader_ Length + aPreambleLength) + if (fsdu!fTot = (fsdu! fCur+1)) then 0 else ((2*aSifsTime)+(calcDur (txrate,stuff(aMpdu_ DurationFactor,sAck_ CtsLng)) + aPlcpHeader_ Length + aPreambleLen_
+ aPreambleLength))))
Wait_Cts_ _Sifs
Tifs
*
*
Ack signal is generated when Ack, CfAck, or Data+CfAck received.
Ack (endRx, txrate)
Trsp
end_ fx
cNack:= inc(cNack)
export(cNack)
ack_ fail
Backoff (ccw,-1)
TxC_Backoff


Process Tx_Coordination_AP
ap_retry_4c(7)
cts_ fail
mFxIP:=false
export (mFxIP)
ccw:= if ccw = aCWmax
then aCWmax else (2*ccw)+1 fi
Backoff (ccw, -1)
fsdu!pdus (fsdu!fCur):= setRetry (fsdu!pdus (fsdu!fCur),1)
tpdu:= setRetry (tpdu,1),
((length (tpdu) +
sCrcLng) > import(aRts_ Threshold))
(true)
(false)
ack_ fail
mFxIP:=false
export (mFxIP)
ccw:= if ccw = aCWmax
then aCWmax else (2*ccw)+1 fi
Backoff (ccw, -1)
fsdu!lrc:= fsdu!lrc+1
slrc:= slrc+1,
ssrc:= ssrc+1,
fsdu!src:= fsdu!src+1
import(aLong_ RetryLimit)
slrc =
(true)
(false)
ssrc =
(false)
(true)
import(aShort_ RetryLimit)
ccw:= aCwMin
ccw:= aCwMin
import(aLong_ RetryLimit)
fsdu!lrc =
(false)
(false)
fsdu!src =
import(aShort_ RetryLimit)
(true)
(true)
PduConfirm (fsdu, retryLimit)
cFail:= inc(cFail), cont:= false
cont:= true
export(cFail)
TxC_Backoff
This shows the case where the same pdu is retried after the backoff. It is also allowable to return this fsdu to PM_Filter with status=partial, and to go to TxC_Backoff state with cont=false. This will allow a different pdu (if available) to be sent as the next transmission.


Process Tx_Coordination_AP
ap_dwell_5c(7)
TxC_Backoff
Done
(bstat)
*
cont
(false)
(true)
TxC_Idle
cont:= false
send_
frag
*
SwChnl
(chan,
doBkoff)
ChangeNav
(0,cswitch)
'channel
change is
Phy-specific'
'PlmeSet._
request
(chan stuff)'
Wait_
_Channel
'PlmeSet.
confirm
(status stuff)
*
SwDone
doBkoff
(false)
(true)
Backoff
(ccw,-1)
SwChnl_
_Backoff
Done
(bstat)
*
TxC_Idle


Process Tx_Coordination_AP
ap_pcf_6a(7)
TxC_Cfp
Transitions on this page are only present for point coordinator.
Attach CfPoll and/or generate CfPoll without data based on polling list if mCfPoll=true.
Pdu_ Request (fsdu)
not import (mCfp)
TxCfAck (endRx)
*
pack:= ftype(tpdu)
tpdu mkFrame (cfend,
import(mBssId), import(mBssId),)
rtype:= cfAck
tpdu:= fdsu!pdus (fsdu!fCur)
tx_ sifs
tpdu:= mkFrame(rtype,
import(mBssId), import(mBssId),)
fsdu!eol
else
(=0)
fsdu!sqf:= seqnum,
seqnum:= seqnum+1, fsdu!eol:= now + import (aMaxTransmit_ MsduLifetime)
Wait_Cfp_ Sifs
tpdu:= setSeq(tpdu, fsdu!sqf)
Trsp
TxCfAck (,)
Change data to data+cfAck if appropriate.
tpdu:= setFtype (tpdu,
ftype(tpdu) or pack)
TxRequest (tpdu, txrate)
tpdu:= setFtype (tpdu,data_ack)
See 9.6 for rules about selecting transmit data rate.
'txrate:= selected tx data rate'
cTfrg:= inc(cTfrg),
cTmcfrg:= if fsdu!grpa then inc(cTmcfrg) else cTmcfrg fi
-
export (cTfrg, cTmcfrg)
Wait_Cfp_ TxDone
TxConfirm
set(now+ aSifsTime, Trsp)
Wait_CfAck


Process Tx_Coordination_AP
ap_cf_retry_7a(7)
Wait_CfAck
Ack
(endRx,)
Trsp
*
TxC_Cfp
cNack:=
inc(cNack),
export
(cNack)
fsdu!pdus
(fsdu!fCur):=
setRetry
(fsdu!pdus
(fsdu!fCur),1)
tpdu:=
setRetry
(tpdu,1),
fsdu!src:=
fsdu!src+1
fsdu!src =
import(aLong_
RetryLimit))
(true)
(false)
PduConfirm
(fsdu,
retryLimit)
PduConfirm
(fsdu,
partial)
This returns the fsdu
to the queue. At the
next cf-poll, either
the same fsdu or a
different fsdu may
be selected for
transmission.
cFail:=
inc(cFail)
cRtry:=
inc(cRtry)
export(cFail)
export(cRtry)
set(now+
aSifsTime,
Trsp)
TxC_Wait_
Pifs
Trsp
*
TxC_Cfp

IEEE

Annex D

(normative)

ASN.1 encoding of the MAC and PHY MIB

Publisher's note:

It has come to our attention that the definition of the management information base (MIB) in the approved draft standard contains inconsistencies between the definitions in Clause 11, Clause 13, and Annex D. Because the definitions in Annex D are not correct, Annex D is not being published at this time. Purchasers of this standard will be notified when Annex D is published and will receive it at no additional cost.

IEEE

Annex E

(informative)

Bibliography

E.1 General

[B1] ANSI Z136.1-1993, American National Standard for the Safe Use of Lasers.

[B2] IEC 60825-1 (1993), Safety of laser products—Part 1: Equipment classification, requirements and user's guide.

[B3] IEEE Std 802.10-1992, IEEE Standards for Local and Metropolitan Area Networks: Interoperable LAN/MAN Security (SILS) (ANSI).

[B4] Schneier, Bruce, "Applied Cryptography, Protocols, Algorithms and Source Code in C," New York: Wiley: 1994.

E.2 Specification and description language (SDL) documentation

[B5] Belina, Ferenc, Dieter Hogrefe, and Amardeo Sarma, *SDL with Applications from Protocol Specification.* Prentice Hall Europe, Hertfordshire, UK, 1991.

An introductory text on SDL, also useful as a language reference (for SDL-88).

[B6] Ellsberger, Jan, Dieter Hogrefe, and Amardeo Sarma, *SDL, Formal Object-Oriented Language for Communicating Systems*; (Prentice Hall Europe, Hertfordshire, UK, 1997.

A recently published book, which appears to be the most comprehensive single-volume introduction and reference for SDL-92, including its object-oriented extensions.

[B7] Faergemand, Ove and Anders Olsen, *New Features in SDL-92*; SDL Newsletter (ISSN 1023-7151), no. 16 (May, 1993), pp. 10–29. Also available online at http://www.tdr.dk/public/SDL/SDL.html.

This provides a summary of the changes from SDL-88 to SDL-92.

[B8] Olsen, Anders, Ove Faergemand, Birger Moller-Pedersen, Rick Reed, and T. R. W. Smith, *Systems Engineering Using SDL-92.* Elsevier Science B.V., Amsterdam, the Netherlands, 1994.
A detailed guide to using SDL-92, including a thorough explanation of abstract data type mechanism and SDL combined with ASN.1 (Z.105).