~~4. Media Access Control~~

99. Full-duplex media access control

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 LAN ~~CSMA/CD~~ full-duplex MAC sublayer.

The MAC sublayer defines a medium-independent facility, built on the medium-dependent physical facility provided by the Physical Layer, and under the access-layer-independent LAN LLC sublayer (or other MAC client). It is applicable to a general class of ~~local area broadcast~~ point-to-point and point-to-multi-point media suitable for use with the full-duplex media access ~~discipline known as Carrier Sense Multiple Access with Collision Detection (CSMA/CD)~~discipline.

The LLC sublayer and the MAC sublayer together are intended to have the same function as that described in the OSI model for the Data Link Layer alone. ~~In a broadcast network,~~ The major functionality in the ~~notion of a~~ MAC sublayer is limited to data ~~link between two network entities does not correspond directly to a distinct physical connection. Nevertheless,~~ encapsulation (transmit and receive) along with the ~~partitioning of~~ associated minor functions ~~presented in this standard requires two main functions generally associated with a data link control procedure to be performed in the MAC sublayer. They are as follows~~including:

- ~~a) Data encapsulation (transmit and receive)~~
- a) Framing (frame boundary delimitation, frame synchronization)
- b) Addressing (handling of source and destination addresses)
- c) Error detection (detection of physical medium transmission errors)
- ~~d) Media Access Management~~
  - ~~1) Medium allocation (collision avoidance)~~
  - ~~2) Contention resolution (collision handling)~~

This MAC does not support the *half duplex* mode of operation so there is no need for collision avoidance or handling. 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~~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.

~~This standard provides for two modes of operation of the MAC sublayer:~~

- ~~a) In *half duplex* mode, stations contend for the use of the physical medium, using the CSMA/CD algorithms specified. Bidirectional communication is accomplished by rapid exchange of frames, rather than full duplex operation. Half duplex operation is possible on all supported media; it is required on those media that are incapable of supporting simultaneous transmission and reception without interference, for example, 10BASE2 and 100BASE-T4.~~
- ~~b) The *full duplex* mode of operation can be used when all of the following are true:~~
  - ~~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.

~~In half duplex mode, Transmit Media Access Management attempts to avoid contention with other traffic on the medium by monitoring the carrier sense signal provided by the Physical Layer Signaling (PLS) compo-~~

~~nent and deferring to passing traffic. When the medium is clear, frame transmission is initiated (after a brief interframe delay to provide recovery time for other CSMA/CD MAC sublayers and for the physical medium). The MAC sublayer then provides a serial stream of bits to the Physical Layer for transmission.~~

~~In half duplex mode, at an operating speed of 1000 Mb/s, the minimum frame size is insufficient to ensure the proper operation of the CSMA/CD protocol for the desired network topologies. To circumvent this problem, the MAC sublayer will append a sequence of extension bits to frames which are less than slotTime bits in length so that the duration of the resulting transmission is sufficient to ensure proper operation of the CSMA/CD protocol.~~

~~In half duplex mode, at an operating speed of 1000 Mb/s, the CSMA/CD MAC may optionally transmit additional frames without relinquishing control of the transmission medium, up to a specified limit.~~

~~In full duplex mode, there is no need for Transmit Media Access Management to avoid contention with other traffic on the medium. Frame transmission may be initiated after the interframe delay, regardless of the presence of receive activity. In full duplex mode, the MAC sublayer does not perform either carrier extension or frame bursting.~~

When operating in point-to-multi-point mode, contention avoidance with other traffic on the medium cannot be managed by this MAC sublayer as there are multiple MACs in parallel with this one. Sublayers other than this must be responsible for contention avoidance.

The Physical Layer performs the task of generating the signals on the medium that represent the bits of the frame. ~~Simultaneously, it monitors the medium and generates the collision detect signal, which in the contention-free case under discussion, remains off for the duration of the frame.~~ A functional description of the Physical Layer is given in Clause 7 and beyond.

When transmission has ~~completed without contention~~completed, the ~~CSMA/CD~~ MAC sublayer so informs the MAC client and awaits the next request for frame transmission.

~~**99.1.3.2.1 Reception without contention**~~

99.1.3.3 Reception

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

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

Receive Data Decapsulation checks the frame's Destination Address field to decide whether the frame should be received by this station. If so, it passes the Destination Address (DA), the Source Address (SA), the Length/~~Type~~ Type, the ~~Data~~ Data, and (optionally) the Frame Check Sequence (FCS) fields to the MAC client, along with an appropriate status code, as defined in ~~4.3.2~~99.3.2. It also checks for invalid MAC frames by inspecting the frame check sequence to detect any damage to the frame enroute, and by checking for proper octet-boundary alignment of the end of the frame. Frames with a valid FCS may also be checked for proper octet-boundary alignment.

~~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.~~

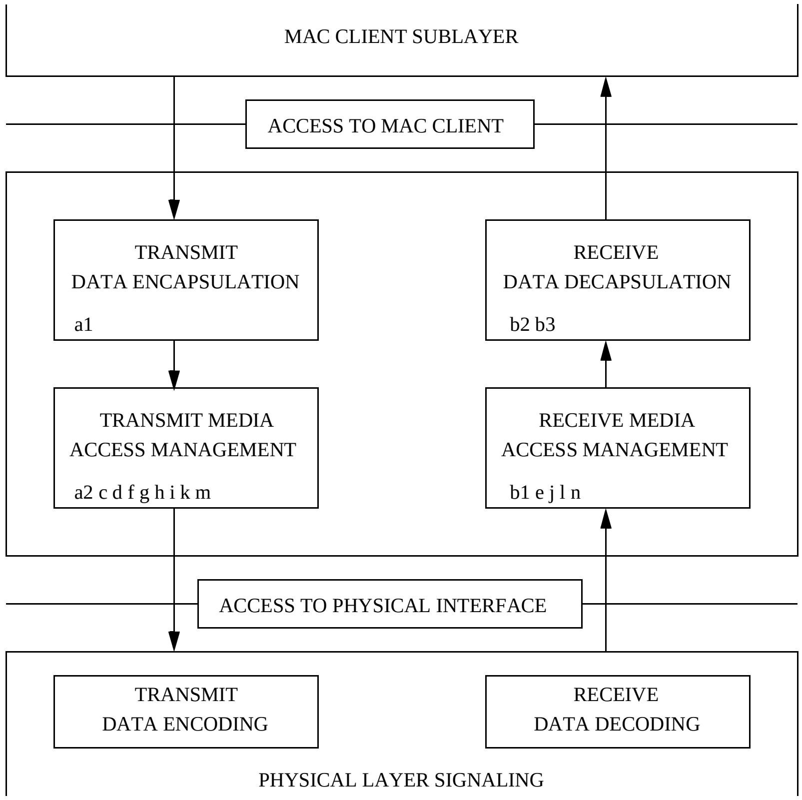
~~At the receiving end, the bits resulting from a collision are received and decoded by the PLS just as are the bits of a valid frame. Fragmentary frames received during collisions are distinguished from valid transmissions by the MAC sublayer's Receive Media Access Management component.~~

99.1.4 Relationships to the MAC client and ~~Physical Layers~~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.

99.1.5 ~~CSMA/CD access~~ Access method functional capabilities

The following summary of the functional capabilities of the ~~CSMA/CD~~ MAC sublayer is intended as a quick reference guide to the capabilities of the standard, as shown in Figure 99–2:



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

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) In half duplex mode, defers transmission of a bit-serial stream whenever the physical medium is busy.~~

d) Appends proper FCS value to outgoing frames and verifies full octet boundary alignment.

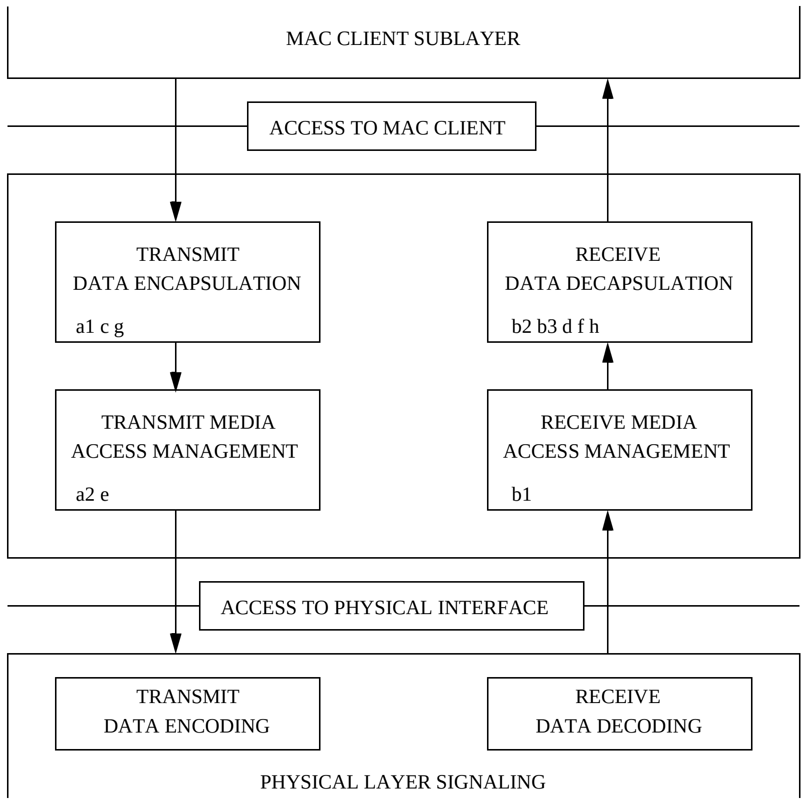
e) Checks incoming frames for transmission errors by way of FCS and verifies octet boundary alignment

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

~~g) In half duplex mode, halts transmission when collision is detected.~~

~~h) In half duplex mode, schedules retransmission after a collision until a specified retry limit is reached.~~

~~i) In half duplex mode, enforces collision to ensure propagation throughout network by sending jam message.~~



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

Figure 99–2—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~~ access control (MAC) method: ~~Precise~~ 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.

99.2.2 Overview of the procedural model

The functions of the ~~CSMA/CD~~ MAC method are presented below, modeled as a program written in the computer language Pascal. This procedural model is intended as the primary specification of the functions to be provided in any ~~CSMA/CD~~ MAC sublayer implementation. It is important to distinguish, however, between the model and a real implementation. The model is optimized for simplicity and clarity of presentation, while any realistic implementation shall place heavier emphasis on such constraints as efficiency and suitability to a particular implementation technology or computer architecture. In this context, several important properties of the procedural model shall be considered.

99.2.2.1 Ground rules for the procedural model

a) First, it shall be emphasized that *the description of the MAC sublayer in a computer language is in no way intended to imply that procedures shall be implemented as a program executed by a computer.* The implementation may consist of any appropriate technology including hardware, firmware, software, or any combination.
b) Similarly, it shall be emphasized that it is the behavior of any MAC sublayer implementations that shall match the standard, not their internal structure. The internal details of the procedural model are useful only to the extent that they help specify that behavior clearly and precisely.
c) The handling of incoming and outgoing frames is rather stylized in the procedural model, in the sense that frames are handled as single entities by most of the MAC sublayer and are only serialized for presentation to the Physical Layer. In reality, many implementations will instead handle frames serially on a bit, octet or word basis. This approach has not been reflected in the procedural model, since this only complicates the description of the functions without changing them in any way.
d) The model consists of algorithms designed to be executed by a number of concurrent processes; these algorithms collectively implement the ~~CSMA/CD~~ MAC procedure. The timing dependencies introduced by the need for concurrent activity are resolved in two ways:
    1) *Processes Versus External Events.* It is assumed that the algorithms are executed "very fast" relative to external events, in the sense that a process never falls behind in its work and fails to respond to an external event in a timely manner. For example, when a frame is to be received, it is assumed that the Media Access procedure ReceiveFrame is always called well before the frame in question has started to arrive.
    2) *Processes Versus Processes.* Among processes, no assumptions are made about relative speeds of execution. This means that each interaction between two processes shall be structured to work correctly independent of their respective speeds. Note, however, that the timing of interactions among processes is often, in part, an indirect reflection of the timing of external events, in which case appropriate timing assumptions may still be made.

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

99.2.2.2 Use of ~~Pascal~~ 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 ~~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:

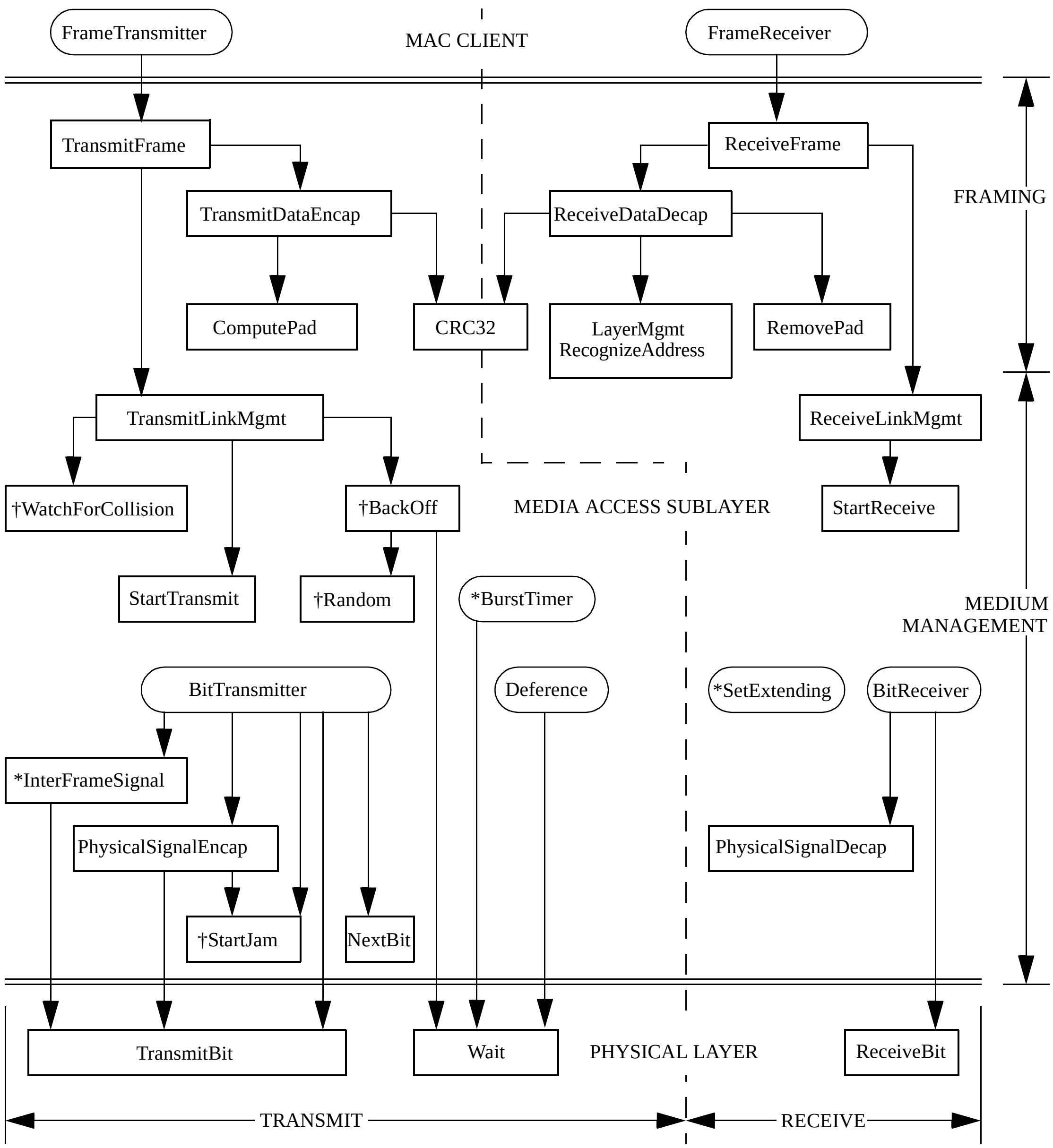
a) Frame Transmitter process
b) Frame Receiver process
c) Bit Transmitter process
d) Bit Receiver process
e) Deference process
~~f) BurstTimer process~~
~~g) SetExtending process~~

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, 99–5b, 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 99–5b 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



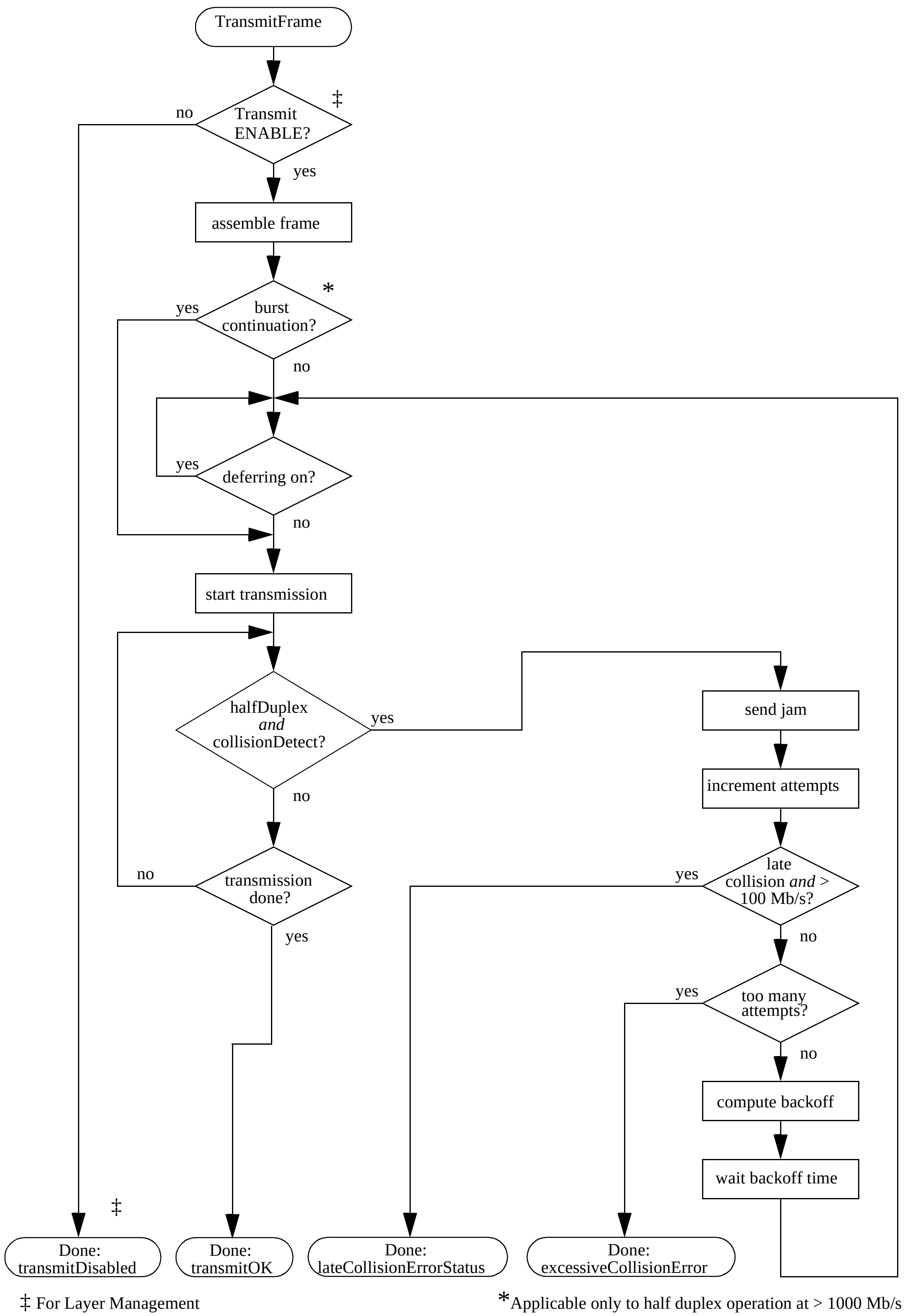
† Not applicable to full duplex operation.
* Applicable only to half duplex operation at 1000 Mb/s.

Figure 99–2—Relationship among CSMA/CD procedures

only required if Layer Management is being implemented. These changes do not affect any aspect of the MAC behavior as observed at the LLC-MAC and MAC-PLS interfaces of ISO/IEC 8802-3: 1990.

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

The Layer Management facilities provided by the ~~CSMA/CD~~ MAC and Physical Layer management definitions provide the ability to manipulate management counters and initiate actions within the layers. The



a) TransmitFrame

Figure 99–3a—Control flow summary

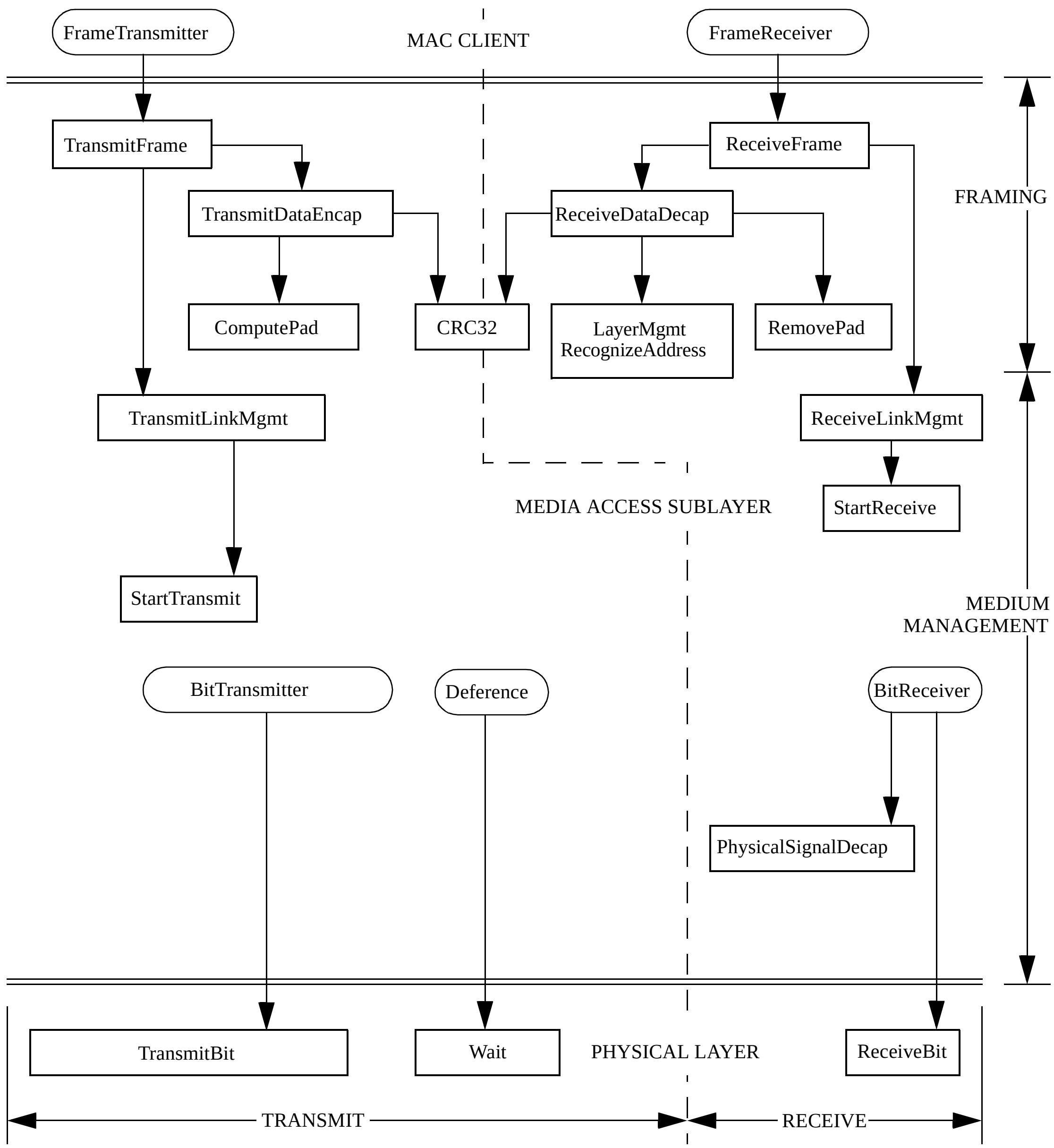


Figure 99–4—Relationship among CSMA/CD procedures

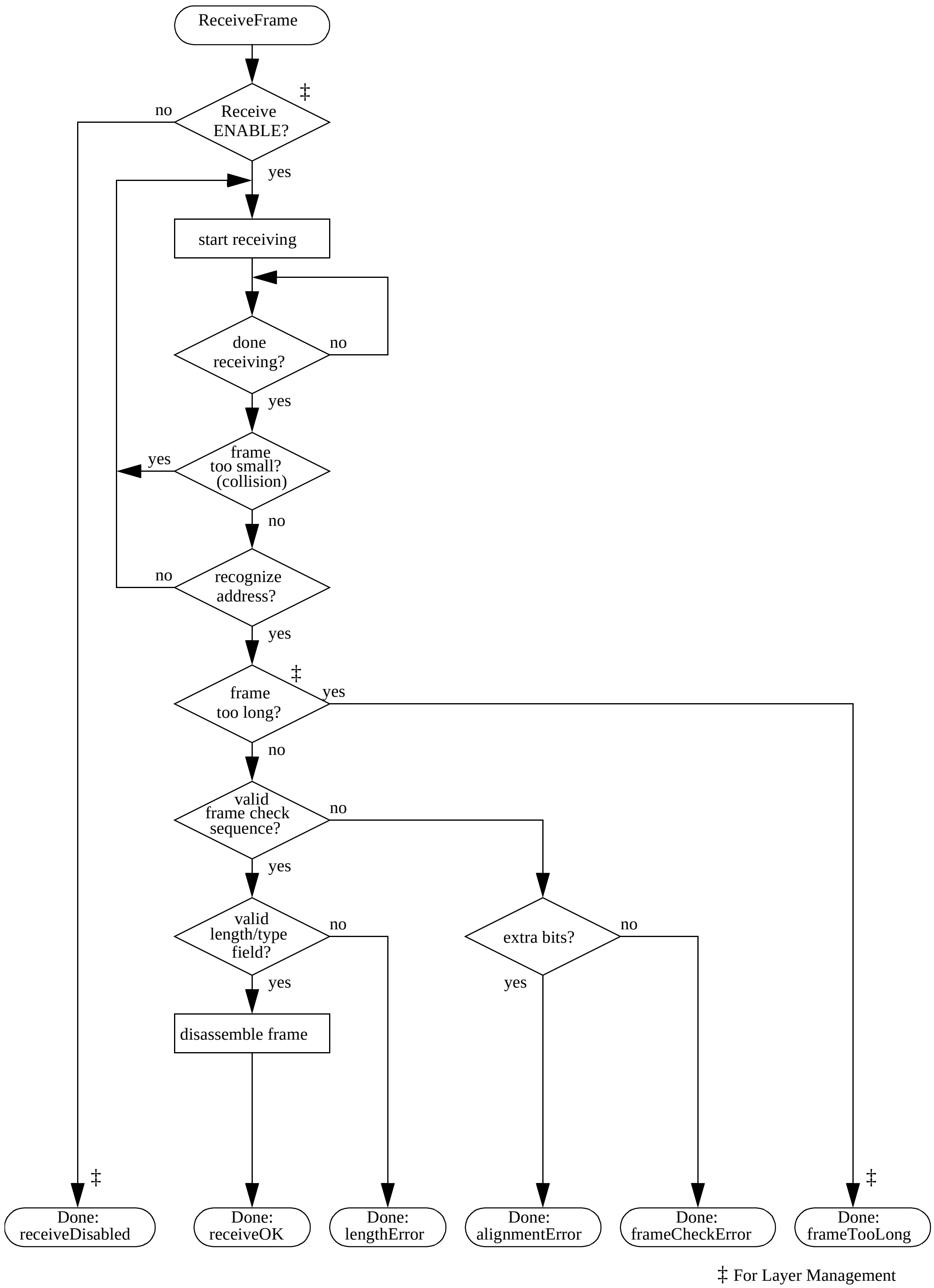
managed objects within this standard are defined as sets of attributes, actions, notifications, and behaviors in accordance with IEEE Std 802.1F-1993, and ISO/IEC International Standards for network management.

99.2.3 Frame transmission model

Frame transmission includes data encapsulation and Media Access management aspects:

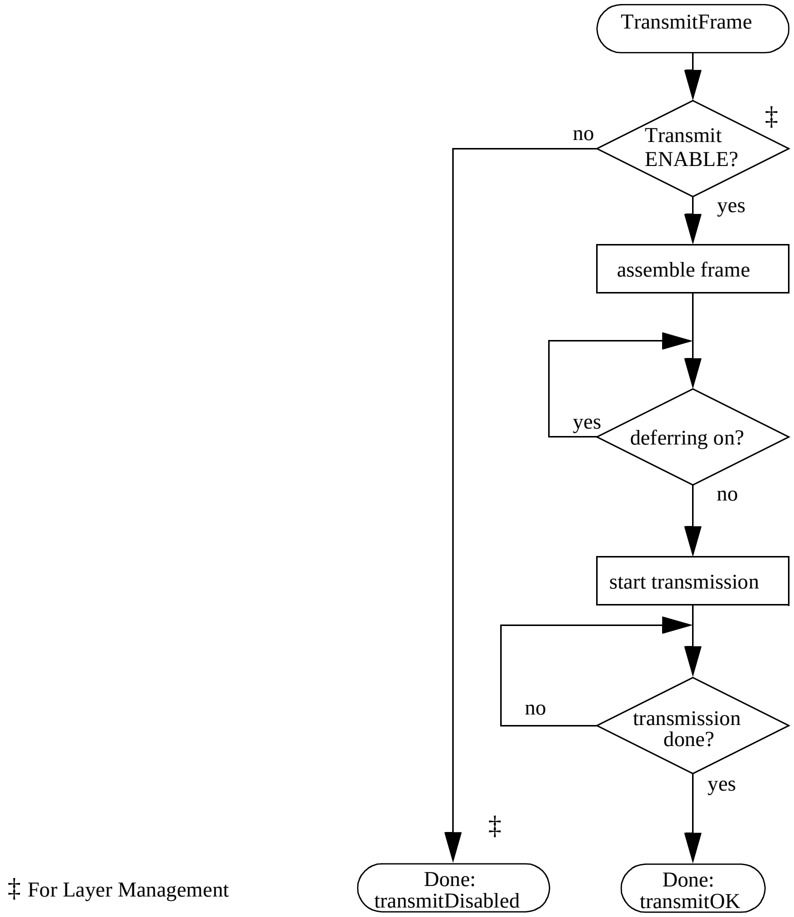
a) Transmit Data Encapsulation includes the assembly of the outgoing frame (from the values provided by the MAC client) and frame check sequence generation.

b) Transmit Media Access Management includes ~~carrier deference,~~ interframe ~~spacing, collision detection~~ spacing and ~~enforcement, collision backoff and retransmission, carrier extension and frame bursting~~bit transmission.



b) ReceiveFrame

Figure 99–4a—Control flow summary



a) TransmitFrame

Figure 99–5a—Control flow summary

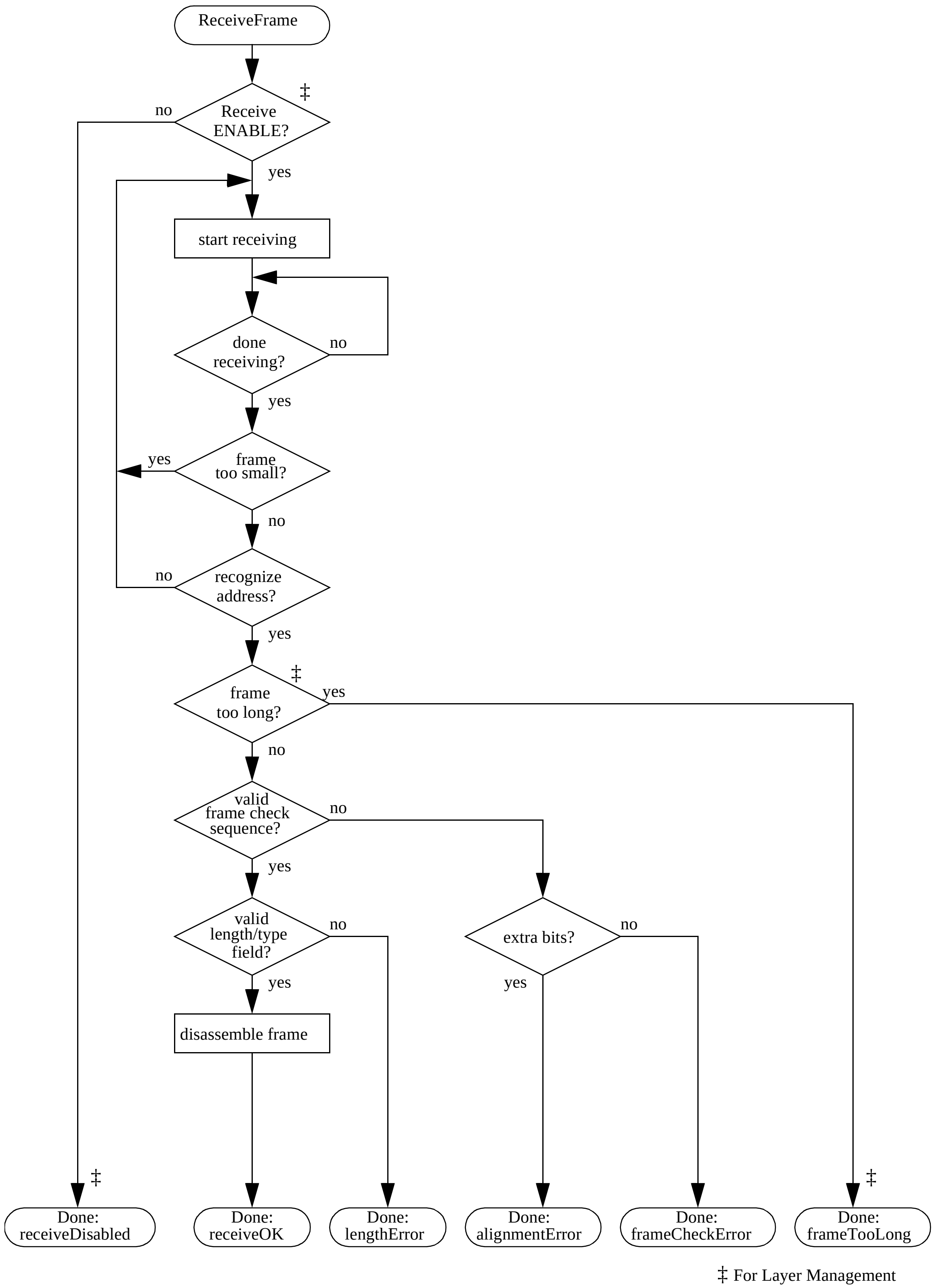
99.2.3.1 Transmit data encapsulation

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 99–5b—Control flow summary

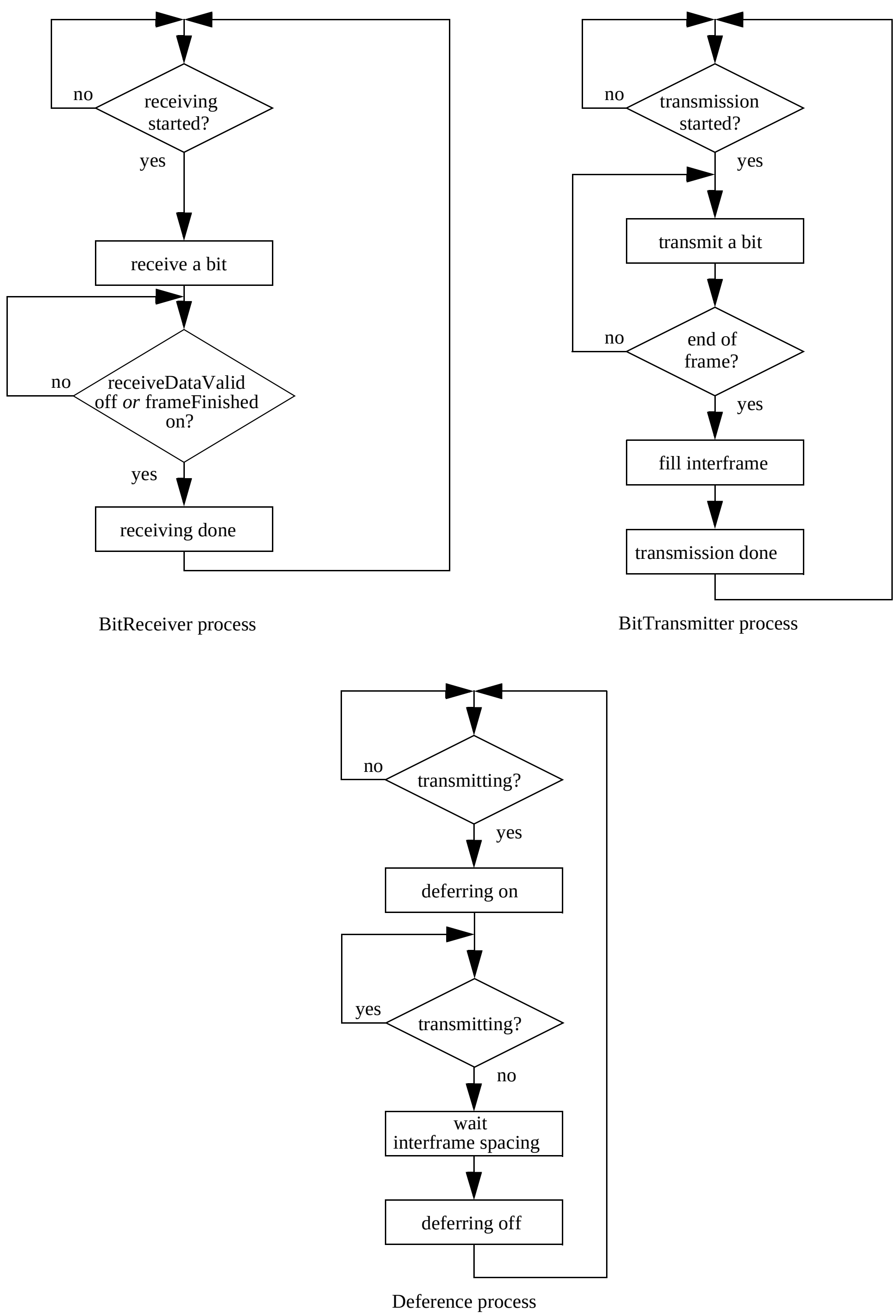
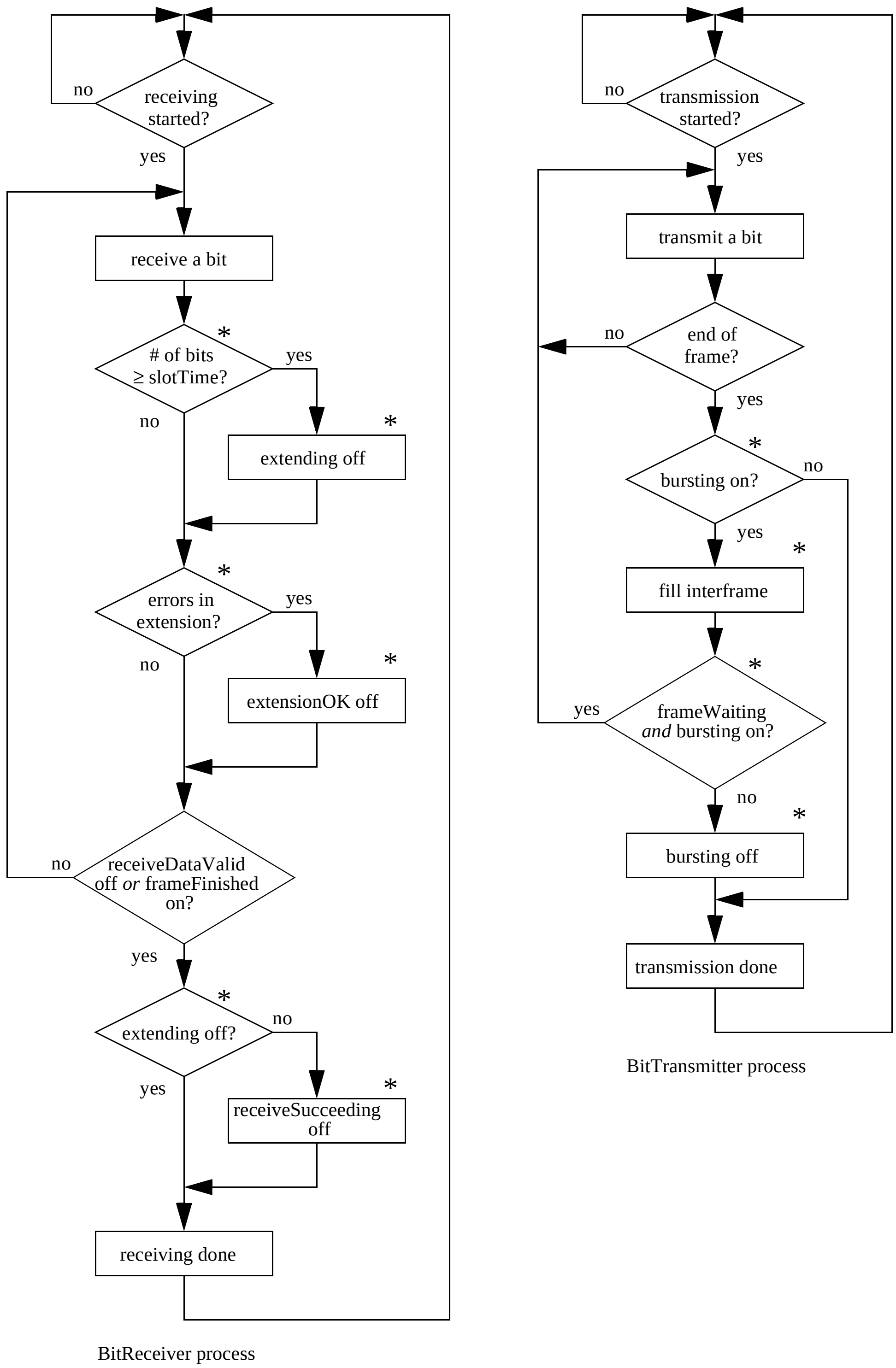


Figure 99–6—Control flow

~~a) *Half duplex mode*~~

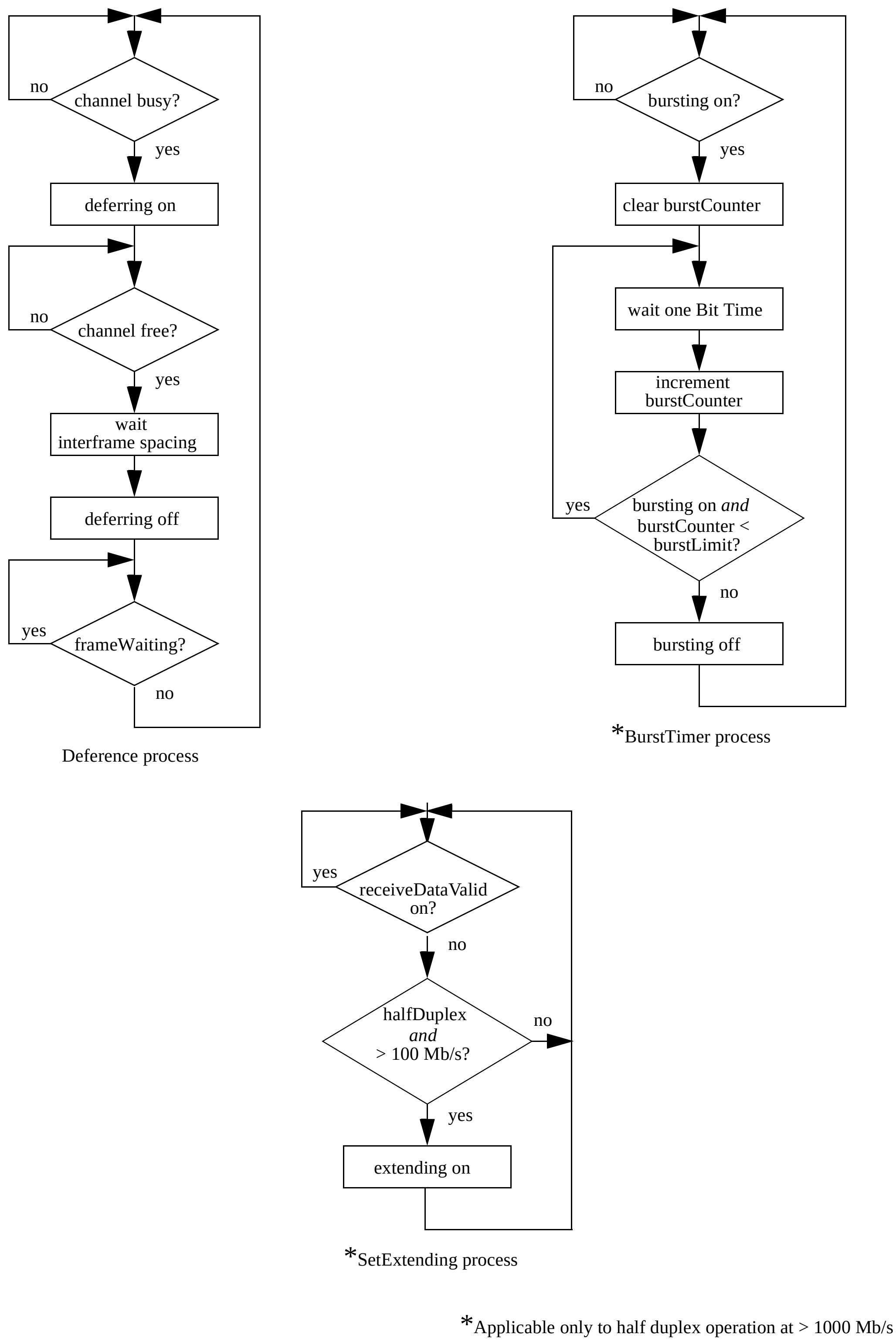
~~Even when it has nothing to transmit, the CSMA/CD MAC sublayer monitors the physical medium for traffic by watching the carrierSense signal provided by the PLS. Whenever the medium is busy, the CSMA/CD MAC defers to the passing frame by delaying any pending transmission of its own. After the last bit of the passing frame (that is, when *carrierSense* changes from true to false), the CSMA/CD MAC continues to defer for a proper interFrameSpacing (see 99.2.3.2.2).~~



*Applicable only to half duplex operation at > 1000 Mb/s

a) MAC sublayer

Figure 99–7a—Control flow



b) MAC sublayer

Figure 99–7b—Control flow

~~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,~~ When a frame is submitted by the ~~CSMA/CD~~ MAC ~~does not defer pending transmissions based on~~ client for transmission, the ~~carrierSense signal from~~ transmission is initiated as soon as possible, but in conformance with the ~~PLS~~following rule. ~~Instead, it~~ The MAC uses the internal variable *transmitting* to maintain proper MAC state while ~~the~~ a 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.~~
~~b) It is an upper bound on the length of a frame fragment generated by a collision.~~
~~c) It is the scheduling quantum for retransmission.~~

~~To fulfill all three functions, the slot time shall be larger than the sum of the Physical Layer round-trip propagation time and the Media Access Layer maximum jam time. The slot time is determined by the parameters of the implementation, see 99.4.~~

~~99.2.3.2.4 Collision detection and enforcement (half duplex mode only)~~

~~Collisions are detected by monitoring the collisionDetect signal provided by the Physical Layer. When a collision is detected during a frame transmission, the transmission is not terminated immediately. Instead, the transmission continues until additional bits specified by jamSize have been transmitted (counting from the time collisionDetect went on). This collision enforcement or jam guarantees that the duration of the collision is sufficient to ensure its detection by all transmitting stations on the network. The content of the jam is unspecified; it may be any fixed or variable pattern convenient to the Media Access implementation; however, the implementation shall not be intentionally designed to be the 32-bit CRC value corresponding to the (partial) frame transmitted prior to the jam.~~

~~99.2.3.2.5 Collision backoff and retransmission (half duplex mode only)~~

~~When a transmission attempt has terminated due to a collision, it is retried by the transmitting CSMA/CD sublayer until either it is successful or a maximum number of attempts (attemptLimit) have been made and all have terminated due to collisions. Note that all attempts to transmit a given frame are completed before any subsequent outgoing frames are transmitted. The scheduling of the retransmissions is determined by a controlled randomization process called "truncated binary exponential backoff." At the end of enforcing a collision (jamming), the CSMA/CD sublayer delays before attempting to retransmit the frame. The delay is an integer multiple of slotTime. The number of slot times to delay before the nth retransmission attempt is chosen as a uniformly distributed random integer r in the range:~~

$$0 \le r < 2^k$$

~~where~~

$$k = \min(n, 10)$$

~~If all attemptLimit attempts fail, this event is reported as an error. Algorithms used to generate the integer r should be designed to minimize the correlation between the numbers generated by any two stations at any given time.~~

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

~~99.2.3.2.6 Full duplex transmission~~

~~In full duplex mode, there is never contention for a shared physical medium. The Physical Layer may indicate to the MAC that there are simultaneous transmissions by both stations, but since these transmissions do not interfere with each other, a MAC operating in full duplex mode must not react to such Physical Layer indications. Full duplex stations do not defer to received traffic, nor abort transmission, jam, backoff, and reschedule transmissions as part of Transmit Media Access Management. Transmissions may be initiated whenever the station has a frame queued, subject only to the interframe spacing required to allow recovery for other sublayers and for the physical 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.

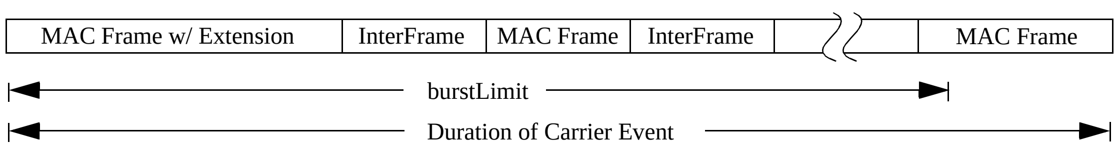


Figure 99–5—Frame bursting

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.

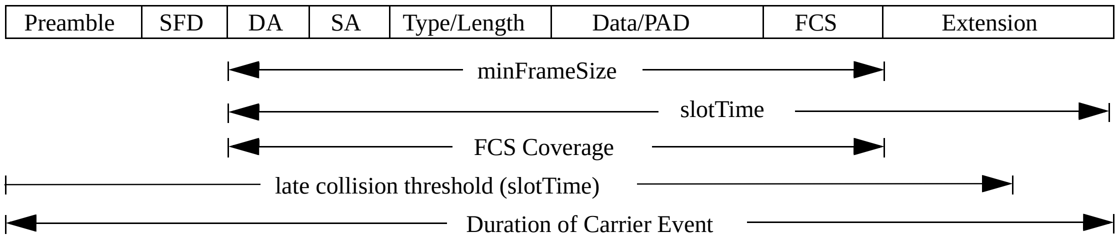


Figure 99–6—Frame with carrier extension

99.2.3.4.1 Transmission

Transmissions may be initiated whenever the station has a frame queued, subject only to the interframe spacing required to allow recovery for the physical medium.

99.2.4 Frame reception model

~~CSMA/CD~~ The MAC sublayer frame reception includes both data decapsulation and Media Access management aspects:

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

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

99.2.4.1 Receive data decapsulation

99.2.4.1.1 Address recognition

The ~~CSMA/CD~~ MAC sublayer is capable of recognizing individual and group addresses.

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

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

The ~~CSMA/CD~~ MAC sublayer is capable of activating some number of group addresses as specified by higher layers. The ~~CSMA/CD~~ 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 ~~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~~

~~Frame Delimiter. If, while transmitting~~ the ~~preamble or~~ Start Frame Delimiter~~, the collision detect variable becomes true, any remaining preamble and Start Frame Delimiter bits shall be sent~~. 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."

99.2.6 Start frame sequence

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

99.2.7 Global declarations

This subclause provides detailed formal specifications for the ~~CSMA/CD~~ MAC sublayer. It is a specification of generic features and parameters to be used in systems implementing this media access method. Subclause 99.4 provides values for these sets of parameters for recommended implementations of this media access mechanism.

99.2.7.1 Common constants, types, and variables

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

const

addressSize = 48; {In bits, in compliance with 3.2.3}
lengthOrTypeSize = 16; {In bits}
clientDataSize = ...; {In bits, size of MAC client data; see ~~4.2.2.2~~99.2.2.2, ~~a~~a) ~~3~~3)}
padSize = ...; {~~iIn~~ In bits, = max (0, minFrameSize − (2 x addressSize + lengthOrTypeSize + clientDataSize + crcSize))}
dataSize = ...; {In bits, = clientDataSize + padSize}
crcSize = 32; {In bits, 32-bit CRC}
frameSize = ...; {In bits, = 2 x addressSize + lengthOrTypeSize + dataSize + crcSize; see 4.2.2.2, a)}
~~minFrameSize = ... ; {In bits, implementation-dependent, see 4.4}~~
~~qTagPrefixSize~~ minFrameSize = ~~4~~...; {In ~~octets~~bits, ~~length of QTag Prefix~~implementation-dependent, see ~~3~~4.~~5~~4}
~~extend~~ maxUntaggedFrameSize = ...~~-~~; {~~Boolean~~In octets, ~~true if (slotTime — minFrameSize) > 0~~implementation-dependent, ~~false otherwise~~see 4.4}
~~extensionBit~~ qTagPrefixSize = ~~...~~4; {~~A nondata value which is used for carrier extension and interframe during bursts~~In octets, length of QTag Prefix, see 3.5}
~~extensionErrorBit = ...; {A nondata value which is used to jam during carrier extension}~~
minTypeValue = 1536; {Minimum value of the Length/Type field for Type interpretation}
maxValidFrame = maxUntaggedFrameSize − (2 x addressSize + lengthOrTypeSize + crcSize) / 8; {In octets, the maximum length of the MAC client data field. This constant is defined for editorial convenience, as a function of other constants}
~~slotTime = ... ; {In bit times, unit of time for collision handling, implementation-dependent, see 4.4}~~
preambleSize = 56; {In bits, see 4.2.5}
sfdSize = 8; {In bits, start frame delimiter}
headerSize = 64; {In bits, sum of preambleSize and sfdSize}

type

~~PhysicalBit~~ Bit = (0, 1~~, extensionBit, extensionErrorBit~~);
~~{Bits transmitted to the Physical Layer can be either 0, 1, extensionBit or extensionErrorBit.~~ PhysicalBit = (0, 1); {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 divided by the bit period~~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}
~~attemptLimit = ...; {Max number of times to attempt transmission}~~
~~backOffLimit = ...; {Limit on number of times to back off}~~
~~burstLimit= ...; {In bits, limit for initiation of frame transmission in Burst Mode, implementation dependent, see 4.4}~~
~~jamSize = ...; {In bits, the value depends upon medium and collision detect implementation}~~

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;
~~frameWaiting~~deferring: Boolean; {~~Indicates that outgoingFrame is deferring~~Implies any pending transmission must wait}
~~attempts~~frameWaiting: ~~0..attemptLimit~~Boolean; {~~Number of transmission attempts on outgoingFrame~~Indicates that outgoingFrame is deferring}
~~newCollision~~ifsStretchMode: Boolean; {Indicates ~~that a collision has occurred but has not yet been jammed}~~the desired mode of operation, and enables the lowering of the
~~transmitSucceeding: Boolean; {Running indicator of whether transmission is succeeding}~~
~~burstMode: Boolean; {Indicates the desired mode of operation, and enables the transmission of multiple frames in a single carrier event. burstMode is a static variable; its value shall only be changed by the invocation of the Initialize procedure}~~
~~bursting: Boolean; {In burstMode, the given station has acquired the medium and the burst timer has not yet expired}~~
~~burstStart: Boolean; {In burstMode, indicates that the first frame transmission is in progress}~~
~~extendError: Boolean; {Indicates a collision occurred while sending extension bits}~~
~~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}

99.2.7.3 Receive state variables

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

var

incomingFrame: Frame; {The frame being received}
receiving: Boolean; {Indicates that a frame reception is in progress}
excessBits: 0..7; {Count of excess trailing bits beyond octet boundary}
receiveSucceeding: Boolean; {Running indicator of whether reception is succeeding}
validLength: Boolean; {Indicator of whether received frame has a length error}
exceedsMaxLength: Boolean; {Indicator of whether received frame has a length longer than the

maximum permitted length}

~~extending~~passReceiveFCSMode: Boolean; {Indicates ~~whether~~ the ~~current frame is subject to carrier extension}~~desired mode of operation, and enables passing of

~~extensionOK: Boolean; {Indicates whether any bit errors were found in the extension part of a frame, 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);

~~TransmitStatus = (transmitDisabled, transmitOK, excessiveCollisionError, lateCollisionErrorStatus);~~

{Result of TransmitFrame operation~~, reporting of lateCollisionErrorStatus is~~} ~~optional for MACs operating at speeds at or below 100Mb/s}~~

ReceiveStatus = (receiveDisabled, receiveOK, frameTooLong, frameCheckError, lengthError, ~~alignmentError); {Result of ReceiveFrame operation}~~ ~~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}

99.2.7.5 State variable initialization

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

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

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

procedure Initialize;
begin
~~frameWaiting := false;~~
~~deferring := false;~~
frameWaiting: Boolean; {Indicates that outgoingFrame is deferring}
~~newCollision~~ deferring := false;
transmitting := false; {An interface to Physical Layer; see below}
receiving := false;
~~halfDuplex~~ passReceiveFCSMode := ...; {True ~~for half duplex operation, false for full duplex operation. For operation at~~when enabling the passing of the frame check sequence of all
~~bursting := false;~~
~~burstMode := ...; { True for half duplex operation at an operating speed of 1000 Mb/s, when multiple frames' transmission in a single carrier event~~ received frames from the MAC sublayer to the MAC client is desired and supported, false otherwise~~} between invocations of the Initialize procedure~~}
~~extending := extend *and* halfDuplex;~~
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;
~~passReceiveFCSMode~~ p2mpMode := ...; {True ~~when enabling the passing of the frame check sequence of all~~for Point-to-Multi-Point implementations, false otherwise}
~~received frames from the MAC sublayer to the MAC client is desired and supported, false otherwise}~~
~~*if* halfDuplex *then while* carrierSense *or* receiveDataValid *do* nothing~~
~~*else*~~ *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:

```
procedure TransmitDataEncap;
begin
    with outgoingFrame do
        begin {Assemble frame}
            view := fields;
            destinationField := destinationParam;
            sourceField := sourceParam;
            lengthOrTypeField := lengthOrTypeParam;
            if fcsParamPresent then
                begin
                    dataField := dataParam; {No need to generate pad if the FCS is passed from MAC client}
                    fcsField := fcsParamValue {Use the FCS passed from MAC client}
                end
            else
                begin
                    dataField := ComputePad(dataParam);
                    fcsField := CRC32(outgoingFrame)
                end;
            view := bits
        end {Assemble frame}
    with outgoingHeader do
        begin
            headerView := headerFields;
            preamble := ...; {* ‘1010...10,’ LSB to MSB*}
            sfd := ...; {* ‘10101011,’ LSB to MSB*}
            headerView := headerBits
        end
end; {TransmitDataEncap}
```

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

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

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

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

~~TransmitLinkMgmt attempts to transmit the frame. In half duplex mode, it first defers to any passing traffic. In half duplex mode, if a collision occurs, transmission is terminated properly and retransmission is scheduled following a suitable backoff interval:~~

~~*function* TransmitLinkMgmt: TransmitStatus;~~
~~begin~~
~~attempts := 0;~~
~~transmitSucceeding := false;~~
~~lateCollisionCount := 0;~~
~~deferred := false; {Initialize}~~

TransmitLinkMgmt attempts to transmit the frame. It first defers to ensure proper interframe spacing:

~~excessDefer := false~~*function* TransmitLinkMgmt: TransmitStatus;
~~*while* (attempts < attemptLimit) *and* (*not* transmitSucceeding)~~
~~*and* (*not* extend *or* lateCollisionCount = 0) *do*~~
~~{No retransmission after late collision if operating at 1000 Mb/s}~~
~~*begin* {Loop}~~
~~*if* bursting *then* {This is a burst continuation}~~
~~frameWaiting := true {Start transmission without checking deference}~~
~~*else* {Non bursting case, or first frame of a burst}~~
begin
~~*if* attempts > 0 *then* BackOff;~~
frameWaiting := true;
if not p2mpMode then *while* deferring *do* nothing {Defer to ~~passing frame, if any~~[1] ensure proper interframe spacing}
~~*if* halfDuplex *then* deferred := true;~~
~~nothing;~~
‡ ~~*if* halfDuplex *then* deferred := true~~
~~*end*~~StartTransmit;
~~burstStart~~ frameWaiting := ~~true~~false;
~~*if* burstMode *then* bursting := true~~
~~*end*;~~
~~lateCollisionError := false;~~
~~StartTransmit;~~
~~frameWaiting := false;~~
~~*if* halfDuplex *then*~~
~~*begin*~~
~~*while* transmitting *do* WatchForCollision;~~

~~[1] The Deference process ensures that the reception of traffic does not cause deferring to be true when in full duplex mode. Deferring is used in full duplex mode to enforce the minimum interpacket gap spacing.~~

~~*if* lateCollisionError *then* lateCollisionCount := lateCollisionCount + 1;~~
~~attempts := attempts + 1~~
~~*end* {Half 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*~~ TransmitLinkMgmt := ~~excessiveCollisionError~~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;~~
~~transmitSucceeding := true;~~
~~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;~~
~~*if* burstMode *then*~~
~~*begin*~~
~~bursting := false;~~
~~*if not* burstStart *then*~~
~~lateCollisionError := true {Every collision is late, unless it hits the first frame in a burst}~~
~~*end*~~
~~*end*~~
~~*end*; {WatchForCollision}~~

WatchForCollision, upon detecting a collision, updates newCollision to ensure proper jamming by the BitTransmitter process. The current transmit bit number is checked to see if this is a late collision. If the collision occurs later than a collision window of slotTime bits into the packet, it is considered as evidence of a late collision. The point at which the collision is received is determined by the network media propagation time and the delay time through a station and, as such, is implementation-dependent (see 4.1.2.2). While operating at speeds of 100 Mb/s or lower, an implementation may optionally elect to end retransmission attempts after a late collision is detected. While operating at the speed of 1000 Mb/s, an implementation shall end retransmission attempts after a late collision is detected.

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

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

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

```
var maxBackOff: 2..1024; {Working variable of BackOff}
procedure BackOff;
begin
    if attempts = 1 then maxBackOff := 2
    else if attempts ≤ backOffLimit then maxBackOff := maxBackOff x 2;
    Wait(slotTime x Random(0, maxBackOff))
end; {BackOff}
```

BurstTimer is a process that does nothing unless the bursting variable is true. When bursting is true, BurstTimer increments burstCounter until the burstLimit limit is reached, whereupon BurstTimer assigns the value false to bursting:

```
    var burstCounter: integer;
begin
    cycle
        while not bursting do nothing; {wait for a burst}
        burstCounter := 0;
        while bursting and (burstCounter < burstLimit) do
            begin
                Wait(1);
                burstCounter := burstCounter + 1
            end;
        bursting := false
    end {burstMode cycle}
end; {BurstTimer}

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 Deferenceprocedure StartTransmit;
   var realTimeCounter: integer; wasTransmitting: Boolean;
begin
   if halfDuplex then cycle{Half duplex loop}
      while not carrierSense do nothing; {Watch for carrier to appear}
      deferring := true;  {Delay start of new 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
            StartRealTimeDelaycurrentTransmitBit := 1;
            repeat
            while carrierSense do StartRealTimeDelay
            until not RealTimeDelay(interFrameSpacingPart1)
            realTimeCounter := interFrameSpacingPart1;
            repeat
               while carrierSense do realTimeCounter lastTransmitBit :=
interFrameSpacingPart1frameSize;
               Wait(1)transmitting := true;
               realTimeCounter lastHeaderBit:= realTimeCounter – 1headerSize
            until (realTimeCounter = 0)
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:

```
         endprocess Deference;
      Wait(interFrameSpacingPart2); {Time out second part of interframe gap}
      deferring := false; Allow new transmissions to proceed}
      while frameWaiting do nothing {Allow waiting transmission, if any}
   end {Half duplex loop}
begin
   else cycle {Full duplex Main 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);
            ifsStretchCount := 0
```

```
            end
        deferring := false {Don't inhibit transmission}
    end {~~Full duplex ~~Main loop}
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 bit times is reflected by the variable ifsStretchSize. If the variable ifsStretchCount is less than ifsStretchRatio and the next frame is ready for transmission (variable frameWaiting is true), the Deference process enforces interframe spacing only for the integer number of octets, as indicated by ifsStretchSize, and saves ifsStretchCount for the next frame's transmission. If the next frame is not ready for transmission (variable frameWaiting is false), then the Deference process initializes the ifsStretchCount variable to zero.

```
    ~~begin~~
        ~~{reset the realtime timer and start it timing}~~
    ~~end; {StartRealTimeDelay}~~

~~function RealTimeDelay (μsec:real): Boolean;~~
    ~~begin~~
        ~~{return the value true if the specified number of microseconds have~~
        ~~not elapsed since the most recent invocation of StartRealTimeDelay,~~
        ~~otherwise return the value false}~~
    ~~end; {RealTimeDelay}~~
```

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

```
process BitTransmitter;
begin
    cycle {Outer loop}
            ~~begin {Inner loop}~~if transmitting then
                ~~extendError := false;~~begin {Inner loop}
                if ifsStretchMode then {Calculate the counter values}
                    begin
                        ifsStretchSize := (ifsStretchCount + headerSize + frameSize + interFrameSpacing) div
                                                    ifsStretchRatio; {Extension of the interframe spacing}
                        ifsStretchCount := (ifsStretchCount + headerSize + frameSize + interFrameSpacing)
                                mod ifsStretchRatio {Remainder to carry over into the next frame's transmission}
                    ~~end;~~
                ~~PhysicalSignalEncap; {Send preamble and start of frame delimiter}~~end;
                while transmitting do
                    begin
                        ~~if (currentTransmitBit > lastTransmitBit) then ~~TransmitBit(~~extensionBit~~outgoingFrame[currentTransmitBit]);
                        ~~else if extendError then TransmitBit(extensionErrorBit) {Jam in extension}~~currentTransmitBit := currentTransmitBit + 1;
                                ~~TransmitBit~~transmitting := (~~extensionErrorBit) {jam in extension}~~currentTransmitBit ≤ lastTransmitBit)
                        ~~else TransmitBit(outgoingFrame[currentTransmitBit])~~end;
                        ~~if newCollision then StartJam else NextBit~~end {Inner loop}
    end {Outer loop}
end; {BitTransmitter}
```

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;~~

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.

~~*var* interFrameCount, interFrameTotal~~*function* ReceiveDataDecap: ~~integer~~ReceiveStatus;
‡ *var* status: ReceiveStatus; {Holds receive status information}
begin
‡ *with* incomingFrame *do*
‡ *begin*
‡ ~~interFrameCount~~ view := ~~0~~fields;
‡ receiveSucceeding := LayerMgmtRecognizeAddress(destinationField);
~~interFrameTotal := interFrameSpacing;~~*if* receiveSucceeding *then*
~~*while* interFrameCount < interFrameTotal *do*~~
~~*begin*~~
begin {Disassemble frame}
destinationParam := destinationField;
~~*if not* extendError *then* TransmitBit(extensionBit)~~sourceParam := sourceField;
~~*else* TransmitBit(extensionErrorBit)~~lengthOrTypeParam := lengthOrTypeField;
dataParam := RemovePad(lengthOrTypeField, dataField);
~~interFrameCount~~ fcsParamValue := ~~interFrameCount + 1~~fcsField;
~~*if* collisionDetect *and not* extendError *then*~~fcsParamPresent := passReceiveFCSMode;
~~*begin*~~
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.}
~~bursting~~ *if* exceedsMaxLength *then* status := ~~false;~~frameTooLong
else if fcsField = CRC32(incomingFrame) *then*
‡ *if* validLength *then* status := receiveOK *else* status := lengthError
‡ *else if* excessBits = 0 *then* status := frameCheckError
‡ ~~extendError~~ *else* status := ~~true~~alignmentError;
‡ LayerMgmtReceiveCounters(status); {Update receive counters in 5.2.4.3}
~~interFrameCount~~ view := ~~0;~~bits
~~interFrameTotal := jamSize~~*end* {Disassemble frame}
‡ *end*; {With incomingFrame}
‡ ~~interFrameTotal~~ ReceiveDataDecap := ~~jamSize~~status
~~*end*~~
end; {~~InterFrameSignal~~ReceiveDataDecap}

~~*procedure* NextBit;~~
~~*begin*~~
~~currentTransmitBit := currentTransmitBit + 1;~~
~~*if* halfDuplex *and* burstStart *and* transmitSucceeding *then* {Carrier extension may be required}~~
~~transmitting := (currentTransmitBit ≤ max(lastTransmitBit, slotTime))~~
~~*else* transmitting := (currentTransmitBit ≤ lastTransmitBit)~~
~~*end*; {NextBit}~~
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}

~~*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

```
        until receiveSucceedingbegin
            validLength := {For length interpretations of the Length/Type field, check to determine if value
                        represented by Length/Type field matches the received clientDataSize};
            if validLength and not passReceiveFCSMode then
                RemovePad := {Truncate the dataParam (when present) to the value represented by the
                        lengthOrTypeParam (in octets) and return the result}
            else RemovePad := dataParam
        else ReceiveFrame := receiveDisabledend
    end; {ReceiveFrameRemovePad}
```

~~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.~~

ReceiveLinkMgmt attempts repeatedly to receive the bits of a frame, discarding any fragments smaller than the minimum valid frame size:

```
    function ReceiveDataDecap: ReceiveStatusprocedure ReceiveLinkMgmt;
ǂ       var status: ReceiveStatus; {Holds receive status information}
    begin
ǂ       with incomingFrame dorepeat
ǂ           beginStartReceive;
            while receiving do nothing; {Wait for frame to finish arriving}
ǂ               view excessBits := fieldsframeSize mod 8;
            frameSize := frameSize – excessBits; {Truncate to octet boundary}
ǂ           receiveSucceeding := LayerMgmtRecognizeAddressreceiveSucceeding and (destinationField
frameSize ≥ minFrameSize);
                                                            {Reject frames too small}
                if receiveSucceeding thenuntil receiveSucceeding
                    begin end; {Disassemble frameReceiveLinkMgmt}

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

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

```
    process BitReceiver;
        var     b: PhysicalBit;
                incomingFrameSize: integer; {Count of all bits received in frame including extension}
                        destinationParam := destinationFieldframeFinished: Boolean;
                        sourceParam := sourceFieldenableBitReceiver: Boolean;
                currentReceiveBit: 1..frameSize; {Position of current bit in incomingFrame}
    begin
        cycle {Outer loop}
                        lengthOrTypeParam := lengthOrTypeField;if receiveEnabled then
                begin {Receive next frame from physical layer}
                        dataParam currentReceiveBit := RemovePad(lengthOrTypeField, dataField)1;
                        fcsParamValue incomingFrameSize := fcsField0;
                        fcsParamPresent frameFinished := passReceiveFCSModefalse;
```

```
            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,b :=
ReceiveBit; {Next bit from physical medium}
                                    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
incomingFrameSize := incomingFrameSize + 1;
                                    that the larger value be used. The use of the smaller value
                                    in this case may result in valid tagged frames exceeding theif
enableBitReceiver then {Append to frame}
                                    maximum permitted frame size.}begin
            if exceedsMaxLength then status incomingFrame[currentReceiveBit] := frameTooLongb;
            else if fcsField = CRC32(incomingFrame) and extensionOK then
‡               if validLength then status currentReceiveBit := receiveOK else status := lengthError
currentReceiveBit + 1
‡             else if excessBits = 0 or not extensionOK then status := frameCheckErrorend
‡           else status := alignmentError;end; {Inner loop}
                beginif enableBitReceiver then
‡                 if validLength then status: = receiveOKbegin
‡                 else status: frameSize := lengthErrorcurrentReceiveBit – 1;
                end
              else
                begin
‡                 if excessBits = 0 or not extensionOK then statusreceiveSucceeding := fra-
meCheckErrortrue;
‡                 else status receiving := alignmentErrorfalse
                end;
‡           LayerMgmtReceiveCounters(status); {Update receive counters in 5.2.4.3}
            view := bitsend {Enabled}
          end {Disassemble frameOuter loop}
‡     end; {With incomingFrameBitReceiver}
‡   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;
    if address = ... {One of the addresses on the multicast list and multicast reception is enabled} then
                                      LayerMgmtRecognizeAddress := true;
```

```
        LayerMgmtRecognizeAddress := false
    end; {LayerMgmtRecognizeAddress}
```

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

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

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

```
    procedure ReceiveLinkMgmt;
    begin
        repeat
            StartReceive;
            while receiving do nothing; {Wait for frame to finish arriving}
            excessBits := frameSize mod 8;
            frameSize := frameSize – excessBits; {Truncate to octet boundary}
            receiveSucceeding := receiveSucceeding and (frameSize ≥ minFrameSize)
                                            {Reject collision fragments}
        until receiveSucceeding
    end; {ReceiveLinkMgmt}

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

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

```
    process BitReceiver;
        var    b: PhysicalBit;
```

```
            incomingFrameSize: integer; {Count of all bits received in frame including extension}
            frameFinished: Boolean;
            enableBitReceiver: Boolean;
            currentReceiveBit: 1..frameSize; {Position of current bit in incomingFrame}
   begin
      cycle {Outer loop}
         if receiveEnabled then
            begin {Receive next frame from physical layer}
               currentReceiveBit := 1;
               incomingFrameSize := 0;
               frameFinished := false;
               enableBitReceiver := receiving;
               PhysicalSignalDecap; {Skip idle 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:~~

~~*process* SetExtending~~*procedure* PhysicalSignalDecap;
begin
~~*cycle* {Loop forever}~~
~~*while* receiveDataValid *do* nothing;~~
~~extending := extend~~ Receive one bit at a time from physical medium until a valid sfd is detected, discard bits and ~~halfDuplex~~
~~*end* {Loop~~return.}
end; {~~SetExtending~~PhysicalSignalDecap}

99.2.11 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.

99.3 Interfaces to/from adjacent layers

99.3.1 Overview

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

The notation used here is the Pascal language, in keeping with the procedural nature of the precise MAC sublayer specification (see 99.2). Each interface is described as a set of procedures or shared variables, or both, that collectively provide the only valid interactions between layers. The accompanying text describes the meaning of each procedure or variable and points out any implicit interactions among them.

Note that the description of the interfaces in Pascal is a notational technique, and in no way implies that they can or should be implemented in software. This point is discussed more fully in 99.2, that provides complete Pascal declarations for the data types used in the remainder of this clause. Note also that the synchronous (one frame at a time) nature of the frame transmission and reception operations is a property of the architectural interface between the MAC client and MAC sublayers, and need not be reflected in the implementation interface between a station and its sublayer.

99.3.2 Services provided by the MAC sublayer

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

Functions:

TransmitFrame

ReceiveFrame

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

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

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

The MAC client transmits a frame by invoking TransmitFrame:

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

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

~~*type* TransmitStatus = (transmitOK, excessiveCollisionError, lateCollisionErrorStatus);~~

‡ *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, lengthErrorframeTooLong, frameCheckError, alignmentError);,

‡ *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 + qTagPrefixSize, for tagged frames.

99.3.3 Services required from the physical layer

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

Function	Procedures	Variables
ReceiveBit	TransmitBit	collisionDetect
	Wait	carrierSense
		receiveDataValid
		transmitting

Function	Procedures	Variables
ReceiveBit	TransmitBit	receiveDataValid
	Wait	transmitting

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

procedure TransmitBit (bitParam: PhysicalBit);

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

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

var transmitting: Boolean*;*

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

~~The 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.~~

The Physical Layer also provides the procedure Wait:

procedure Wait (bitTimes: integer)*;*

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

~~Another important property of the Physical Layer, which is an implicit part of the interface presented to the MAC sublayer, is the round-trip propagation time of the physical medium. Its value represents the maximum time required for a signal to propagate from one end of the network to the other, and for a collision to propagate back. The round-trip propagation time is primarily (but not entirely) a function of the physical size of the network. The round-trip propagation time of the Physical Layer is defined in 99.4 for a selection of physical media.~~

99.4 Specific implementations

99.4.1 Compatibility overview

To provide total compatibility at all levels of the standard, it is required that each network component implementing the ~~CSMA/CD~~ MAC sublayer procedure adheres rigidly to these specifications. The information provided in 99.4.2 provides design parameters for specific implementations of this access method. Variations from these values result in a system implementation that violates the standard.

~~A DTE shall be capable of operating in half duplex mode, full duplex mode, or both. In any given instantiation of a network conforming to this standard, all stations shall be configured to use the same mode of operation, either half duplex or full duplex.~~

~~All DTEs connected to a repeater or a mixing segment shall be configured to use the half duplex mode of operation. When a pair of DTEs are connected to each other with a link segment, both devices shall be configured to use the same mode of operation, either half duplex or full duplex.~~

99.4.2 Allowable implementations

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

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 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.

NOTE 5—For 10 Gb/s implementations, the value of ifsStretchRatio of 104 bits adapts the average data rate of the MAC sublayer to SONET/SDH STS-192 data rate (with frame granularity), for WAN-compatible applications of this standard.

WARNING

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

~~4.4.2.1 Parameterized values~~

~~See 99.4.2.~~

~~4.4.2.2 Parameterized values~~

~~See 99.4.2.~~

~~4.4.2.3 Parameterized values~~

~~See 99.4.2.~~

~~4.4.2.4 Parameterized values~~

~~See 99.4.2.~~

~~4.4.3 Configuration guidelines~~

~~The operational mode of the MAC may be determined either by the Auto-Negotiation functions specified in Clause 28 and Clause 37, or through manual configuration. When manual configuration is used, the devices on both ends of a link segment must be configured to matching modes to ensure proper operation. When Auto-Negotiation is used, the MAC must be configured to the mode determined by Auto-Negotiation before assuming normal operation.~~

~~NOTE—Improper configuration of duplex modes may result in improper network behavior.~~