~~4. Media Access Control~~

~~1) The physical medium is capable of supporting simultaneous transmission and reception without interference (e.g., 10BASE-T, 10BASE-FL, and 100BASE-TX/FX).~~

~~2) There are exactly two stations on the LAN. This allows the physical medium to be treated as a full duplex point-to-point link between the stations. Since there is no contention for use of a shared medium, the multiple access (i.e., CSMA/CD) algorithms are unnecessary.~~

~~3) Both stations on the LAN are capable of and have been configured to use full duplex operation.~~

~~The most common configuration envisioned for full duplex operation consists of a central bridge (also known as a switch) with a dedicated LAN connecting each bridge port to a single device.~~

The ~~formal specification of the MAC in 99.2 comprises both the half duplex and full duplex modes of operation. The~~ remainder of this clause provides a functional model of ~~the CSMA/CD~~ this MAC method.

~~99.1.2 CSMA/CD operation~~

99.1.3 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 ~~CSMA/CD~~ 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 the receive frame operations. A transmitted frame addressed to the originating station will be received and passed to the MAC client at that station. This characteristic of the MAC sublayer may be implemented by functionality within the MAC sublayer or full duplex characteristics of portions of the lower layers.~~

~~99.1.3.1 Normal operation~~

~~99.1.3.1.1 Transmission without contention~~

Transmit frame operations are independent from receive frame operations.

99.1.3.2 Transmission

When a MAC client requests the transmission of a frame, the Transmit Data Encapsulation component of the ~~CSMA/CD~~ 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 ~~CSMA/CD~~ 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 4.2.3.3)~~requirement. 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. ~~The frame is then handed to~~ Frame transmission may be initiated after the ~~Transmit Media Access Management component in~~ interframe delay, regardless of the ~~MAC sublayer for transmission~~presence of receive activity.

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.

~~In half duplex mode, at an operating speed of 1000 Mb/s, frames may be extended by the transmitting station under the conditions described in 4.2.3.4. The extension is discarded by the MAC sublayer of the receiving station, as defined in the procedural model in 4.2.9.~~

~~**99.1.3.4 Access interference and recovery**~~

~~In half duplex mode, if multiple stations attempt to transmit at the same time, it is possible for them to interfere with each other's transmissions, in spite of their attempts to avoid this by deferring. When transmissions from two stations overlap, the resulting contention is called a collision. Collisions occur only in half duplex mode, where a collision indicates that there is more than one station attempting to use the shared physical medium. In full duplex mode, two stations may transmit to each other simultaneously without causing interference. The Physical Layer may generate a collision indication, but this is ignored by the full duplex MAC.~~

~~A given station can experience a collision during the initial part of its transmission (the collision window) before its transmitted signal has had time to propagate to all stations on the CSMA/CD medium. Once the collision window has passed, a transmitting station is said to have acquired the medium; subsequent collisions are avoided since all other (properly functioning) stations can be assumed to have noticed the signal and to be deferring to it. The time to acquire the medium is thus based on the round-trip propagation time of the Physical Layer whose elements include the PLS, PMA, and physical medium.~~

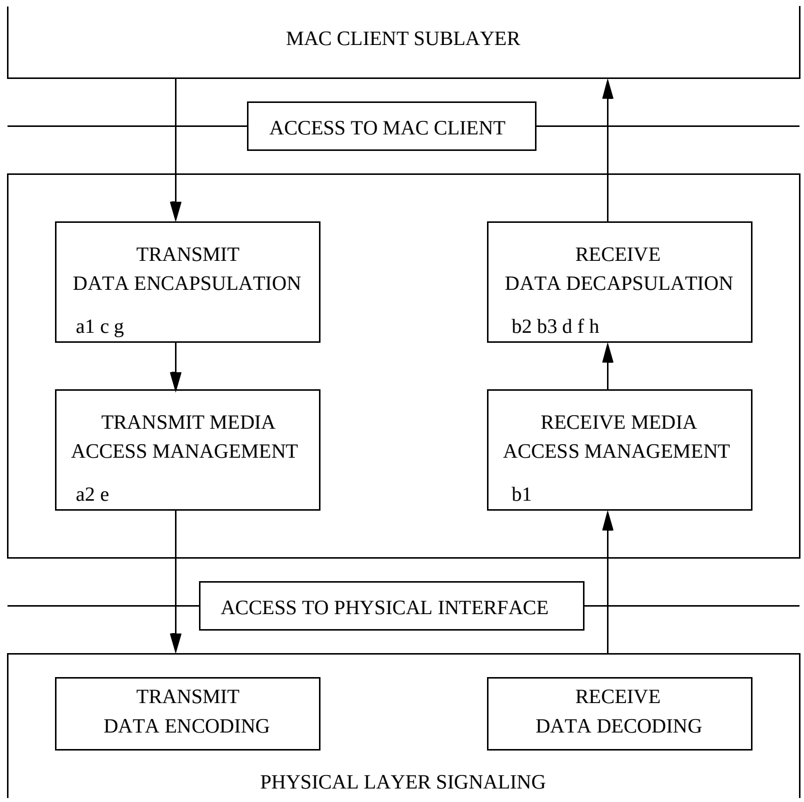
~~In the event of a collision, the transmitting station's Physical Layer initially notices the interference on the medium and then turns on the collision detect signal. In half duplex mode, this is noticed in turn by the Transmit Media Access Management component of the MAC sublayer, and collision handling begins. First, Transmit Media Access Management enforces the collision by transmitting a bit sequence called jam. In 99.4, implementations that use this enforcement procedure are provided. This ensures that the duration of the collision is sufficient to be noticed by the other transmitting station(s) involved in the collision. After the jam is sent, Transmit Media Access Management terminates the transmission and schedules another transmission attempt after a randomly selected time interval. Retransmission is attempted again in the face of repeated collisions. Since repeated collisions indicate a busy medium, however, Transmit Media Access Management attempts to adjust to the medium load by backing off (voluntarily delaying its own retransmissions to reduce its load on the medium). This is accomplished by expanding the interval from which the random retransmission time is selected on each successive transmit attempt. Eventually, either the transmission succeeds, or the attempt is abandoned on the assumption that the medium has failed or has become overloaded.~~

~~In full duplex mode, a station ignores any collision detect signal generated by the Physical Layer. Transmit Media Access Management in a full duplex station will always be able to transmit its frames without contention, so there is never any need to jam or reschedule transmissions.~~

~~At the receiving end, the bits resulting from a collision are received and decoded by the PLS just as are the bits of a valid frame. Fragmentary frames received during collisions are distinguished from valid transmissions by the MAC sublayer's Receive Media Access Management component.~~

99.1.4 Relationships to the MAC client and Physical Layers

The ~~CSMA/CD~~ MAC sublayer provides services to the MAC client required for the transmission and reception of frames. Access to these services is specified in 99.3. The ~~CSMA/CD~~ MAC sublayer makes a best effort to ~~acquire the medium and~~ 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.



NOTE—a1, b2, etc., refer to functions listed in 99.1.5.

Figure 99–2—CSMA/CD Media Access Control functions

j) Discards received transmissions that are less than a minimum length.
k) 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.
l) Removes preamble, Start Frame Delimiter, DA, SA, Length/Type field, FCS, and PAD field (if necessary) from received frames.
~~m) Appends extension bits to the first (or only) frame of a burst if it is less than slotTime bits in length when in half duplex mode at an operating speed of 1000 Mb/s.~~
~~n) Strips extension bits from received frames when in half duplex mode at an operating speed of 1000 Mb/s.~~

99.2 ~~CSMA/CD~~ Media Access Control (MAC) method: Precise specification

99.2.1 Introduction

A precise algorithmic definition is given in this subclause, providing procedural model for the ~~CSMA/CD~~ 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 ~~CSMA/CD~~ MAC method, it is the Pascal procedural specification in 99.2.7 through 99.2.11 ~~which~~ that should be consulted for the definitive statement. Subclauses 99.2.2 through 99.2.6 provide, in prose, a description of the access mechanism with the formal terminology to be used in the remaining subclauses.

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 ~~seven~~ 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 ~~five~~ three processes are defined to reside in the MAC sublayer. The ~~seven~~ five processes are as follows:

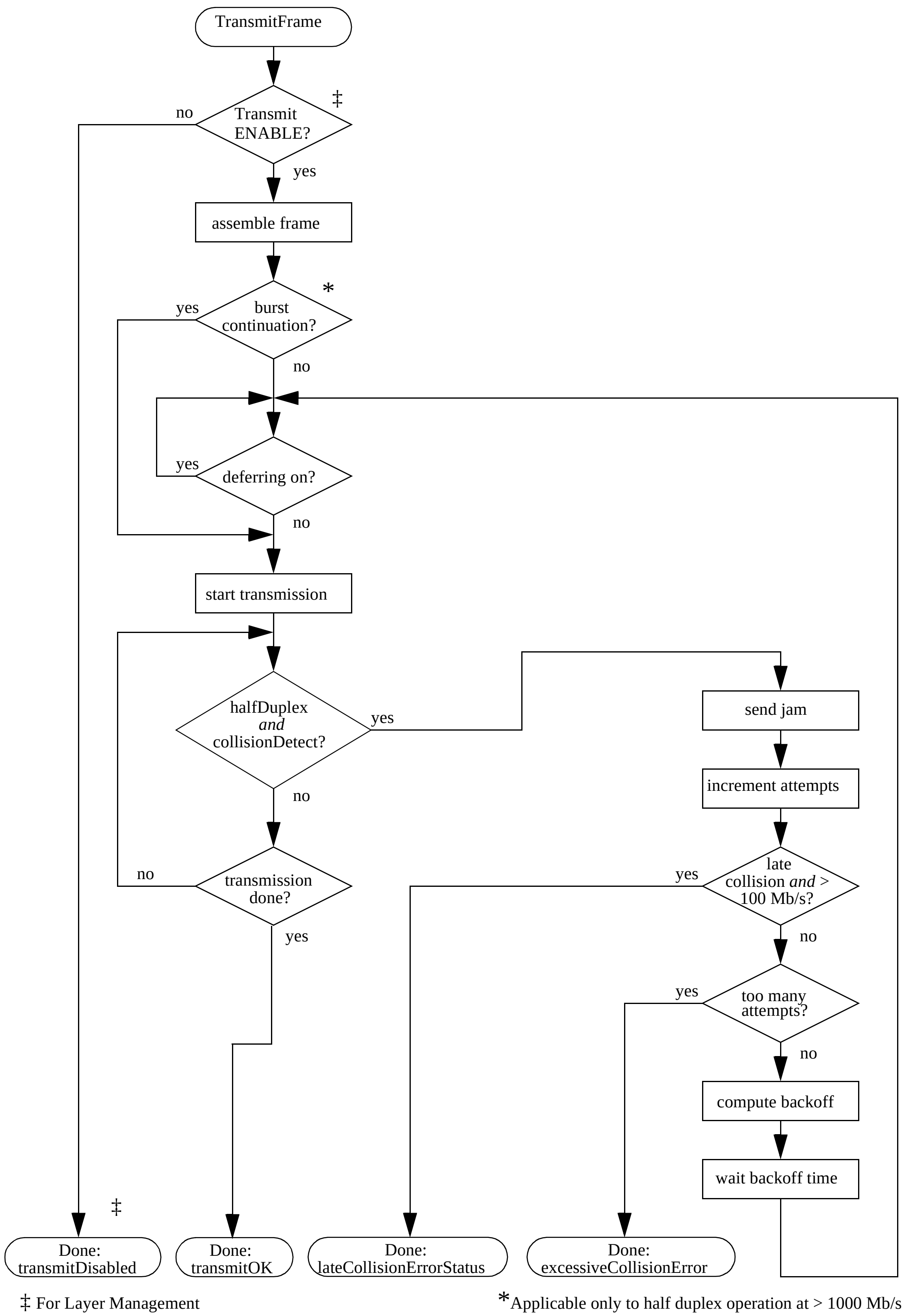
a) Frame Transmitter process
b) Frame Receiver process
c) Bit Transmitter process
d) Bit Receiver process
e) Deference process
~~f) BurstTimer process~~
~~g) SetExtending process~~

This organization of the model is illustrated in Figure 99–4 and reflects the fact that the communication of entire frames is initiated by the client of the MAC sublayer, while the timing of ~~collision backoff and of~~ individual bit transfers is based on interactions between the MAC sublayer and the Physical-Layer-dependent bit time.

Figure 99–4 depicts the static structure of the procedural model, showing how the various processes and procedures interact by invoking each other. Figures 99–5a, 99–5c, and 99–6~~, and 99–7b~~ 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.

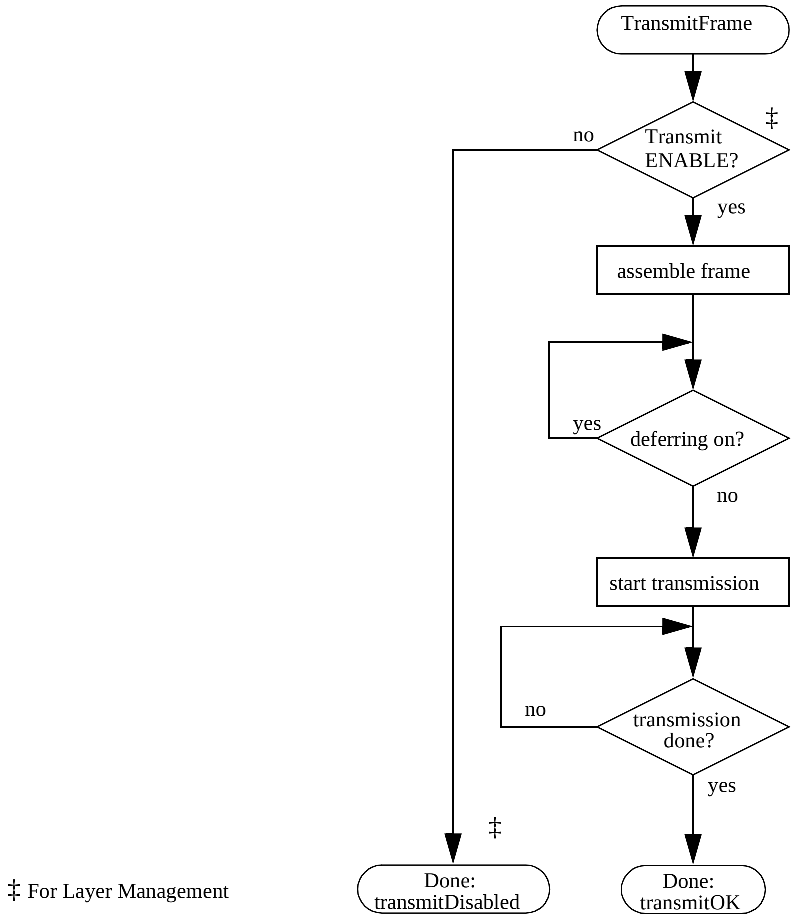
99.2.2.4 Layer management extensions to procedural model

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, 99.2.7–99.2.11, 99.3.2, Figure 99–5a, and Figure 99–5c have been modified and expanded to provide management services. Second, Layer Management procedures have been added as 5.2.4. Note that Pascal variables are shared between Clauses 99 and 5. Within the Pascal descriptions provided in Clause 99, a "‡" in the left margin indicates a line that has been added to support management services. These lines are



a) TransmitFrame

Figure 99–3a—Control flow summary



a) TransmitFrame

Figure 99–5a—Control flow summary

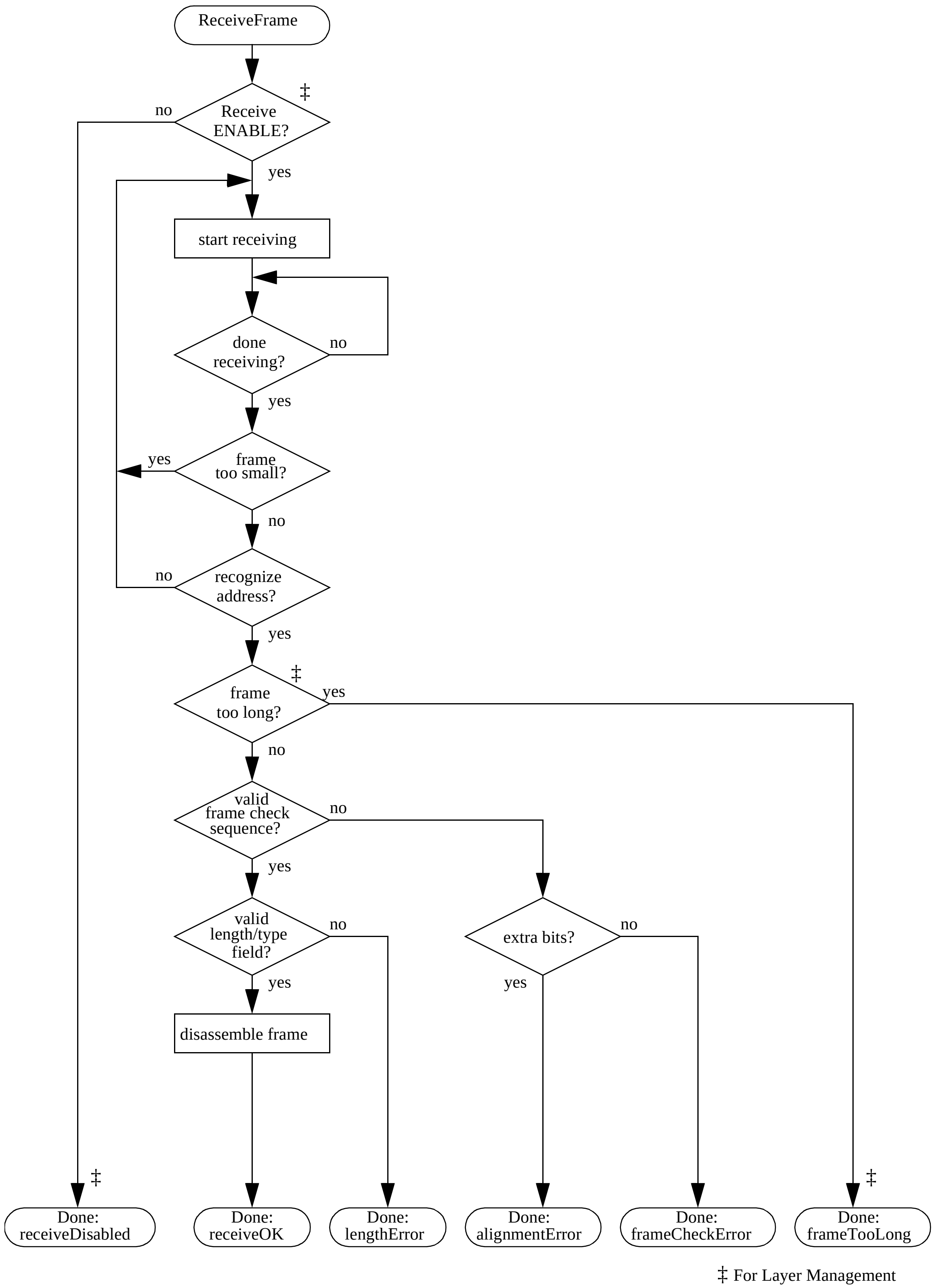
99.2.3.1 Transmit data encapsulation

The fields of the ~~CSMA/CD~~ MAC frame are set to the values provided by the MAC client as arguments to the TransmitFrame operation (see ~~4.3~~99.3) with the following possible exceptions: the padding ~~field, the extension field,~~ field and the frame check sequence. The padding field is necessary to enforce the minimum frame size~~. The extension field is necessary to enforce the minimum carrier event duration on the medium in half duplex mode at an operating speed of 1000 Mb/s~~. 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.

99.2.3.2 Transmit media access management

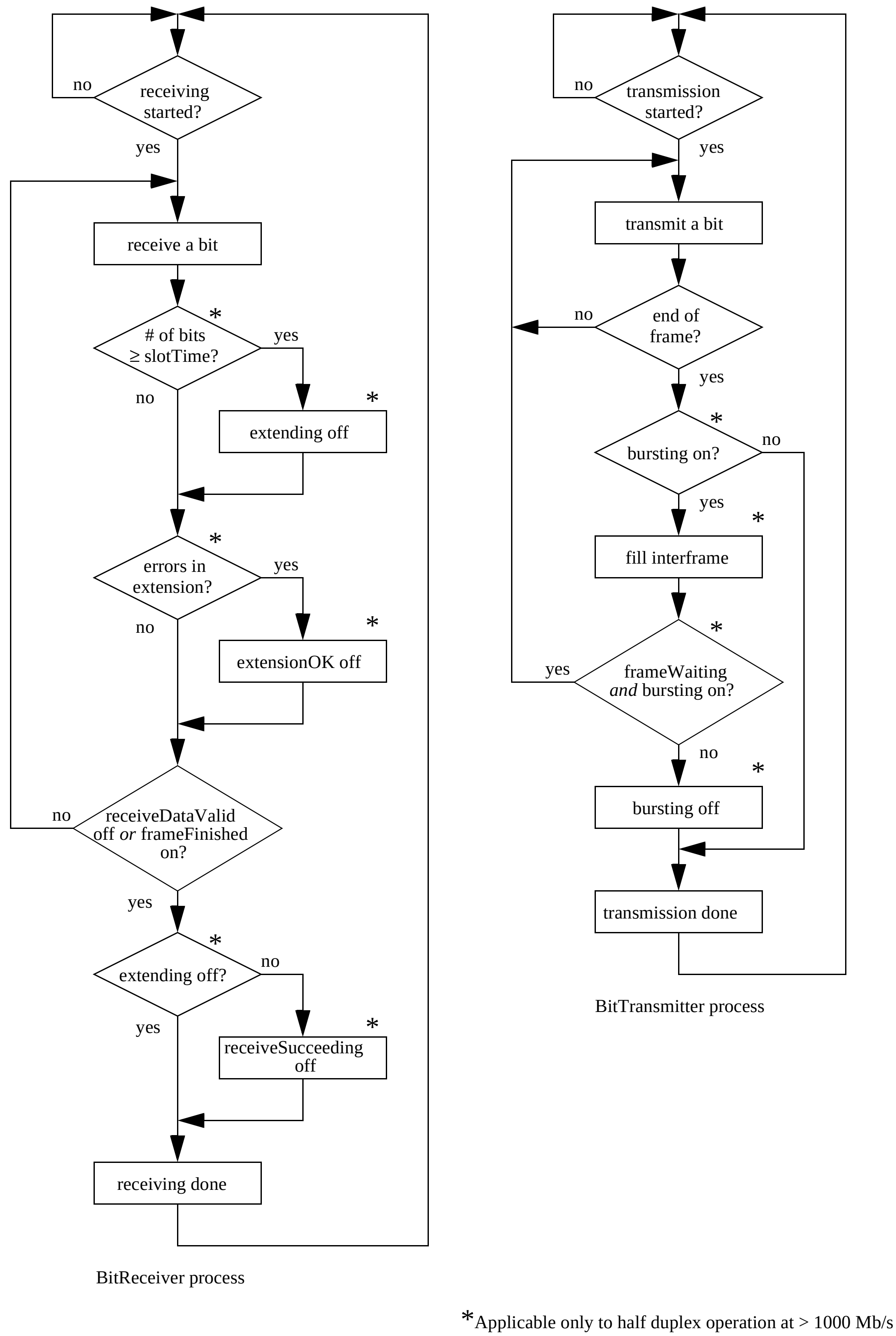
99.2.3.2.1 Deference

~~When a frame is submitted by the MAC client for transmission, the transmission is initiated as soon as possible, but in conformance with the rules of deference stated below. The rules of deference differ between half duplex and full duplex modes.~~



b) ReceiveFrame

Figure 99–5c—Control flow summary



a) MAC sublayer

Figure 99–7a—Control flow

bit of ~~the passing frame~~ a transmitted frame, (that is, when ~~*carrierSense*~~ *transmitting* changes from true to false), the ~~CSMA/CD~~ MAC continues to defer for a proper interFrameSpacing (see 99.2.3.2.2).

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

~~NOTE—It is possible for the PLS carrier sense indication to fail to be asserted briefly during a collision on the media. If the Deference process simply times the interframe gap based on this indication it is possible for a short interframe gap to be generated, leading to a potential reception failure of a subsequent frame. To enhance system robustness the following optional measures, as specified in 99.2.8, are recommended when interFrameSpacingPart1 is other than zero:~~

~~Start the timing of the interFrameSpacing as soon as transmitting and carrierSense are both false. Reset the interFrameSpacing timer if carrierSense becomes true during the first 2/3 of the interFrameSpacing timing interval. During the final 1/3 of the interval, the timer shall not be reset to ensure fair access to the medium. An initial period shorter than 2/3 of the interval is permissible including zero.~~

~~b) *Full duplex mode*~~

~~In full duplex mode, the CSMA/CD MAC does not defer pending transmissions based on the carrierSense signal from the PLS. Instead, it uses the internal variable *transmitting* to maintain proper MAC state while the transmission is in progress. After the last bit of a transmitted frame, (that is, when *transmitting* changes from true to false), the MAC continues to defer for a proper interFrameSpacing (see 99.2.3.2.2).~~

99.2.3.2.2 Interframe spacing

As defined in 99.2.3.2.1, the ~~rules~~ rule for deferring to passing frames ~~ensure~~ ensures a minimum interframe spacing of interFrameSpacing bit times. This is intended to provide interframe recovery time ~~for other CSMA/CD sublayers and for~~ 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 99.4.

A larger value for interframe spacing is used for dynamically adapting the nominal data rate of the MAC sublayer to SONET/SDH data rates (with packet granularity) for WAN-compatible applications of this standard. While in this optional mode of operation, the MAC sublayer counts the number of bits sent during a frame's transmission. After the frame's transmission has been completed, the MAC sublayer extends the minimum interframe spacing by a number of bits that is proportional to the length of the previously transmitted frame. For more details, see ~~4.2.7~~ 99.2.7 and ~~4.2.8~~99.2.8.

~~99.2.3.2.3 Collision handling (half duplex mode only)~~

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

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

~~a) It is an upper bound on the acquisition time of the medium.~~

~~99.2.3.2.7 Frame bursting (half duplex mode only)~~

~~At an operating speed of 1000 Mb/s, an implementation may optionally transmit a series of frames without relinquishing control of the transmission medium. This mode of operation is referred to as *burst mode*. Once a frame has been successfully transmitted, the transmitting station can begin transmission of another frame without contending for the medium because all of the other stations on the network will continue to defer to its transmission, provided that it does not allow the medium to assume an idle condition between frames. The transmitting station fills the interframe spacing interval with extension bits, which are readily distinguished from data bits at the receiving stations, and which maintain the detection of carrier in the receiving stations. The transmitting station is allowed to initiate frame transmission until a specified limit, referred to as burstLimit, is reached. The value of burstLimit is specified in 99.4.2. Figure 99–5 shows an example of transmission with frame bursting.~~

~~The first frame of a burst will be extended, if necessary, as described in 99.2.3.4. Subsequent frames within a burst do not require extension. In a properly configured network, and in the absence of errors, collisions cannot occur during a burst at any time after the first frame of a burst (including any extension) has been transmitted. Therefore, the MAC will treat any collision that occurs after the first frame of a burst, or that occurs after the slotTime has been reached in the first frame of a burst, as a late collision.~~

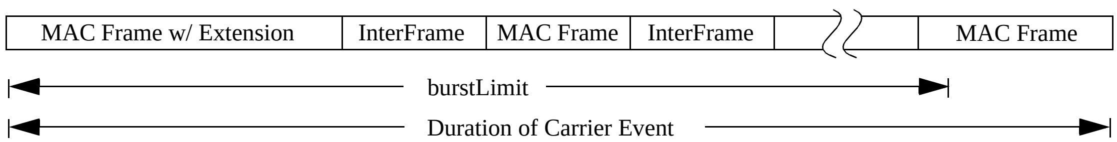


Figure 99–5—Frame bursting

~~99.2.3.3 Minimum frame size~~

~~The CSMA/CD Media Access mechanism requires that a minimum frame length of minFrameSize bits be transmitted. If frameSize is less than minFrameSize, then the CSMA/CD 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.~~

~~99.2.3.4 Carrier extension (half duplex mode only)~~

~~At an operating speed of 1000 Mb/s, the slotTime employed at slower speeds is inadequate to accommodate network topologies of the desired physical extent. Carrier Extension provides a means by which the slotTime can be increased to a sufficient value for the desired topologies, without increasing the minFrameSize parameter, as this would have deleterious effects. Nondata bits, referred to as extension bits, are appended to frames that are less than slotTime bits in length so that the resulting transmission is at least one slotTime in duration. Carrier Extension can be performed only if the underlying physical layer is capable of sending and receiving symbols that are readily distinguished from data symbols, as is the case in most physical layers that use a block encoding/decoding scheme. The maximum length of the extension is equal to the quantity (slotTime – minFrameSize). Figure 99–6 depicts a frame with carrier extension.~~

~~The MAC continues to monitor the medium for collisions while it is transmitting extension bits, and it will treat any collision that occurs after the threshold (slotTime) as a late collision.~~

99.2.4.1.3 Frame disassembly

Upon recognition of the Start Frame Delimiter at the end of the preamble sequence, the ~~CSMA/CD~~ 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 ~~CSMA/CD~~ 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 ~~CSMA/CD~~ 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 ~~CSMA/CD~~ 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 as collision fragments; see 99.2.4.2.2)~~ 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.

~~When a burst of frames is received while operating in half duplex mode at an operating speed of 1000 Mb/s, the individual frames within the burst are delimited by sequences of interframe fill symbols, which are conveyed to the receiving MAC sublayer as extension bits. Once the collision filtering requirements for a given frame, as described in 99.2.4.2.2, have been satisfied, the receipt of an extension bit can be used as an indication that all of the data bits of the frame have been received.~~

~~**99.2.4.2.2 Collision filtering**~~

~~In the absence of a collision, the shortest valid transmission in half duplex mode must be at least one slotTime in length. Within a burst of frames, the first frame of a burst must be at least slotTime bits in length in order to be accepted by the receiver, while subsequent frames within a burst must be at least minFrameSize in length. Anything less is presumed to be a fragment resulting from a collision, and is discarded by the receiver. In half duplex mode, occasional collisions are a normal part of the Media Access management procedure. The discarding of such a fragment by a MAC is not reported as an error.~~

~~The shortest valid transmission in full duplex mode must be at least minFrameSize in length. While collisions do not occur in full duplex mode MACs, a full duplex MAC nevertheless discards received frames containing less than minFrameSize bits. The discarding of such a frame by a MAC is not reported as an error.~~

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, ~~PhysicalSignalEncap~~ BitTransmitter shall first transmit the preamble, a bit sequence used for physical medium stabilization and synchronization, followed by ~~the Start~~

Bit = (0, 1);
PhysicalBit = (0, 1~~, extensionBit, extensionErrorBit~~);
~~{Bits transmitted to the Physical Layer can be either 0, 1, extensionBit or extensionErrorBit.~~ {Bits ~~received from~~ transmitted to the Physical Layer can be either ~~0,~~ 0 or 1. Bits received from the Physical Layer can be either 0 or ~~extensionBit.~~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}
~~*var*~~
~~halfDuplex: Boolean; {Indicates the desired mode of operation. halfDuplex is a static variable; its value shall only be changed by the invocation of the Initialize procedure}~~

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 4.4}~~
~~interFrameSpacingPart1 = ...; {In bit times, duration of the first portion of interFrameSpacing. In the range of 0 to 2/3 of interFrameSpacing}~~
~~interFrameSpacingPart2~~ interFrameSpacing = ...; {In bit times, ~~duration of the remainder of interFrameSpacing~~minimum gap between frames. Equal ~~to interFrameSpacing – interFrameSpacingPart1}~~to interFrameGap,
~~interFrameSize = ... ; {in bits, length of interframe fill during a burst. Equal to interFrameGap~~

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}
~~extensionOK~~passReceiveFCSMode: Boolean; {Indicates ~~whether any bit errors were found in~~ the ~~extension part~~ desired mode of ~~a frame,~~operation, and enables passing of ~~which is not checked by the CRC}~~
~~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}

99.2.7.4 Summary of interlayer interfaces

a) The interface to the MAC client, defined in ~~4.3.2~~99.3.2, is summarized below:

type
TransmitStatus = (transmitDisabled, transmitOK~~, excessiveCollisionError, lateCollisionErrorStatus~~);
~~{Result of TransmitFrame operation, reporting of lateCollisionErrorStatus is~~
~~TransmitStatus = (transmitDisabled, transmitOK, excessiveCollisionError, lateCollisionErrorStatus);~~
{Result of TransmitFrame operation~~, reporting of lateCollisionErrorStatus is~~}
ReceiveStatus = (receiveDisabled, receiveOK, frameTooLong, frameCheckError, lengthError,
~~optional for MACs operating at speeds at or below 100Mb/s}~~
~~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 ~~4.3.3~~99.3.3, is summarized in the following:

var
receiveDataValid: Boolean; {Indicates incoming bits}
~~carrierSense: Boolean; {In half duplex mode, indicates that transmission should defer}~~
transmitting: Boolean; {Indicates outgoing bits~~}~~
~~collisionDetect: Boolean; {Indicates medium contention~~}
procedure TransmitBit (bitParam: PhysicalBit); {Transmits one bit}
function ReceiveBit: PhysicalBit; {Receives one bit}
procedure Wait (bitTimes: integer); {Waits for indicated number of bit times}

```
        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 unspec-

```
            frameWaiting := falseStartTransmit;
            if halfDuplex thenframeWaiting := false;
            begin
               while transmitting do WatchForCollision;
               if lateCollisionError then lateCollisionCount := lateCollisionCount + 1;
               attempts := attempts + 1
            end while transmitting do nothing {Half Full duplex mode}
            else while transmitting do nothing {Full duplex mode}
         end; {Loop}
      LayerMgmtTransmitCounters; {Update transmit and transmit error counters in 5.2.4.2}
      if transmitSucceeding then
         begin
            if burstMode then burstStart := false; {Can't be the first frame anymore}
            TransmitLinkMgmt := transmitOK
         end
      else if (extend and lateCollisionCount > 0) then TransmitLinkMgmt := lateCollisionErrorStatus;
      else  TransmitLinkMgmt := excessiveCollisionError
   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;
      transmitSucceeding := true;
      transmitting := true;
      lastHeaderBit:= headerSize
      currentTransmitBit := 1;
      lastTransmitBit := frameSize;
      transmitting := true;
      lastHeaderBit:= headerSize
   end; {StartTransmit}
```

In half duplex mode, TransmitLinkMgmt monitors the medium for contention by repeatedly calling WatchForCollision, once frame transmission has been initiated:

```
   procedure WatchForCollision;
   begin
      if transmitSucceeding and collisionDetect then
         begin
            if currentTransmitBit > (slotTime – headerSize) then lateCollisionError := true;
            newCollision := true;
            transmitSucceeding := false;
```

~~*process* BurstTimer;~~
~~*begin*~~
~~*cycle*~~
~~*while not* bursting *do* nothing; {Wait for a burst}~~
~~Wait(burstLimit);~~
~~bursting := false~~
~~*end* {burstMode cycle}~~
~~*end*; {BurstTimer}~~

The Deference process runs asynchronously to continuously compute the proper value for the variable deferring. ~~In the case of half duplex burst mode, deferring remains true throughout the entire burst.~~ 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
~~*if* halfDuplex *then cycle* {Half duplex loop}~~
while not ~~carrierSense~~ transmitting *do* nothing; {~~Watch~~ Wait for ~~carrier to appear~~the start of a transmission}
deferring := true; {~~Delay start of new~~ Inhibit future transmissions}
~~wasTransmitting := transmitting;~~
~~*while* carrierSense *or* transmitting *do* wasTransmitting := wasTransmitting *or* transmitting;~~
~~*if* wasTransmitting *then* Wait(interFrameSpacingPart1) {Time out first part of interframe gap}~~
~~*else*~~
~~*begin*~~
~~StartRealTimeDelay;~~
~~*repeat*~~
~~*while* carrierSense *do* StartRealTimeDelay~~
~~*until not* RealTimeDelay(interFrameSpacingPart1)~~
~~realTimeCounter := interFrameSpacingPart1;~~
~~*repeat*~~
~~*while* carrierSense *do* realTimeCounter := interFrameSpacingPart1;~~
~~Wait(1);~~
~~realTimeCounter := realTimeCounter – 1~~
~~*until* (realTimeCounter = 0)~~
~~*end*;~~
~~Wait(interFrameSpacingPart2)~~*while* transmitting *do* nothing; {~~Time out second part~~ Wait for the end of ~~interframe gap~~the current transmission}
~~deferring := false; Allow new transmissions to proceed}~~
~~*while* frameWaiting *do* nothing {Allow waiting transmission, if any}~~
~~*end* {Half duplex loop}~~
~~*else cycle* {Full duplex loop}~~
~~*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);

99.2.9 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 (
~~*end*~~*var* destinationParam: AddressValue;
~~*if* bursting *then*~~
~~*begin*~~*var* sourceParam: AddressValue;
~~InterFrameSignal~~*var* lengthOrTypeParam: LengthOrTypeValue;
~~*if* extendError *then*~~
~~*if* transmitting *then* transmitting :=~~
~~called during InterFrameSignal}~~
~~{TransmitFrame may have been called during InterFrameSignal}~~
~~*else* IncLargeCounter(lateCollision);~~
~~{Count late collisions which were missed by TransmitLinkMgmt}~~
~~bursting := bursting *and* (frameWaiting *or* transmitting)~~
~~*end*~~*var* dataParam: DataValue;
~~*end* {Inner loop}~~
~~*end* {Outer loop}~~*var* fcsParamValue: CRCValue;
~~*end*; {BitTransmitter}~~

~~The bits transmitted to the physical layer can take one of four values: data zero (0), data one (1), extensionBit (EXTEND), or extensionErrorBit (EXTEND_ERROR). The values extensionBit and extensionErrorBit are not transmitted between the first preamble bit of a frame and the last data bit of a frame under any circumstances. The BitTransmitter calls the procedure TransmitBit with bitParam = extensionBit only when it is necessary to perform carrier extension on a frame after all of the data bits of a frame have been transmitted. The BitTransmitter calls the procedure TransmitBit with bitParam = extensionErrorBit only when it is necessary to jam during carrier extension.~~

~~*procedure* PhysicalSignalEncap;~~
var fcsParamPresent: Bit): ReceiveStatus;
function ReceiveDataDecap: ReceiveStatus; {Nested function; see body below}
begin
~~*while* currentTransmitBit ≤ lastHeaderBit *do*~~*if* receiveEnabled *then*
~~*begin*~~repeat
~~TransmitBit(outgoingHeader[currentTransmitBit]); {Transmit header one bit at a time}~~
ReceiveLinkMgmt;
~~currentTransmitBit~~ ReceiveFrame := ~~currentTransmitBit + 1~~ReceiveDataDecap;
~~*end*;~~*until* receiveSucceeding
~~*if* newCollision *then* StartJam~~ *else* ~~currentTransmitBit~~ ReceiveFrame := ~~1~~receiveDisabled
end; {~~PhysicalSignalEncap~~ReceiveFrame}

~~The procedure InterFrameSignal fills the interframe interval between the frames of a burst with extensionBits. InterFrameSignal also monitors the variable collisionDetect during the interframe interval between the frames of a burst, and will end a burst if a collision occurs during the interframe interval. The procedural model is defined such that a MAC operating in the burstMode will emit an extraneous sequence of interFrameSize extensionBits in the event that there are no additional frames ready for transmission after InterFrameSignal returns. Implementations may be able to avoid sending this extraneous sequence of extensionBits if they have access to information (such as the occupancy of a transmit queue) that is not assumed to be available to the procedural model.~~

~~*procedure* InterFrameSignal;~~

begin
RecognizeAddress := ...; {Returns true for the set of physical, broadcast,
and multicast-group addresses corresponding
to this station}
end; {RecognizeAddress}

~~*procedure* StartJam;~~
~~*begin*~~
~~extendError := currentTransmitBit > lastTransmitBit;~~
~~currentTransmitBit := 1;~~
~~lastTransmitBit := jamSize;~~
~~newCollision := false~~
~~*end*; {StartJam}~~

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

~~**99.2.10 Frame reception**~~

~~The algorithms in this subclause define CSMA/CD Media Access sublayer frame reception.~~

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

function ~~ReceiveFrame~~ 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;
if address = ... {One of the addresses on the multicast list and multicast reception is enabled} *then*
LayerMgmtRecognizeAddress := true;
~~*var* destinationParam: AddressValue;~~LayerMgmtRecognizeAddress := false
~~*var* sourceParam: AddressValue;~~
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;
~~*var* fcsParamValue: CRCValue;~~
~~*var* fcsParamPresent: Bit): ReceiveStatus;~~
~~*function* ReceiveDataDecap: ReceiveStatus; {Nested function; see body below}~~
begin
if ~~receiveEnabled~~ lengthOrTypeParam ≥ minTypeValue then
~~repeat~~begin
validLength := true; {Don't perform length checking for Type field interpretations}
~~ReceiveLinkMgmt;~~RemovePad := dataParam
~~ReceiveFrame := ReceiveDataDecap;~~end
else if lengthOrTypeParam ≤ maxValidFrame then


```
                exceedsMaxLength enableBitReceiver := ...; {Check to determine if receive frame size ex-
ceeds the maximumreceiving;
                                    permitted frame size. MAC implementations may use either
            PhysicalSignalDecap; {Skip idle and extension, strip off preamble and sfd}
            while receiveDataValid and not frameFinished do
                                    maxUntaggedFrameSize  or  (maxUntaggedFrameSize  +begin
{Inner loop to receive the rest of an incoming frame}
                                    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.b := ReceiveBit; {Next bit from
physical medium}
                if exceedsMaxLength then status := frameTooLong
                else if fcsField incomingFrameSize := CRC32(incomingFrame) and extensionOK thenin-
comingFrameSize + 1;
‡                 if validLength enableBitReceiver then status := receiveOK else status := lengthEr-
ror{Append to frame}
‡                 else if excessBits = 0 or not extensionOK then status := frameCheckErrorbegin
‡                   else status incomingFrame[currentReceiveBit] := alignmentErrorb;
‡                     if validLength then status: currentReceiveBit := receiveOKcurrentReceiveBit + 1
‡                     else status: = lengthErrorend
                    end; {Inner loop}
                  else
                    beginif enableBitReceiver then
‡                     if excessBits = 0 or not extensionOK then status:= frameCheckErrorbegin
‡                     else status frameSize := alignmentErrorcurrentReceiveBit – 1;
                    endreceiveSucceeding := true;
‡               LayerMgmtReceiveCounters(status); {Update receive counters in 5.2.4.3}
                view receiving := bitsfalse
            end {Disassemble frame}end
‡         end; end {With incomingFrameEnabled}
‡     ReceiveDataDecap := status
  end; end {ReceiveDataDecapOuter loop}

  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;
    if address = ... {One of the addresses on the multicast list and multicast reception is enabled} then
                                    LayerMgmtRecognizeAddress := true;
    LayerMgmtRecognizeAddress := false
  end; {LayerMgmtRecognizeAddressBitReceiver}
```

```
        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}
                if enableBitReceiver then extensionOK := true;
                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 b = 0 or b = 1 then {Normal case}
                            if enableBitReceiver then {Append to frame}
                            begin
                                if incomingFrameSize > currentReceiveBit then extensionOK := false;
                            ——                     {Errors in the extension get mapped to data bits on input}
                                incomingFrame[currentReceiveBit] := b;
                                currentReceiveBit := currentReceiveBit + 1
                            end
                        else if not extending then frameFinished := true; {b must be an extensionBit}
                        if incomingFrameSize ≥ slotTime then extending := false
                    end; {iInner loop}
                if enableBitReceiver then
                    begin
                        frameSize := currentReceiveBit – 1;
                        receiveSucceeding := not extending;
                        receiving := false
                    end
            end {Enabled}
    end {Outer loop}
end; {BitReceiver}
```

The bits received from the physical layer can take one of three values: data zero (0), data one (1), or extensionBit (EXTEND). The value extensionBit will not occur between the first preamble bit of a frame and the last data bit of a frame in normal circumstances. Extension bits are counted by the BitReceiver but are not appended to the incoming frame. The BitReceiver checks whether the bit received from the physical layer is a data bit or an extensionBit before appending it to the incoming frame. Thus, the array of bits in incomingFrame will only contain data bits. The underlying Reconciliation Sublayer (RS) maps incoming EXTEND_ERROR bits to normal data bits. Thus, the reception of additional data bits after the frame extension has started is an indication that the frame should be discarded.

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

The process SetExtending controls the extending variable, which determines whether a received frame must be at least slotTime bits in length or merely minFrameSize bits in length to be considered valid by the BitReceiver. SetExtending sets the extending variable to true whenever receiveDataValid is de-asserted, while in half duplex mode at an operating speed of 1000 Mb/s:

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 code excessiveCollisionError indicates that the transmission attempt was aborted due to excessive collisions, because of heavy traffic or a network failure. MACs operating in the half duplex mode at the speed of 1000 Mb/s are required to report lateCollisionErrorStatus in response to a late collision; MACs operating in the half duplex mode at speeds of 100 Mb/s and below are not required to do so.~~ 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, ~~lengthError~~frameTooLong, frameCheckError~~, alignmentError);~~;

var transmitting: Boolean*;*

Before sending the first bit of a frame, the MAC sublayer sets transmitting to true, to inform the Physical ~~Media Access~~ 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 presence of a collision in the physical medium is signaled to the MAC sublayer by the variable collisionDetect:~~

~~*var* collisionDetect: Boolean;~~

~~The collisionDetect signal remains true during the duration of the collision.~~

~~NOTE—In full duplex mode, collision indications may still be generated by the Physical Layer; however, they are ignored by the full duplex MAC.~~

~~The collisionDetect signal is generated only during transmission and is never true at any other time; in particular, it cannot be used during frame reception to detect collisions between overlapping transmissions from two or more other stations.~~

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 99.2 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 99.2 for details.)

~~NOTE—When a burst of frames is received in half duplex mode at an operating speed of 1000 Mb/s, the variable receiveDataValid will remain true throughout the burst. Furthermore, the variable receiveDataValid remains true throughout the extension field. In these respects, the behavior of the variable receiveDataValid is different from the underlying GMII signal RX_DV, from which it may be derived. See 35.2.1.7.~~

~~The overall event of activity on the physical medium is signaled to the MAC sublayer by the variable carrierSense:~~

~~*var* carrierSense: Boolean;~~

~~In half duplex mode, the MAC sublayer shall monitor the value of carrierSense to defer its own transmissions when the medium is busy. The Physical Layer sets carrierSense to true immediately upon detection of activity on the physical medium. After the activity on the physical medium ceases, carrierSense is set to false. Note that the true/false transitions of carrierSense are not defined to be precisely synchronized with the beginning and the end of the frame, but may precede the beginning and lag the end, respectively. (See 99.2 for details.) In full duplex mode, carrierSense is undefined.~~

Parameters	Values		
	10 Mb/s 1BASE-5 100 Mb/s	**1 Gb/s**	**10 Gb/s**
slotTime	512 bit times	4096 bit times	not applicable
interFrameGap	96 bits	96 bits	96 bits
attemptLimit	16	16	not applicable
backoffLimit	10	10	not applicable
jamSize	32 bits	32 bits	not applicable
maxUntaggedFrameSize	1518 octets	1518 octets	1518 octets
minFrameSize	512 bits (64 octets)	512 bits (64 octets)	512 bits (64 octets)
burstLimit	not applicable	65 536 bits	not applicable
ifsStretchRatio	not applicable	not applicable	104 bits

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 ~~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