This is an alternative suggested remedy for the comment #182.

The Draft 2.0 has the following text:

```
ProcessTimestamp( Plid, Timestamp )
{
   if( FirstTimestamp[Plid] )
   {
      // The following line is executed only in the ONU
      LocalTime = Timestamp;
      Rtt[Plid] = LocalTime - Timestamp;
      TimestampDrift[Plid] = false;
      FirstTimestamp[Plid] = false;
   }
   else
      TimestampDrift[Plid] = abs(LocalTime - Timestamp) > DRIFT_THOLD
}
```

Note that the *LocalTime* value in this function represents the MPCP local time latched when the Envelope Start Header (ESH) containing the given MPCPDU was received, and not the time at which the *ProcessTimestamp(...)* function was executed. See 144.3.1.1 for more details.

The draft does not formally show the *LocalTime* being latched when ESH is received. Note that the *ProcessTimestamp()* function is executed only after the MAC has received the entire MPCPDU and has verified the FCS, i.e., much later than timestamp reference time. This is the same approach we used in 802.3av and we never formally showed latching the MPCPDU receive time there either. However, if we decide to be more formal in 802.3ca, here is the way to do it:

1) MCRS Output process generates *MCRS_ESH[ch].indication()* every time it reads an ESH from the *ENV_RX* buffer. This indication is similar to *TS_RX.indication()* primitive used by the Clause 90 for time synchronization.

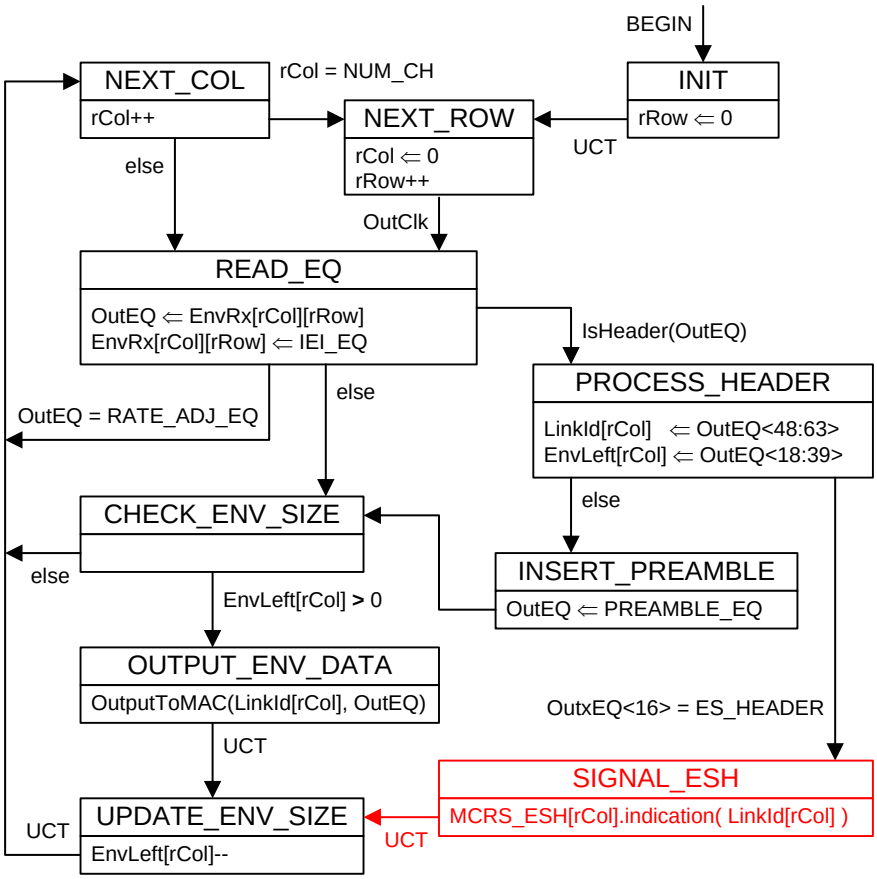

Figure 143–16—MCRS Receive Function, Output Process state diagram

143.3.1.2.3 MCRS_ECH[ch].indication(Llid) primitive

The Output Process (see Figure 143–16) generates the *MCRS_ECH[ch].indication(Llid)* primitive every time an ESH EQ is read from the *ENV_RX* buffer. This primitive causes the MPMC Control Parser Process (see 144.2.1) to generate a local timestamp (i.e., to latch the local MPCP time) representing the arrival time of ESH EQ.

2) The MPMC Control Parser latches *LocalTime* value into *LatchedTime[LLid]* variable at the moment it receives *MCRS_ESH[ch].indication()* primitive.

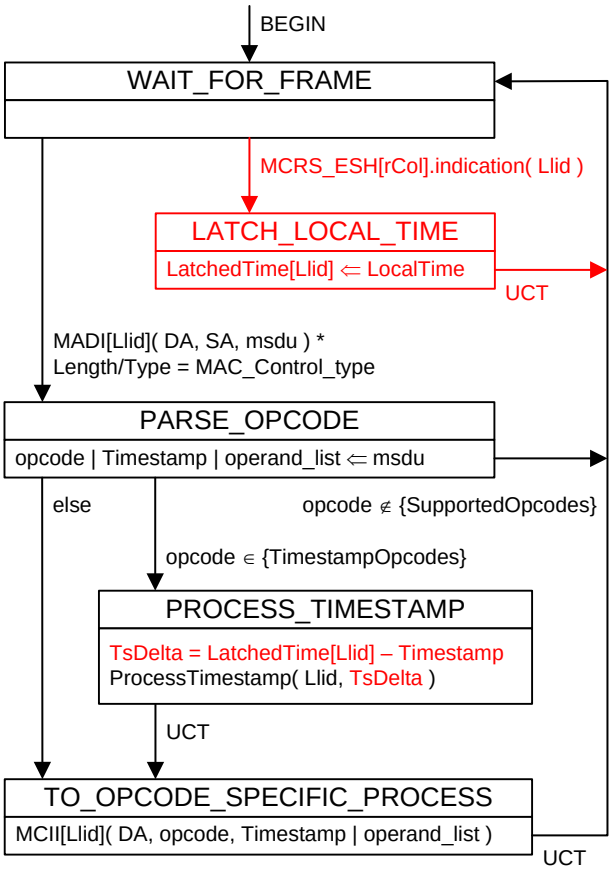

Figure 144–5—Control Parser state diagram

LatchedTime[]

Type: An array of 32-bit unsigned integers

Description: Each element of this array represents the value of *LocalTime* variable (see 144.2.1.2) latched at the moment when the *MCRS_ESH.indication(Llid)* primitive was generated. The elements of the array are indexed by the *Llid* values.

TsDelta

Type: 32-bit signed integer

Description: This variable represents the difference between the time that ESH was read from the MCRS *ENV_RX* buffer (i.e., the *LatchedTime[Plid]*) and the *Timestamp* value in an MPCPDU that followed that ESH. In the ONU, the *TsDelta* is used to adjust the *LocalTime* value, while in the OLT it is used as the measurement of the round-trip time to the ONU that sourced the given MPCPDU.

3) Finally, the *ProcessTimestamp()* function is modified to use *LatchedTime* instead of *LocalTime*. Once we have MPCPDU *Timestamp* and *LatchedTime* values stored, it doesn't matter when exactly the *ProcessTimestamp()* function is executed

```
ProcessTimestamp( Plid, TsDelta )
{
      if( FirstTimestamp[Plid] )
      {
            // The following line is executed only in the ONU
            LocalTime -= TsDelta;

            // The following line is executed only in the OLT
            Rtt[Plid] = TsDelta;

            TimestampDrift[Plid] = false;
            FirstTimestamp[Plid] = false;
      }
      else
            TimestampDrift[Plid] = abs(TsDelta) > DRIFT_THOLD
}
```

Some explanation is needed for the following line:

```
LocalTime -= TsDelta;
```

What we really need to do here is to set *LocalTime* to be equal to the *Timestamp* in received MPCPDU:

```
LocalTime = Timestamp;
```

But since the *LocalTime* constantly increments, the above statement would only be correct at the exact moment of time when an ESH was read from the ENV_RX buffer. In reality, the *ProcessTimestamp()* function is executed at a later time -- after the entire MPCPDU is received and FCS is checked. In situations when we have multiple MPCPDUs following the same ESH, the delay to call the *ProcessTimestamp()* function can be even larger.

That additional time elapsed since latching the ESH reception time and until the current moment is equal to `LocalTime - LatchedTime[Plid]`. So, at the time when we set the *LocalTime* to a new value, we need to account for the additional elapsed time, i.e.:

```
LocalTime = Timestamp + (LocalTime – LatchedTime[Plid]);
```

The *ProcessTimestamp()* function doesn't take the *Timestamp* value as an argument anymore. But it takes *TsDelta*, which by definition is equal `LatchedTime[Plid] - Timestamp`. So we can substitute the *TsDelta* into the above equation:

```
LocalTime = LocalTime – TsDelta;
```

Or using "**-=**" operator, we get

```
LocalTime -= TsDelta;
```