99. Media Access Control for physical links with multiple MACs

99.1 Functional model of the MAC method

99.1.1 Overview

The architectural model described in Clause 1 is used in this clause to provide a functional description of the MAC sublayer for physical links with multiple MACs.

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 local area media suitable for use with media access mechanisms supporting multiple MACs over a single physical link.

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. In a network with multiple MACs and a single physical link, a distinct physical connection may correspond to multiple data links between two or more network entities. In this type of network, the major functionality in the MAC sublayer is limited to data encapsulation (transmit and receive) along with the associated minor functions including:

a) Framing (frame boundary delimitation, frame synchronization)
b) Addressing (handling of source and destination addresses)
c) Error detection (detection of physical medium transmission errors)

This MAC does not support the *half duplex* mode of operation so there is no need for collision avoidance or handling. Also, with multiple data links over a single physical link, one MAC cannot be responsible for media access and contention. The responsibility for coordinating multiple data streams over the one link lies with sublayers other than this MAC. Therefore, Media Access Management is limited to the transmission of bits to the physical layer and delaying any transmission for an interframe gap.

An optional MAC control sublayer, architecturally positioned between LLC (or other MAC client) and the MAC, is specified in Clause 31 and Clause 65. 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.

99.1.2 Full duplex operation

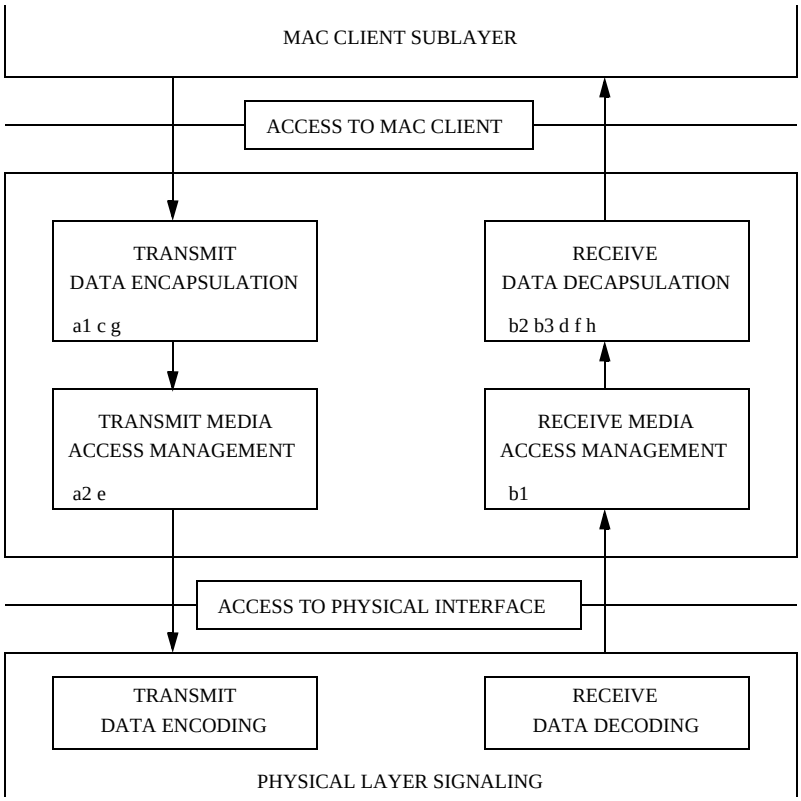
This subclause provides an overview of frame transmission and reception in terms of the functional model of the architecture. This overview is descriptive, rather than definitional; the formal specifications of the operations described here are given in 99.2 and 99.3. Specific implementations for full duplex mechanisms that meet this standard are given in 99.4. Figure 1–1 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 99.2 for the MAC functions and in Clause 7 for PLS.

Transmit frame operations are independent from receive frame operations.

99.1.4 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 99–1:



NOTE—a1, b2, etc., refer to functions listed in 99.1.4.

Figure 99–1—CSMA/CD 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) Appends proper FCS value to outgoing frames and verifies full octet boundary alignment.

d) Checks incoming frames for transmission errors by way of FCS and verifies octet boundary alignment

e) Delays transmission of frame bit stream for specified interframe gap period.

f) Discards received transmissions that are less than a minimum length.

g) 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.

h) Removes preamble, Start Frame Delimiter, DA, SA, Length/Type field, FCS, and PAD field (if necessary) from received frames.

99.2.2.2 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.)

99.2.2.3 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 99–2 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. Figures 99–3a, 99–3b, and 99–4 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.

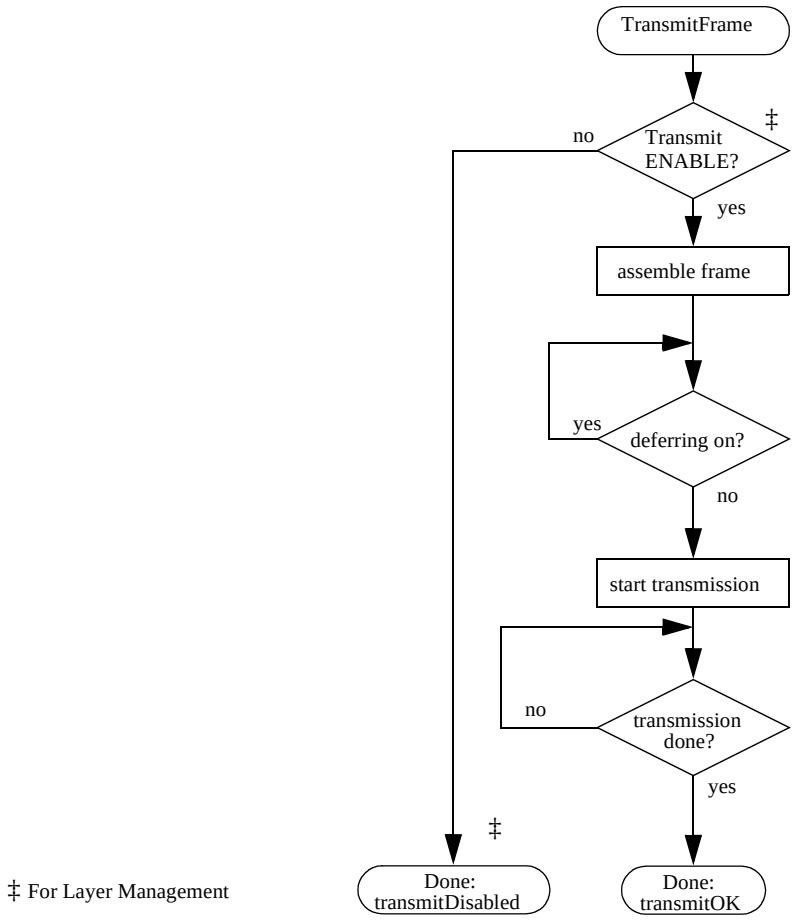


Figure 99–3a—Control flow summary

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.

99.2.3 Frame transmission model

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 interframe spacing and bit transmission.

99.2.3.1 Transmit data encapsulation

The fields of the MAC frame are set to the values provided by the MAC client as arguments to the Transmit-Frame operation (see 99.3) 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

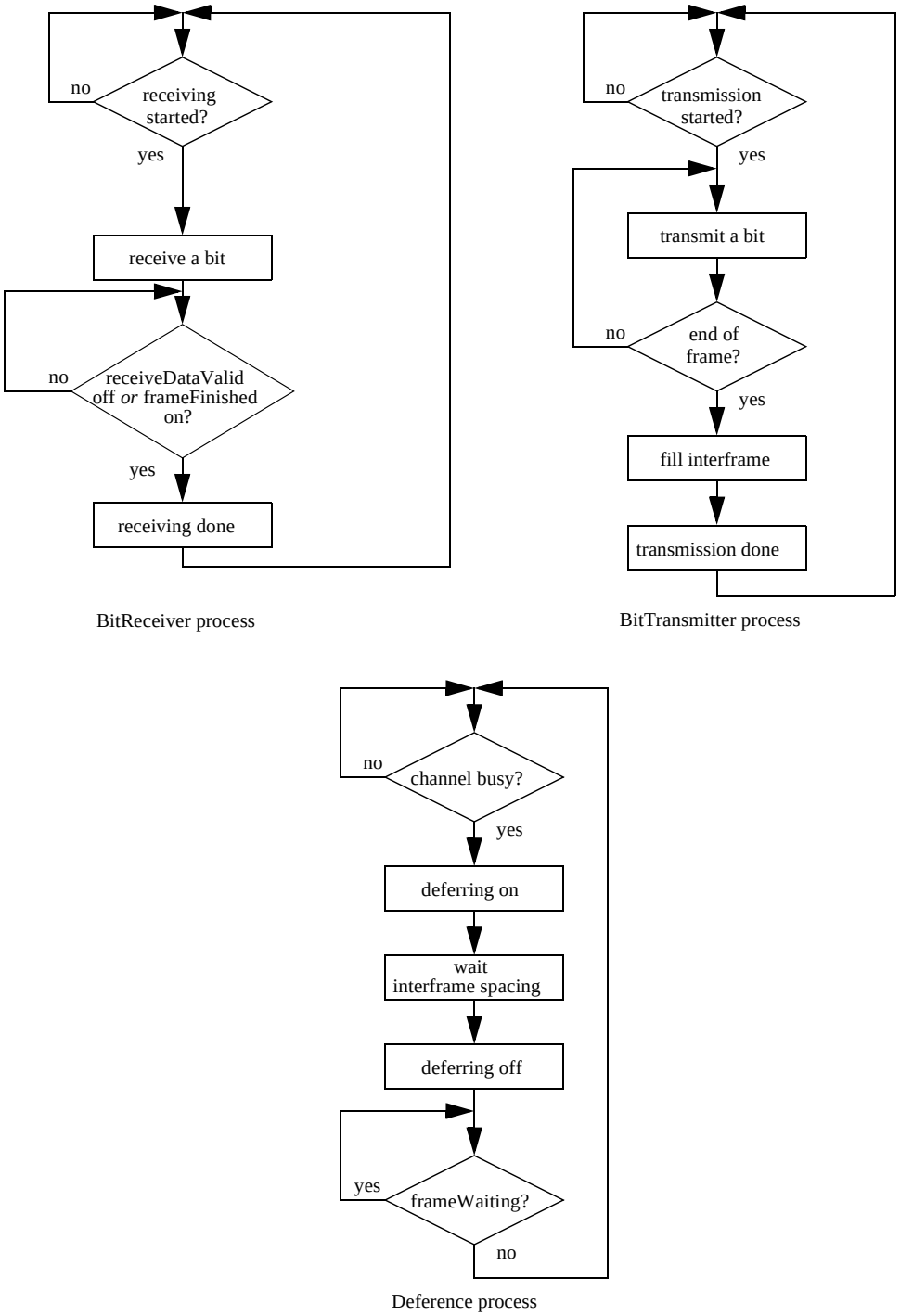


Figure 99–4—Control flow

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.

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.

99.2.4.1.2 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.

99.2.4.1.3 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.

99.2.4.2 Receive media access management

99.2.4.2.1 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 an error can produce a frame with a length that is not an integer multiple of 8 bits. Complete frames 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.

99.2.5 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.”

```
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}
```

99.2.7.2 Transmit state variables

The following items are specific to frame transmission. (See also 99.4.)

```
const
   interFrameSpacing = ...; {In bit times, minimum gap between frames. Equal to interFrameGap,
                              see 99.4}
   ifsStretchRatio = ...; {In bits, determines the number of bits in a frame that require one octet of
                           interFrameSpacing extension, when ifsStretchMode is enabled; implementation
                           dependent, see 4.4}
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}
   frameWaiting: Boolean; {Indicates that outgoingFrame is deferring}
   ifsStretchMode: Boolean; {Indicates the desired mode of operation, and enables the lowering of the
                             average data rate of the MAC sublayer (with frame granularity), using
                             extension of the minimum interFrameSpacing. ifsStretchMode is a static
                             variable; its value shall only be changed by the invocation of the Initialize
                             procedure}
   ifsStretchCount: 0..ifsStretchRatio; {In bits, a running counter that counts the number of bits during a
                                         frame's transmission that are to be considered for the minimum
                                         interFrameSpacing extension, while operating in ifsStretchMode}
   ifsStretchSize: 0..(((maxUntaggedFrameSize + qTagPrefixSize) x 8 + headerSize + interFrameSpacing
                       + ifsStretchRatio – 1) div ifsStretchRatio);
                       {In octets, a running counter that counts the integer number of octets that are to be
                       added to the minimum interFrameSpacing, while operating in ifsStretchMode}
   p2mpMode: Boolean; {Indicates the desired mode of operation, and disables waiting for the deferring
                       variable before transmitting}
```

variables are appropriately reinitialized before each use.) Initialize then waits for the medium to be idle, and starts operation of the various processes.

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
   frameWaiting: Boolean; {Indicates that outgoingFrame is deferring}
   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}
   ifsStretchMode := ...; {True for operating speeds above 1000 Mb/s when lowering the average data rate
                          of the MAC sublayer (with frame granularity) is desired and supported, false
                          otherwise}
   ifsStretchCount := 0;
   ifsStretchSize := 0;
   p2mpMode := ...; {True for Point-to-Multi-Point implementations, false otherwise}
   while receiveDataValid do nothing
   {Start execution of all processes}
end; {Initialize}
```

99.2.8 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:

```
function TransmitLinkMgmt: TransmitStatus;
begin
    frameWaiting := true;
    if not p2mpMode then while deferring do nothing {Defer to ensure proper interframe spacing}
    StartTransmit;
    frameWaiting := false;
    while transmitting do nothing {Full duplex mode}
    LayerMgmtTransmitCounters; {Update transmit and transmit error counters in 5.2.4.2}
    TransmitLinkMgmt := transmitOK
end; {TransmitLinkMgmt}
```

If the p2mpMode is enabled, then IPG is enforced outside this sublayer. If it is not enabled, then the IPG is timed using the Deference process.

Editors note: *To be removed prior to final publication*

This test for p2mpMode is option #1 to making the IPG optional for P2MP.

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. Interframe spacing may be used to lower the average data rate of a MAC at operating speeds above 1000 Mb/s in the full duplex mode, when it is necessary to adapt it to the data rate of a WAN-based physical layer. When interframe stretching is enabled, deferring remains true throughout the entire extended interframe gap, which includes the sum of interFrameSpacing and the interframe extension as determined by the BitTransmitter:

```
process Deference;
    var realTimeCounter: integer; wasTransmitting: Boolean;
begin
    while not transmitting do nothing; {Wait for the start of a transmission}
    deferring := true; {Inhibit future transmissions}
    while transmitting do nothing; {Wait for the end of the current transmission}
    Wait(interFrameSpacing + ifsStretchSize x 8); {Time out entire interframe gap and IFS extension}
    if not frameWaiting then {Don't roll over the remainder into the next frame}
        begin
            Wait(8);
            ifsStretchCount := 0
        end
    deferring := false {Don't inhibit transmission}
end; {Deference}
```

If the ifsStretchMode is enabled, the Deference process continues to enforce interframe spacing for an additional number of bit times, after the completion of timing the interFrameSpacing. The additional number of

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 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 LayerMgmtRecognizeAddress(address: AddressValue): Boolean;
    begin
        if {promiscuous receive enabled} then LayerMgmtRecognizeAddress := true;
        if address = ... {MAC station address} then LayerMgmtRecognizeAddress := true;
        if address = ... {Broadcast address} then LayerMgmtRecognizeAddress := true;
```

```
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 and extension, 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}
```

99.2.10 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.

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*);*

The transmitDisabled status code indicates that the transmitter is not enabled. Successful transmission is indicated by the status code transmitOK.. 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.

99.3.3 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 two Boolean variables:

Function	Procedures	Variables
ReceiveBit	TransmitBit	receiveDataValid
	Wait	transmitting

99.4.2 Allowable implementations

The following parameter values shall be used for all implementations:

Parameters	Values			
	10 Mb/s 1BASE-5 100 Mb/s	**1 Gb/s**	**P2MP**	**10 Gb/s**
interFrameGap	96 bits	96 bits	0 bits	96 bits
maxUntaggedFrameSize	1518 octets	1518 octets	1518 octets	1518 octets
minFrameSize	512 bits (64 octets)	512 bits (64 octets)	512 bits (64 octets)	512 bits (64 octets)
ifsStretchRatio	not applicable	not applicable	not applicable	104 bits

Editors note: *To be removed prior to final publication*

This P2MP column in the parameter table is option #2 to making the IPG optional for P2MP.

NOTE 1—For 10 Mb/s implementations, the spacing between two successive 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 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.