Annex 4A

(normative)

Simplified full duplex media access control

Editors' Notes: To be removed prior to final publication.

References:

None.

Definitions:

None.

Abbreviations:

None.

Issues:

None.

Revision History:

Draft 3.1	January 2003	Revised draft for IEEE 802.3 Working Group Ballot Recirculation, incorporating comments resolved at the September, 2003 Interim meeting in Portonovo, Italy.

This annex is based on the Clause 4 MAC, with simplifications for use in networks that don't require the half-duplex operational mode. This annex stands alone and does not rely on information within Clause 4 to be implemented.

4A.1 Functional model of the MAC method

4A.1.1 Overview

The architectural model described in Clause 1 is used in this clause to provide a functional description of the LAN full duplex MAC sublayer.

The MAC sublayer defines a medium-independent facility, built on the medium-dependent physical facility provided by the Physical Layer, and under the access-layer-independent LAN LLC sublayer (or other MAC client). It is applicable to a general class of media suitable for use with the full duplex media access discipline.

The LLC sublayer and the MAC sublayer together are intended to have the same function as that described in the OSI model for the Data Link Layer alone. The partitioning of functions presented in this standard requires two main functions generally associated with a data link control procedure to be performed in the MAC sublayer. They are as follows:

a) Data encapsulation (transmit and receive)
    1) Framing (frame boundary delimitation, frame synchronization)
    2) Addressing (handling of source and destination addresses)
    3) Error detection (detection of physical medium transmission errors)
b) Media access management (physical layer contention)

This MAC does not support the *half duplex* mode of operation so there is no need for collision avoidance or handling. However, this MAC does have the ability to avoid contention within the physical layer. Therefore, Media Access Management comprises the transmission of bits to the physical layer and delaying any transmission for an interframe gap or for a longer period of time based on contention within the physical layer.

An optional MAC control sublayer, architecturally positioned between LLC (or other MAC client) and the MAC, is specified in Clause 31 and Clause 64. This MAC Control sublayer is transparent to both the underlying MAC and its client (typically LLC). The MAC sublayer operates independently of its client; i.e., it is unaware whether the client is LLC or the MAC Control sublayer. This allows the MAC to be specified and implemented in one manner, whether or not the MAC Control sublayer is implemented. References to LLC as the MAC client in text and figures apply equally to the MAC Control sublayer, if implemented.

The remainder of this clause provides a functional model of this MAC method.

4A.1.2 Full duplex operation

This subclause provides an overview of frame transmission and reception in terms of the functional model of the architecture. This overview is descriptive, rather than definitional; the formal specifications of the operations described here are given in 4A.2 and 4A.3. Specific implementations for full duplex mechanisms that meet this standard are given in 4A.4. Figure 4A–1 provides the architectural model described functionally in the subclauses that follow.

The Physical Layer Signaling (PLS) component of the Physical Layer provides an interface to the MAC sublayer for the serial transmission of bits onto the physical media. For completeness, in the operational

description that follows some of these functions are included as descriptive material. The concise specification of these functions is given in 4A.2 for the MAC functions and in Clause 7 for PLS.

Transmit frame operations are independent from receive frame operations.

4A.1.2.1 Transmission

When a MAC client requests the transmission of a frame, the Transmit Data Encapsulation component of the full duplex MAC sublayer constructs the frame from the client-supplied data. It prepends a preamble and a Start Frame Delimiter to the beginning of the frame. Using information provided by the client, the MAC sublayer also appends a PAD at the end of the MAC information field of sufficient length to ensure that the transmitted frame length satisfies a minimum frame-size requirement (see 4A.2.3.2.4). It also prepends destination and source addresses, the length/type field, and appends a frame check sequence to provide for error detection. If the MAC supports the use of client-supplied frame check sequence values, then it shall use the client-supplied value, when present. If the use of client-supplied frame check sequence values is not supported, or if the client-supplied frame check sequence value is not present, then the MAC shall compute this value. Frame transmission may be initiated once there is no contention at the physical layer and after the interframe delay, regardless of the presence of receive activity.

The Physical Layer performs the task of generating the signals on the medium that represent the bits of the frame. A functional description of the Physical Layer is given in Clause 7 and beyond.

When transmission has completed, the MAC sublayer so informs the MAC client and awaits the next request for frame transmission.

4A.1.2.2 Reception

At each receiving station, the arrival of a frame is first detected by the Physical Layer, which responds by synchronizing with the incoming preamble, and by turning on the receiveDataValid signal. As the encoded bits arrive from the medium, they are decoded and translated back into binary data. The Physical Layer passes subsequent bits up to the MAC sublayer, where the leading bits are discarded, up to and including the end of the preamble and Start Frame Delimiter.

Meanwhile, the Receive Media Access Management component of the MAC sublayer, having observed receiveDataValid, has been waiting for the incoming bits to be delivered. Receive Media Access Management collects bits from the Physical Layer entity as long as the receiveDataValid signal remains on. When the receiveDataValid signal is removed, the frame is truncated to an octet boundary, if necessary, and passed to Receive Data Decapsulation for processing.

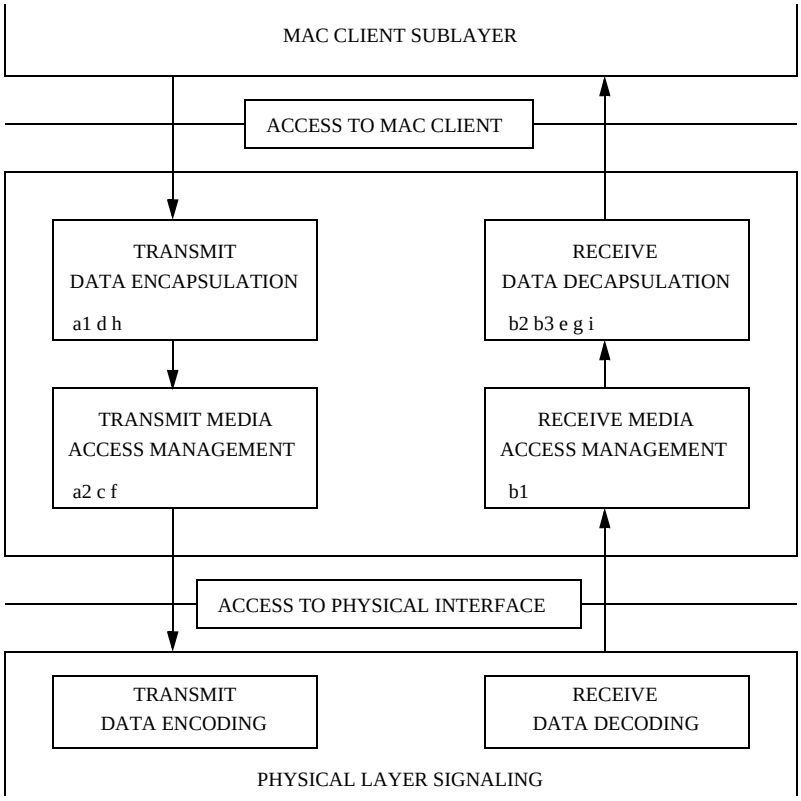
Receive Data Decapsulation checks the frame's Destination Address field to decide whether the frame should be received by this station. If so, it passes the Destination Address (DA), the Source Address (SA), the Length/Type, the Data, and (optionally) the Frame Check Sequence (FCS) fields to the MAC client, along with an appropriate status code, as defined in 4A.3.2. It also checks for invalid MAC frames by inspecting the frame check sequence to detect any damage to the frame enroute, and by checking for proper octet-boundary alignment of the end of the frame. Frames with a valid FCS may also be checked for proper octet-boundary alignment.

4A.1.3 Relationships to the MAC client and physical layers

The MAC sublayer provides services to the MAC client required for the transmission and reception of frames. Access to these services is specified in 4A.3. The MAC sublayer makes a best effort to transfer a serial stream of bits to the Physical Layer. Although certain errors are reported to the client, error recovery is not provided by MAC. Error recovery may be provided by the MAC client or higher (sub)layers.

4A.1.4 Access method functional capabilities

The following summary of the functional capabilities of the MAC sublayer is intended as a quick reference guide to the capabilities of the standard, as shown in Figure 4A–1:



NOTE—a1, b2, etc., refer to functions listed in 4A.1.4.

Figure 4A–1—Media access control functions

a) For Frame Transmission
   1) Accepts data from the MAC client and constructs a frame.
   2) Presents a bit-serial data stream to the Physical Layer for transmission on the medium.

   NOTE—Assumes data passed from the client sublayer are octet multiples.

b) For Frame Reception
   1) Receives a bit-serial data stream from the Physical Layer.
   2) Presents to the MAC client sublayer frames that are either broadcast frames or directly addressed to the local station.
   3) Discards or passes to Network Management all frames not addressed to the receiving station.

c) Defers transmission of a bit-serial stream whenever the physical layer is busy.
d) Appends proper FCS value to outgoing frames and verifies full octet boundary alignment.
e) Checks incoming frames for transmission errors by way of FCS and verifies octet boundary alignment
f) Delays transmission of frame bit stream for specified interframe gap period.
g) Discards received transmissions that are less than a minimum length.
h) Appends preamble, Start Frame Delimiter, DA, SA, Length/Type field, and FCS to all frames, and inserts PAD field for frames whose data length is less than a minimum value.
i) Removes preamble, Start Frame Delimiter, DA, SA, Length/Type field, FCS, and PAD field (if necessary) from received frames.

4A.2 Media access control (MAC) method: precise specification

4A.2.1 Introduction

A precise algorithmic definition is given in this subclause, providing a procedural model for the MAC process with a program in the computer language Pascal. See references [B11] and [B34] for resource material. Note whenever there is any apparent ambiguity concerning the definition of some aspect of the MAC method, it is the Pascal procedural specification in 4A.2.7 through 4A.2.10 that should be consulted for the definitive statement. Subclauses 4A.2.2 through 4A.2.6 provide, in prose, a description of the access mechanism with the formal terminology to be used in the remaining subclauses.

4A.2.2 Overview of the procedural model

The functions of the MAC method are presented below, modeled as a program written in the computer language Pascal. This procedural model is intended as the primary specification of the functions to be provided in any MAC sublayer 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 shall 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 shall be considered.

4A.2.2.1 Ground rules for the procedural model

a) First, it shall be emphasized that *the description of the MAC sublayer in a computer language is in no way intended to imply that procedures shall 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 shall be emphasized that it is the behavior of any MAC sublayer implementations that shall 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 MAC 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 only complicates 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 MAC procedure. The timing dependencies introduced by the need for concurrent activity are resolved in two ways:

   1) *Processes Versus 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 Media Access procedure ReceiveFrame is always called well before the frame in question has started to arrive.

   2) *Processes Versus Processes.* Among processes, no assumptions are made about relative speeds of execution. This means that each interaction between two processes shall 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 MAC client and MAC procedures, although the actual parallel structure of the implementations is likely to vary.

4A.2.2.2 Use of Pascal in the procedural model

Several observations need to be made regarding the method with which Pascal is used for the model. Some of these observations are as follows:

a) The following limitations of the language have been circumvented to simplify the specification:
   1) The elements of the program (variables and procedures, for example) 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 Concurrent Pascal, a Pascal derivative, 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). 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 (as fields and as bits) follows the spirit but not the letter 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 by way of 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 (microcode, hardware state machines, etc.)

4A.2.2.3 Organization of the procedural model

The procedural model used here is based on five cooperating concurrent processes. The Frame Transmitter process and the Frame Receiver process are provided by the clients of the MAC sublayer (which may include the LLC sublayer) and make use of the interface operations provided by the MAC sublayer. The other three processes are defined to reside in the MAC sublayer. The five processes are as follows:

a) Frame Transmitter process
b) Frame Receiver process
c) Bit Transmitter process
d) Bit Receiver process
e) Deference process

This organization of the model is illustrated in Figure 4A–2 and reflects the fact that the communication of entire frames is initiated by the client of the MAC sublayer, while the timing of individual bit transfers is based on interactions between the MAC sublayer and the Physical-Layer-dependent bit time.

Figure 99–2 depicts the static structure of the procedural model, showing how the various processes and procedures interact by invoking each other. Figure 4A–3a, Figure 4A–3b, and Figure 4A–3c summarize the dynamic behavior of the model during transmission and reception, focusing on the steps that shall be performed, rather than the procedural structure that 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 subclauses.

4A.2.2.4 Layer management extensions to procedural model

In order to incorporate network management functions, this Procedural Model has been expanded beyond that provided in ISO/IEC 8802-3: 1990. Network management functions have been incorporated in two ways. First, 4A.2.7–4A.2.10, 4A.3.2, Figure 4A–3a, and Figure 4A–3b have been modified and expanded to

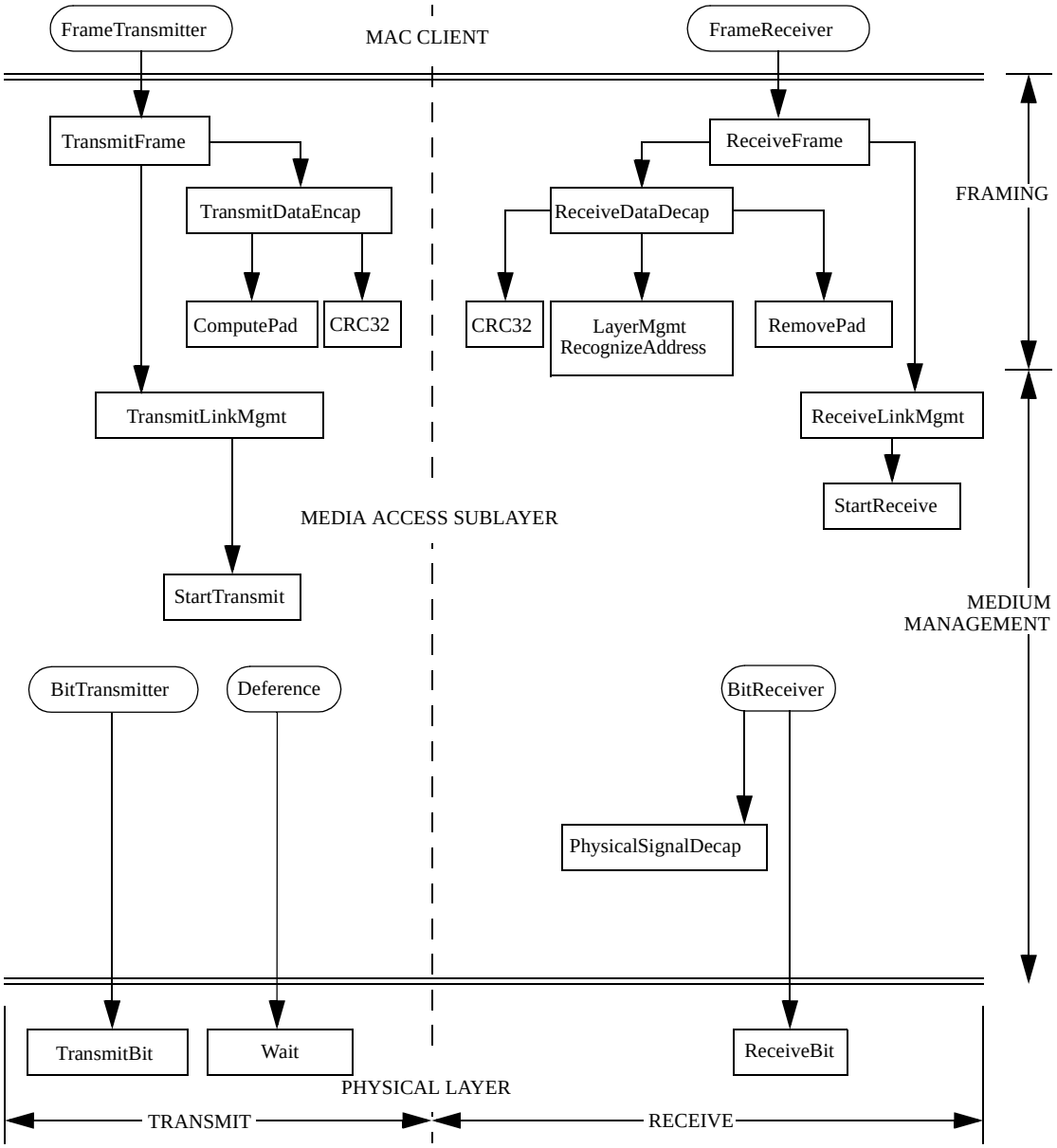


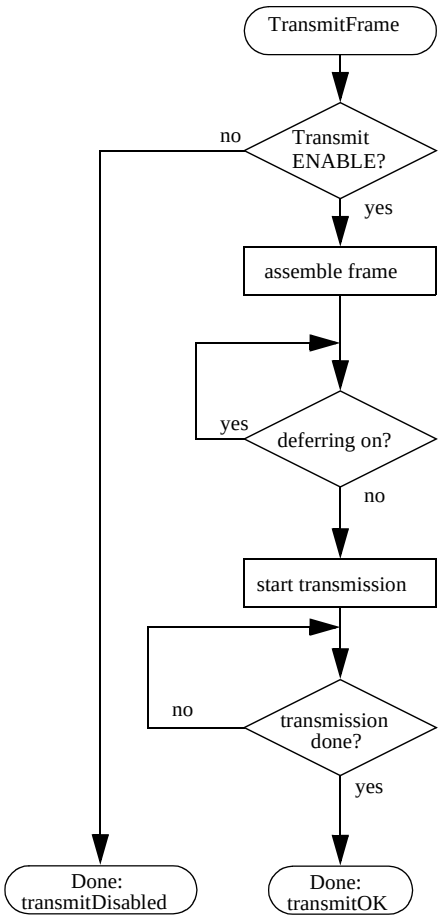
Figure 4A–2—Relationship among MAC procedures

provide management services. Second, Layer Management procedures have been added as 5.2.4. Note that Pascal variables are shared between Annex 4A and Clause 5.

The Pascal procedural specification shall be consulted for the definitive statement when there is any apparent ambiguity concerning the definition of some aspect of the MAC access method.

The Layer Management facilities provided by the MAC and Physical Layer management definitions provide the ability to manipulate management counters and initiate actions within the layers. The managed objects within this standard are defined as sets of attributes, actions, notifications, and behaviors in accordance with IEEE Std 802.1F-1993, and ISO/IEC International Standards for network management.

4A.2.3 Frame transmission model



a) TransmitFrame

Figure 4A–3a—Control flow summary

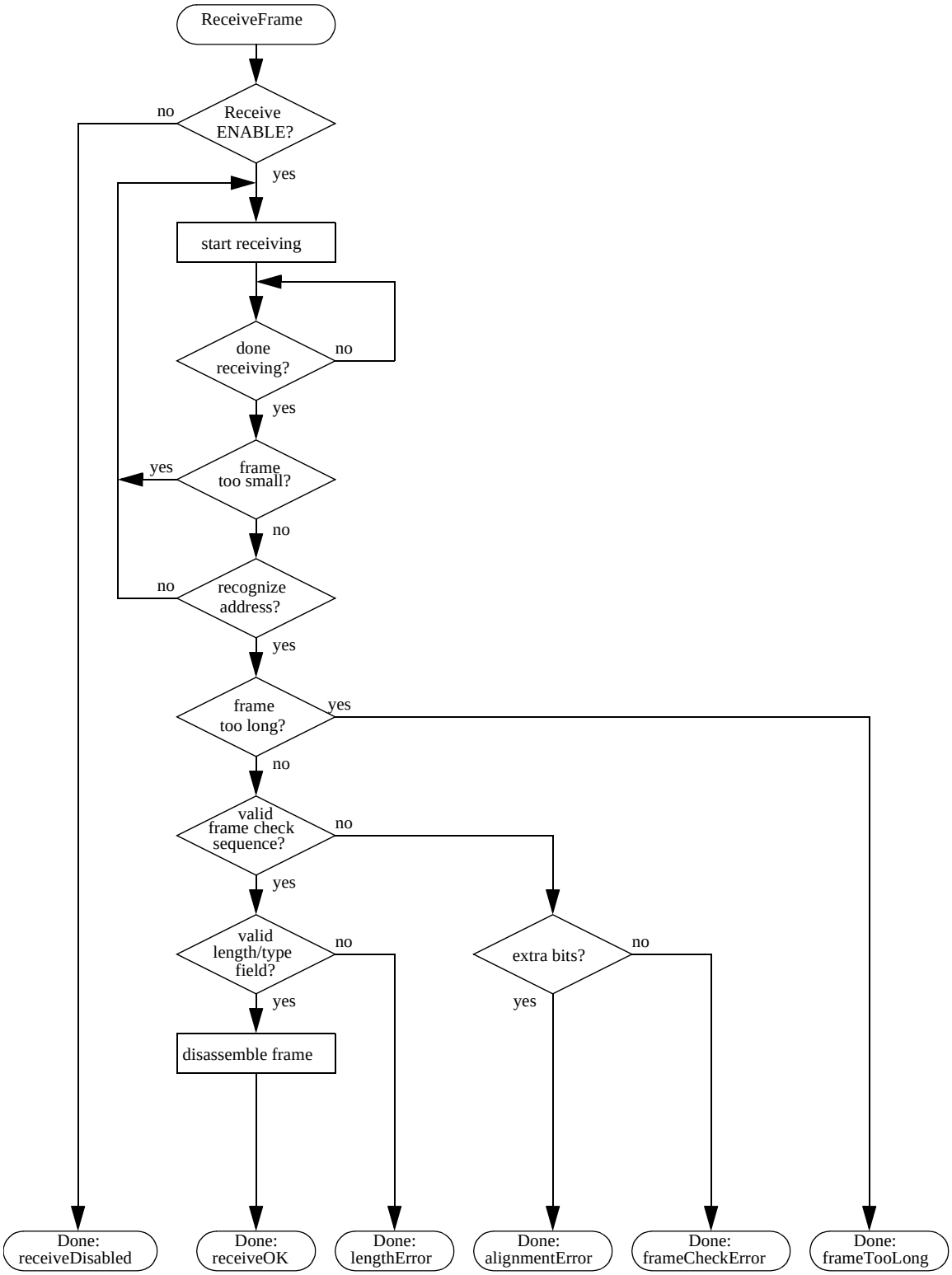
Frame transmission includes data encapsulation and Media Access management aspects:

a) Transmit Data Encapsulation includes the assembly of the outgoing frame (from the values provided by the MAC client) and frame check sequence generation.
b) Transmit Media Access Management includes carrier deference, interframe spacing and bit transmission.

4A.2.3.1 Transmit data encapsulation

The fields of the MAC frame are set to the values provided by the MAC client as arguments to the TransmitFrame operation (see 4A.3) with the following possible exceptions: the padding field and the frame check sequence. The padding field is necessary to enforce the minimum frame size. The frame check sequence field may be (optionally) provided as an argument to the MAC sublayer. It is optional for a MAC to support the provision of the frame check sequence in such an argument. If this field is provided by the MAC client, the padding field shall also be provided by the MAC client, if necessary. If this field is not provided by the MAC client, or if the MAC does not support the provision of the frame check sequence as an external argument, it is set to the CRC value generated by the MAC sublayer, after appending the padding field, if necessary.

4A.2.3.2 Transmit media access management



b) ReceiveFrame

Figure 4A–3b—Control flow summary

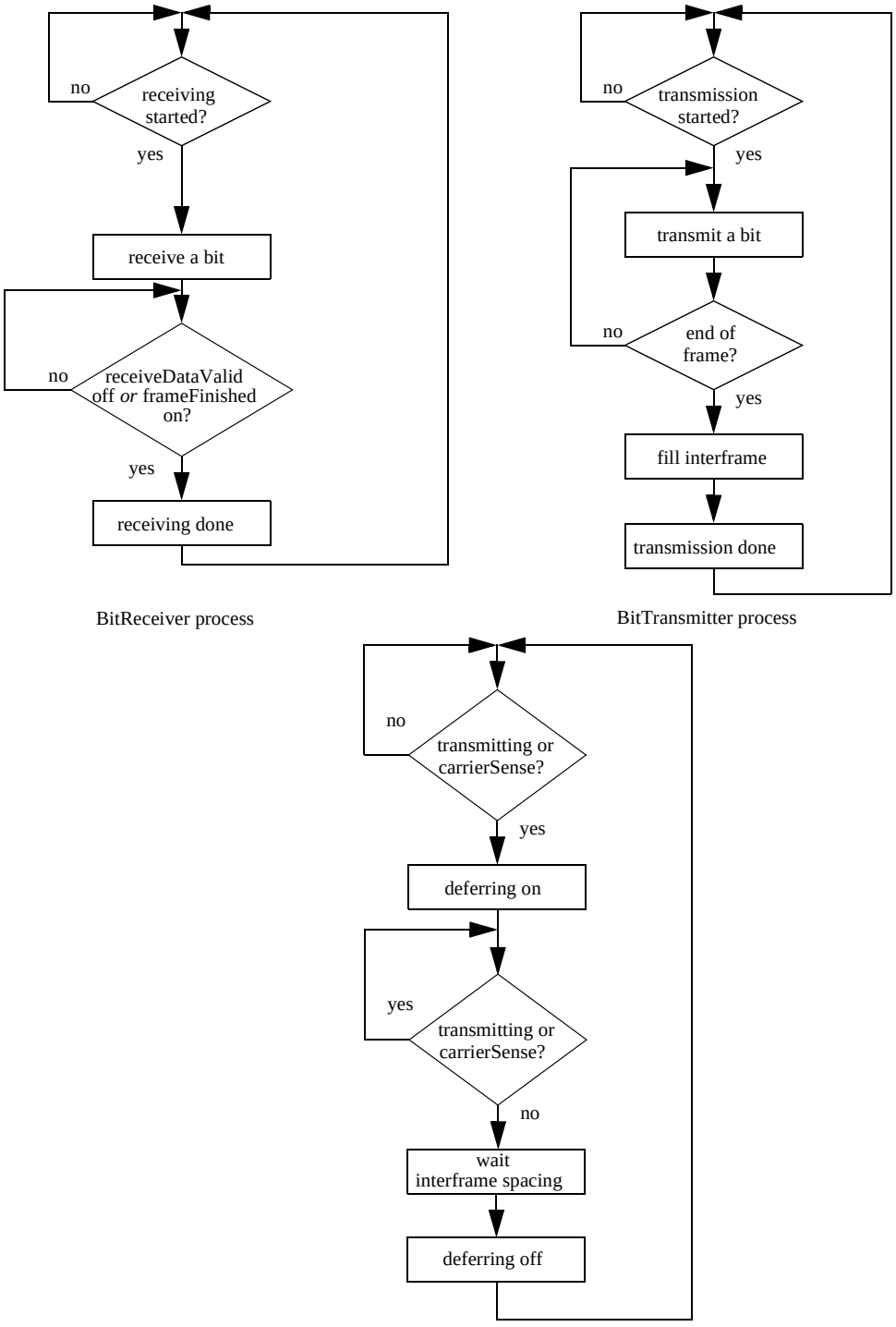


Figure 4A–3c—Control flow

4A.2.3.2.1 Deference

When a frame is submitted by the MAC client for transmission, the transmission is initiated as soon as possible, but in conformance with the following rules.

The MAC sublayer monitors the transmitting variable, which indicates the MAC is transmitting data to the physical layer, as well as the carrierSense signal provided by the PLS, which indicates the physical layer is not ready for the next frame. When either transmitting or carrierSense is true, the MAC delays any pending transmission. When both are false, the MAC continues to defer for a proper interFrameSpacing (see 4A.2.3.2.2).

If, at the end of the interFrameSpacing, a frame is waiting to be transmitted, transmission is initiated. When transmission has completed (or immediately, if there was nothing to transmit) the MAC sublayer resumes its original monitoring of transmitting and carrierSense.

4A.2.3.2.2 Interframe spacing

As defined in 99.2.3.2.1, the rules for deferring ensure a minimum interframe spacing of interFrameSpacing bit times. This is intended to provide interframe recovery time to aid in frame delineation on the physical medium.

Note that interFrameSpacing 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. The larger value is determined by the parameters of the implementation, see 4A.4.

4A.2.3.2.3 Transmission

Transmissions may be initiated whenever the station has a frame queued, subject only to the physical layer contention and interframe spacing required to allow recovery for the physical medium. In certain implementations, interframe spacing is accomplished outside this layer. These implementations are allowed to always initiate transmissions immediately, subject only to conditions enforced outside this layer.

4A.2.3.2.4 Minimum frame size

The MAC requires that a minimum frame length of minFrameSize bits be transmitted. If frameSize is less than minFrameSize, then the MAC sublayer shall append extra bits in units of octets (pad), after the end of the MAC client data field but prior to calculating, and appending, the FCS (if not provided by the MAC client). The number of extra bits shall be sufficient to ensure that the frame, from the DA field through the FCS field inclusive, is at least minFrameSize bits. If the FCS is (optionally) provided by the MAC client, the pad shall also be provided by the MAC client. The content of the pad is unspecified.

4A.2.4 Frame reception model

The MAC sublayer frame reception includes both data decapsulation and Media Access management aspects:

a) Receive Data Decapsulation comprises address recognition, frame check sequence validation, and frame disassembly to pass the fields of the received frame to the MAC client.

b) Receive Media Access Management comprises recognition of collision fragments from incoming frames and truncation of frames to octet boundaries.

4A.2.4.1 Receive data decapsulation

4A.2.4.1.1 Address recognition

The MAC sublayer is capable of recognizing individual and group addresses.

a) *Individual Addresses*. The MAC sublayer recognizes and accepts any frame whose DA field contains the individual address of the station.

b) *Group Addresses.* The MAC sublayer recognizes and accepts any frame whose DA field contains the Broadcast address.

The MAC sublayer is capable of activating some number of group addresses as specified by higher layers. The MAC sublayer recognizes and accepts any frame whose Destination Address field contains an active group address. An active group address may be deactivated.

The MAC sublayer may also provide the capability of operating in the promiscuous receive mode. In this mode of operation, the MAC sublayer recognizes and accepts all valid frames, regardless of their Destination Address field values.

4A.2.4.1.2 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 the frame is identified as invalid.

4A.2.4.1.3 Frame disassembly

Upon recognition of the Start Frame Delimiter at the end of the preamble sequence, the MAC sublayer accepts the frame. If there are no errors, the frame is disassembled and the fields are passed to the MAC client by way of the output parameters of the ReceiveFrame operation.

4A.2.4.2 Receive media access management

4A.2.4.2.1 Framing

The MAC sublayer recognizes the boundaries of an incoming frame by monitoring the receiveDataValid signal provided by the Physical Layer. Two possible length errors can occur that indicate ill-framed data: the frame may be too long, or its length may not be an integer number of octets.

a) *Maximum Frame Size.* The receiving MAC sublayer is not required to enforce the frame size limit, but it is allowed to truncate frames longer than maxUntaggedFrameSize octets and report this event as an (implementation-dependent) error. A receiving MAC sublayer that supports tagged MAC frames (see 3.5) may similarly truncate frames longer than (maxUntaggedFrameSize + qTagPrefixSize) octets in length, and report this event as an (implementation-dependent) error.

b) *Integer Number of Octets in Frame.* Since the format of a valid frame specifies an integer number of octets, only a collision or an error can produce a frame with a length that is not an integer multiple of 8 bits. Complete frames (that is, not rejected for being too small) that do not contain an integer 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

4A.2.5 Preamble generation

In a LAN implementation, most of the Physical Layer components are allowed to provide valid output some number of bit times after being presented valid input signals. Thus it is necessary for a preamble to be sent before the start of data, to allow the PLS circuitry to reach its steady state. Upon request by TransmitLinkMgmt to transmit the first bit of a new frame, BitTransmitter shall first transmit the preamble, a bit sequence used for physical medium stabilization and synchronization, followed by the Start Frame Delimiter. The preamble pattern is:

10101010 10101010 10101010 10101010 10101010 10101010 10101010

The bits are transmitted in order, from left to right. The nature of the pattern is such that, for Manchester encoding, it appears as a periodic waveform on the medium that enables bit synchronization. It should be noted that the preamble ends with a "0."

4A.2.6 Start frame sequence

The receiveDataValid signal is the indication to the MAC that the frame reception process should begin. Upon reception of the sequence 10101011 following the assertion of receiveDataValid, PhysicalSignalDecap shall begin passing successive bits to ReceiveLinkMgmt for passing to the MAC client.

4A.2.7 Global declarations

This subclause provides detailed formal specifications for the MAC sublayer. It is a specification of generic features and parameters to be used in systems implementing this media access method. 4A.4 provides values for these sets of parameters for recommended implementations of this media access mechanism.

4A.2.7.1 Common constants, types, and variables

The following declarations of constants, types and variables are used by the frame transmission and reception sections of each MAC sublayer:

```
const
    addressSize = 48; {In bits, in compliance with 3.2.3}
    lengthOrTypeSize = 16; {In bits}
    clientDataSize = ...; {In bits, size of MAC client data; see 4A.2.2.2, a) 3)}
    padSize = ...; {In bits, = max (0, minFrameSize ─ (2 x addressSize + lengthOrTypeSize +
                                                    clientDataSize + crcSize))}
    dataSize = ...; {In bits, = clientDataSize + padSize}
    crcSize = 32; {In bits, 32-bit CRC}
    frameSize = ...; {In bits, = 2 x addressSize + lengthOrTypeSize + dataSize + crcSize; see 4A.2.2.2, a)}
    minFrameSize = ...-; {In bits, implementation-dependent, see 4A.4}
    maxUntaggedFrameSize = ...-; {~~i~~In octets, implementation-dependent, see 4A.4}
    qTagPrefixSize = 4; {In octets, length of QTag Prefix, see 3.5}
    minTypeValue = 1536; {Minimum value of the Length/Type field for Type interpretation}
    maxValidFrame = maxUntaggedFrameSize ─ (2 x addressSize + lengthOrTypeSize + crcSize) / 8;
                    {In octets, the maximum length of the MAC client data field. This constant is
                    defined for editorial convenience, as a function of other constants}
    preambleSize = 56; {In bits, see 4A.2.5}
    sfdSize = 8; {In bits, start frame delimiter}
        headerSize = 64; {In bits, sum of preambleSize and sfdSize}
type
    Bit = (0, 1);
    PhysicalBit = (0, 1); {Bits transmitted to the Physical Layer can be either 0 or 1. Bits received
                    from the Physical Layer can be either 0 or 1}
    AddressValue = array [1..addressSize] of Bit;
    LengthOrTypeValue = array [1..lengthOrTypeSize] of Bit;
    DataValue = array [1..dataSize] of Bit; {Contains the portion of the frame that starts with the first bit
                                    following the Length/Type field and ends with the last bit
                                    prior to the FCS field. For VLAN Tagged frames, this value
                                    includes the Tag Control Information field and the original
                                    MAC client Length/Type field. See 3.5}
    CRCValue = array [1..crcSize] of Bit;
    PreambleValue = array [1..preambleSize] of Bit;
```

```
SfdValue = array [1..sfdSize] of Bit;
ViewPoint = (fields, bits); {Two ways to view the contents of a frame}
HeaderViewPoint = (headerFields, headerBits);
Frame = record {Format of Media Access frame}
   case view: ViewPoint of
      fields: (
         destinationField: AddressValue;
         sourceField: AddressValue;
         lengthOrTypeField: LengthOrTypeValue;
         dataField: DataValue;
         fcsField: CRCValue);
      bits: (contents: array [1..frameSize] of Bit)
   end; {Frame}

Header = record {Format of preamble and start frame delimiter}
   case headerView: HeaderViewPoint of
      headerFields: (
         preamble: PreambleValue;
         sfd: SfdValue);
         headerContents: array [1..headerSize] of Bit)
      headerBits: (headerContents: array [1..headerSize] of Bit)
   end; {Defines header for MAC frame}
```

4A.2.7.2 Transmit state variables

The following items are specific to frame transmission. (See also 4A.4.)

```
const
   interFrameSpacing = ...; {In bit times, minimum gap between frames. Equal to interFrameGap,
                             see 4A.4}
var
   outgoingFrame: Frame; {The frame to be transmitted}
   outgoingHeader: Header;
   currentTransmitBit, lastTransmitBit: 1..frameSize; {Positions of current and last outgoing bits in
                                                       outgoingFrame}
   lastHeaderBit: 1..headerSize;
   deferring: Boolean; {Implies any pending transmission must wait for the physical layer to be ready for
                        the next packet and for the interframe spacing}
   deferenceMode: Boolean; {Indicates the desired mode of operation, and enables waiting for the
                            deferring variable before transmitting}
```

4A.2.7.3 Receive state variables

The following items are specific to frame reception. (See also 4A.4.)

```
var
   incomingFrame: Frame; {The frame being received}
   receiving: Boolean; {Indicates that a frame reception is in progress}
   excessBits: 0..7; {Count of excess trailing bits beyond octet boundary}
   receiveSucceeding: Boolean; {Running indicator of whether reception is succeeding}
   validLength: Boolean; {Indicator of whether received frame has a length error}
   exceedsMaxLength: Boolean; {Indicator of whether received frame has a length longer than the
                               maximum permitted length}
   passReceiveFCSMode: Boolean; {Indicates the desired mode of operation, and enables passing of
                                 the frame check sequence field of all received frames from the
```

MAC sublayer to the MAC client. passReceiveFCSMode is a static variable}

4A.2.7.4 Summary of interlayer interfaces

a) The interface to the MAC client, defined in 4A.3.2, is summarized below:

```
type
    TransmitStatus = (transmitDisabled, transmitOK, excessiveCollisionError, lateCollisionErrorStatus);
                {Result of TransmitFrame operation, the values excessiveCollisionError and
                    lateCollisionError are never generated by this MAC but maintained here as
                    artifacts of the original CMSA/CD MAC}
    ReceiveStatus = (receiveDisabled, receiveOK, frameTooLong, frameCheckError, lengthError,
                alignmentError); {Result of ReceiveFrame operation}
function TransmitFrame (
    destinationParam: AddressValue;
    sourceParam: AddressValue;
    lengthOrTypeParam: LengthOrTypeValue;
    dataParam: DataValue;
    fcsParamValue: CRCValue;
    fcsParamPresent: Bit): TransmitStatus;  {Transmits one frame}
function ReceiveFrame (
    var destinationParam: AddressValue;
    var sourceParam: AddressValue;
    var lengthOrTypeParam: LengthOrTypeValue;
    var dataParam: DataValue;
    var fcsParamValue: CRCValue;
    var fcsParamPresent: Bit): ReceiveStatus; {Receives one frame}
```

b) The interface to the Physical Layer, defined in 4A.3.3, is summarized in the following:

```
var
    receiveDataValid: Boolean; {Indicates incoming bits}
    carrierSense: Boolean; {Indicates that physical layer is not ready for the next packet and that
                        transmission should defer}
    transmitting: Boolean; {Indicates outgoing bits}
    collisionDetect: Boolean; {Unused by this MAC but maintained as an artifact of the CSMA/CD MAC}
procedure TransmitBit (bitParam: PhysicalBit); {Transmits one bit}
function ReceiveBit: PhysicalBit; {Receives one bit}
procedure Wait (bitTimes: integer); {Waits for indicated number of bit times}
```

4A.2.7.5 State variable initialization

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

NOTE—Care should be taken to ensure that the time from the completion of the Initialize process to when the first packet transmission begins is at least an interFrameGap.

If Layer Management is implemented, the Initialize procedure shall only be called as the result of the initializeMAC action (30.3.1.2.1).

```
procedure Initialize;
```

```
begin
   deferring := false;
   transmitting := false; {An interface to Physical Layer; see below}
   receiving := false;
   passReceiveFCSMode := ...; {True when enabling the passing of the frame check sequence of all
                               received frames from the MAC sublayer to the MAC client is desired and
                               supported, false otherwise}
   deferenceMode := ...; {False for implementations that cannot rely on deference within the MAC to
                          provide an interframe gap, true otherwise}
   while carrierSense or receiveDataValid do nothing
   {Start execution of all processes}
end; {Initialize}
```

4A.2.8 Frame transmission

The algorithms in this subclause define MAC sublayer frame transmission. The function TransmitFrame implements the frame transmission operation provided to the MAC client:

```
function TransmitFrame (
   destinationParam: AddressValue;
   sourceParam: AddressValue;
   lengthOrTypeParam: LengthOrTypeValue;
   dataParam: DataValue;
   fcsParamValue: CRCValue;
   fcsParamPresent: Bit): TransmitStatus;
procedure TransmitDataEncap; {Nested procedure; see body below}
begin
   if transmitEnabled then
      begin
         TransmitDataEncap;
         TransmitFrame := TransmitLinkMgmt
      end
   else TransmitFrame := transmitDisabled
end; {TransmitFrame}
```

If transmission is enabled, TransmitFrame calls the internal procedure TransmitDataEncap 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.

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;
         lengthOrTypeField := lengthOrTypeParam;
         if fcsParamPresent then
            begin
               dataField := dataParam; {No need to generate pad if the FCS is passed from MAC client}
```

```
                    fcsField := fcsParamValue {Use the FCS passed from MAC client}
                end
            else
                begin
                    dataField := ComputePad(dataParam);
                    fcsField := CRC32(outgoingFrame)
                end;
            view := bits
        end {Assemble frame}
    with outgoingHeader do
        begin
            headerView := headerFields;
            preamble := ...; {* '1010...10,' LSB to MSB*}
            sfd := ...; {* '10101011,' LSB to MSB*}
            headerView := headerBits
        end
end; {TransmitDataEncap}
```

If the MAC client chooses to generate the frame check sequence field for the frame, it passes this field to the MAC sublayer via the fcsParamValue parameter. If the fcsParamPresent parameter is true, TransmitDataEncap uses the fcsParamValue parameter as the frame check sequence field for the frame. Such a frame shall not require any padding, since it is the responsibility of the MAC client to ensure that the frame meets the minFrameSize constraint. If the fcsParamPresent parameter is false, the fcsParamValue parameter is unspecified. TransmitDataEncap first calls the ComputePad function, followed by a call to the CRC32 function to generate the padding (if necessary) and the frame check sequence field for the frame internally to the MAC sublayer.

ComputePad appends an array of arbitrary bits to the MAC client data to pad the frame to the minimum frame size:

```
begin
    ComputePad := {Append an array of size padSize of arbitrary bits to the MAC client dataField}
end; {ComputePadParam}

function ComputePad(var dataParam: DataValue): DataValue;
begin
    ComputePad := {Append an array of size padSize of arbitrary bits to the MAC client dataField}
end; {ComputePad}
```

TransmitLinkMgmt attempts to transmit the frame. When deferenceMode is true, it first defers to the physical layer if it is not ready for the next packet and to ensure proper interframe spacing. When deferenceMode is false, it begins transmitting immediately:

```
function TransmitLinkMgmt: TransmitStatus;
begin
    if deferenceMode then while deferring do {Defer to physical layer contention and IFS}
    StartTransmit;
    while transmitting do nothing
    LayerMgmtTransmitCounters; {Update transmit and transmit error counters in 5.2.4.2}
    TransmitLinkMgmt := transmitOK
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;
   transmitting := true;
   lastHeaderBit:= headerSize
end; {StartTransmit}
```

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

```
process Deference;
begin
   cycle {Main loop}
      while not transmitting and not carrierSense do nothing; {Wait for the start of transmission or
                                                                contention}
      deferring := true; {Inhibit future transmissions}
      while transmitting or carrierSense do nothing; {Wait for the end of transmission and contention}
      Wait(interFrameSpacing); {Time out entire interframe gap}
      deferring := false {Don't inhibit transmission}
   end {Main loop}
end; {Deference}
```

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

```
process BitTransmitter;
begin
   cycle {Outer loop}
      if transmitting then
         begin {Inner loop}
            while transmitting do
               begin
                  TransmitBit(outgoingFrame[currentTransmitBit]);
                  currentTransmitBit := currentTransmitBit + 1;
                  transmitting := (currentTransmitBit ≤ lastTransmitBit)
               end
         end {Inner loop}
   end {Outer loop}
end; {BitTransmitter}
```

4A.2.9 Frame reception

The algorithms in this subclause define the MAC sublayer frame reception.

The function ReceiveFrame implements the frame reception operation provided to the MAC client:

```
function ReceiveFrame (
   var destinationParam: AddressValue;
   var sourceParam: AddressValue;
   var lengthOrTypeParam: LengthOrTypeValue;
```

```
        var dataParam: DataValue;
        var fcsParamValue: CRCValue;
        var fcsParamPresent: Bit): ReceiveStatus;
    function ReceiveDataDecap: ReceiveStatus; {Nested function; see body below}
    begin
        if receiveEnabled then
            repeat
                ReceiveLinkMgmt;
                ReceiveFrame := ReceiveDataDecap;
            until receiveSucceeding
        else ReceiveFrame := receiveDisabled
    end; {ReceiveFrame}
```

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

```
    function ReceiveDataDecap: ReceiveStatus;
        var status: ReceiveStatus; {Holds receive status information}
    begin
        with incomingFrame do
            begin
                view := fields;
                receiveSucceeding := LayerMgmtRecognizeAddress(destinationField);
                if receiveSucceeding then
                    begin {Disassemble frame}
                        destinationParam := destinationField;
                        sourceParam := sourceField;
                        lengthOrTypeParam := lengthOrTypeField;
                        dataParam := RemovePad(lengthOrTypeField, dataField);
                        fcsParamValue := fcsField;
                        fcsParamPresent := passReceiveFCSMode;
                        exceedsMaxLength := ...; {Check to determine if receive frame size exceeds the maximum
                                                  permitted frame size. MAC implementations may use either
                                                  maxUntaggedFrameSize or (maxUntaggedFrameSize +
                                                  qTagPrefixSize) for the maximum permitted frame size,
                                                  either as a constant or as a function of whether the frame being
                                                  received is a basic or tagged frame (see 3.2, 3.5). In
                                                  implementations that treat this as a constant, it is recommended
                                                  that the larger value be used. The use of the smaller value
                                                  in this case may result in valid tagged frames exceeding the
                                                  maximum permitted frame size.}
                        if exceedsMaxLength then status := frameTooLong
                        else if fcsField = CRC32(incomingFrame) then
                                if validLength then status := receiveOK else status := lengthError
                        else if excessBits = 0 then status := frameCheckError
                        else status := alignmentError;
                        LayerMgmtReceiveCounters(status); {Update receive counters in 5.2.4.3}
                        view := bits
                    end {Disassemble frame}
            end; {With incomingFrame}
        ReceiveDataDecap := status
    end; {ReceiveDataDecap}
```

```
function LayerMgmtRecognizeAddress(address: AddressValue): Boolean;
begin
   if {promiscuous receive enabled} then LayerMgmtRecognizeAddress := true;
   else if address = ... {MAC station address} then LayerMgmtRecognizeAddress := true;
   else if address = ... {Broadcast address} then LayerMgmtRecognizeAddress := true;
   else if address = ... {One of the addresses on the multicast list and multicast reception is enabled} then
                                                          LayerMgmtRecognizeAddress := true;
   else LayerMgmtRecognizeAddress := false
end; {LayerMgmtRecognizeAddress}
```

The function RemovePad strips any padding that was generated to meet the minFrameSize constraint, if possible. When the MAC sublayer operates in the mode that enables passing of the frame check sequence field of all received frames to the MAC client (passReceiveFCSMode variable is true), it shall not strip the padding and it shall leave the data field of the frame intact. Length checking is provided for Length interpretations of the Length/Type field. For Length/Type field values in the range between maxValidFrame and minTypeValue, the behavior of the RemovePad function is unspecified:

```
function RemovePad(var lengthOrTypeParam: LengthOrTypeValue; dataParam: DataValue): DataValue;
begin
   if lengthOrTypeParam ≥ minTypeValue then
      begin
         validLength := true; {Don't perform length checking for Type field interpretations}
         RemovePad := dataParam
      end
   else if lengthOrTypeParam ≤ maxValidFrame then
      begin
         validLength := {For length interpretations of the Length/Type field, check to determine if value
                        represented by Length/Type field matches the received clientDataSize};
         if validLength and not passReceiveFCSMode then
            RemovePad := {Truncate the dataParam (when present) to the value represented by the
                           lengthOrTypeParam (in octets) and return the result}
         else RemovePad := dataParam
      end
end; {RemovePad}
```

ReceiveLinkMgmt attempts repeatedly to receive the bits of a frame, discarding any fragments smaller than 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 := receiveSucceeding and (frameSize ≥ minFrameSize)
                                                          {Reject frames too small}
   until receiveSucceeding
end; {ReceiveLinkMgmt}

procedure StartReceive;
begin
   receiveSucceeding := true;
```

```
        receiving := true
    end; {StartReceive}
```

The BitReceiver process runs asynchronously, receiving bits from the medium at the rate determined by the Physical Layer's ReceiveBit operation, partitioning them into frames, and optionally receiving them:

```
    process BitReceiver;
        var   b: PhysicalBit;
              incomingFrameSize: integer; {Count of all bits received in frame including extension}
              frameFinished: Boolean;
              enableBitReceiver: Boolean;
              currentReceiveBit: 1..frameSize; {Position of current bit in incomingFrame}
    begin
        cycle {Outer loop}
            if receiveEnabled then
                begin {Receive next frame from physical layer}
                    currentReceiveBit := 1;
                    incomingFrameSize := 0;
                    frameFinished := false;
                    enableBitReceiver := receiving;
                    PhysicalSignalDecap; {Skip idle, strip off preamble and sfd}
                    while receiveDataValid and not frameFinished do
                        begin {Inner loop to receive the rest of an incoming frame}
                            b := ReceiveBit; {Next bit from physical medium}
                            incomingFrameSize := incomingFrameSize + 1;
                            if enableBitReceiver then {Append to frame}
                            begin
                                incomingFrame[currentReceiveBit] := b;
                                currentReceiveBit := currentReceiveBit + 1
                            end
                        end; {Inner loop}
                    if enableBitReceiver then
                        begin
                            frameSize := currentReceiveBit – 1;
                            receiveSucceeding := true;
                            receiving := false
                        end
                end {Enabled}
        end {Outer loop}
    end; {BitReceiver}

    procedure PhysicalSignalDecap;
    begin
        {Receive one bit at a time from physical medium until a valid sfd is detected, discard bits and return.}
    end; {PhysicalSignalDecap}
```

4A.2.10 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 for the entire frame, excluding the FCS field (if present)}
```

end; {CRC32}

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

procedure nothing; *begin end*;

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

4A.3 Interfaces to/from adjacent layers

4A.3.1 Overview

The purpose of this clause is to provide precise definitions of the interfaces between the architectural layers defined in Clause 1 in compliance with the Media Access Service Specification given in Clause 2. In addition, the services required from the physical medium are defined.

The notation used here is the Pascal language, in keeping with the procedural nature of the precise MAC sublayer specification (see 4A.2). Each interface is described as a set of procedures or shared variables, or both, that collectively provide the only valid interactions between layers. The accompanying text describes the meaning of each procedure or variable and points out any implicit interactions among them.

Note that the description of the interfaces in Pascal is a notational technique, and in no way implies that they can or should be implemented in software. This point is discussed more fully in 4A.2, that provides complete Pascal declarations for the data types used in the remainder of this clause. Note also that the synchronous (one frame at a time) nature of the frame transmission and reception operations is a property of the architectural interface between the MAC client and MAC sublayers, and need not be reflected in the implementation interface between a station and its sublayer.

4A.3.2 Services provided by the MAC sublayer

The services provided to the MAC client by the MAC sublayer are transmission and reception of frames. The interface through which the MAC client uses the facilities of the MAC sublayer therefore consists of a pair of functions.

Functions:

TransmitFrame

ReceiveFrame

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

NOTE 1—The frame_check_sequence parameter defined in 2.3.1 and 2.3.2 is mapped here into two variables: fcsParamValue and fcsParamPresent. This mapping has been defined for editorial convenience. The fcsParamPresent variable indicates the presence or absence of the fcsParamValue variable in the two function calls. If the fcsParamPresent variable is true, the fcsParamValue variable contains the frame check sequence for the corresponding frame. If the fcsParamPresent variable is false, the fcsParamValue variable is unspecified. If the MAC sublayer does not support client-supplied frame check sequence values, then the fcsParamPresent variable in TransmitFrame shall always be false.

NOTE 2—The mac_service_data_unit parameter defined in 2.3.1 and 2.3.2 is mapped here into two variables: lengthOrTypeParam and dataParam. This mapping has been defined for editorial convenience. The first two octets of the mac_service_data_unit parameter contain the lengthOrTypeParam variable. The remaining octets of the mac_service_data_unit parameter form the dataParam variable.

The MAC client transmits a frame by invoking TransmitFrame:

```
function TransmitFrame (
    destinationParam: AddressValue;
    sourceParam: AddressValue;
    lengthOrTypeParam: LengthOrTypeValue;
    dataParam: DataValue;
    fcsParamValue: CRCValue;
    fcsParamPresent: Bit): TransmitStatus;
```

The TransmitFrame operation is synchronous. Its duration is the entire attempt to transmit the frame; when the operation completes, transmission has either succeeded or failed, as indicated by the resulting status code:

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

The transmitDisabled status code indicates that the transmitter is not enabled. Successful transmission is indicated by the status code transmitOK. The codes excessiveCollisionError and lateCollisionErrorStatus are artifacts of the CSMA/CD MAC and maintained here for historical purposes. These codes are never generated by this full duplex MAC. TransmitStatus is not used by the service interface defined in 2.3.1. TransmitStatus may be used in an implementation dependent manner.

The MAC client accepts incoming frames by invoking ReceiveFrame:

```
function ReceiveFrame (
    var destinationParam: AddressValue;
    var sourceParam: AddressValue;
    var lengthOrTypeParam: LengthOrTypeValue;
    var dataParam: DataValue;
    var fcsParamValue: CRCValue;
    var fcsParamPresent: Bit): ReceiveStatus;
```

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

```
type ReceiveStatus = (receiveDisabled, receiveOK, frameTooLong, frameCheckError,
                      lengthError, alignmentError);
```

The receiveDisabled status indicates that the receiver is not enabled. Successful reception is indicated by the status code receiveOK. The frameTooLong error indicates that a frame was received whose frameSize was beyond the maximum allowable frame size. The code frameCheckError indicates that the frame received was damaged by a transmission error. The lengthError indicates that the lengthOrTypeParam value was both consistent with a length interpretation of this field (i.e., its value was less than or equal to maxValidFrame), and inconsistent with the frameSize of the received frame. The code alignmentError indicates that the frame received was damaged, and that in addition, its length was not an integer number of octets. ReceiveStatus is not mapped to any MAC client parameter by the service interface defined in 2.3.2. ReceiveStatus may be used in an implementation dependent manner.

Note that maxValidFrame represents the maximum number of octets that can be carried in the MAC client data field of a frame and is a constant, regardless of whether the frame is a basic or tagged frame (see 3.2 and 3.5). The maximum length of a frame (including all fields from the Destination address through the FCS,

inclusive) is either maxUntaggedFrameSize (for basic frames) or maxUntaggedFrameSize + qTagPrefix-Size, for tagged frames.

4A.3.3 Services required from the physical layer

The interface through which the MAC sublayer uses the facilities of the Physical Layer consists of a function, a pair of procedures and four Boolean variables :

Function	Procedures	Variables
ReceiveBit	TransmitBit	collisionDetect
	Wait	carrierSense
		receiveDataValid
		transmitting

During transmission, the contents of an outgoing frame are passed from the MAC sublayer to the Physical Layer by way of repeated use of the TransmitBit operation:

procedure TransmitBit (bitParam: PhysicalBit);

Each invocation of TransmitBit passes one new bit of the outgoing frame to the Physical Layer. The TransmitBit operation is synchronous. The duration of the operation is the entire transmission of the bit. The operation completes when the Physical Layer is ready to accept the next bit and it transfers control to the MAC sublayer.

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

var transmitting: Boolean*;*

Before sending the first bit of a frame, the MAC sublayer sets transmitting to true, to inform the physical layer that a stream of bits will be presented via the TransmitBit operation. After the last bit of the frame has been presented, the MAC sublayer sets transmitting to false to indicate the end of the frame.

The collisionDetect variable is not used by this full duplex MAC but maintained as an artifact of the CSMA/CD MAC's interface to the physical layer.

var collisionDetect: Boolean*;*

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

function ReceiveBit: PhysicalBit;

Each invocation of ReceiveBit retrieves one new bit of the incoming frame from the Physical Layer. The ReceiveBit operation is synchronous. Its duration is the entire reception of a single bit. Upon receiving a bit, the MAC sublayer shall immediately request the next bit until all bits of the frame have been received. (See 4A.2 for details.)

The overall event of data being received is signaled to the MAC sublayer by the variable receiveDataValid:

var receiveDataValid: Boolean*;*

When the Physical Layer sets receiveDataValid to true, the MAC sublayer shall immediately begin retrieving the incoming bits by the ReceiveBit operation. When receiveDataValid subsequently becomes false, the MAC sublayer can begin processing the received bits as a completed frame. If an invocation of ReceiveBit is pending when receiveDataValid becomes false, ReceiveBit returns an undefined value, which should be discarded by the MAC sublayer. (See 4A.2 for details.)

The overall event of contention at the physical layer, indicating that the physical layer is not ready to accept the next packet, is signaled to the MAC sublayer by the variable carrierSense:

var carrierSense: Boolean*;*

The MAC sublayer shall monitor the value of carrierSense to defer its own transmissions when the physical layer is busy. The physical layer sets carrierSense to true immediately upon contention within the physical layer. After the contention ceases, carrierSense is set to false.

While the label carrierSense does not accurately describe the condition presented by this variable, the name is maintained as an artifact of the CSMA/CD MAC interface to the physical layer.

The Physical Layer also provides the procedure Wait:

procedure Wait (bitTimes: integer)*;*

This procedure waits for the specified number of bit times. This allows the MAC sublayer to measure time intervals in units of the (physical-medium-dependent) bit time.

4A.4 Specific implementations

4A.4.1 Compatibility overview

To provide total compatibility at all levels of the standard, it is required that each network component implementing the MAC sublayer procedure adheres rigidly to these specifications. The information provided in 4A.4.2 provides design parameters for specific implementations of this access method. Variations from these values result in a system implementation that violates the standard.

4A.4.2 Allowable implementations

The following parameter values shall be used for their corresponding implementations:

Parameters	Values
interFrameGap	96 bits
maxUntaggedFrameSize	1518 octets
minFrameSize	512 bits (64 octets)

NOTE 1—For 10 Mb/s implementations, the spacing between two successive non-colliding packets, from start of idle at the end of the first packet to start of preamble of the subsequent packet, can have a minimum value of 47 BT (bit times), at the AUI receive line of the DTE. This interFrameGap shrinkage is caused by variable network delays, added preamble bits, and clock skew.

NOTE 2—For 1BASE-5 implementations, see also DTE Deference Delay in 12.9.2.

NOTE 3—For 1 Gb/s implementations, the spacing between two non-colliding packets, from the last bit of the FCS field of the first packet to the first bit of the preamble of the second packet, can have a minimum value of 64 BT (bit times), as measured at the GMII receive signals at the DTE. This interFrameGap shrinkage may be caused by variable network delays, added preamble bits, and clock tolerances.

NOTE 4—For 10 Gb/s implementations, the spacing between two packets, from the last bit of the FCS field of the first packet to the first bit of the preamble of the second packet, can have a minimum value of 40 BT (bit times), as measured at the XGMII receive signals at the DTE. This interFrameGap shrinkage may be caused by variable network delays and clock tolerances.

WARNING

Any deviation from the above specified values may affect proper operation of the network.