102.2.2 Multipoint transmission control, Control Parser, and Control Multiplexer

The purpose of the multipoint transmission control is to allow only one of the multiple MAC clients to transmit to its associated MAC and subsequently to the RS layer at one time by only asserting one transmitEnable signal at a time.

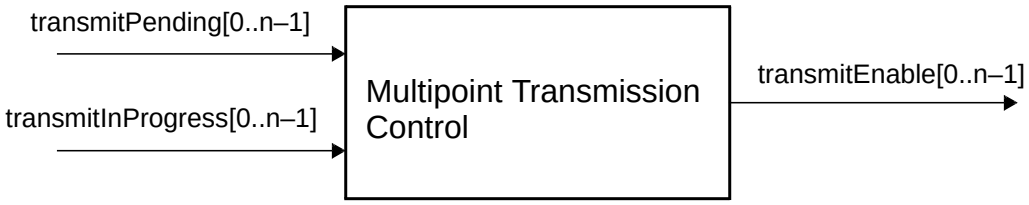


Figure 102–6—Multipoint Transmission Control service interfaces

Multipoint MAC Control Instance n function block communicates with the Multipoint Transmission Control using transmitEnable[n], transmitPending[n], and transmitInProgress[n] state variables (see Figure 102–4).

The Control Parser is responsible for opcode independent parsing of MAC frames in the reception path. By identifying MAC Control frames, demultiplexing into multiple entities for event handling is possible. Interfaces are provided to existing Clause 31 entities, functional blocks associated with MPCP, and the MAC Client.

The Control Multiplexer is responsible for forwarding frames from the MAC Control opcode-specific functions and the MAC Client to the MAC. Multiplexing is performed in the transmission direction. Given multiple MCF:MA_DATA.request primitives from the MAC Client, and MA_CONTROL.request primitives from the MAC Control Clients, a single MAC:MA_DATA.request service primitive is generated for transmission. At the CLT, multiple MAC instances share the same Multipoint MAC Control, as a result, the transmit block is enabled based on an external control signal housed in Multipoint Transmission Control for transmission overlap avoidance. At the CNU, the Gate Processing functional block interfaces for upstream transmission administration.

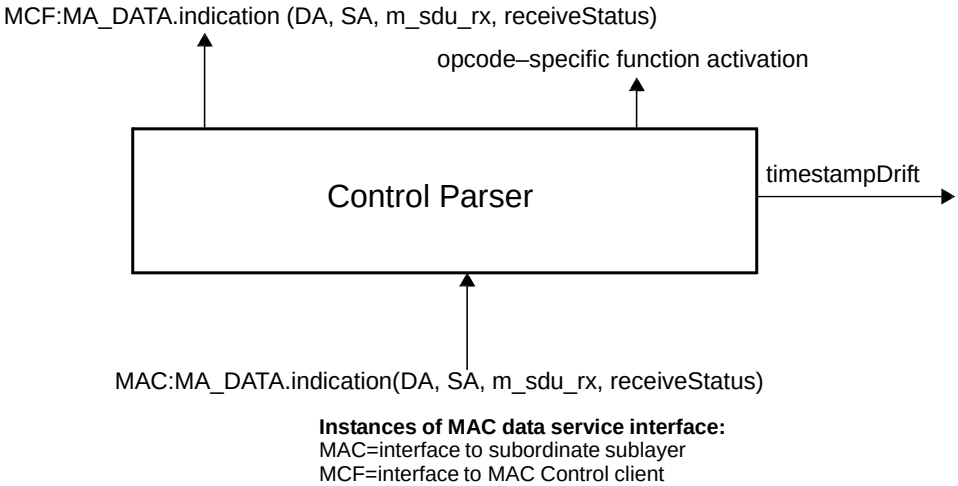
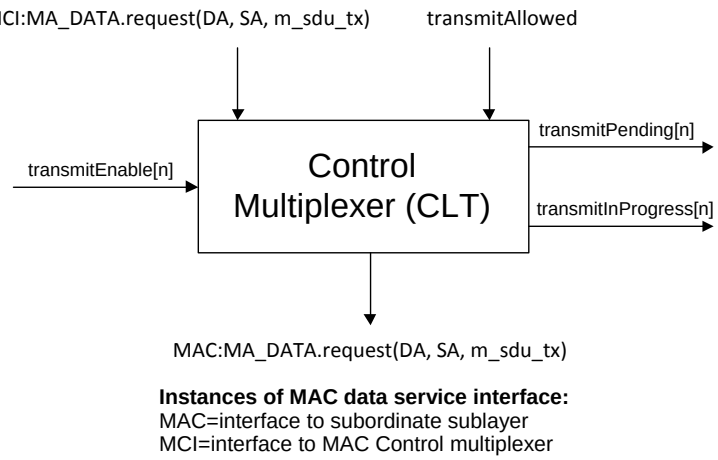
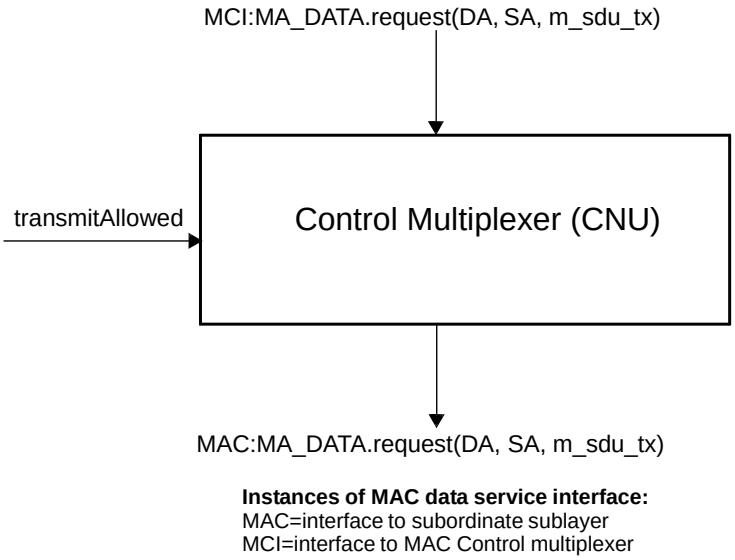


Figure 102–7—Control Parser service interfaces



NOTE—MAC:MA_DATA.request primitive may be issued from multiple MAC Control processing blocks.

Figure 102–8—CLT Control Multiplexer service interfaces



NOTE—MAC:MA_DATA.request primitive may be issued from multiple MAC Control processing blocks.

Figure 102–9—CNU Control Multiplexer service interfaces

102.2.2.1 Constants

FEC_CODEWORD_SIZE
TYPE: integer
This constant represents the size of FEC codeword in octets (FEC_PAYLOAD_SIZE + FEC_PARITY_SIZE).
Value: {TBD}

FEC_PARITY_SIZE
TYPE: integer

This constant represents the size of FEC codeword parity field in octets.
Value: {TBD}

FEC_PAYLOAD_SIZE
TYPE: integer
This constant represents the size of FEC codeword payload in octets.
VALUE: {TBD}

guardThresholdCLT
TYPE: integer
This constant holds the maximum amount of drift allowed for a timestamp received at the CLT. This value is measured in units of time_quantum.
VALUE: 12

guardThresholdCNU
TYPE: integer
This constant holds the maximum amount of drift allowed for a timestamp received at the CNU. This value is measured in units of time_quantum.
VALUE: 8

MAC_Control_type
TYPE: integer
The value of the Length/Type field as defined in 31.4.1.3.
VALUE: 0x8808

tailGuard
TYPE: integer
This constant holds the value used to reserve space at the end of the upstream transmission at the CNU in addition to the size of last MAC service data unit (m_sdu) in units of octets. Space is reserved for the MAC overheads including: preamble, SFD, DA, SA, Length/Type, FCS, and minimum interpacket gap. The sizes of the above listed MAC overhead items are described in 3.1.1. The size of the minimum IPG is described in 4A.4.2.
VALUE: 38

time_quantum
This variable is defined in 64.2.2.1.

tqSize
TYPE: integer
This constant represents time_quantum in octet transmission times.
VALUE: 20

Note that the list of constants will be updated per technical decision #44 (http://www.ieee802.org/3/bn/public/decisions/decisions.html) once EPoC-specific FEC and PMD overhead details are settled.

102.2.2.2 Counters

localTime
TYPE: 32 bit unsigned
This variable holds the value of the local timer used to control MPCP operation. This variable is advanced by a timer at 62.5 MHz, and counts in time_quanta. At the CLT the counter shall track the transmit clock, while at the CNU the counter shall track the receive clock. For accuracy of receive clock see Y.4.1.2. It is reloaded with the received timestamp value (from the CLT) by the Control Parser (see Figure 102–12). Changing the value of this variable while running using Layer Management is highly undesirable and is unspecified.

102.2.2.3 Variables

BEGIN

TYPE: Boolean

This variable is used when initiating operation of the functional block state diagram. It is set to true following initialization and every reset.

fecOffset

TYPE: 32 bit unsigned

A variable that advances by PHY_DATA_SIZE after every 8×(PHY_DATA_SIZE + PHY_OVERHEAD_SIZE) bit times (see EPoC de-rating equation 102-1). After reaching the value of FEC_CODEWORD_SIZE, this variable is reset to zero. In the CLT, this variable is initialized to 0 at system initialization. In the CNU, this variable is assigned in the GATE Processing CNU Activation state diagram (see Figure 102–14).

NOTE—Notation fecOffset[1:0] refers to two least significant bits of this variable.

data_rx

TYPE: bit array

This variable represents a 0–based bit array corresponding to the payload of a received MPCPDU. This variable is used to parse incoming MPCPDU frames.

data_tx

TYPE: bit array

This variable represents a 0-based bit array corresponding to the payload of an MPCPDU being transmitted. This variable is used to access payload of outgoing MPCPDU frames, for example to set the timestamp value.

grantStart

TYPE: Boolean

This variable indicates beginning of a grant transmission. It is set to true in the GATE Processing Activation state diagram (see Figures 102-28b and 102–30) when a new grant activates. It is reset to false after the transmission of the first frame in the grant (see Figure 102–14). This variable is defined in CNU and, for TDD mode only, also in the CLT.

newRTT

TYPE: 16 bit unsigned

This variable temporary holds a newly–measured Round Trip Time to the CNU. The new RTT value is represented in units of time_quanta.

m_sdu_rx

TYPE: bit array

Equal to the concatenation of the Length/Type and data_rx variables.

m_sdu_tx

TYPE: bit array

Equal to the concatenation of the Length/Type and data_tx variables.

m_sdu_ctl

TYPE: bit array

Equal to the concatenation of the MAC_Control_type and data_tx variables.

OctetsRemaining

TYPE: 32 bit unsigned

This variable is an alias for the expression (((stopTime – localTime) ×tqSize) – tqOffset). It denotes the number of octets that can be transmitted between the current time and the end of the grant.

OctetsRequired

TYPE: 16 bit unsigned

This variable represents a total transmission time of next packet and is used to check whether the next packet fits in the remainder of the transmission window. The value of OctetsRequired includes packet transmission time, tailGuard defined in 102.2.2.1, and FEC parity data overhead. This variable is measured in units of octets.

opcode_rx

TYPE: 16 bit unsigned

This variable holds an opcode of the last received MPCPDU.

opcode_tx

TYPE: 16 bit unsigned

This variable holds an opcode of an outgoing MPCPDU.

packet_initiate_delay

TYPE: 16 bit unsigned

This variable is used to set the time–out interval for packet_initiate_timer defined in 102.2.2.5. The packet_initiate_delay value is represented in units of octets.

RTT

TYPE: 16 bit unsigned

This variable holds the measured Round Trip Time to the CNU. The RTT value is represented in units of time_quanta.

stopTime

TYPE: 32 bit unsigned

This variable holds the value of the localTime counter corresponding to the end of the nearest grant. This value is set by the Gate Processing function as described in 102.3.5.

timestamp

TYPE: 32 bit unsigned

This variable holds the value of timestamp of the last received MPCPDU frame.

timestampDrift

TYPE: Boolean

This variable is used to indicate whether an error is signaled as a result of uncorrectable timestamp drift.

tqOffset

TYPE: 8 bit unsigned

This variable denotes the offset (in octet times) of the current actual time from the localTime variable (which maintains the current time in units of time_quanta).

transmitAllowed

TYPE: Boolean

This variable is used to control PDU transmission at the CNU and at the CLT. It is set to true when the transmit path is enabled, and is set to false when the transmit path is being shut down. transmitAllowed changes its value according to the state of the Gate Processing functional block.

transmitEnable

TYPE: Boolean array

This array contains one element per each Multipoint MAC Control instance. Elements of this array are used to control the transmit path in the Multipoint MAC Control instance at the CLT. Setting an element to TRUE indicates that the selected instance is permitted to transmit a frame. Setting it to FALSE inhibits the transmission of frames in the selected instance. Only one element of transmitEnable should be set to TRUE at a time.

transmitInProgress

TYPE: Boolean array

This array contains one element per each Multipoint MAC Control instance. The element *j* of this array set to on indicates that the Multipoint MAC Control instance *j* is in the process of transmitting a frame.

transmitPending

TYPE: Boolean array

This array contains one element per each Multipoint MAC Control instance. The element *j* of this array set to on indicates that the Multipoint MAC Control instance *j* is ready to transmit a frame.

PHY_DATA_SIZE

TYPE: integer

The number of octets constituting the denominator in the EPoC de-rating Equation (102–1). To normalize the effective data rate, the MPCP control multiplexer waits PHY_OVERHEAD_SIZE octets per every PHY_DATA_SIZE octets transmitted.

Value: {TBD}

$$B = \frac{\text{XGMII\_Rate}}{\text{PCS\_Rate}} = \frac{\text{PHY\_DATA\_SIZE+PHY\_OVERHEAD\_SIZE}}{\text{PHY\_DATA\_SIZE}} \quad (102\text{-}1)$$

PHY_OVERHEAD_SIZE

TYPE: integer

The number of octets constituting (together with PHY_DATA_SIZE) the numerator in the EPoC de-rating Equation (102–1). To normalize the effective data rate, the MPCP control multiplexer waits PHY_OVERHEAD_SIZE octets per every PHY_DATA_SIZE octets transmitted.

Value: {TBD}

Note that the list of variables will be updated per technical decision #44 (http://www.ieee802.org/3/bn/public/decisions/decisions.html) once EPoC-specific FEC and PMD overhead details are settled.

102.2.2.4 Functions

abs(n)

This function returns the absolute value of the parameter n.

Opcode–specific function(opcode)

Functions exported from opcode specific blocks that are invoked on the arrival of a MAC Control message of the appropriate opcode.

CheckGrantSize(length)

This function calculates the future time at which the transmission of the current frame (including the FEC parity overhead) is completed.

$$\text{CheckGrantSize}(\text{length}) = \left\lceil \frac{\text{fecOffset} + \text{length}}{\text{FEC\_PAYLOAD\_SIZE}} \right\rceil \times \text{FEC\_CODEWORD\_SIZE} - \text{fecOffset}$$

NOTE—The notation $\lceil x \rceil$ represents a *ceiling* function, which returns the value of its argument *x* rounded up to the nearest integer.

PMD_Overhead(length)

This function calculates the additional amount of time (in octet times) that the MPCP control multiplexer waits following transmission of a frame of size 'length' by the MAC. The additional time is added to allow the insertion of parity data into the frame by the PHY layer and to adjust the data rate to the effective data rate supported by the PCS and PMD. PMD_Overhead() returns

the number of octets that the PHY inserts during transmission of a particular packet and its subsequent IPG. Parameter 'length' represents the size of an entire frame including preamble, SFD, DA, SA, Length/Type, FCS, and IPG. The following formula is used to calculate the overhead:

$$\text{PMD\_Overhead}(\text{length}, \text{B}) = 12 + \left\lceil (\text{B} - 1) \times \text{length} + \text{B} \times \left(\text{FEC\_PARITY\_SIZE} \times \left\lfloor \frac{\text{fecOffset} + \text{length}}{\text{FEC\_PAYLOAD\_SIZE}} \right\rfloor \right) \right\rceil$$

NOTE – The notation $\lceil x \rceil$ represents a *celing* function, which returns the value of its argument *x* rounded up to the nearest integer.

NOTE—The notation $\lfloor x \rfloor$ represents a *floor* function, which returns the value of its argument *x* rounded down to the nearest integer.

select()

This function selects the next Multipoint MAC Control instance allowed to initiate transmission of a frame. The function returns an index to the transmitPending array for which the value is not false. The selection criteria in the presence of multiple active elements in the list is implementation dependent.

SelectFrame()

This function enables the interface, which has a pending frame. If multiple interfaces have frames waiting at the same time, only one interface is enabled. The selection criteria is not specified, except for the case when some of the pending frames have Length/Type = MAC_Control. In this case, one of the interfaces with a pending MAC Control frame shall be enabled.

sizeof(sdu)

This function returns the size of the sdu in octets.

transmissionPending()

This function returns true if any of the Multipoint MAC Control instances has a frame waiting to be transmitted. The function can be represented as:

```
transmissionPending() =
   transmitPending[0] +
   transmitPending[1] +
   ... +
   transmitPending[n-1]
```

where n is the total number of Multipoint MAC Control instances.

Note that the list of function will be updated per technical decision #44 (http://www.ieee802.org/3/bn/public/decisions/decisions.html) once EPoC-specific FEC and PMD overhead details are settled. In particular, further checks are needed for the function CheckGrantSize(), in relation to data rata adaption changes.

102.2.2.5 Timers

packet_initiate_timer

This timer is used to delay frame transmission from MAC Control to avoid variable MAC delay while MAC enforces IPG after a previous frame. In addition, this timer increases interframe spacing just enough to accommodate the extra parity data to be added by the FEC encoder.

102.2.2.6 Messages

MAC:MA_DATA.indication(DA, SA, m_sdu, receiveStatus)

The service primitive is defined in 31.3.

MCF:MA_DATA.indication(DA, SA, m_sdu, receiveStatus)

The service primitive is defined in 31.3.

MAC:MA_DATA.request (DA, SA, m_sdu)

The service primitive is defined in 31.3. The action invoked by this service primitive is not considered to end until the transmission of the frame by the MAC has concluded. The ability of the MAC control layer to determine this is implementation dependent.

MCF:MA_DATA.request (DA, SA, m_sdu)

The service primitive is defined in 31.3.

102.2.2.7 State diagrams

The Multipoint transmission control function in the CLT shall implement state diagram shown in Figure Z–10. Control parser function in the CLT shall implement state diagram shown in Figure 102–11. Control parser function in the CNU shall implement state diagram shown in Figure 102–12. Control multiplexer function in the CLT shall implement state diagram shown in Figure 102–13. Control multiplexer function in the CNU shall implement state diagram shown in Figure 102–14.

Note that Figure 102–13 and Figure 102–14 will be updated per technical decision #44 (http://www.ieee802.org/3/bn/public/decisions/decisions.html) once EPoC-specific FEC and PMD over-head details are settled.

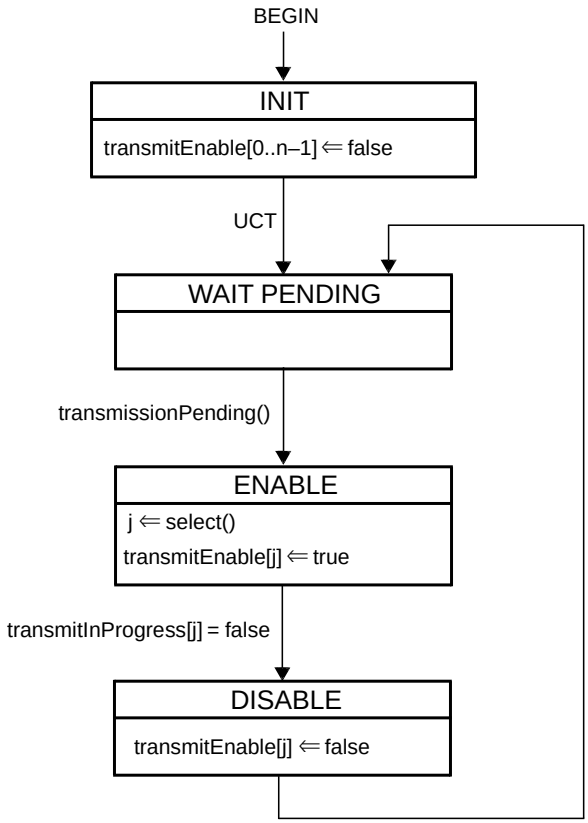
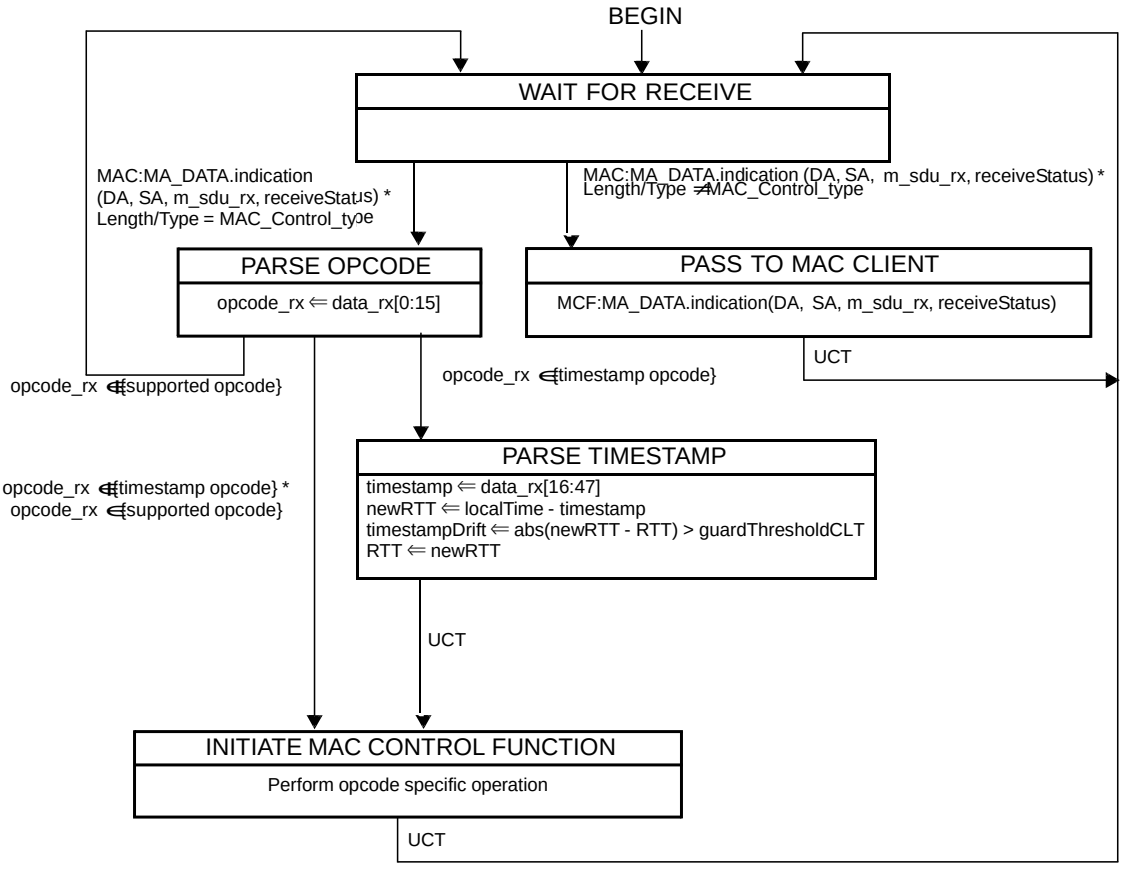


Figure 102–10—CLT Multipoint Transmission Control state diagram

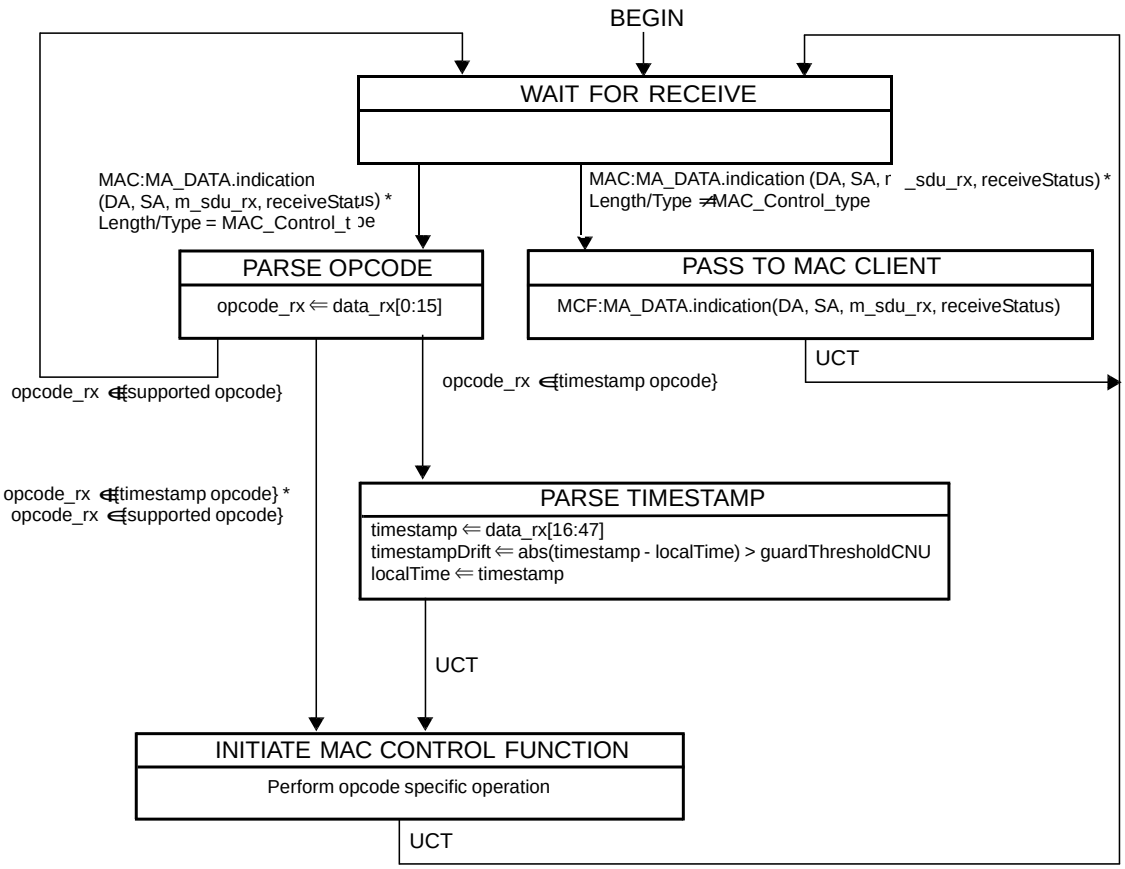


Instances of MAC data service interface:
MAC=interface to subordinate sublayer
MCF=interface to MAC Control client

NOTE—The opcode–specific operation is launched as a parallel process by the MAC Control sublayer, and not as a synchronous function. Progress of the generic MAC Control Receive state diagram (as shown in this figure) is not implicitly impeded by the launching of the opcode specific function.

Refer to Annex 31A for list of supported opcodes and timestamp opcodes.

Figure 102–11—CLT Control Parser state diagram

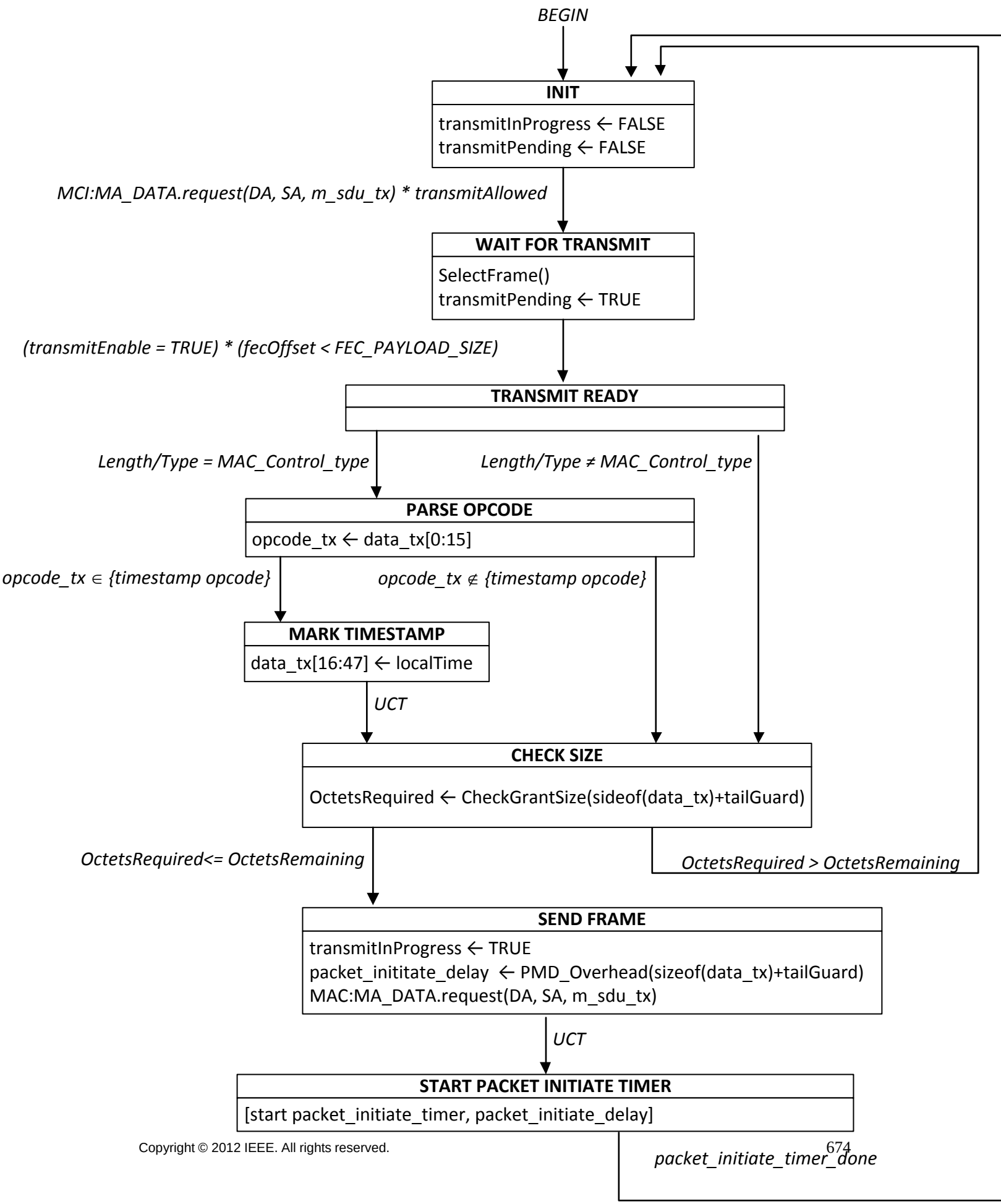


Instances of MAC data service interface:
MAC=interface to subordinate sublayer
MCF=interface to MAC Control client

NOTE—The opcode–specific operation is launched as a parallel process by the MAC Control sublayer, and not as a synchronous function. Progress of the generic MAC Control Receive state diagram (as shown in this figure) is not implicitly impeded by the launching of the opcode specific function.

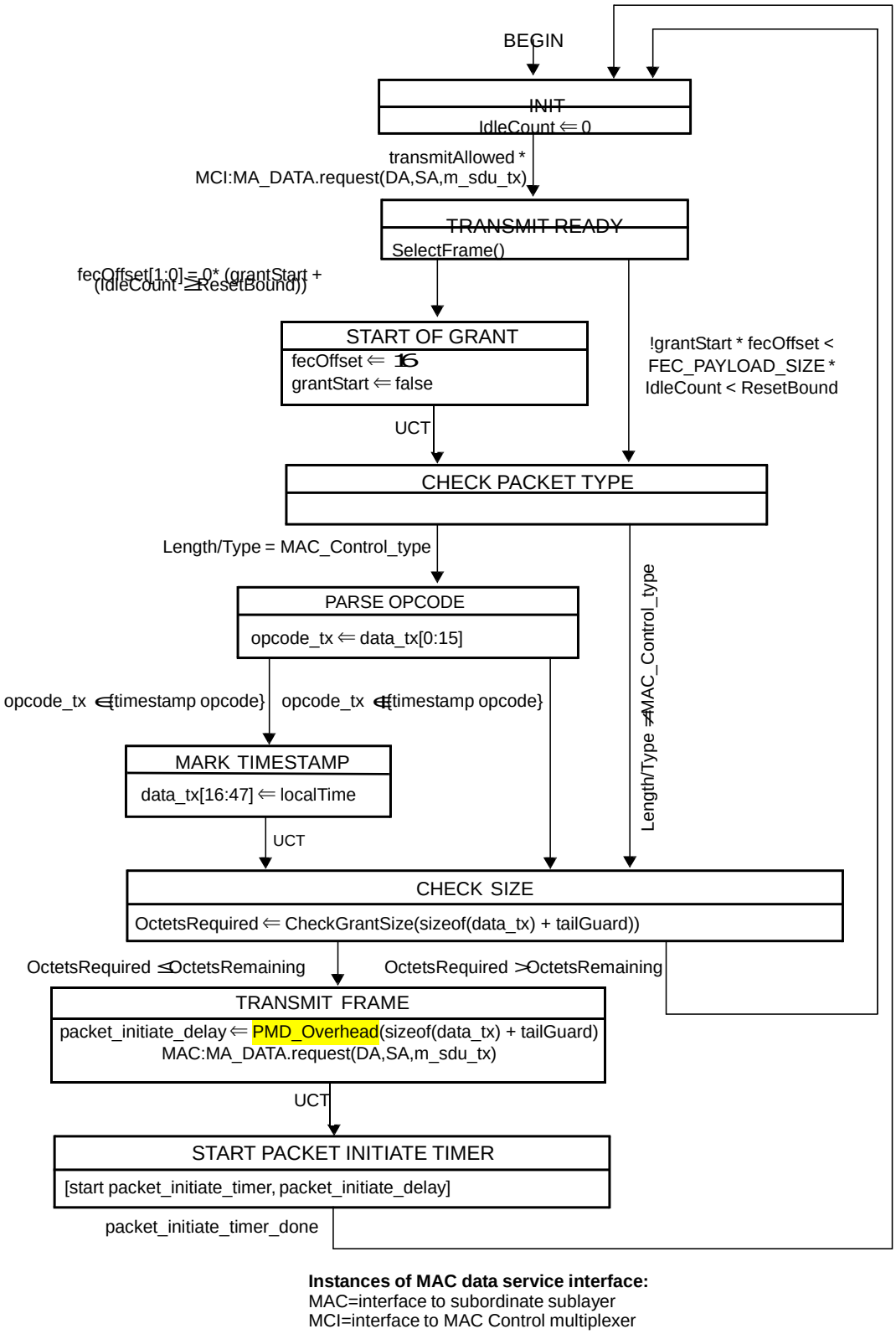
Refer to Annex 31A for list of supported opcodes and timestamp opcodes.

Figure 102–12—CNU Control Parser state diagram



Instances of MAC data service interface:
MAC=interface to subordinate sublayer
MCI=interface to MAC Control multiplexer

Figure 102–13—CLT Control Multiplexer state diagram



Instances of MAC data service interface:
MAC=interface to subordinate sublayer
MCI=interface to MAC Control multiplexer

Figure 102–14—CNU Control Multiplexer state diagram