ECMA

EUROPEAN COMPUTER MANUFACTURERS ASSOCIATION

STANDARD ECMA-82

LOCAL AREA NETWORKS (CSMA/CD BASEBAND)

LINK LAYER

September 1982

ECMA

EUROPEAN COMPUTER MANUFACTURERS ASSOCIATION

STANDARD ECMA-82

LOCAL AREA NETWORKS (CSMA/CD BASEBAND)

LINK LAYER

September 1982

BRIEF HISTORY

Open Systems Interconnection standards are intended to facilitate homogeneous interconnection of heterogeneous information processing systems. This Standard is within the framework for the coordination of standards for Open Systems Interconnection which is defined by ISO 7498. It is based on the practical experience of ECMA Member Companies world-wide, and the results of their active participation in the current work of ISO, CCITT, IEEE and national standardization bodies in Europe and the USA. It represents a pragmatic and widely based consensus.

This standard is one of a series of standards to be developed and published by ECMA in the field of Local Area Networks. These ECMA standards refer to several LAN techniques:

- Carrier Sense Multiple Access with Collision Detection (CSMA/CD)
- Token Techniques (Bus, Ring).

This Standard ECMA-82 was adopted by the General Assembly of ECMA in June 1982.

TABLE OF CONTENTS

Table of Contents (cont'd)

1. GENERAL

1.1 Scope

For the purpose of compatible interconnection of data processing equipment via a local area network using the carrier sense multiple access with collision detection technique (CSMA/CD), this Standard ECMA-82;

- defines in abstract form the services provided by the Link Layer (see 3.1),
- defines the services assumed to be available at the conceptual interface between the Link Layer and the underlying Physical Layer (see 3.2),
- specifies the functions within the Link Layer (see 4),
- specifies the structure and encoding of the link-protocol-data-unit (see 5),
- specifies Link Layer protocol size and timing parameters (see 6).

A particular emphasis of this Standard is to specify the homogeneous externally visible characteristics needed for interconnection compatibility, while avoiding unnecessary constraints upon and changes to the internal design and implementation of the heterogeneous processing equipment to be interconnected.

The operation of this protocol is of the connectionless-data-transmission type. Extension to include connection-oriented operation is for future study. The specification of hardware and software implementation interfaces is outside the scope of this Standard.

1.2 References

ISO 7498	Data Processing - Open Systems Interconnection - Basic Reference Model.
ECMA-72	Transport Protocol.
ECMA-81	Local Area Networks (CSMA/CD Baseband) - Physical Layer.
ECMA-80	Local Area Networks (CSMA/CD Baseband) - Coaxial Cable System.
ECMA-TR/14	Local Area Networks Layers 1 to 4 Architecture and Protocols.

1.3 Definitions

For the purpose of this Standard the following definitions apply.

1.3.1 Broadcast medium

The class of media in which all stations are capable of receiving a signal transmitted by any other station.

1.3.2 Broadcast transmission

A transmission addressed to all stations on a broadcast medium.

1.3.3 Carrier Sense Multiple Access with Collision Detection (CSMA/CD)

The generic term for a class of link management procedure which allows:

- multiple stations to access a broadcast channel at will (without explicit prior coordination),
- avoids contention via carrier sense and deference, and
- resolves contention via collision detection and retransmission.

1.3.4 Collision

The result of multiple transmission overlapping in the transmission medium, resulting in corrupted data and necessitating retransmission.

1.3.5 Data Terminal Equipment (DTE)

The source and sink for all communication on the network. This includes all equipments attached to the medium, including the means of connection.

1.3.6 Jam

An encoded bit sequence used for collision enforcement.

1.3.7 Link Service Access Point (LSAP)

The point at which Link Level Services are provided by Link Level entity to a Network Level entity.

1.3.8 Link Service Data Unit (LSDU)

A unit of information which is exchanged between Link Service Access Points (LSAP).

1.3.9 Logical Link Control Data Unit (LLCDU)

An element of the Link Level protocol frame. This includes the Destination and Source Link Service Access Points, a protocol control field and the service user data.

1.3.10 Medium Access Data Unit (MADU)

A unit of Link Level data which is encapsulated by the physical transmission frame. This includes the destination and source DTE addresses, the LLCDU octet count, the LLCDU, any padding and a frame check sequence.

1.3.11 Multicast

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

1.3.12 Padding

Additional non-significant data required to prevent short frames.

1.3.13 Preamble

A sequence of encoded bits which the Physical Layer transmits before each frame to allow synchronization of clocks and other Physical circuitry at other DTEs on the network.

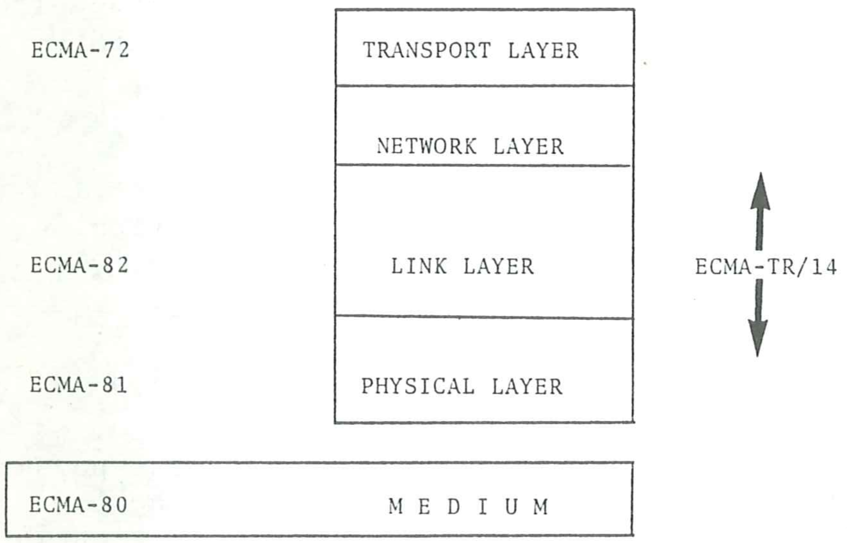
1.4 Conformance

Equipment conforming to this Standard shall implement the provisions specified in sections 4, 5 and 6.

There are no conformance requirements for the layer services defined in section 3.

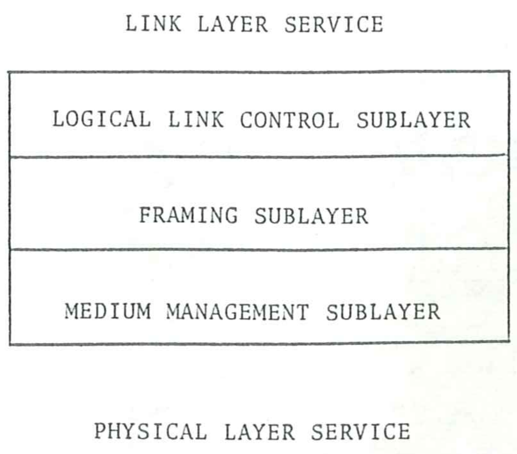
2. GENERAL DESCRIPTION

ECMA-TR/14 discusses the relationship between this Standard and other ECMA Standards for Local Area Network interconnection. This relationship is illustrated diagrammatically below.



The Link Layer provides the means of data transmission via the services provided by the Physical Layer.

The general structure of the Link Layer is illustrated in the following figure.



3. SERVICE DEFINITION

3.1 Link Layer Service

The service provided by the Link Layer to the Network Layer is defined in accordance with the ISO Basic Reference Model for Open Systems Interconnection.

3.1.1 Connectionless-Data-Transmission Service

The connectionless data transmission service provides a means of communication without the establishment of a Link Layer connection.

The data transfer may be point to point or multicast.

The primitives for connectionless-data-transmission are

- L DATA request
- L DATA indication

The L DATA request primitive is passed to the Link Layer to request that an LSDU be sent to one or more remote LSAPs. The L DATA indication primitive is passed from the Link Layer to indicate the arrival of an LSDU.

3.1.1.1 L DATA request

Function:

This primitive is the service request primitive for the connectionless-data-transmission service.

Semantics:

The parameters are as follows:

L DATA request (source address, destination address, LSDU).

The source address and destination address parameters specify the DTEs and LSAPs involved in the transfer. The destination address may specify either an individual address or a multicast address. The LSDU parameter specifies the Link Service Data Unit to be transmitted.

Generated:

This primitive is passed from the Network Layer to the Link Layer to request that a LSDU be sent to one or more remote LSAPs.

Effect:

Receipt of this primitive causes the Link Layer to attempt to send the LSDU using connectionless procedures.

3.1.1.2 L DATA indication

Function:

This primitive is the service indication primitive for the connectionless-data-transmission service.

Semantics:

The parameters are as follows:

L DATA indication (source address, destination address, LSDU).

The source address and destination address parameters specify the DTEs and LSAPs involved in the transfer. The destination address may be the address of a local LSAP, or may be a multicast address specifying multiple LSAPs including a local LSAP. The LSDU parameter specifies the Link Service Data Unit which has been received.

Generated:

This primitive is passed from the Link Layer to the Network Layer to indicate the arrival of an LSDU.

Effect:

The effect of receipt of this primitive by the Network Layer is outside the scope of this Standard.

3.1.2 Connection Oriented Service

Connection oriented services are not defined in this Standard. The relationship between this Standard and ECMA-71 (BA2 8-LAP B) is for future study.

3.2 Physical Layer Service

The services assumed to be available from the Physical Layer at the conceptual interface between the Link Layer and the underlying Physical Layer, are as defined in Standard ECMA-81.

The names of the Physical Layer services referred to in this Standard are:

- Signal Error Indication,
- Input Data Indication,
- Carrier Indication,
- Output Data Request,
- Output Complete Indication,
- Output Data Response,
- Output Request.

4. FUNCTIONS WITHIN THE LAYER

For the purposes of description, the functions within the Link Layer are defined in terms of a functional partitioning into sublayers. There is no requirement for a corresponding internal structure of implementations of the protocol. The requirements are that the Link Layer behaviour externally visible on the network shall be as defined below.

4.1 Logical Link Control Sublayer

The function of the Logical Link Control Sublayer is to request the transmission of a Link Service Data Unit (LSDU). It concatenates the Destination Link Service Access Point (DLSAP), the Source Link Service Acces Point (SLSAP) and the Logical Link Control Data Unit (LLCDU) into a suitable format for the framing sublayer.

4.2 Functions within the Framing Sublayer

4.2.1 Transmission

When the Logical Link Control Sublayer requests the transmission of a frame, the Framing Sublayer constructs the frame from the Logical Link Control Sublayer supplied data. The frame structure and content is defines in section 5.

The CSMA/CD media access mechanism necessitates a minimum frame length. The Framing Sublayer adds padding as specified in section 5.

4.2.2 Reception

The Framing Sublayer checks the frame's destination DTE address field to decide whether the frame should be received by this station. If so, it passes the contents of the frame to the Logical Link Control Sublayer to set up an appropriate status code. The status code is generated by inspecting the Frame Check Sequence to detect any damage to the frame, and by checking for proper octet boundary alignment of the end of the frame.

The procedure for invalid frames is not defined.

4.3 Functions within the Media Management Sublayer

The functions within the Media Management Sublayer are primarily concerned with avoiding, detecting and recovering from contention with other DTEs on the Network. The procedures defined below shall be used.

4.3.1 Normal operation

4.3.1.1 Transmission without contention

When a LLC sublayer of a transmitting DTE requests transmission of a frame, the Framing Sublayer constructs the MADU from the LLC supplied data and appends a Frame Check Sequence to provide error detection. The frame is then handed to the Medium Management Sublayer for transmission. The Medium Management Sublayer initiates transmission by using the Physical Layer primitive: Output Request. The Medium Management Sublayer then send each bit of the MADU by using the Physical Layer primitive: Output Data Response, in response to a Physical Layer primitive: Output Data Request.

When transmission has completed without contention the Medium Management Sublayer so informs the Physical Layer by sending the primitive: Output Complete Indication, and awaits the next request for frame transmission.

4.3.1.2 Reception without contention

At the receiving DTE, the arrival of a frame is first detected by the Physical Layer which responds by synchornizing with the incoming preamble, and by asserting the Carrier Indication. As the encoded bits arrive from the medium, they are decoded and translated back into binary data. The Physical Layer sends an Input Data Indication for each bit to the Medium Management Sublayer, where the leading bits are discarded up to and including the end of the preamble and start of information field delimiter.

The end of the Media Access Data Unit is signalled by removing the Carrier Indication. The Media Access Data Unit is then passed to the framing sublayer.

4.3.1.3 Carrier Deference

Traffic on the physical medium is monitored by means of the Carrier Indication.

When the Carrier Indication is removed, there is a further delay before any transmission is initiated. This is the Inter Frame Gap defined in 4.3.1.4 and section 6. After this, any pending transmission is initiated before resuming monitoring of the Carrier Indication.

4.3.1.3 Inter Frame Gap

As defined in 4.3.1.3, the rules for deferring to passing frames ensure a minimum Inter Frame Gap. This is intended to provide interframe recovery time.

The Inter Frame Gap shall be of at least the duration defined in Section 6. If necessary for implementation reasons, a transmitting DTE may use a larger value with a resulting decrease in its throughput.

4.3.2 Contention Operation

4.3.2.1 Collision Handling

After a DTE has finished deferring and has started transmission, it is possible for collisions to occur until acquisition of the network has been accomplished through the deference of all other DTEs.

The dynamics of collision handling are largely determined by a single parameter called the Slot Time.

This parameter describes three important aspects of collision handling:

- It sets an upper bound on the acquisition time of the network,
- It sets an upper bound on the length of a frame generated by a collision,
- It is scheduling quantum for retransmission. In order to fulfill all three functions, the Slot Time is required to be larger than the sum of the Physical Layer round trip delay and the Media Access Layer maximum Jam Size Time (see section 6). The Slot Time defined in section 6 shall be used.

4.3.2.2 <u>Collision Detection and Enforcement</u>

Collisions are detected by monitoring the Signal Error Indication provided by the Physical Layer. When a collision is detected during a frame transmission, the transmission is not terminated immediately. Instead, the transmission continues until additional bits specified by Jam Size (see section 6) have been transmitted (counting from when the Signal Error Indication was sent). This collision enforcement or Jam guaranteeds that the duration of the collision is sufficient to ensure its detection by all transmitting DTEs on the network. The content of the Jam is unspecified: it may be any fixed or variable pattern convenient to the Data Link implemenation, but should not be the 32-bit FCS value corresponding to the (partial) frame transmitted prior to the Jam.

If while transmitting the preamble, the Signal Error Indication occurs, any remaining preamble bits shall be sent immediately followed by the Jam.

4.3.2.3 <u>Collision Backoff and Retransmission</u>

When a transmission attempt has terminated due to a collision, it is retried by the transmitting DTE until either it is successful, or a maximum number of attempts (Attempt Limit defined in section 6) have been made and all have terminated due to collisions. No subsequent outgoing frame are transmitted until all attempts to transmit a given frame are completed. The scheduling of the retransmission is determined by controlled randomization process called "truncated binary exponential backoff". At the end of enforcing a collision (Jamming, see section 4.3.2.2), the DTE delays before attempting to retransmit the frame. The delay is an integral multiple of Slot Time. The number of Slot Times to delay before the retransmission attempt is chosen as a uniformly distributed random integer r in the range 0 r 2'''k where k = min (n, Backoff Limit, defined in section 6).

The above defines the most aggressive behaviour that a DTE may exhibit in attempting to retransmit after a collision. In the course of implementing the retransmission scheduling procedure, a DTE may introduce extra delays which will degrade its own throughput, but in no case shall a DTE's retransmission scheduling result in a lower average delay between retransmission attempts than the procdure defined above.

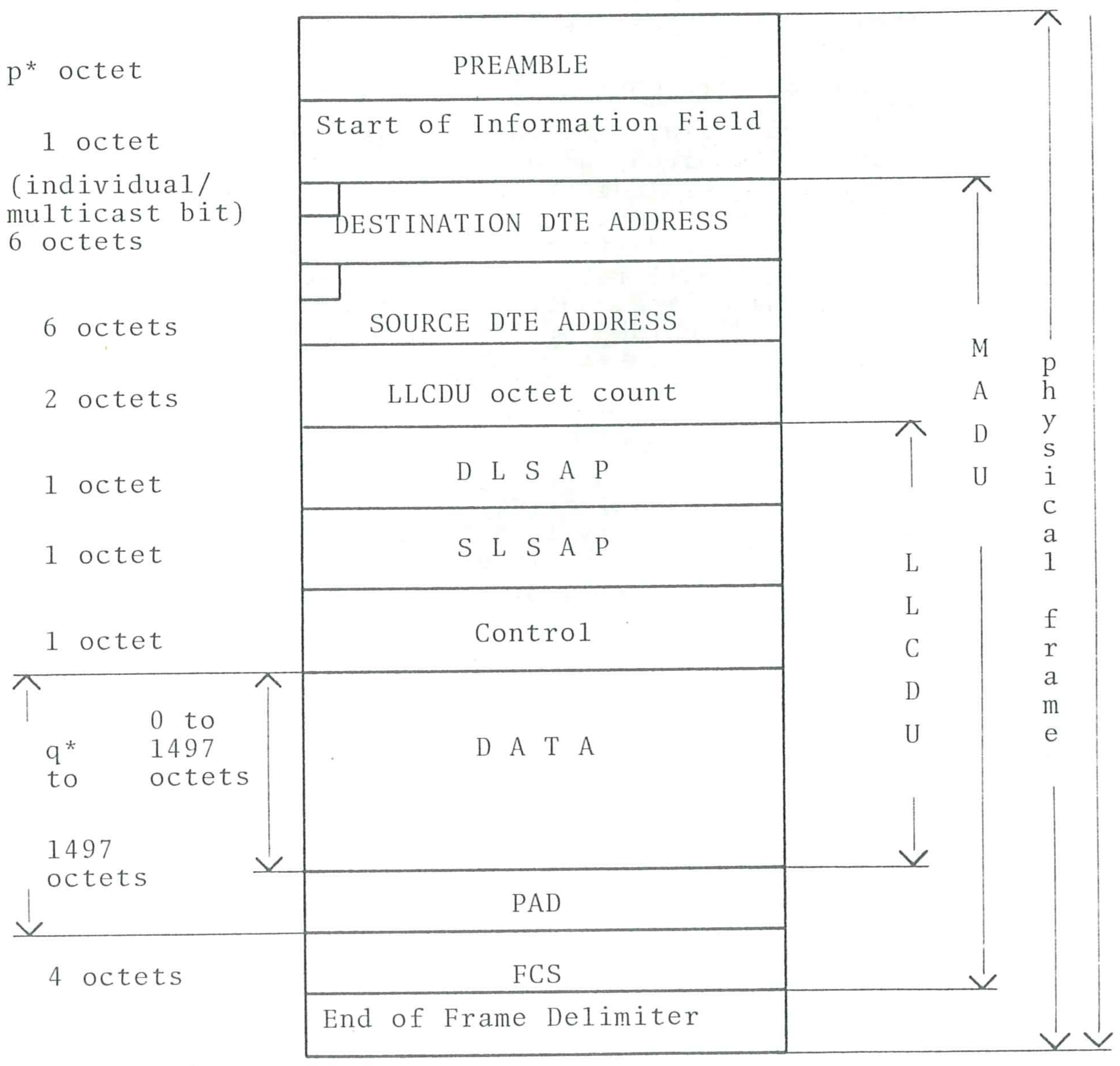
4.3.2.4 Reception with Contention

At the receiving DTE, the bits resulting from a collision are received and decoded by the Physical Layer. A Signal Error Indication at the receiving DTE is ignored. Instead, the fragmentary frames received during collisions are distinguished from valid frames by always being smaller than the shortest valid frame (see Minimum Madu Size, defined in section 6). Such fragments are discarded by receiving Medium Management Sublayer.

5. PROTOCOL FORMATS

5.1 Frame Structure

The frame structure shall be as defined in sections 5.2, 5.3, and 5.4. This structure is illustrated in figure 3. The octets of the frame are transmitted from top to bottom and the bits of each octet are transmitted from left to right.



*p and q are defined in Section 6

FIGURE 3 - FRAME STRUCTURE

5.2 Physical Frame

The Physical Frame comprises a Preamble (see 5.2.1), followed by a Start of Information Field (see 5.2.2); then a Media Access Data Unit (see 5.3). It is terminated by an End of Frame Delimiter (see 5.2.3).

5.2.1 Preamble

The Preamble is sent before the Start of Information Field to allow the DTE circuitry to reach its steady-state, with valid outputs throughout the system.

Upon request by the Link Layer to transmit the first bit of a new MADU, the DTE shall first transmit the Preamble.

The Preamble bit pattern is presented to the Physical Layer in the same manner as Link Layer information. The pattern is an octet of value 10101010 (the left hand ONE is the first transmitted) repeated the number of times defined in section 6.

5.2.2 Start of Information Field

The Start of Information Field (SIF) indicates the start of a MADU and follows the Preamble. It is one octet in length and has the pattern 10101011.

5.2.3 End of frame delimiter

A condition which is considered to exist for more than 3,5 but less than 5 bit times after the last FCS bit.

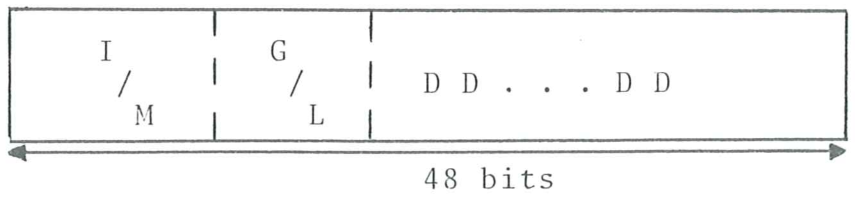
5.3 Media Access Data Unit

The Media Access Data Unit (MADU) is comprised of the following sequence of fields: a Destination DTE Address Field (see 5.3.1), a Source DTE Address Field (see 5.3.2), an LLCDU octet count (see 5.3.3), a LLCDU (see 5.4), a Padding Field if required (see 5.3.4), and a Frame Check Sequence Field (see 5.3.5).

The beginning and end of a MADU is defined by the Physical Frame which encloses it.

5.3.1 Destination DTE Address Field

The Destination DTE Address Field itdentifies the DTE(s) for which the frame is intended. It may be an individual address or a multicast address. It may be a globally or locally administrated address (the I/M bit of this field is the first transmitted.



Individual address: the unique address associated with a particular DTE on the LAN. A DTE's individual address shall be distinct from the individual address of any other DTE on the same LAN.

Multicast address: a multi-destination address, may be associated with one or more DTEs on a given LAN. There are two kinds of multicast address: multicast group address and broadcast address.

- Multicast group address: an address associated by higher level convention with a group of logically related DTEs.
- Braodcast address: a distinguished, predefined multicast address which always denotes the set of all DTEs on a given LAN. This field consists of 48 ONEs.

The first bit of the address distinguishes individual from multicast addresses:

ZERO = individual address (I)

ONE = multicast address (M)

Address Administration: there are two methods of assigning addresses, Global Administration and Local Administration.

- Global Administration: with this method all individual addresses are required to be distinct from the individual address of any other DTE on any LAN. All multicast addresses are required to be different such that no group of logically related DTEs has the same address as any other logically related DTEs. The procedure for administration of these addresses is not defined in this Standard.
- Local Administration: Each individual and multicast address in any given LAN is assigned by the local administration of the given LANs. The procedure for administration of these addresses is not defined in this Standard.

The second bit of the address distinguishes between globally administrated and locally administrated addresses:

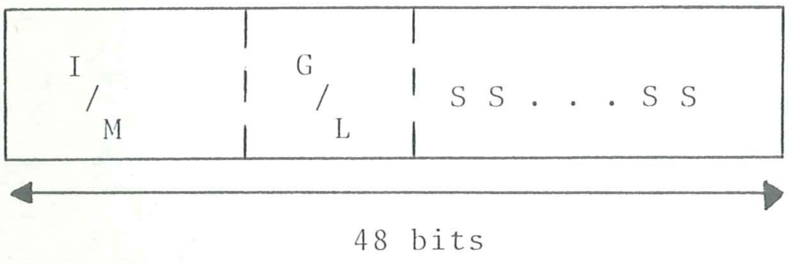
ZERO = global administration

ONE = local administration

5.3.2 Source DTE Address Field

The Source DTE Address Field identifies the DTE sending the frame. The source DTE Address Field is not interpreted at the Link Layer. It is specified at the Link Layer because a uniform convention for the placement of the field is crucial for most higher level protocols.

The source address may be Globally or Locally administrated. The I/M bit shall normally be ZERO; the significance of it being set to ONE is undefined in this Standard and is for future study. The second bit is encoded as defined in 5.3.1 (the I/M bit of this field is the first transmitted).



5.3.3 LLCDU Octet Count

These two octets contain a binary value of the number of octets in the LLCDU. The high order octet is transmitted first. Each octet is transmitted low order bit first.

5.3.4 Padding

Padding is used to fill a frame in order to meet the minimum MADU size requirements (see 6). The padding octet values are not defined in this Standard.

5.3.5 Frame Check Sequence Field

The Frame Check Sequence (FCS) field contains a 4-octet (32-bit) cyclic redundancy check (CRC) value. This value is computed as a function of the contents of all MADU fields except the frame check sequence field itself. The encoding is defined by the generating polynomial:

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

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

- The first 32 bits of the frame are complemented.
- The n bits of the frame are the coefficients of a polynomial M(x) of degree n-1. (The first bit of the destination address field corresponds to the x^{n-1} term and the last bit of the data field corresponds to the x^0 term).
- M(x) is multiplied by x^{32} and divided by G(x), producing a remainder R(x) of degree smaller than 31.

- The coefficients of R(x) are a 32-bit sequence.
- The bit sequence of complemented and the result is the CRC.
- The 32 bits of the CRC are transmitted in the order $x^{31}, x^{30} \ldots x^{1}, x^{0}$.

5.4 Logical Link Control Data Unit

The Logical Link Control Data Unit (LLCDU) relates to the Logical Control Sublayer defined in section 4.1

All LLCDUs shall have the following format:

DLSAP (8 bits)	SLSAP (8 bits)	CONTROL (8 bits)	DATA (variable)

where

DLSAP	= Destination Link Service Access Point address field (see 5.4.1).
SLSAP	= Source Link Service Access Point address field (see 5.4.1).
CONTROL	= Control field (see 5.4.2).
DATA	= Data field (see 5.4.3).

5.4.1 Address Fields

Each LLCDU shall contain two logical address fields: the Destination Kink Service Access Point address and the Source Link Service Access Point address in that order. The DLSAP address field shall specify the LSAP for which the LLCDU is intended. The SLSAP address field shall consist of an individual LSAP address and it identifies the LSAP from which the transmission of the LLCDU was initiated.

DLSAP may be an individual address or a group address.

The format is

I/G	D	D	D	D	D	D	D

I/G value ZERO = INDIVIDUAL DLSAP (or SLSAP)

I/G value ONE = GROUP ADDRESS DLSAP

The pattern 11111111 is for braodcast DLSAP address. (the I/G bit of this field shall be the first transmitted).

5.4.2 Control Field

This Control field is of one octet. For Connectionless-Data-Transmission the value is 11000011.
The coding for other LLCDU types is for further studies.

5.4.3 Data Field

The Data field shall consist of any integral number (including zero) of octets.

Because the format of a valid frame specifies an integral number of octets, only a collision or an error can produce a frame with a length that is not an integral multiple of 8 bits.

6. SPECIFIC PARAMETERS

The following specific parameter values referred to in Section 4 and 5 shall be used:

Slot Time 512 bit times
Inter Frame Gap 9,6 us
Attempt Limit 16
Backoff Limit 10
Jam size 32 to 48 bits
Preamble length (p) 8 octets
Max MADU size 1518 octets
Min MADU size 64 octets (hence q = 43 octets).

These values are based on a 10 Mbit/s data rate, using the physical medium defined in Standard ECMA-ZZ with a maximum of length of 2,5 km.

APPENDIX A

FORMAL SPECIFICATION

A.1 Introduction

An algorithmic definition is given in this section, providing a procedural model for the ECMA Media Access Method in the form of a program in the language Pascal. Note that whenever there is any apparent ambiguity concerning the definition of some aspect of the EMCA Media Access Method, it is the Chapters 4, 5, and 6 which should be consulted for the definitive statement.

A.2 The ECMA CSMA/CD Media Access Procedural Model

A.2.1 Overview of the Procedural Model

The functions of the ECMA CSMA/CD Medium Management and Framing Sublayers are presented below, modeled as a program written in Pascal. This procedural model indicates the functions to be provided in any ECMA CSMA/CD implementation. It is important to distinguish, however, between the model and a real implementation. The model is optimized for simplicity and clarity of presentation, while any realistic implementation must place heavier emphasis on such constraints as efficiency and suitability to a particular implementation technology or computer architecture. In this context, several important properties of the procedural model must be considered.

A.2.2 Ground Rules for the Procedural Model

a) First, it must be emphasized that the description of the Medium Management and Framing Sublayers in a programming language is in no way intended to imply that procedures must be implemented as a program executed by a computer. The implementation may consist of any appropriate technology including hardware, firmware, software, or any combination.

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

c) The handling of incoming and outgoing frames is rather stylized in the procedural model, in the sense that frames are handled as single entities by most of the Medium Management Sublayer and are only serialized for presentation to the Physical Layer. In reality, many implementations will instead handle frames serially on a bit, octet or word basis. This approach has not been reflected in the procedural model, since this would only complicate the description of the functions without changing them in any way.

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

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

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

It is intended that the concurrency in the model reflect the parallelism intrinsic to the task of implementing the ECMA Link Layer procedures, although the actual parallel structure of the implementations is likely to vary.

A.2.3 Use of Pascal in the Procedural Model

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

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

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

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

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

4) The use of a variant record to represent a frame (both as fields and as bits) follows the letter but not the spirit of the Pascal Report, since it allows the underlying representation to be viewed as two different data types.

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

A.2.4 Organization of the Procedural Model

The procedural model used here is based on five cooperating concurrent processes. Three are actually defined in the Medium Management and Framing Sublayers. The remaining two processes are provided by the LLC Sublayer and utilize the Framing Sublayer. The five processes are thus:

LLC Sublayer :

Frame Transmitter Process	**Frame Receiver Process**

Medium Management Sublayer:

Bit Transmitter Process	**Bit Receiver Process**
Deference Process	

This organization of the model is illustrated in Figure A-1 and reflects the fact that the communication of entire frames is initiated by the LLC Sublayer, while the timing of collision backoff and of individual bit transfers is based on interactions between the Medium Management Sublayer and the Physical-Layer-dependent bit-time.

Figure A-1 depicts the static structure of the procedural model, showing how the various processes and procedures interact by invoking each other. Figures A-2 and A-3 summarize the dynamic behavior of the model during transmission and reception, focusing on the steps that must be performed, rather than the procedural structure which performs them. The usage of the shared state variables is not depicted in the figures, but is described in the comments and prose in the following sections.

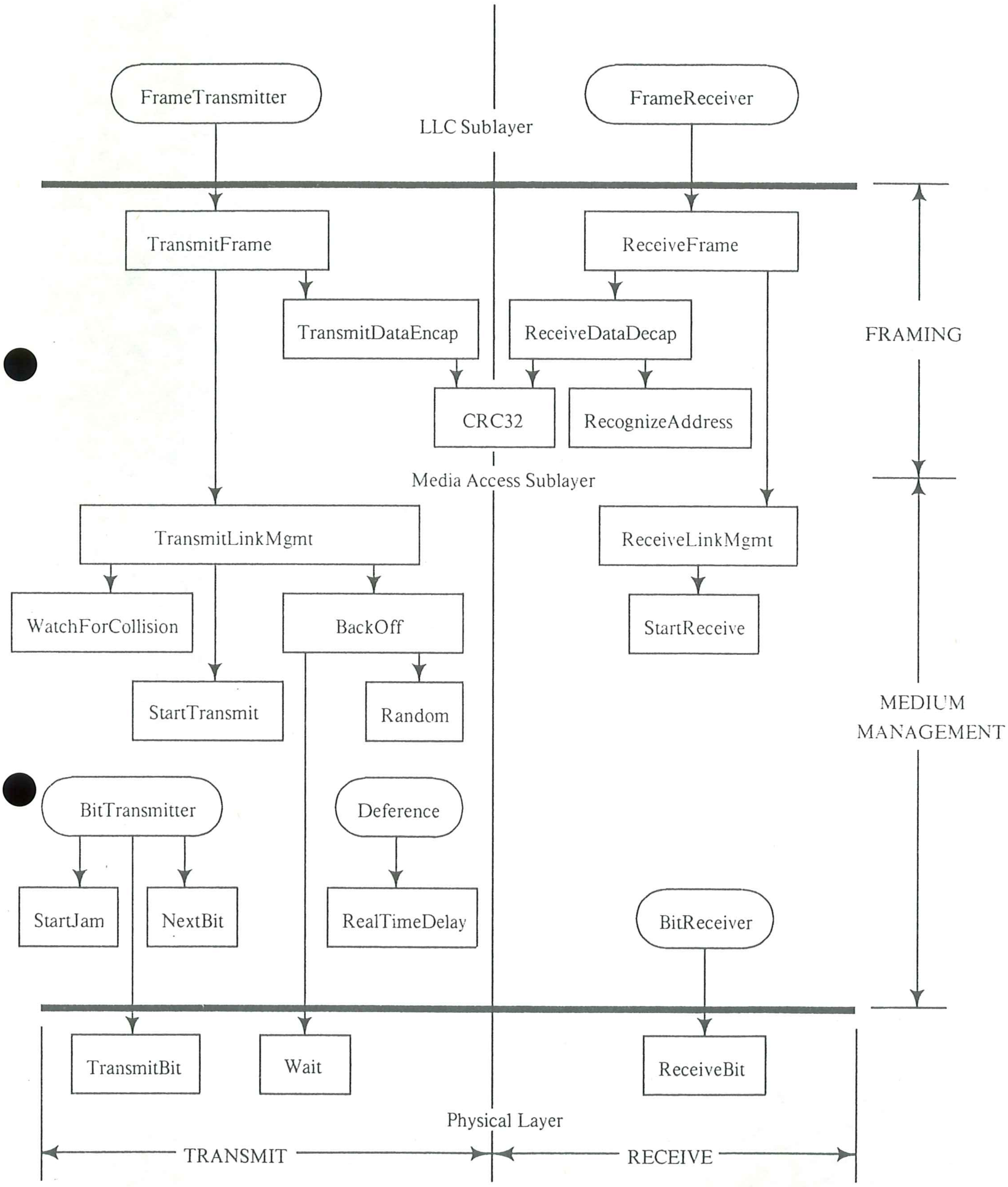


Figure A-1 Relationship among CSMA/CD procedures

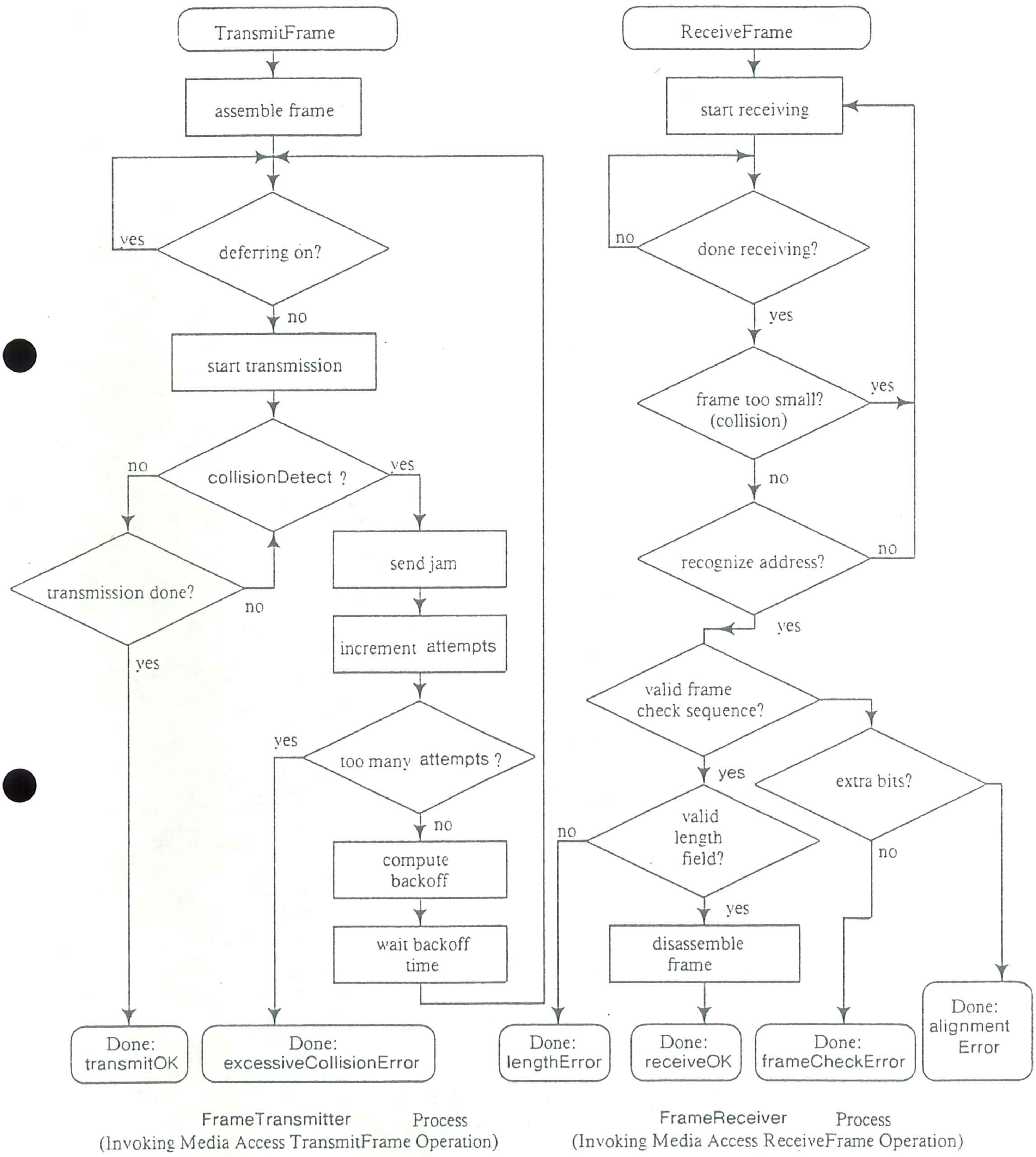
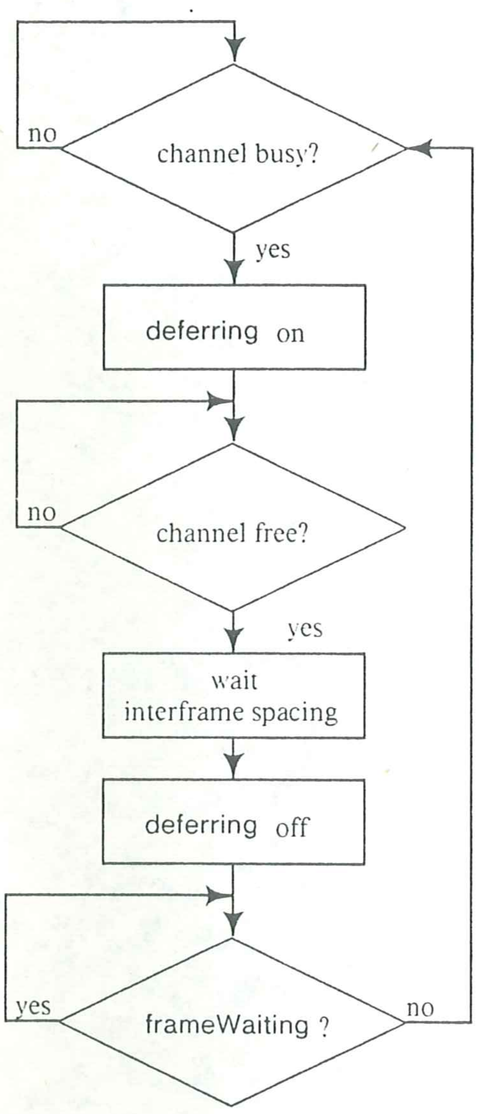
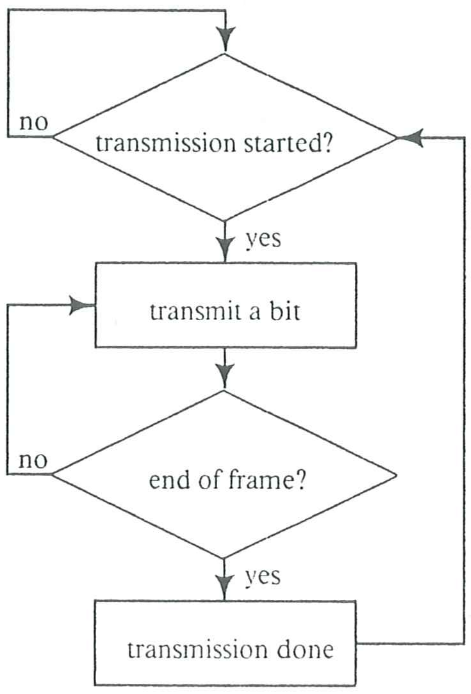


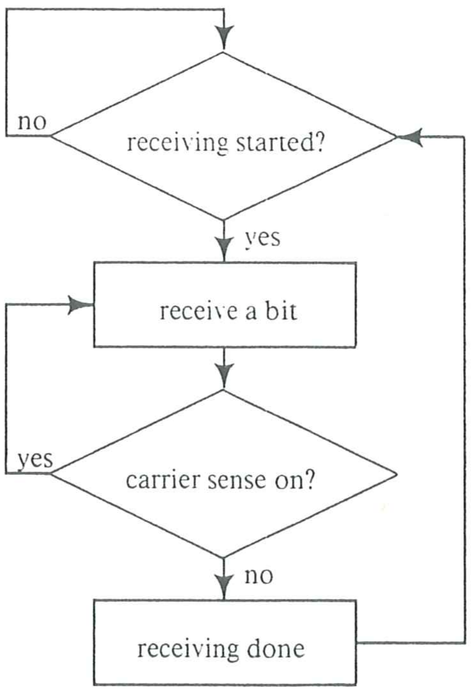
Figure A-2 Control Flow Summary



Deference process



Bit Transmitter process



Bit Receiver process

Figure A-3 Control Flow -- Media Access Sublayer

A.3 Frame Transmission Model

Frame transmission includes Framing and Medium Management aspects:

Transmit Framing includes the assembly of the outgoing frame (from the values provided by the LLC Sublayer), minimum frame size enforcement, and frame check sequence generation.

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

The performance of these functions by a transmitting Link Layer interacts with corresponding actions by other Link Layers to jointly implement the ECMA CSMA/CD protocol.

A.3.1 Transmit Framing

A.3.1.1 Frame Assembly

The fields of the CSMA/CD frame are set to the values provided by the LLC Sublayer as arguments to the *TransmitFrame* operation, with the exception of the padding necessary to enforce the minimum frame size, and the frame check sequence, which is set to the CRC value generated by the Framing Sublayer .

A.3.1.2 Frame Check Sequence Generation

The CRC value is generated and inserted in the frame check sequence field.

A.3.2 Transmit Medium Management

A.3.2.1 Carrier Deference

Even when it has nothing to transmit, the CSMA/CD Medium Management Sublayer monitors the physical medium for traffic by watching the *carrierSense* signal provided by the Physical Layer. (Note: Assertion of Carrier Indication by the Physical Layer causes the *carrierSense* signal to become true and removing Carrier Indication causes it to become false). Whenever the medium is busy, the CSMA/CD Medium Management Sublayer *defers* to the passing frame by delaying any pending transmission of its own. After the last bit of the passing frame (i.e., when *carrierSense* changes from true to false), the CSMA/CD Medium Management Sublayer continues to defer for a proper interframe spacing, *interFrameGap* (see Section A.3.2.2). At the end of that time, if it has a frame waiting to be transmitted, transmission is initiated independent of the value of *carrierSense*. When transmission has completed (or immediately, if there was nothing to transmit) the CSMA/CD Medium Management Sublayer resumes its original monitoring of *carrierSense*.

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

A.3.2.2 Interframe Spacing

As defined in Section A.3.2.1, the rules for deferring to passing frames insure a minimum interframe spacing of *interFrameGap* sec. This is intended to provide interframe recovery time for other CSMA/CD Medium Management Sublayer and for the physical medium.

Note that *interFrameGap* is the minimum value of the interframe spacing. If necessary for implementation reasons, a transmitting sublayer may use a larger value with a resulting decrease in its throughput.

A.3.2.3 Collision Handling

Once a Medium Management Sublayer has finished deferring and has started transmission, it is still possible for it to experience contention for the medium. Collisions can occur until acquisition of the network has been accomplished through the deference of all other stations' Medium Management sublayers.

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

- It is an upper bound on the aquisition time of the medium.
- It is an upper bound on the length of a frame generated by a collision.
- It is the scheduling quantum for retransmission.

In order to fulfill all three functions, the slot time must be larger than the sum of the Physical Layer round-trip propagation time and the Medium Management Sublayer maximum jam time.

A.3.2.3.1 Collision Detection and Enforcement

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

A.3.2.3.2 Collision Backoff and Retransmission

When a transmission attempt has terminated due to a collision, it is retried by the transmitting Medium Management Sublayer until either it is successful, or a maximum number of attempts, *attemptLimit*, have been made and all have terminated due to collisions. Note that all attempts to transmit a given frame are completed before any subsequent outgoing frames are transmitted.

The scheduling of the retransmissions is determined by a controlled randomization process called "truncated binary exponential backoff". At the end of enforcing a collision (jamming), the Medium Management Sublayer delays before attempting to retransmit the frame. The delay is an integral multiple of *slotTime*. The number of slot times to delay before the nth retransmission attempt is chosen as a uniformly distributed random integer r in the range $0 \leq r < 2^k$ where $k = \min(n, 10)$. If all *attemptLimit* attempts fail, this event is reported as an error.

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

A.3.3 Minimum Frame Size

The CSMA/CD Media Access mechanism requires that a minimum frame length of *slotTime* bits be transmitted. If *frameSize* is less than *slotTime*, then the CSMA/CD Framing Sublayer must append extraoctets, or a *pad*, after the end of the LLC data field but prior to calculating, and appending, the FCS. The number of *pad* units must be sufficient to ensure that the frame is at least *slotTime* bits. The pad is determined using the value in the length field passed by the LLC Sublayer.

A.4 Frame Reception Model

Frame reception includes both data decapsulation and Medium Management aspects:

Receive Frame Handling comprises address recognition, frame check sequence validation, and frame disassembly to pass the fields of the received frame to the LLC Sublayer.

Receive Medium Management' recognizes collision fragments from incoming frames .

The performance of these functions by a receivingLink Layer interacts with corresponding actions by other Link Layers to jointly implement the ECMA CSMA/CD protocol.

A.4.1 Receive Frame Handling

A.4.1.1 Address Recognition

The Framing Sublayer is capable of recognizing individual and group addresses.

A.4.1.1.1 Individual Addresses

The Framing Sublayer recognizes and accepts any frame whose destination field contains the individual address of the station.

A.4.1.1.2 Multicast Addresses

The Framing Sublayer recognizes and accepts any frame whose destination field contains the broadcast address, or selected multicast group addresses as specified by higher layers..

A.4.1.2 Frame Check Sequence Validation

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

A.4.1.3 Frame Disassembly

The frame is disassembled and the fields are passed to the LLC Sublayer via the output parameters of the *ReceiveFrame* operation.

A.4.1.4 Physical Frame Management

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

A.4.1.4.1 Maximum Frame Size

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

A.4.1.4.2 Minimum Frame Handling

The receiving Framing Sublayer will remove any padding added by the transmitting DTE.

A.4.1.4.3 Integral Number of Octets in Frame

Since the format of a valid frame specifies an integral number of octets, only a collision or an error can produce a frame with a length that is not an integral multiple of 8. Complete frames (i.e., not rejected as collision fragments; see Section A.4.2.1) that do not contain an integral number of octets are truncated to the nearest octet boundary. If frame check sequence validation detects an error in such a frame, the status code *alignmentError* is reported.

A.4.2 Receive Media Access Management

A.4.2.1 Collision Filtering

The smallest valid frame must be a least one *slotTime* in length. Any frame containing less than *slotTime* bits is presumed to be a fragment resulting from a collision and is discarded by the receiving Medium Management Sublayer. Since occasional collisions are a normal part of the Medium Management procedure, the discarding of such a fragment is not reported as an error to the LLC Sublayer.

A.5 Global Declarations

This section provides detailed formal specifications for the CSMA/CD Media Access Mechanism. It is a specification of generic features and parameters to be used in systems implementing this ECMA media access method.

A.5.1 Common Constants and Types

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

```
const
    addressSize = 48 ; {in bits}
    lengthSize = 16; {in bits}
    dataSize = ... ; {LLCData, see A.2.3, note 3}
    padSize = ...; {in bits, = max(0,slotTime-dataSize), see A.2.3, note 3}
    crcSize = 32; {32 bit CRC = 4 octets}
    frameSize = ... ; { = 2*addressSize + lengthSize + dataSize +padSize + crcSize,
    see A.2.3, note 3}

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


type
    Bit = 0..1;
    AddressValue = array [1..addressSize] of Bit;
    LengthValue= array [1..lengthSize] of Bit
    DataValue = array [1..dataSize] of Bit;
    PadValue = array [0..padSize] of Bit;
    CRCValue = array [1..crcSize] of Bit;
    ViewPoint = (fields, bits); {Two ways to view the contents of a frame}
    Frame = record {Format of Media Access frame}
       case view: ViewPoint of
          fields: (
              destinationField: AddressValue;
              sourceField: AddressValue;
              lengthField: LengthValue;
              dataField: DataValue;
              padField: PadValue;
              fcsField: CRCValue);
          bits: (
              contents: array [1..frameSize] of Bit)
       end; {Frame}
```

A.5.2 Transmit State Variables

The following items are specific to frame transmission.

```
const
```

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

A.5.3 Receive State Variables

The following items are specific to frame reception.

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

A.5.4 Summary of Interlayer Interfaces

The interface to the LLC Sublayer is summarized below:

```
type
    TransmitStatus = (transmitOK, excessiveCollisionError); {Result of
    TransmitFrame operation}
    ReceiveStatus = (
        receiveOK, lengthError, frameCheckError, alignmentError); {Result of
        ReceiveFrame operation}
function TransmitFrame (
        destinationParam: AddressValue;
        sourceParam: AddressValue;
        lengthParam: LengthValue;
        dataParam: DataValue): TransmitStatus; {Transmits one frame}
function ReceiveFrame (
        var destinationParam: AddressValue;
        var sourceParam: AddressValue;
        var lengthParam: LengthValue;
        var dataParam: DataValue): ReceiveStatus; {Receives one frame}
```

The interface to the Physical Layer is summarized below:

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

A.5.5 State Variable Initialization

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

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

A.6 Frame Transmission

The algorithms in this section define frame transmission.

The function *TransmitFrame* implements the frame transmission operation provided to the LLC Sublayer:

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

First, *TransmitFrame* calls the internal procedure *ComputePadParam* to generate an arbitrary array of type *PadValue*. *TransmitDataEncap* then is called to construct the frame. Next, *TransmitLinkMgmt* is called to perform the actual transmission. The *TransmitStatus* returned indicates the success or failure of the transmission attempt.

ComputePadParam builds the array *padParam* used in *TransmitDataEncap* to pad the frame to the minimum frame size.

```
function ComputePadParam: PadValue;
begin
    ComputePadParam := {Build an array of size PadSize of arbitrary bits}
end; {ComputePadParam}
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

BitTransmitter, upon detecting a new collision, immediately enforces it by calling *StartJam* to initiate the transmission of the jam. The jam should contain a sufficient number of bits of arbitrary data so that it is assured that both comunicating stations detect the collision. (*StartJam* uses the first set of bits of the frame up to *JamSize*, merely to simplify this program).

A.7 Frame Reception

The algorithms in this section defines ECMA CSMA/CD frame reception:

The procedure *ReceiveFrame* implements the frame reception operation provided to the LLC Sublayer:

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

ReceiveFrame calls *ReceiveLinkMgmt* to receive the next valid frame, and then calls the internal procedure *ReceiveDataDecap* to return the frame's fields to the LLC Sublayer if the frame's address indicates that it should do so. It also removes any padding added by the transmitting DTE. The returned *ReceiveStatus* indicates the presence or absence of detected transmission errors in the frame.

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

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


function RemovePad(
    var lengthParam: LengthValue
    var dataParam: DataValue): DataValue;
begin
    RemovePad := {Strips lengthParam bits from the data field and returns the
    LLCDataField};
    validLength:= {Check to determine if value represented by lengthParam matches
    received dataSize}
end; {RemovePad}
```

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

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

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

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

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

3.12 Common procedures

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

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

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

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

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