Interoperability with a One-Step Clock on Receive in 802.1ASbt

Geoffrey M. Garner
Consultant

IEEE 802.1 AVB TG
2013.11.12

gmgarner@alum.mit.edu

Introduction - 1

- This presentation describes changes needed in 802.1AS (potentially to be included in 802.1ASbt) for a clock (time-aware system) to interoperate with a one-step clock
  - The clock interoperating with the one-step clock would need to handle (process) messages received from the one-step clock
  - However, the clock would not have to be a one-step clock, nor transmit one-step messages
- This was discussed in previous presentations, most recently in the initial version and in Revision 1 of [1]
  - **Revision 2 of [1] described additional changes needed for 802.1AS for a clock to transmit one-step messages; this material is not included and not discussed in the current presentation**
- The present presentation expands on Revision 1 of [1] in that the changes needed for the MDSyncReceive state machine and related functions are indicated

Introduction - 2

- Reference [1] (all revisions) also indicates that if one-step Pdelay messages are to be handled, we need to decide how information needed for neighbor rate ratio computation will be transported
  - Here, we review the possible approaches described in [1]

Review of Two-Step Handling of Sync and Follow_Up

- When a two-step clock (802.1AS time-aware system) sends Sync and Follow_Up:
  - originTimestamp field of Sync is set to 0 (all the Sync fields after the header are shown as reserved in 11.4.3)
  - correctionField of Sync is set to 0 (Table 11-5 of the 802.1ASbt draft
  - PreciseOrigin timestamp field of Follow_Up message contains the timestamp of the grandmaster (GM) where the Sync information originated, except for any sub-ns portion
  - correctionField of Follow_Up contains the sum of the sub-ns portion of the timestamp of the GM where the Sync information originated, the acumulated residence times in the path, the accumulated mean link delays in the path, and any asymmetry corrections
- A time-aware system that receives Sync and Follow_Up from a two step clock does the following on transmitting Sync and Follow_Up
  - Timestamp the outgoing Sync message and compute the residence time, as indicated in 11.2.14 (MDSyncSend state machine)
  - Add the residence time and mean propagation delay on the incoming link to the correctionField of the incoming Follow_Up message
  - Transmit the Follow_Up message with the incoming preciseOrigin timestamp and the new correctionField value computed in the above bullet item

Handling of Sync and Follow_Up from One-Step Clock - 1

- 802.1AS does not currently specify a one-step clock
  - The most logical place to go to for a description of what a one-step clock sends is IEEE 1588
- When a one-step boundary clock sends Sync (it does not send Follow_Up because it is one-step):
  - The originTimestamp contains the recovered GM time, except for any sub-ns part
  - The correctionField contains any sub-ns portion of the recovered GM time
  - The correctionField does not contain any residence time corrections, nor link delays, when the Sync message is transmitted by a BC; these are included in the correction field only if the Sync message is transmitted by a transparent clock (TC)
    - When the Sync message is transmitted by a BC, these corrections are included in the originTimestamp (except for any sub-ns part)

Handling of Sync and Follow_Up from One-Step Clock - 2

- 802.1AS does not specify one-step on transmission (as stated above)
- For two-step transmission, 802.1AS places all components of the time, except the GM time when it transmits the Sync information excluding any sub-ns portion, in the correctionField
- It is conceivable that a one-step clock whose Sync messages are being received might also place all components of time (except the GM time when it transmits the Sync information excluding any sub-ns portion) in the correctionField
    - The main constraint is that when we add the originTimestamp and correctionField, the result is the time the Sync message was transmitted by the next upstream node
- Therefore, we will assume that the time the Sync message was transmitted by the next upstream node is the sum of the Sync message originTimestamp and correctionField, for the case where the upstream boundary clock or ordinary clock is one-step
- Finally, it is assumed that the Follow_Up information TLV is attached to the Sync message when the transmission is from a one-step clock

Additional Point for Two-Step Clocks

- If a clock is two-step, the correctionField of Sync ought to not contain timing information (i.e., it ought to be zero)
  - This is because IEEE 1588 – 2008 (subclause 9.5.9.4) and IEEE 802.1AS – 2011 (subclause 11.2.14.2.1) specify that the Sync correctionField shall be set to zero on transmit
  - However, it would be safest to check the correctionField of Sync anyway since, given that we are describing interaction with one-step clocks, which are not specified in 802.1AS, we might also have non-standard behavior that results in the Sync correctionField non-zero when transmitted from a two-step boundary clock; for example
    - The message could have come from a system that interoperates with one-step clocks and leaves the Sync correctionField intact
    - There could be a one-step transparent clock in between the upstream two-step boundary clock and the receiving boundary clock
  - Therefore, when receiving messages from a two-step clock, the correctionFields of the Sync and Follow_Up messages should be added to the preciseOriginTimestamp to obtain the time the Sync message was transmitted by the next upstream node

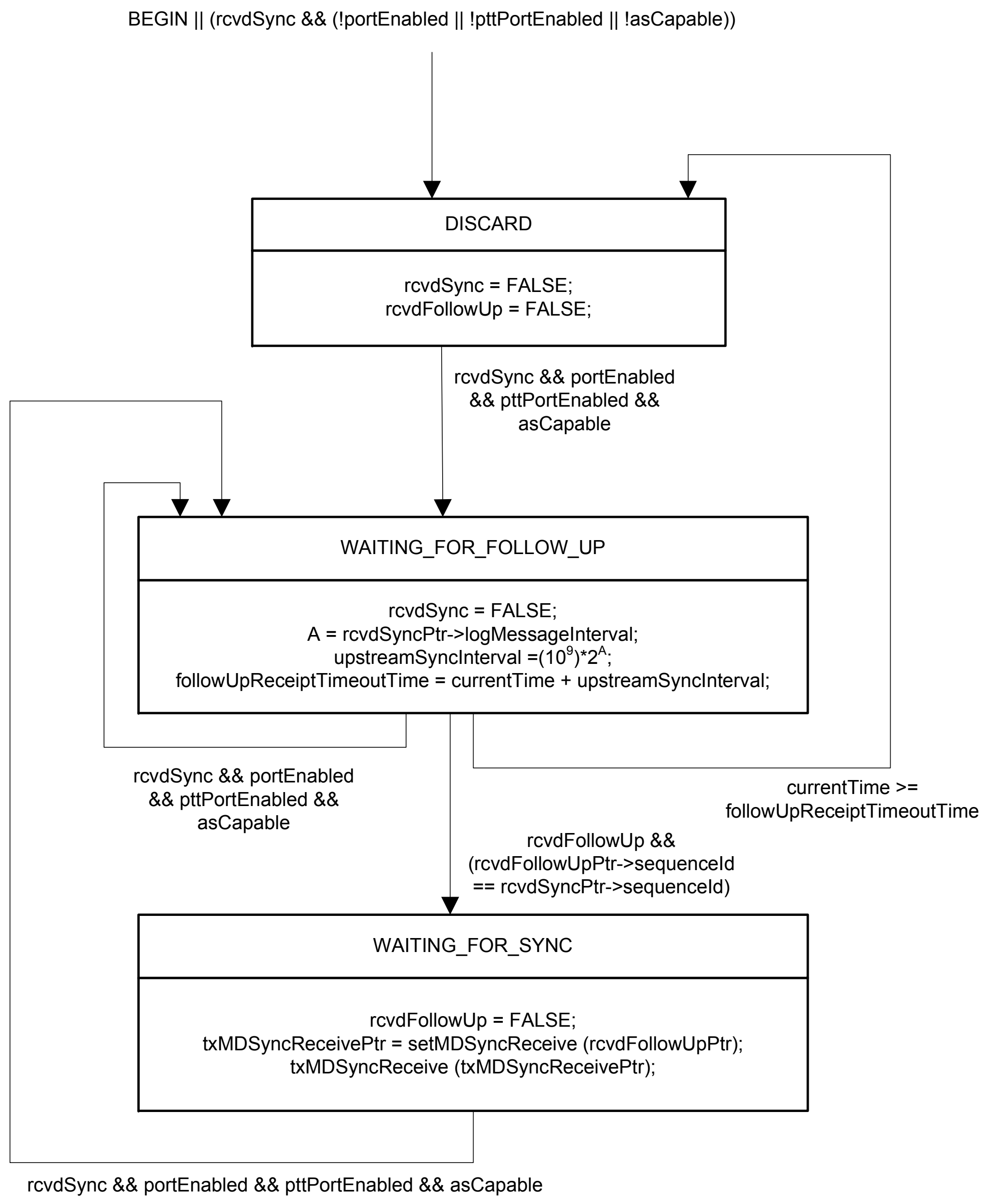
Summary of Changes to Handle One-Step on Receive (See R1 of [1]) - 1

- ❑Allow twoStepFlag to be FALSE on receive, in Table 11-4
- ❑Also in Table 11-4, now must pay attention to twoStepFlag on receive (and not ignore it on receive)
- ❑In Table 11-5, need to indicate that the correctionField of Sync now can contain corrections for fractional ns, residence times, and link delays (and any asymmetry corrections) in the case where the messages are from a one-step clock
  - ▪However, the correctionField may contain timing information in the two-step case as indicated on the previous slide; therefore, it will be included in computing the GM time
- ❑In MDSyncReceiveSM state machine (11.2.13 and Figure 11-6), need to add logic for case where twoStepFlag is FALSE
  - ▪If twoStepFlag is FALSE, do not wait for Follow_Up; process the correctionField in Sync as the Follow_Up correctionField would be processed.

Summary of Changes to Handle One-Step on Receive (See R1 of [1]) - 1

- ❑ If twoStepFlag is TRUE, the correctionField of Sync should be added to the preciseOriginTimestamp and correctionField of the Follow_Up message to obtain the GM time when the Sync message was transmitted from the next node upstream
- ❑ MDSyncSendSM state machine does not change, as messages are sent as two-step

Current MDSyncReceive State Machine



BEGIN || (rcvdSync && (!portEnabled || !pttPortEnabled || !asCapable))
DISCARD
rcvdSync = FALSE;
rcvdFollowUp = FALSE;
rcvdSync && portEnabled && pttPortEnabled && asCapable
WAITING_FOR_FOLLOW_UP
rcvdSync = FALSE;
A = rcvdSyncPtr->logMessageInterval;
upstreamSyncInterval =(10^9)*2^A;
followUpReceiptTimeoutTime = currentTime + upstreamSyncInterval;
rcvdSync && portEnabled && pttPortEnabled && asCapable
currentTime >= followUpReceiptTimeoutTime
rcvdFollowUp && (rcvdFollowUpPtr->sequenceId == rcvdSyncPtr->sequenceId)
WAITING_FOR_SYNC
rcvdFollowUp = FALSE;
txMDSyncReceivePtr = setMDSyncReceive (rcvdFollowUpPtr);
txMDSyncReceive (txMDSyncReceivePtr);
rcvdSync && portEnabled && pttPortEnabled && asCapable

setMDSyncReceive(): creates an MDSyncReceive structure, and returns a pointer to this structure. The members of this structure are set as follows:

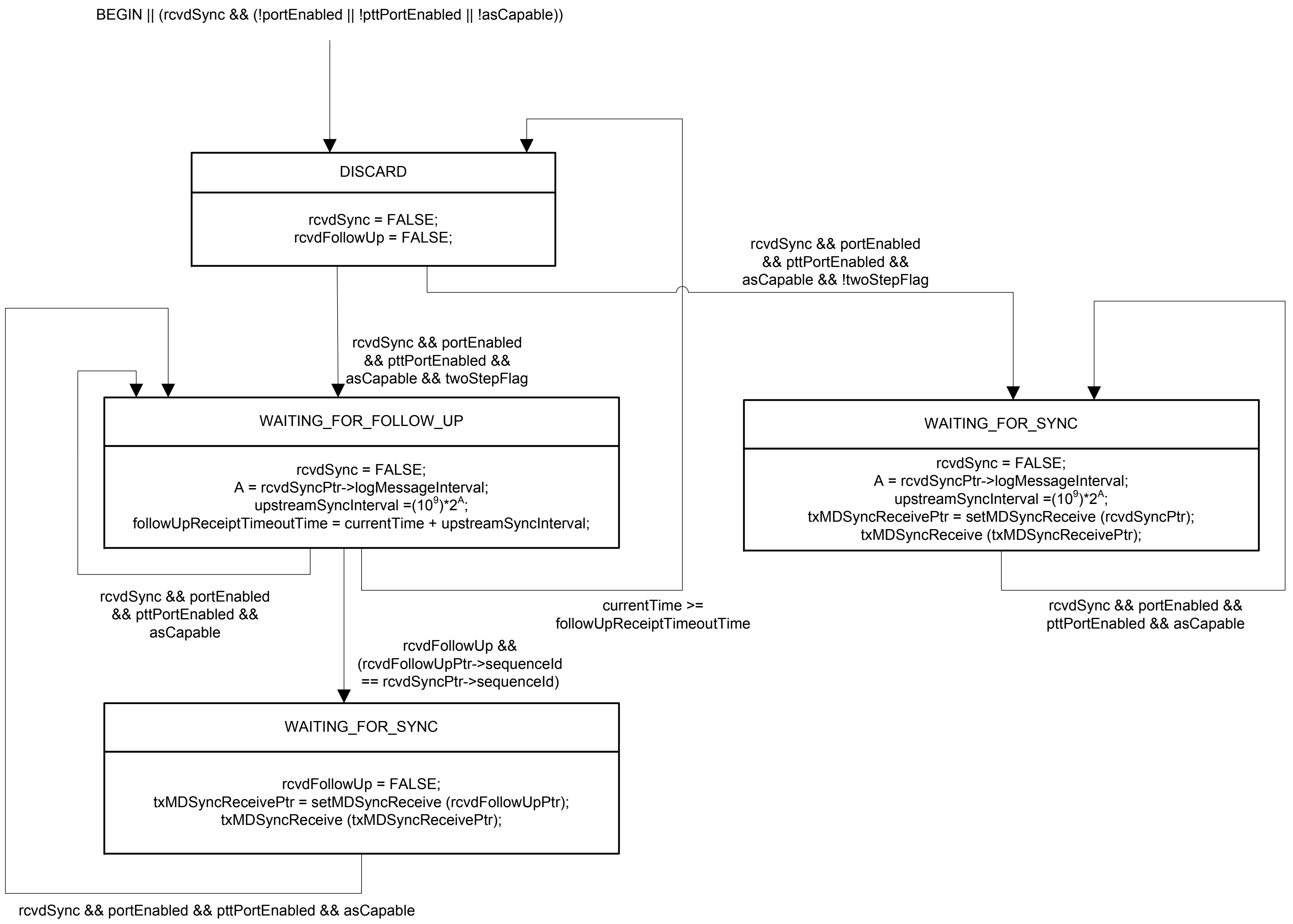
a) followUpCorrectionField is set equal to the correctionField (see Error! Reference source not found.) of the most recently received FollowUp message,
b) sourcePortIdentity is set equal to the sourcePortIdentity (see Error! Reference source not found.) of the most recently received Sync message,
c) logMessageInterval is set equal to the logMessageInterval (see Error! Reference source not found.) of the most recently received Sync message,
d) preciseOriginTimestamp is set equal to the preciseOriginTimestamp (see Error! Reference source not found.) of the most recently received Follow_Up message,
e) rateRatio is set equal to the quantity (cumulativeScaledRateOffset$\times 2^{-41}$)+1.0, where the cumulativeScaledRateOffset field is for the most recently received Follow_Up message (see Error! Reference source not found.),
f) upstreamTxTime is set equal to the <syncEventIngressTimestamp> for the most recently received Sync message, minus the mean propagation time on the link attached to this port (neighborPropDelay, see 10.2.4.7) divided by neighborRateRatio (see 10.2.4.6), minus delayAsymmetry (see 10.2.4.8) for this port divided by rateRatio [see rateRatio is set equal to the quantity (cumulativeScaledRateOffset above]. The <syncEventIngressTimestamp> is equal to the timestamp value measured relative to the timestamp measurement plane, minus any ingressLatency (see 8.4.3),

NOTE 1—The mean propagation time is divided by neighborRateRatio to convert it from the time base of the time-aware system at the other end of the attached link to the time base of the current time-aware system. The delayAsymmetry is divided by rateRatio to convert it from the time base of the grandmaster to the time base of the current time-aware system. The two quotients are then subtracted from <syncEventIngressTimestamp>, which is measured relative to the time base of the current time-aware system.

NOTE 2—The difference between the mean propagation time in the grandmaster time base, the time base of the time-aware system at the other end of the link, and the time base of the current time-aware system is usually negligible. The same is true of any delayAsymmetry. See NOTE 2 of Error! Reference source not found..

g) gmTimeBaseIndicator is set equal to the gmTimeBaseIndicator of the most recently received Follow_Up message (see Error! Reference source not found.),
h) lastGmPhaseChange is set equal to the lastGmPhaseChange of the most recently received Follow_Up message (see Error! Reference source not found.), and
i) lastGmFreqChange is set equal to the lastGmFreqChange of the most recently received Follow_Up message (see Error! Reference source not found.).

New MDSyncReceive State Machine


BEGIN || (rcvdSync && (!portEnabled || !pttPortEnabled || !asCapable))
DISCARD
rcvdSync = FALSE;
rcvdFollowUp = FALSE;
rcvdSync && portEnabled && pttPortEnabled && asCapable && !twoStepFlag
rcvdSync && portEnabled && pttPortEnabled && asCapable && twoStepFlag
WAITING_FOR_FOLLOW_UP
rcvdSync = FALSE;
A = rcvdSyncPtr->logMessageInterval;
upstreamSyncInterval =(10^9)*2^A;
followUpReceiptTimeoutTime = currentTime + upstreamSyncInterval;
WAITING_FOR_SYNC
rcvdSync = FALSE;
A = rcvdSyncPtr->logMessageInterval;
upstreamSyncInterval =(10^9)*2^A;
txMDSyncReceivePtr = setMDSyncReceive (rcvdSyncPtr);
txMDSyncReceive (txMDSyncReceivePtr);
rcvdSync && portEnabled && pttPortEnabled && asCapable
currentTime >= followUpReceiptTimeoutTime
rcvdSync && portEnabled && pttPortEnabled && asCapable
rcvdFollowUp && (rcvdFollowUpPtr->sequenceId == rcvdSyncPtr->sequenceId)
WAITING_FOR_SYNC
rcvdFollowUp = FALSE;
txMDSyncReceivePtr = setMDSyncReceive (rcvdFollowUpPtr);
txMDSyncReceive (txMDSyncReceivePtr);
rcvdSync && portEnabled && pttPortEnabled && asCapable

setMDSyncReceive(): creates an MDSyncReceive structure, and returns a pointer to this structure. The members of this structure are set as follows:

a) If twoStepFlag of the most recently received Sync message is TRUE, followUpCorrectionField is set equal to the sum of the correctionField (see Error! Reference source not found.~~11.4.2.4~~) of the most recently received Sync message, plus the correctionField of the most recently received Follow_Up message, else followUpCorrectionField is set equal to the correctionField of the most recently received Sync message,

b) sourcePortIdentity is set equal to the sourcePortIdentity (see Error! Reference source not found.~~11.4.2.5~~) of the most recently received Sync message,

c) logMessageInterval is set equal to the logMessageInterval (see Error! Reference source not found.~~11.4.2.8~~) of the most recently received Sync message,

d) if twoStepFlag of the most recently received Sync message is TRUE, preciseOriginTimestamp is set equal to the preciseOriginTimestamp (see Error! Reference source not found.~~11.4.4.2.1~~) of the most recently received Follow_Up message, else preciseOriginTimestamp is set equal to the originTimestamp (see 11.4.3.2.1) of the most recently received Sync message,

e) rateRatio is set equal to the quantity (cumulativeScaledRateOffset$\times 2^{-41}$)+1.0, where the cumulativeScaledRateOffset field is for the most recently received Follow_Up message (see Error! Reference source not found.~~11.4.4.3.6~~),

f) upstreamTxTime is set equal to the <syncEventIngressTimestamp> for the most recently received Sync message, minus the mean propagation time on the link attached to this port (neighborPropDelay, see 10.2.4.7) divided by neighborRateRatio (see 10.2.4.6), minus delayAsymmetry (see 10.2.4.8) for this port divided by rateRatio [see rateRatio is set equal to the quantity (cumulativeScaledRateOffset~~e)~~ above]. The <syncEventIngressTimestamp> is equal to the timestamp value measured relative to the timestamp measurement plane, minus any ingressLatency (see 8.4.3),

NOTE 1—The mean propagation time is divided by neighborRateRatio to convert it from the time base of the time-aware system at the other end of the attached link to the time base of the current time-aware system. The delayAsymmetry is divided by rateRatio to convert it from the time base of the grandmaster to the time base of the current time-aware system. The two quotients are then subtracted from <syncEventIngressTimestamp>, which is measured relative to the time base of the current time-aware system.

NOTE 2—The difference between the mean propagation time in the grandmaster time base, the time base of the time-aware system at the other end of the link, and the time base of the current time-aware system is usually negligible. The same is true of any delayAsymmetry. See NOTE 2 of Error! Reference source not found.~~11.2.15.2.4~~.

g) gmTimeBaseIndicator is set equal to the gmTimeBaseIndicator of the most recently received Follow_Up message (see Error! Reference source not found.~~11.4.4~~),
h) lastGmPhaseChange is set equal to the lastGmPhaseChange of the most recently received Follow_Up message (see Error! Reference source not found.~~11.4.4~~), and
i) lastGmFreqChange is set equal to the lastGmFreqChange of the most recently received Follow_Up message (see Error! Reference source not found.~~11.4.4~~).

Sync Message Tables

- The Sync message structure (subclause 11.4.3) must be shown separately for the one-step and two-step cases
- If twoStepFlag is TRUE, the Sync message is as it is shown in 11.4.3 (Table 11-8)
  - In this case, the originTimestamp field, which follows the header, is shown as reserved
- If twoStepFlag is FALSE, the Sync message looks like the Follow_Up message (Table 11-9), except that the preciseOriginTimestamp is labeled originTimestamp
  - In this case, the Follow_Up information TLV is present

Processing of Pdelay Messages – 1 (taken from [1])

- Need to decide what to do regarding the Pdelay messages
  - Both Pdelay_Resp and Pdelay_Resp_Follow_Up messages are used in nearest neighbor rate ratio measurement
  - In one-step Pdelay, only Pdelay_Resp is sent, and it carries difference between its send time and the Pdelay_Req receipt time
    - This is sufficient information for propagation delay measurement, but not for neighbor rate ratio measurement
  - Some possible solutions are
    - Don’t handle one-step Pdelay messages on receive (i.e., only handle one-step Sync)
    - Carry the responseOriginTimestamp (i.e., the timestamp of the sending of Pdelay_Resp) in the requestReceiptTimestamp field of Pdelay_Resp
      - This can be done because IEEE 1588 specifies that in the one-step case the requestReceiptTimestamp field is set to zero, and the difference t3 – t2 is carried in the correctionField
      - But this would not allow any sub-ns component of the timestamp of sending Pdelay_Resp to be carried
      - Also, this would be a specification in 802.1AS; it would be necessary that the one-step system that sends Pdelay_Resp complies with this (probably would want to request this be added to 1588v3)

Processing of Pdelay Messages – 2 (taken from [1])

❑Need to decide what to do regarding the Pdelay messages (cont.)

- Some possible solutions (cont.)
  - Carry the responseOriginTimestamp in a TLV attached to Pdelay_Resp
    - In this approach, a TLV of the same length must be added to Pdelay_Req so that Pdelay_Req and Pdelay_Resp have the same length
    - This is done to ensure that there is no error in measured propagation delay in the event there is an unknown or undetected asymmetry that depends on message length
    - This Pdelay_Req TLV would likely not carry any useful information; its purpose would be only to ensure that Pdelay_Req and Pdelay_Resp have the same length
  - Invent a new mechanism for neighbor rate ratio measurement for this case (i.e., other than the new mechanisms above)
  - Others?
- The intent is that the AVB TG would pick a solution for measuring rate ratio (either one of the above or another solution)
  - It is **not** intended to have multiple options

References - 1

[1] Geoffrey M. Garner, *Discussion of Assumptions for 802.1ASbt Features*, presentation for IEEE 802.1 AVB TG, July 16, 2012, Revision 1, July 16, 2012, Revision 2, July 19, 2012.