Description of Use of IEEE 1588 ~~Followup~~ Peer-to-Peer Transparent Clocks in A/V Bridging Networks

Revision ~~10.01~~2.0

2006.~~043.241827~~05.12

Geoffrey M. Garner
Samsung (Consultant)
gmgarner@comcast.net

Revision history:

1.0 – Addressed numerous initial comments; completed Sections 2.2.2.3 and 4.2 (Pdelay mechanism); completed details on frequency compensation algorithm in Sections 2.2.2.5 and 4.4; added details on architectural implications of P2P TC and OC functions being combined in a single box and resulting simplifications (Sections 2.1 and 4.1.1)

1.1 – Added description of on-the-fly operation (allowed as an option) and, to reflect this, removed "followup" from the title of the document; added revision history; addressed comments on correction field resolution (Section 2.2.2.8) made in 4/19/2006 AVB call and removed author's note at end of that section; added an author's note reflecting discussion in the 4/19/2006 AVB call that multicast Pdelay messages cannot be transmitted on blocked links.

2.0 – Revised frame formats in Section 3 to reflect the latest agreements in the IEEE 1588 Short Frames Subcommittee, given in References [12] – [15].

1. Introduction

The Audio/Video Bridging (AVB) Task Group (TG) within IEEE 802.1 is considering the use of a subset of IEEE 1588 Version 2 Precise Time Protocol (PTP) (currently under development) to provide timing and synchronization to the various AVB network nodes. Specifically, the Draft PAR for AVB Timing/Synchronization (IEEE 802.1as) indicates that IEEE 802.1as "specifies the use of IEEE 1588 specifications where applicable in the context of IEEE Stds 802.1D and 802.1Q" and that it will "leverage the work of the IEEE 1588 WG to develop the additional specifications needed to address these requirements" [1]. A joint IEEE 802.1 AVB/IEEE 1588 Design Meeting was held February 21, 2006 [2], in which it was suggested that the Follow-up Peer-to-Peer (P2P) Transparent Clock (TC) being developed as part if IEEE 1588 Version 2 could be used advantageously to provide synchronization to AVB networks [2]. A possible way of using the Follow-up P2P TC, consistent with the current Draft Working Technical Description [3] was discussed verbally in the meeting and then documented in [4]. Reference [4], augmented by [3] plus other IEEE 1588 Version 2 Working Documents (e.g., relevant documents include, but are not necessarily limited to [5] and portions of [6] referenced in [5]) and the published IEEE 1588 Version 1 [7] provide a description of how IEEE 1588 can be used to provide timing and synchronization to AVB networks.[1] However, an initial presentation of [4] to the AVB TG indicated it would be desirable to provide a more self-contained description of the use of IEEE 1588 in AVB networks.

The main purpose of this document is to provide a detailed description of the use of IEEE 1588 for timing/synchronization in AVB networks, in a manner that is easily accessible and understandable to the user. The intent is to provide a detailed description of the protocols; nonetheless, this document is not a substitute for the eventual standards documents that are published (i.e., IEEE 1588 Version 2, IEEE 802.1as).

The document is organized as follows. Section 2 is an overview of the subset of the PTP clock synchronization model that will be used by AVB networks and additional assumptions and requirements for AVB that are not part of the PTP specification but are part of a PTP profile for AVB. Section 3 describes the message and frame formats. This description is taken from Reference [6]. Section 4 describes how each message is processed as it originates at the ingress node, arrives at the egress node, arrives at a node that is not the egress node, and is transmitted by a node that is not the ingress node.

2. Subset of PTP clock synchronization model used in Audio/Video Bridging networks

Some of the material and text in this section is either taken from or uses text in [9] as a starting point.

2.1 Overview of PTP systems used in A/V Bridging networks

[1] Additional information, referred to as a *1588 profile*, is also needed. This information will be specified in an IEEE 802.1 document. Profile information is typically specific to a respective application, and is not specified directly in IEEE 1588 because IEEE 1588 is used in a wide variety of applications that have different requirements.

General PTP systems (i.e., not necessarily limited to AVB network applications) are distributed, networked systems consisting of some combination of PTP and non-PTP devices. PTP devices include ordinary clocks, boundary clocks, transparent clocks, and administrative nodes. Transparent clocks may be further subdivided into two types: (a) peer-to-peer (P2P), and (b) end-to-end (E2E). Non-PTP devices include ordinary network switches (i.e., bridges), routers, and/or other infrastructure devices, and possibly end application devices such as computers, printers, displays, video or audio players, etc.

The PTP protocol is a distributed protocol that specifies how the real-time PTP clocks in the system synchronize with each other. The ordinary and boundary clocks are organized into a master-slave synchronization hierarchy with the clock at the top of the hierarchy, i.e., the Grandmaster (GM) clock, determining the reference time for the entire system. The synchronization is achieved by exchanging PTP timing messages with the slaves using the timing information to adjust their clocks to the time of their master in the hierarchy. The transparent clocks are not part of the master slave hierarchy; however, each transparent clock may syntonize (i.e., synchronize in frequency but not time) to a master boundary or ordinary clock. In addition, an ordinary or boundary clock may be collocated with a transparent clock, in which case a timing signal synchronized to a master is available at the transparent clock. Every AVB network node will contain a collocated ordinary and peer-to-peer transparent clock. One of the nodes will be the master, and all the other nodes will be slaves to this master (and therefore the master will also be the grandmaster). An AVB network will not contain any non-AVB devices (i.e., any nodes that do not have PTP clocks).[2]

Devices in a PTP system communicate with each other via a communication network. In general, the network may include bridges between segments implementing different network communication protocols; in the case of AVB, the network is Ethernet. An ordinary clock (OC) has a single physical or logical connection to the network, i.e., a single port. A boundary clock (BC) may have multiple physical or logical connections to the network, i.e., multiple ports. An E2E or P2P transparent clock (TC) may have multiple physical connections to the network, i.e., multiple ports. In an AVB network, there is an implied single connection between an OC and the collocated P2P TC. This is illustrated in Figure 1.

Figure 1 indicates that an AVB network node will contain both OC and P2P TC functions. In describing how AVB nodes synchronize to the GM (i.e., by processing PTP messages, which results in the computation of frequency adjustment factors, slave offsets, and new states for the OC and its port and in the synchronization of each slave OC to the GM OC), two approaches are possible. We may consider the OC and P2P TC as individual functions and describe them in a general way. We would indicate that an AVB network node must have both functions present; however, the description would not take advantage of any simplification that resulted from this requirement. In such a description, we would consider the functional link between the OC function and the P2P TC function explicitly, i.e., it would have a state with regard to the processing of Sync and Follow_Up, timestamp measurements would be made for messages on this link, and the Pdelay mechanism would be used to measure propagation delay on this link. This general description is used in IEEE 1588, Version 2, because general PTP systems allow for standalone P2P TCs. However, this description would be more complicated than necessary for AVB networks. For example, since the

[2] In a general PTP system, it is possible for multiple TCs to be connected to a non-PTP device in a star configuration, with the non-PTP device as the hub and the TCs as spokes. If more than 2 of the TCs are P2P TCs, the situation is referred to as a 1:N configuration. Version 2 of IEEE 1588 will not allow 1:N configurations of P2P TCs. In an AVB network, all the nodes must be AVB-enabled and therefore will contain P2P TCs; therefore, 1:N configurations will not arise in AVB networks (at least, for the present AVB, using wired Ethernet links).

OC and P2P TC functions in an AVB NE are collocated, the propagation delay on the functional link connecting them is zero and the timestamp measurements can be made directly on the Sync and Pdelay messages that leave the combined node (right-hand part of Figure 1). In addition, we can consider the state of the combined node and of each link with regard to the Sync and Follow_Up messages (i.e., master or slave state) but not with regard to the Pdelay messages (since P2P TCs are stateless). In the description of the processing of Sync and Follow_Up messages in Section 4.1, we first use the former, general approach; this resembles the description for general PTP systems that include TCs given in [3] (but following the description in [4] to specialize the operation to AVB systems). We then give a simplified description that takes advantage of the fact that both functions are present in an AVB network node.

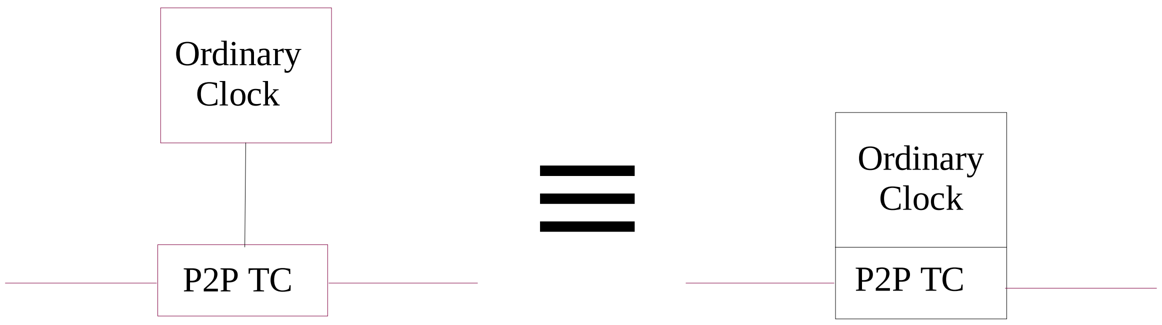


Figure 1. Illustration of implied single connection between an OC and the collocated P2P TC

The PTP protocol executes within a logical scope called a subdomain. All PTP messages, data sets, state machines and any other PTP artifacts are always associated with a particular subdomain. In general, a given physical network and individual devices connected to the network can be associated with multiple subdomains. The time established within a subdomain by the PTP protocol is independent of the time in other subdomains. In the case of an AVB network, the entire network will consist of a single subdomain.

2.2 AVB synchronization overview

There are two phases in the normal execution of the PTP protocol in an AVB network:

a) Establishing the master-slave hierarchy, and
b) Synchronizing the clocks.

2.2.1 Establishing the master-slave hierarchy

In an AVB network, each ordinary clock examines the contents of all Announce messages received. Using the best master clock (BMC) and state decision algorithms ~~these~~ the contents of the Announce messages and the contents of the data sets associated with the OC are analyzed to determine the state of the single implied OC port and of the OC. This process establishes one OC as the GM and the other OCs as slaves, in the single subdomain of the AVB network, as illustrated in Figure 2. Note that the P2P TCs are not part of the master-slave hierarchy.

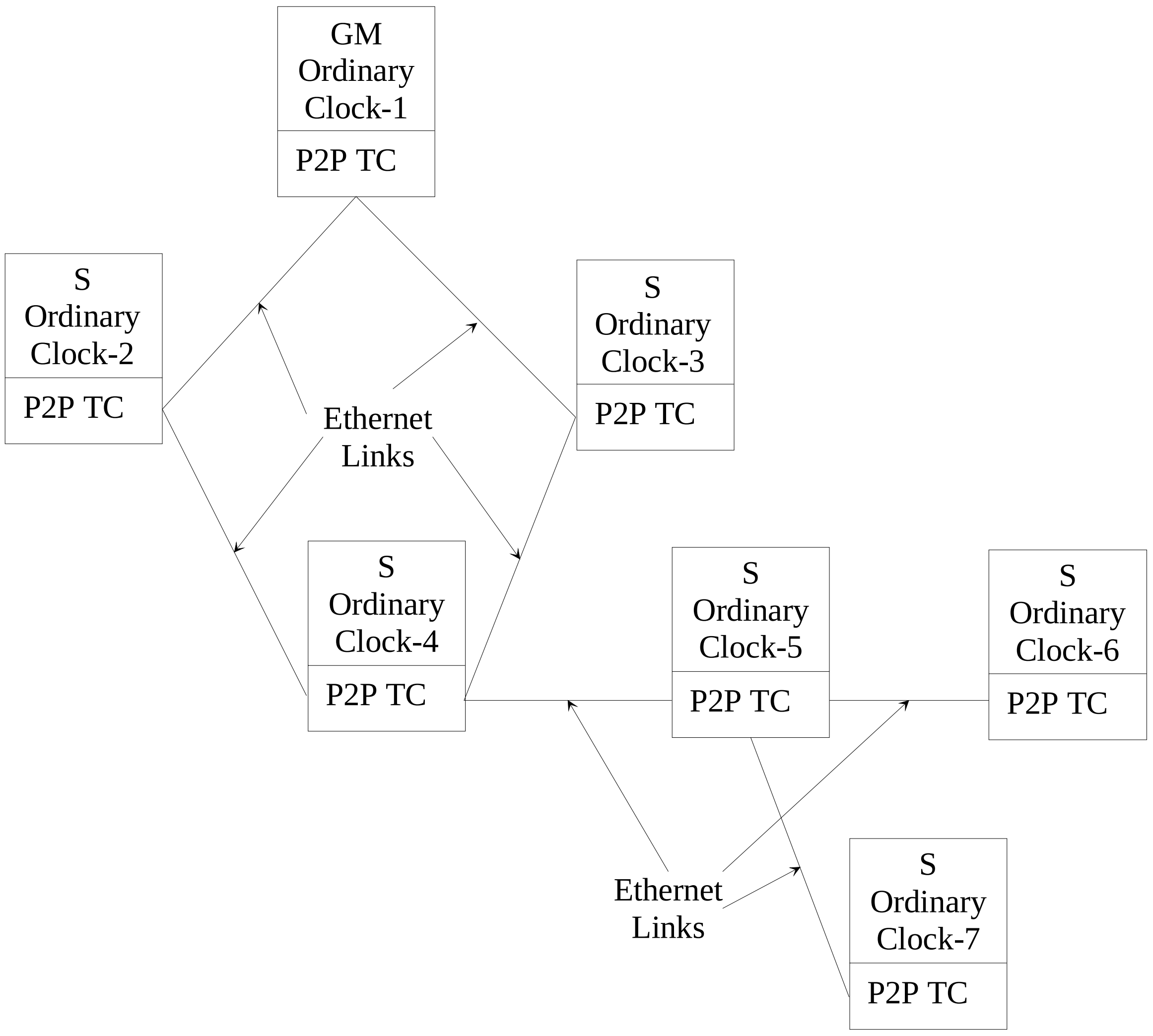


Figure 2. Illustration of AVB clock hierarchy

In Figure 2, the GM and slaves communicate via the Ethernet links. Each node is assumed to have a forwarding database, as would be present in conventional Ethernet, that enables unicast and multicast messages to reach their intended destinations without cycling endlessly. While Figure 2 shows a mesh topology, with clocks 1 through 4 forming a loop, the forwarding databases will reduce this to a spanning tree by not using one of the four links connecting these clocks. The manner in which the forwarding databases are constructed is not specified in IEEE 1588, Version 2. One way in which the forwarding databases can be constructed is through the use of dynamic MAC address learning for unicast addresses and GMRP[3] registration for multicast addresses, with spanning tree algorithm used to avoid loops, but IEEE 1588, Version 2 does not require this.[4]

[3] GMRP stands for GARP Multicast Registration Protocol. GARP stands for Generic Attribute Registration Protocol.

[4] In IEEE 1588, Version 1, TCs are not specified; a node is either a BC or an OC. In addition, BCs do not forward PTP messages. Therefore, the issue of constructing a forwarding database to ensure that PTP messages reach their intended destinations does not arise in a 1588 Version 1 network that has only BCs and OCs (i.e., no TCs and no non-PTP devices). This is because all communication is point-to-point in such a case. Also in such a case, the application of the BMC algorithm is more complicated than in Figure 2 above because now the master-slave hierarchy has multiple levels; nonetheless, the BMC algorithm establishes the

The network in Figure 2 is equivalent to a single PTP communication path. A PTP communication path supports the direct communication among the ports of ordinary and boundary clocks. Boundary clocks do not forward Sync, Follow_Up Delay_Req, Delay_Resp, or Announce messages, and different ports of a boundary clock are on different PTP communication paths. However, for simplicity AVB networks will not contain boundary clocks[5] (and no BCs are present in Figure 2).

2.2.1.2 Description of best master clock algorithm

To be supplied.

2.2.2 Synchronizing the clocks

In a PTP system, a master and slave clock synchronize by exchanging timing messages. For example, in Figure 2 the GM (Ordinary Clock-1) synchronizes a slave, e.g., Ordinary Clock-2, by exchanging messages with the slave. We first describe how a master and slave are synchronized in Version 1 of IEEE 1588, i.e., where the master and slave are directly connected and there are no TCs in between. We then consider the case of an AVB network, where P2P TCs might be present (we do not consider E2E TCs as these are not used in AVB networks). As part of the latter, we consider (1) synchronization in general PTP systems that use P2P TCs, (2) measurement of link propagation times using the ~~Peer_D~~Pdelay mechanism, (3) processing Sync and Follow_Up messages at P2P TCs in AVB, (4) syntonizing P2P TCs to the GM, (5) architectural considerations for AVB node synchronization, and (6) application filter requirements. Note that some of the items described here, e.g., aspects of (3), (4), (5), and (6) are not specified for general PTP systems and are properly part of a 1588 profile for AVB networks.

2.2.2.1 Synchronizing a master and slave that are directly connected

Consider a master and slave that are directly connected, i.e., have no TCs and no non-PTP devices between them. The basic pattern of synchronization message exchange is illustrated in 3.

hierarchy. In Figure 2, the master-slave hierarchy and application of the BMC algorithm is much simpler than in general IEEE 1588, Version 1 networks that have multiple BCs, but the network of Figure 2 does need to ensure that the PTP messages are forwarded properly. Finally, note that Version 1 networks that have non-PTP devices also must make sure that the PTP messages are forwarded properly (and the specification of this is beyond the scope of IEEE 1588).

[5] In principle, there is no reason why AVB networks could not contain Boundary Clocks (just as there is no reason why any 1588 network could not contain boundary clocks). The decision that AVB networks will use only P2P TCs with collocated OCs was made to limit the number of 1588 architectural options and keep the AVB network simple. This will help to ensure both low cost and ease of administration by users (consistent with the 802.1as PAR [1].

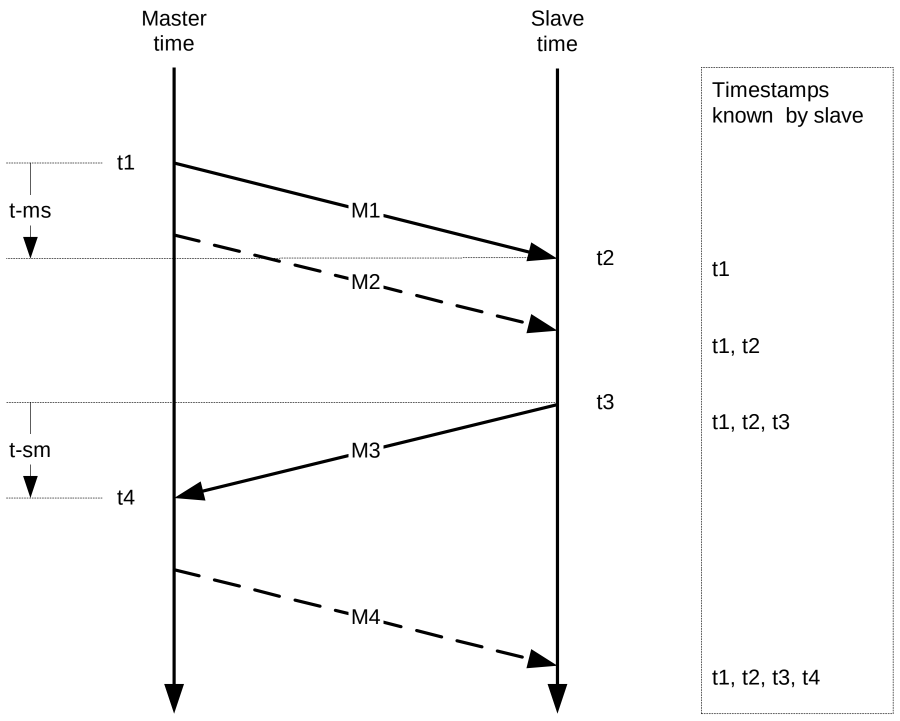


Figure 3. Basic synchronization message exchange (taken from[9])

The message exchange pattern is (items (a) – (f) below are taken from [9]):

a) The master sends a message M1, referred to as Sync, to the slave and notes the time, t1, at which it was sent.

b) The slave receives the message M1 and notes the time of reception, t2.

c) The master conveys to the slave the timestamp t1 by:
   1) Embedding the timestamp t1 in message M1. This requires some sort of hardware processing for highest accuracy, or
   2) Embedding the timestamp t1 in a second message M2, referred to as Follow_Up. This can be done in software since the timing is not critical.

d) The slave sends a message M3, referred to as Delay_Req, to the master and notes the time, t3, at which it was sent.

e) The master receives the message M3 and notes the time of reception, t4.

f) The master conveys to the slave the timestamp t4 by embedding it in a message M4, referred to as Delay_Resp.

At the conclusion of this exchange of messages, the slave possesses all four timestamps. These timestamps may be used to compute the offset of the slave's clock with respect to the master and the mean propagation time of messages between the two clocks, that is the mean of t-ms and t-sm in 3. The computations are

$$t_{ms} = t_2 - t_1$$
$$t_{sm} = t_4 - t_3$$
$$mean\_propagation\_time = \frac{t_{ms} + t_{sm}}{2} \qquad . \qquad (2\text{-}1)$$
$$slave\_offset = t_2 - t_1 - mean\_propagation\_time$$

The expression for slave offset assumes that the master-to-slave and slave-to-master propagation times are equal. Any asymmetry in propagation time will introduce an error in the computed value of slave offset.

The accuracy of the slave offset computation also depends on how accurately the times t_1, t_2, t_3, and t_4 are measured, i.e., the computations in Eqs. (2-1) depend on these times reflecting when the Sync and Delay_Req messages actually are sent and received. This means that the measurement must be made below the Ethernet mac, i.e., at the MII or GMII. In addition, if Follow_Up is not used, i.e., if an accurate measurement of t_1 is placed in the Sync message, the hardware will be more expensive. For this reason, AVB ~~will~~ may use the Follow_Up message (i.e., use of the Follow_Up message will be allowed, though on-the-fly operation will not be precluded).

This basic exchange of messages is used in two ways in the PTP protocol:

g) To synchronize between an ordinary clock and a boundary clock, and

h) To measure link propagation time.

The propagation times usually vary very slowly, if at all. Therefore, it is often possible to perform the Delay_Req/Delay_Resp message exchange much less frequently than the sending of Sync and, if implemented, Follow_Up. In this case, multiple Sync and Follow_Up messages are sent between successive Delay_Req/Delay_Resp exchanges. The mean propagation time measured by a Delay_Req/Delay_Resp exchange is used by the slave in all subsequent clock offset calculations until the next Delay_Req/Delay_Resp exchange.~~5~~ This is illustrated in Figure 4 where, in each computation of slave offset after a Sync and Follow_Up message are received, the values of t_1 and t_2 for that Sync and the most recently measured mean propagation time are used in the final equation of Eq. (2-1).

The time between successive Sync messages is referred to as the synch interval.

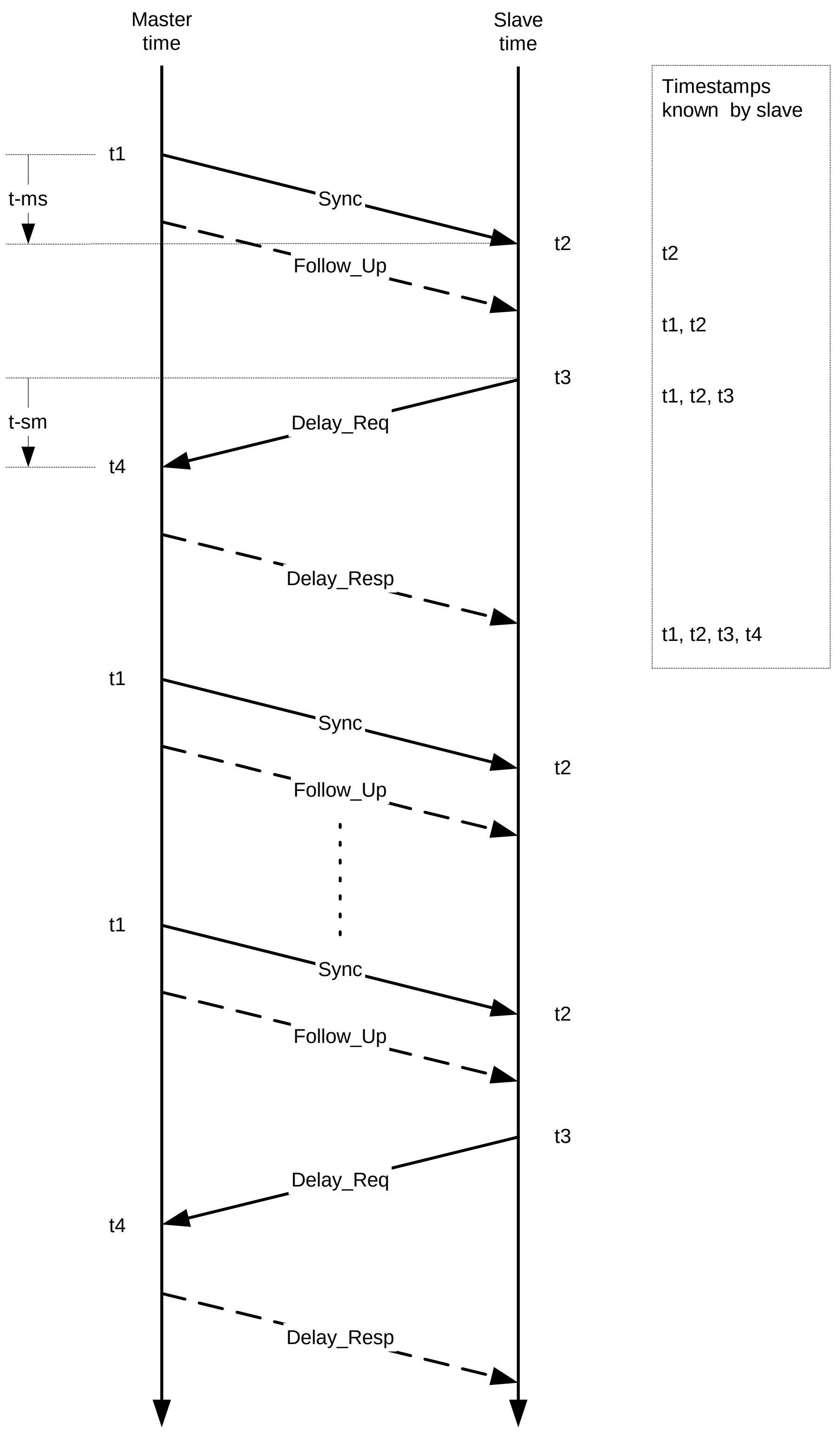


Figure 4. Basic synchronization message exchange, showing multiple Sync and Follow_Up messages between two successive Delay_Req/Delay_Resp exchanges.

2.2.2.2 Synchronizing a master and slave that communicate through one or more P2P TCs

If there are one or more P2P TCs between the master and slave, the master-to-slave and slave-to-master propagation times will, in general, not be equal and will vary appreciably with time. The reason for this is that the Ethernet links in Figure 2 will also carry application traffic; in fact, this will be the majority of the traffic in the network. Even if the PTP messages have highest priority,[6] the priority will be non-preemptive; a large Ethernet frame in service when it is desired to send a Sync message from a TC node can result in addition delay whose value will be between zero and the transmission delay for this frame. For a maximum sized Ethernet frame (1500 bytes of payload), the maximum value for this delay is nearly 125 μs for 100 Mbit/s Ethernet. This is much larger than the delays due to propagation through the PHY and wire.

The TC solves the above problem by measuring the time the Sync message arrives, t_a, and the time the Sync message departs, t_d, and computing the difference, $t_r = t_a - t_d$. This difference is referred to as the residence time. The residence time is accumulated in a field of the Sync or Follow_Up message referred to as the correction field. Specifically, the correction fields of the Sync message and Follow_Up message, respectively, are initialized to zero when the GM creates those messages. When a TC measures residence time, it has two choices. If it is able to make a sufficiently accurate measurement of t_r and add the value of this measurement to the correction field of the Sync message, i.e., if it is an on-the-fly TC, it does this. In this case, it does not alter the correction field of the Follow_Up message when that message is received and sent by the TC. However, if it is a follow-up TC, i.e., if it cannot make an accurate residence time measurement sufficiently quickly to add it to the value of the Sync message correction field as the Sync message is transmitted, it instead adds the residence time to the correction field of the Follow_Up message. As it is more expensive to make on-~~th~~the-fly measurements of residence time, AVB P2P TCs ~~will~~ are allowed to be follow-up TCs.[7]

The P2P TCs also measure propagation times on the Ethernet links that connect them using the three ~~Peer_D~~Pdelay messages, referred to as ~~Peer_D~~Pdelay_Req, ~~Peer_D~~Pdelay_Resp, and ~~Peer_D~~Pdelay_Resp_Follow_Up. The details of this measurement will be described shortly; however, the result is that each P2P TC at the end of a link knows the propagation time on that link. A P2P TC will also accumulate in the correction field of the Sync or Follow_Up message the propagation time for the link on which the Sync message arrived. Specifically, an on-the-fly P2P TC will accumulate the propagation time in the Sync correction field, and a follow-up P2P TC will accumulate the propagation time in the Follow_Up correction field. When the Sync and Follow_Up messages reach their final destination slave clocks, the sum of the correction fields in these messages is a measure of the total residence time in all the intervening TCs plus the propagation times on all the links except the final ingress link at the destination. This is illustrated in the example in Figure 5~~4~~.

[6] AVB will use the priority mechanism of IEEE 802.1D and 802.1Q.

[7] ~~Actually, O~~O~~o~~n-the-fly measurements will be allowed, but will not be required.

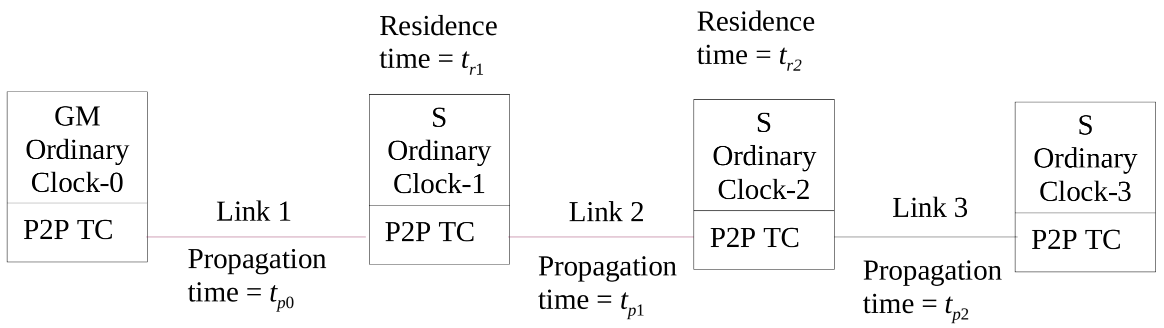

Residence time = t_{r1}
Residence time = t_{r2}
GM Ordinary Clock-0
P2P TC
Link 1
Propagation time = t_{p0}
S Ordinary Clock-1
P2P TC
Link 2
Propagation time = t_{p1}
S Ordinary Clock-2
P2P TC
Link 3
Propagation time = t_{p2}
S Ordinary Clock-3
P2P TC

Figure ~~5~~4. Illustration of accumulation of residence times and propagation times by P2P TCs

Figure ~~4~~ 5 consists of a GM (ordinary clock 0) and three slave clocks (ordinary clocks 1 –3). The link connecting clocks i and i+1 is labeled link i+1. The residence time in clock i is denoted t_{ri}, and the propagation time on link j is denoted t_{pj}. If clock N is the destination clock (i.e., there are a total of N+1 ordinary clocks, with N = 3 in the example here), then residence times are measured in clocks 1 through N-1, and link propagation times are measured by the ~~Peer_D~~Pdelay mechanism on links 1 through N. When the Sync and Follow_Up messages arrive at clock N, the sum of their correction fields is given by

$$Sync\_correction\_field + Follow\_Up\_correction\_field = \sum_{i=1}^{N-1} \left(t_{ri} + t_{pi} \right). \qquad (2\text{-}2)$$

The reason the propagation time on the final link is not included in Eq. (2-2) is that the propagation time is added to the correction field when the Sync or Follow_Up message is transmitted from the node, but at the final destination the Sync and Follow_Up messages are not transmitted to any additional nodes. However, the propagation time on this final link, t_{pN}, is known at the destination node, because the P2P TC in that node (node N) participates in a ~~Peer_D~~Pdelay measurement with the TC at the other end of the link (node N-1). Therefore, the propagation time may be easily added to the sum in Eq. (2-2) to give

$$total\_propagation\_plus\_residence\_time = \sum_{i=1}^{N-1} t_{ri} + \sum_{i=1}^{N} t_{pi} \,. \qquad (2\text{-}3)$$

The slave offset is now computed as

$$slave\_offset = t_2 - t_1 - total\_propagation\_plus\_residence\_time \,, \qquad (2\text{-}4)$$

where we have reverted to the notation of Figure 3 for the times the Sync message is transmitted by the GM and received at the final destination, i.e., t_1 is the time the Sync message is transmitted from the GM (ordinary clock 0), t_2 is the time the Sync message is received by the destination slave clock (ordinary clock N, with N = 3 in Figure 5~~4~~), and *total_propagation_plus_residence_time* is given by Eq. (2-3).

Note that even though a P2P TC is present at the GM, it does not measure residence time. This is because the time t_1 that the Sync message departs the GM network element occurs after the message traverses the P2P TC (because the GM and P2P TC functions are located in the same NE).

With this mechanism, there is no need for the Delay_Req and Delay_Resp messages shown in Figures 3 and 4. Instead, the propagation times are computed using the ~~Peer_D~~Pdelay messages, which are described shortly.

In an actual AVB network, all the slave clock nodes must be synchronized; in addition, the topology will, in general, be a ~~mesh~~ tree topology, which may be reduced from a mesh topology (Figure 2), rather than a linear topology (Figure 54). PTP messages are, by default, multicast. This means that the GM sends a single Sync and a single Follow_Up message to all the nodes; it does not have to generate a separate Sync and separate Follow_Up message for each node. The forwarding database ensures that each slave clock receives each Sync and each Follow_Up message that the GM transmits, and also that Sync and Follow_Up messages do not cycle endlessly. The multicast mechanism means that a Sync message received by a P2P TC on one port may be transmitted on multiple ports. If this is done, the transmission time is measured separately for each port that the message is transmitted on, and a separate residence time is computed for each transmitted port. If the P2P TC is on-the-fly, the correction field of the transmitted Sync message is updated separately on each port using the respective residence time for that port and propagation time for the link attached to the port the message arrived on. When the corresponding Follow_Up message arrives, it is transmitted on the same ports that the Sync message was transmitted on. If the P2P TC is a follow-up TC, the correction field of the transmitted Follow_Up message is updated separately on each port using the respective residence time for that port and propagation time for the link attached to the port the message arrived on.

[***Author's Note: The fact that the PTP messages are multicast likely means that some form of multicast address registration at the AVB nodes is needed. One way to do this is with GMRP There has been initial discussions of the multicast address mechanism in the 4/19/2006 AVB call and via email. Further work is needed, and the mechanism must be described in more detail.***]

2.2.2.3 Measurement of link propagation times using ~~Peer_D~~Pdelay mechanism

~~This section will be filled in. The Short Frames group is considering a 2 timestamp format for the Peer_Delay messages. Note: will the mechanism now be symmetric?~~
The description of this subsection, and Figure 6, are taken from Section 6.5.3 of [9] (with minor modifications made to the text). The mechanism for measuring the propagation time between two P2P TCs using the Pdelay mechanism is illustrated in Figure 6. The measurement is made by each port at the end of every link. Thus, both ports sharing a link will independently make the measurement and both ports will know the propagation time as a result. This allows the corrections described above in Section 2.2.2.2 to be made irrespective of the direction taken by a Sync message. It is important that the propagation time measurement occur even on ports otherwise blocked by non-PTP algorithms used to eliminate cyclic topologies. [***Author's Note: It was indicated in the 4/19/2006 AVB call that the multicast address mechanism in 802.1 bridges (and in most networking technologies) does not permit multicast messages to be sent on blocked ports. Since the Pdelay messages are multicast by default, this implies that these could not be sent on blocked ports. This should not be an issue for AVB, as in the event of a reconfiguration, the propagation delay on a link that becomes unblocked would be learned within one Pdelay message exchange time. Nonetheless, this issue should be investigated more fully in looking at how multicast addressing will work for the PTP messages used in AVB, to determine if it really is a limitation.***]

The propagation time measurement starts with port-1 (Figure 6) issuing and generating a timestamp, t1, for a Pdelay_Req message. Port-2 receives and timestamps, t2, this message. Port-2 returns and timestamps, t3, a Pdelay_Resp message. Port-2 returns the timestamps t2 and t3 either in the

Pdelay_Resp (which requires hardware assist) or in a Pdelay_Resp_Follow_Up message (as is the case for the use of Follow_Up, the use of Pdelay_Resp_Follow_Up will be allowed, though on-the-fly measurement will not be precluded). Port-1 generates a timestamp, t4, upon receiving the Pdelay_Resp message. Port-1 then uses these four timestamps and the first three equations of Eq. (2-1) to compute the mean propagation time.

There will be an error in the propagation time measurement equal to the frequency offset between the two P2P TC nodes multiplied by one-half the time interval between the receipt of Pdelay_Req and the sending of Pdelay_Resp. In general 1588 systems where the P2P TC nodes may not be syntonized, this means that the turnaround time between the receipt of Pdelay_Req and the sending of Pdelay_Resp should be as short as possible, to minimize this source of error. However, this is not expected to be an issue for AVB networks because the AVB nodes will be syntonized.

The Pdelay messages may be sent less frequently than the Sync and Follow_Up messages, as propagation times are not expected to vary rapidly. They will be sent every *P* sync intervals, where *P* is to be determined. There are several possibilities for the synchronization of the sending of these messages: (1) Port-1 sends the messages at times determined by the local free-running oscillator, in which case the measurements of propagation time by the various ports on the various nodes are occurring at times that are not synchronized and rates that are not syntonized, (2) Port-1 sends the messages at times determined by the syntonized frequency that is synthesized from the free-running oscillator times and the Sync and Follow_Up messages received from the GM, in which case the measurements of propagation time by the various ports on the various nodes are occurring at times that are not synchronized but rates that are syntonized, and (3) Port-1 sends the messages at times determined by the synchronized clock (i.e., after adding the slave offset), in which case the measurements of propagation time by the various ports on the various nodes are occurring at times that are synchronized.

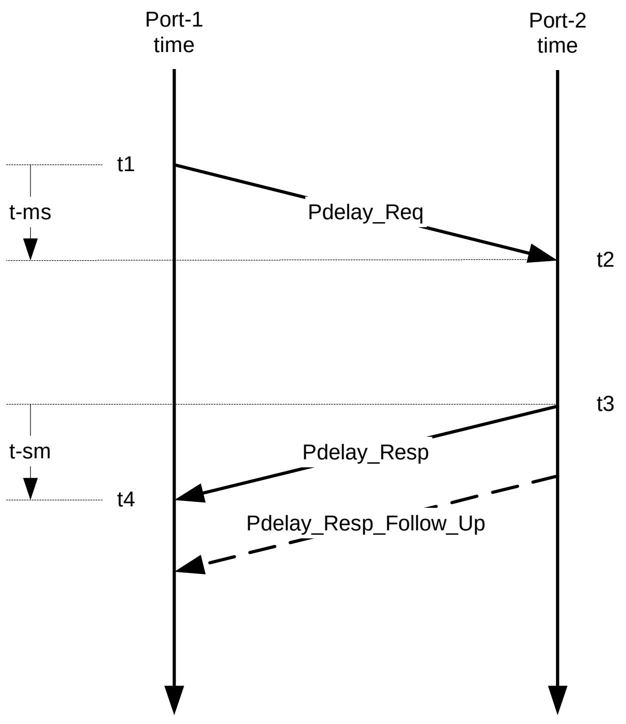
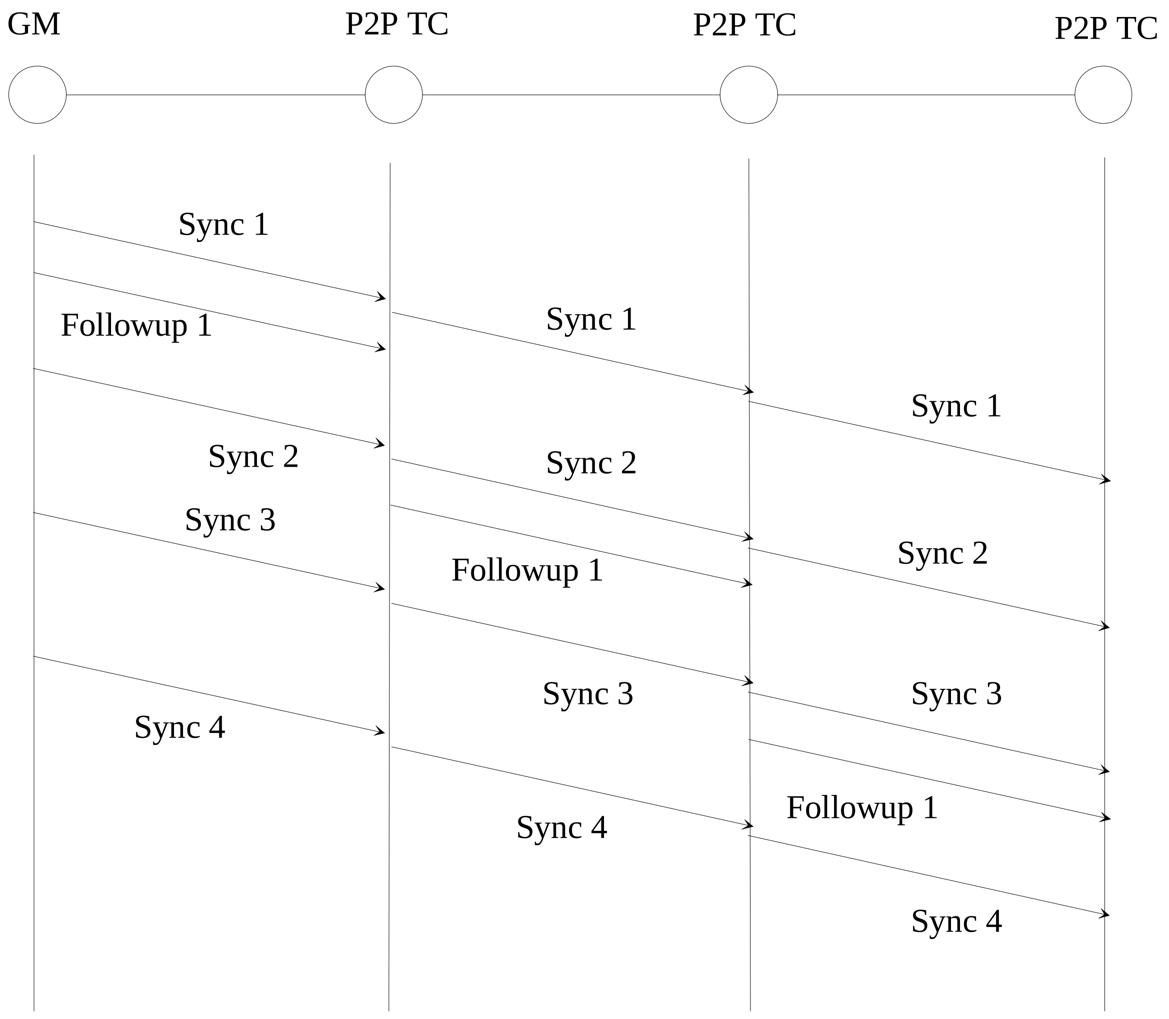


Figure 6. Propagation time measurement using Pdelay mechanism

2.2.2.4 Processing of Sync and Follow_Up messages at P2P TCs in AVB

AVB network nodes will be assumed to have standard Ethernet oscillators, with nominal rates of 25 MHz for 100 Mbit/s Ethernet and 125 MHz for 1 Gbit/s Ethernet. This means that the phase measurement granularity in the TC and OC can be as much as 40 ns. Additional phase error will result from the variable component of latency in the Ethernet PHY (the fixed component can be specified by the manufacturer in the design). Some AVB applications have stringent jitter and wander requirements. For example, uncompressed digital video has jitter requirements of less than 1 ns peak-to-peak measured through a 10 Hz low-pass jitter measurement filter, and maximum frequency offset and drift of 0.23 ppm and 0.023 ppm/s, respectively [10]. Consumer grade digital audio has a jitter requirement of 10 ns peak-to-peak measured through a 10 Hz low-pass jitter measurement filter, and maximum frequency offset of 50 ppm [10]. Additional details on these requirements are given in [10]. It is expected that a Sync interval as short as 10 ms or less will be needed in AVB networks to meet the jitter and wander requirements.

AVB network nodes will also use an inexpensive processor, e.g., the 8051. While Sync messages in theory require minimal processing by a Follow-up P2P TC (i.e., the TC needs only to measure the arrival and departure times of the Sync message), Follow_Up messages require more processing. It is expected that the 8051 processor may take up to 10 ms to process a Follow_Up message, depending on the total load of the processor. This means that for a path through *N* PTP TCs (excluding the GM and final slave node), it ~~will require at least~~may take on the order of 10*N* ms for the Followup message to travel from the master to the slave. Since the Sync messages require little or no processing, they will travel from the master to slave in much less than 10*N* ms (the time for the Sync to travel from the master to the slave will likely be on the order of the AVB latency requirement; values in the range of 2 – 6 ms have been discussed). Figure 7~~5~~ illustrates this scenario for the case of 2 follow-up TCs between the GM and final slave node, i.e., *N* = 2.



Note: The Followup messages corresponding to Sync2 and Sync3 are not shown to keep the diagram from being too cluttered.

Figure 7~~5~~. Illustration of the accumulation of multiple Sync messages at a node before the Follow_Up corresponding to the first Sync arrives. Follow_Up processing time and sync interval are of the same order. Sync processing time is much less than Follow_Up processing time. All TCs are assumed to be follow-up TCs.

In Figure 7~~5~~, when Follow_Up 1 (corresponding to Sync 1) arrives at the final node, three Sync messages have arrived (including Sync 1. It is seen that with each successive hop, the Followup message processing delay causes one additional Sync message to get ahead of the Followup message. If the master sends Sync every 10 ms, this means that each P2P TC would have to maintain state information on the residence times for multiple Sync messages at any given time (the final TC in the chain would have to save information for as many as *N*+1 Sync messages). Note that the Follow_Up message processing time occurs at each successive P2P TC but not at the GM at the beginning of the chain; it is assumed that the GM can send Follow_Up almost immediately after sending Sync (i.e., in a time short compared to the sync interval). If the Follow_Up processing time were also incurred at the GM, then one additional Sync message would have arrived when the Follow_Up message arrives at each TC, i.e., the final TC in the chain would have to save state information for as many as *N*+2 Sync messages.

The need to save state information on multiple Sync messages at each P2P TC may be avoided by having each Sync message held at a TC until the corresponding Follow_Up message arrives. When the Follow_Up message arrives, its correction field is added to the correction field of the Sync, and the Sync is sent. The time the Sync is sent is noted; using this and the time the Sync arrived, a residence time for the Sync in the current TC node is computed. A new Follow_Up message is generated, and the sum of the residence time just measured and the delay on the upstream link on which the Sync message arrived is placed in the correction field of the Follow_Up message. This approach is illustrated in Figure 8~~6~~.

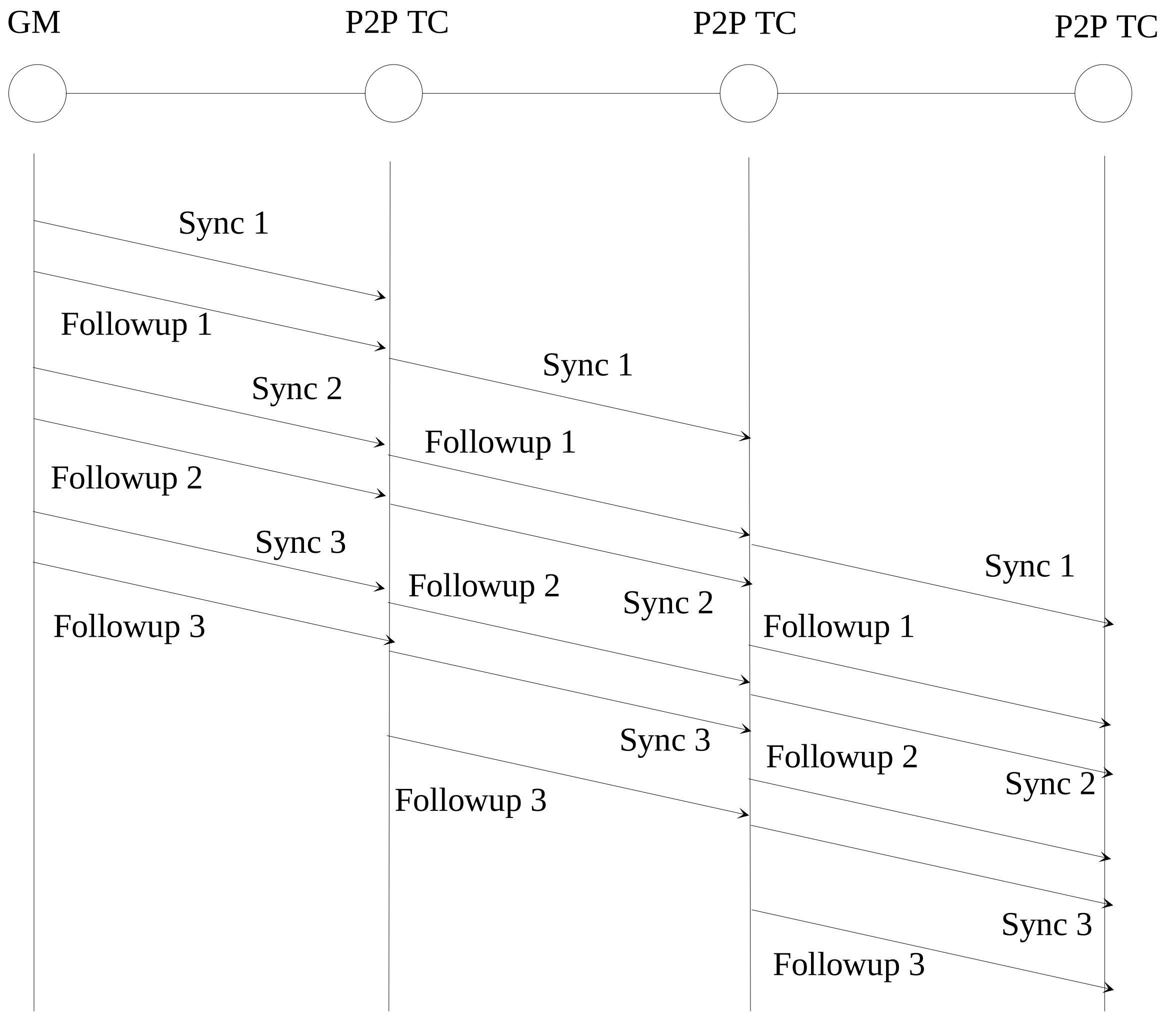


Figure 8~~6~~. Sync and Follow_Up sent with Sync held at P2P TC until corresponding Follow_Up arrives. Follow_Up processing time and sync interval are of the same order. Sync processing time is much less than Follow_Up processing time.

In Figure 8~~6~~, successive Sync messages do not get ahead of Follow_Up messages associated with previous Sync messages. Note that in both cases the Follow_Up message processing time occurs at each successive P2P TC but not at the ~~BC~~ GM at the beginning of the chain; it is assumed that the ~~BC~~ GM can send Follow_Up almost immediately after sending Sync (i.e., in a time short compared to the sync interval). If the Follow_Up processing time were also incurred at the ~~BC~~GM, the Synch messages would be held longer at the first P2P TC in Figure 8~~6~~.

With the approach of Figure 8~~6~~, a Sync message will not get ahead of a Followup message for a previous Sync as long as the time between Sync messages is not shorter than the time required for processing a Followup message at a node (i.e., 10 ms). When a Sync message is sent, at least 10 ms will have elapsed since the previous Sync message, which means that the Followup message will have had enough time to be processed at the next downstream node. It is a general requirement for all PTP systems that use P2P TCs that process Follow_Up messages that the time to process a Follow_Up message not exceed the Sync interval. If this requirement is not met, then an increasing backlog of Follow_Up messages will accumulate over time at each node that is traversed by Sync and corresponding Follow_Up messages.

The approach of Figure 8~~6~~ is fully consistent with the semantics of how a P2P TC handles Sync and Follow_Up, i.e., Eqs. (2-2) – (2-4). Both the approach of Figure 8~~6~~ and the conventional approach (Figure 7~~5~~) result in the sum of the Sync and Follow_Up correction fields being the same on arrival at any P2P TC. Therefore, these equations may be used to compute the slave offset in either approach.

2.2.2.5 Syntonizing the P2P TC to the grandmaster

A P2P TC contains a free-running oscillator with frequency accuracy no worse than ± 100 ppm. If residence time is measured using this oscillator, there will be an error on the order of the residence time multiplied by the actual frequency offset. With the approach of Figure 8~~6~~ described in the previous subsection, the residence time may be on the order of a synch interval, e.g., as much as 10 ms, due to the holding of a Sync message at each P2P TC until the corresponding Follow_Up message arrives. This can result in an error in the residence time measurement on the order of (100000 ns/s)(0.01 s) = 1000 ns. To reduce this error, IEEE 1588 Version 2 allows the P2P TC to be syntonized, i.e., synchronized in frequency, to the Grandmaster. While syntonization of the P2P TC to the GM is not mandatory in IEEE 1588, it will be mandatory for AVB networks.

A P2P TC will syntonize to the GM by comparing a time interval measured by the GM with the same time interval measured by the local, free-running oscillator of the P2P TC. The time interval is equal to *M* Sync intervals. At present, *M* = 10, though this may change based on the results of jitter and wander performance simulations. The measurement is done as follows. The P2P TC already must measure when each Sync message arrives in order to compute the residence time. An estimate of the GM time when the Sync message arrives is given by

$$t_{GM} = t_1 + total\_propagation\_plus\_residence\_time\,, \qquad (2\text{-}5)$$

where t_1 is the time the GM sends the Sync message as defined in Figure 3 and *total_propagation_plus_residence_time* is given by Eq. (2-3) and is computed as the sum of the correction fields in the Sync and corresponding Follow_Up message plus the propagation time on the link that the Sync message arrived on (this computation may be seen to be equivalent to Eq. (2-3) by comparing Eq. (2-3) with Eq. (2-2)). Note that in order to compute t_{GM}, the P2P TC must wait until the Follow_Up message corresponding to the Sync message arrives. ~~If t_2 is the time the time the Sync message arrives, and if *i* indexes the synch interval at which the measurement is made, than a measurement of the frequency offset of the P2P TC free-running oscillator relative to the GM is given by~~

~~$$y_{TC,i+M} = \frac{t_{2,i+M} - t_{2,i}}{t_{GM,i+M} - t_{GM,i}}\,. \qquad (2\text{-}6)$$~~

Define the following notation:

$t_{GM,i}$ = estimated GM time at sync interval *i* based on the received Sync and Follow_Up messages (and using Eq. (2-5))
$t_{b,i}$ = time indicated by free-running oscillator in P2P TC (analogous to the quantity *basetimer* in [11])
$y_{TC,i}$ = measured frequency offset of GM relative to free-running oscillator in P2P TC (note that we define this as the frequency offset of the GM relative to the P2P TC rather than vice-versa only for convenience (we could have used the opposite convention, in which case the resulting equations below would be slightly more complicated)
$t_{f,i}$ = syntonized time, synthesized from the measured frequency offset and the time indicated by the free-running P2P TC oscillator (analogous to the quantity *flextimer* in [11])

The frequency offset is equal to the fractional part of the ratio of the elapsed time indicated by the GM and the elapsed time indicated by the free-running P2P TC oscillator. It is given by

$$y_{TC,kM} = \frac{t_{GM,kM} - t_{GM,(k-1)M}}{t_{b,kM} - t_{b,(k-1)M}} - 1 \quad . \qquad (2\text{-}6)$$
$$y_{TC,kM+i} = y_{TC,kM} \quad \text{for } i = 1,2,\ldots,M$$

In Eq. (2-6), the frequency offset is measured every *M* sync intervals, and the current measured value is used at all sync intervals until the next measurement.

The syntonized time, i.e., syntonized to the GM, is synthesized under the assumption that the frequency offset of the GM relative to the free-running P2P oscillator has been equal to the current measured value since that measurement was made. Then

$$1 + y_{TC,kM} = \frac{t_{f,kM+i} - t_{f,kM}}{t_{b,kM+i} - t_{b,kM}} \quad \text{for } i = 1,2,\ldots,M \; . \qquad (2\text{-}7)$$

Eq. (2-7) may then be used to solve for the syntonized time at an arbitrary sync interval between two successive frequency offset measurements

$$t_{f,kM+i} = t_{f,kM} + (1 + y_{TC,kM})(t_{b,kM+i} - t_{b,kM}) \quad \text{for } i = 1,2,\ldots,M \; . \qquad (2\text{-}8)$$

In applying Eq. (2-8), the value of free-running oscillator time and the syntonized time when the frequency offset measurement is made, $t_{b,kM}$ and $t_{f,kM}$ respectively, are saved until the next frequency offset measurement is made. If it is desired to compute the syntonized time at a sync interval in terms of only the syntonized time at the previous sync interval and the free-running times at the current and previous sync intervals, an alternative result may be obtained by replacing *i* by *i*-1 in Eq. (2-8) and subtracting this from Eq. (2-8)

$$t_{f,kM+i} = t_{f,kM+i-1} + (1 + y_{TC,kM})(t_{b,kM+i} - t_{b,kM+i-1}) \quad \text{for } i = 1,2,\ldots,M \; . \qquad (2\text{-}9)$$

Note that Eqs. (2-6) – (2-9) hold for *i* = 1, 2, …,*M*. When *i* = *M*, a new frequency offset measurement is made, *k* is increased by 1, *i* is set to zero, and the process begins again.

Note that each P2P TC needs only its own local free-running oscillator phase information and the information conveyed by the GM in the Sync and Follow_Up messages. It is not necessary for any P2P TC to convey its local free-running phase information to any other P2P TC or to the GM.

~~Each~~ In applying Eqs. (2-6) – (2-9), a P2P TC ~~will use~~is, in effect, using the measured frequency offset ~~relative to~~of the GM relative to its free-running oscillator to synthesize a frequency signal that is syntonized with the GM. This synthesis, i.e., the implementation of Eqs. (2-6) – (2-9), may be done via hardware, firmware, or software.

The syntonized time $t_{f,i}$ is used as the t_2 value in Eq. (2-4) to obtain slave offset. In addition, the syntonized time at the P2P TCs at the two ends of a link are used in the first two of Eqs. (2-1) in obtaining propagation times using the Pdelay mechanism (see Section 2.2.2.3).

2.2.2.6 AVB Node synchronization architecture

The above subsection describes how the P2P TC contains a free-running oscillator and syntonizes this to the GM frequency by measuring its frequency offset relative to the GM. Similarly, subsections 2.2.2.1, 2.2.2.2, and 2.2.2.4 describe how a slave clock synchronizes to the GM by measuring its phase offset relative to the GM; the specific computation is given by Eqs. (2-2) – (2-4). Eq.~~n.~~ (2-4) indicates that the slave clock measures the time the Sync message arrives, but is not specific on whether the slave clock uses the local free-running oscillator embedded in the P2P TC, the signal synthesized by the P2P TC that is syntonized to the GM, a separate free-running local oscillator in the slave, or the synchronized timing signal produced in the slave using the computed phase offsets. This subsection discusses the relative merits of each choice and concludes that the best approach for AVB is to use the syntonized timing signal synthesized by the P2P TC, i.e., as opposed to using a free-running oscillator embedded in the P2P TC or slave clock. The synchronized timing signal produced in the slave clock using the computed slave_offset may also be used for the timestamp measurements, though the performance difference between using the syntonized and synchronized signals will not be significant (see the paragraph following the next paragraph for details on this).

Since one of the requirements for AVB is low cost, it is desirable to have a single oscillator in an AVB NE for both the P2P TC and slave clock functions. If the P2P TC were not syntonizing to the GM, the slave could still synchronize using Eqs. (2-2) – (2-4). Even if there were no P2P TCs between the slave and GM, the fact that the slave and GM frequencies were different would result in a computed *slave_offset* (Eqs. (2-1) – (2-4)) on the order of the frequency offset between the free-running slave/TC oscillator and the GM multiplied by the synch interval. This could be as large as (100000 ns/s)(0.01 s) = 1000 ns. However, since the frequency offset between the GM and slave/PC oscillator is already being measured and a syntonized frequency is being created, the use of this frequency for the slave offset computation will greatly reduce the magnitude of the computed slave offset phase step. The phase step magnitude will now be on the order of the syntonized frequency measurement accuracy multiplied by the synch interval. For example, if the phase measurement granularity is 40 ns and the P2P TC oscillator offset is measured over 10 synch intervals, i.e., 100 ms, the error in measured frequency offset is 40×10^{-9} s/0.1 s = 400×10^{-9} = 0.4 ppm. The slave offset now is (400 ns/s)(0.01 s) = 4 ns, i.e., is reduced from the 1000 ns computed when the free-running frequency is used for the measurement by a factor of 250. In practice, the reduction will not be this large because other effects are present, e.g., oscillator phase noise and drifts due to temperature effects, phase measurement error due to the variable portion of the PHY latency, and frequency measurement granularity. A better estimate of the synchronization performance will be determined via simulation.

The syntonized signal has the same average frequency as the GM (except for measurement errors described above), but not necessarily the same phase. There may be a large phase difference between the time signal at the P2P TC syntonized to the GM and the GM signal itself (e.g., due to an initial phase offset). However, the only impact of this large initial phase offset is that it is added to the slave time measurement t_2, which is in error by the opposite amount. To see this, consider two cases for Figure 3 and Eq. (2-1). We assume the master and slave frequencies are syntonized in both cases. In the first case, the syntonized, but not synchronized, signal at the slave is used to make timestamp measurements. After the slave receives a Sync message and a Follow_Up message, and assuming propagation times have been measured with the Pdelay mechanism, the slave_offset is computed using the final equation in Eq. (2-1). If the phase measurement granularity were zero and there were no timestamp measurement errors and no clock phase error due to noise or temperature effects, the computed slave_offset would be the exact offset between the master and slave. The actual computed offset will be in error to the extent that these effects are non-zero. In any case. Adding this computed offset to the syntonized slave timing signal will produce a synchronized slave timing signal. If the synchronized timing signal is used for future timestamp measurements, the computed slave_offset will be very small (its actual magnitude will depend on how large the above effects are that produce errors in the offset measurement). If the syntonized timing signal is used for future timestamp measurements, the computed slave offset will simply be the static time difference between the syntonized slave signal and the master signal. ~~–~~The conclusion is that the synchronization performance will be very similar if one uses the syntonized signal to make the slave offset measurement versus a synchronized signal, i.e., a signal that has past computed slave offsets added back in.

Note that in the above paragraph we are referring to an *unfiltered* synchronized signal. After adding the slave offset to the signal used to make the slave offset measurement, the signal may be filtered to reduce any jitter and wander due to the offset addition. If the filtered signal were used to measure the message arrival times, performance could be improved. However, in AVB networks filtering will be application dependent, i.e., applications with more/less stringent jitter and wander requirements can require filters with tighter/looser bandwidth and gain peaking requirements. The reason for this is so that the cost of any expensive filtering will be borne by applications that need the filtering. AVB networks will either not require any level of filtering to be present by default (i.e., regardless of what applications are being demapped at the node) or any required filtering will be minimal. If filtering is not assumed to be present, there is likely no major performance difference between the cases where the syntonized and synchronized signals are used for the slave phase offset measurement.

2.2.2.7 Filtering the synchronized timing signal

The synchronized timing signal obtained by adding the measured slave phase offset to the syntonized (to the GM) signal used to measure the times of arrival and departure at/from the P2P TC will have jitter and wander due to the phase steps caused by the offsets. These signals may be filtered to reduce the jitter and wander. An y filter requirements will be application dependent, so that the cost of any expensive filtering will be associated with the application that requires it. Regardless of the level of filtering, any filter requirements may be expressed generically using a transfer characteristic. An example is shown in Figure 9~~7~~.

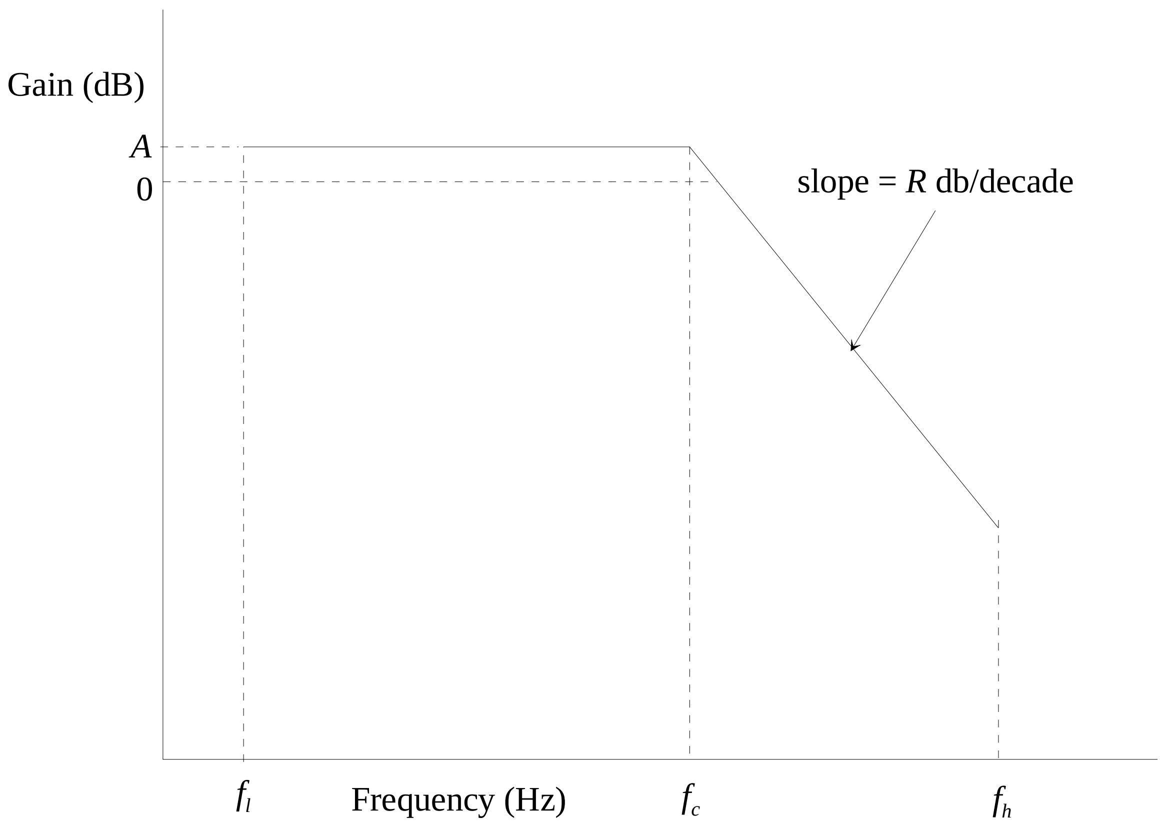


Figure 9~~7~~. Example transfer requirement for filter, specific to application

A filter used for the application in question would be required to have a transfer characteristic that is below the mask in Figure 9~~7~~. The filter would be tested by applying input signals of specified amplitude and frequencies ranging from f_l to f_h, measuring the amplitude of the output, and plotting the gain (in dB) as a function of frequency. The result would have to be below the mask. The quantity *A* is the gain peaking, and the quantity f_c is the maximum bandwidth. Note that any implementation is allowed for the filter (digital versus analog, second order versus higher order, etc); the only requirement is that the filter transfer function meet the specified mask.

2.2.2.8 Time stamp and correction field resolution

[***Author's note: The statements below also will apply to the timestamps and correction fields in the Pdelay messages. This must be indicated after the sections for those messages are filled in.***]

As indicated in Section 3.9, PTP timestamps (both originTimestamp in Sync and preciseOrigingTimestamp in Follow_Up messages) use 4 bytes to represent the number of seconds and 4 bytes to represent the number of ns, along with a 2 byte epochNumber that counts the number of times the 32 bit seconds counter has rolled over. The resolution for a PTP timestamp is therefore 1 ns, and the entire timestamp will roll over after 2^{48} s, or approximately 8.9×10^6 years.

Section 3.9 also indicates that the correction field is 8 bytes and represents the number of units of size 2^{-16} ns (approximately 15.26 fs). Specifically, the most significant 6 bytes represent the number of ns, and the least significant 2 bytes represent a fraction of a ns. The correction field is a signed quantity; the most significant bit is a sign bit. Therefore, the largest absolute value of the whole number of nanoseconds that can be represented is 2^{47}.

If the timestamp measurement precision in a PTP system is 1 ns or greater, the measurement is carried by the originTimestamp and/or the preciseOriginTimestamp fields. If the timestamp measurement precision is less than 1 ns, then the floor of the timestamp value is carried in the originTimestamp or preciseOriginTimestamp fields. The fractional ns portion is carried in the correction field, i.e., is added to the current contents of the correction field when the timestamp is written (the current contents of the correction field may be zero if the message was just created with the correction field initialized to zero). The smallest timestamp measurement resolution supported is $2^{-\cancel{48}16}$ ns (i.e., if the node hardware supports time measurements with precision better than $2^{-\cancel{48}16}$ ns, the value must either be truncated or rounded (IEEE 1588 does not specify whether truncation or rounding is to be done; that is presumably application dependent and part of the respective 1588 profile).

AVB nodes will be required to support phase measurement granularity of 40 ns (i.e., the clock hardware will be no worse than the 25 MHz clock used in 100 Mbit/s Ethernet). However, AVB nodes will be allowed to support smaller granularity (i.e., < 40 ns). In addition, AVB will be required to support timestamp resolution of X (X is TBD, but will require at most 16 bits, i.e., X will not be smaller than 2^{-16} ns). By timestamp resolution, we mean the number of bits of precision in the most significant 16 bits of the correction field actually used. If an AVB node chooses to implement fewer than 16 bits of precision (but at least X bits), then any measurements will truncated/rounded [***Author's note: It must be specified whether AVB nodes will truncate or round.***] to X bits and the remaining most significant bits will be filled with zeros.

~~[***Author's note: Is a statement needed on the case of resolutions better than 2^{-48} ns, e.g., (1) that AVB nodes are not expected to ever need to support such small resolution, or (2) that if such resolution is supported, whether the measurement values should be truncated or rounded, or (3) such cases will be addressed in a future version of this standard, or (4) something else?***]~~

3. Message and frame formats

AVB networks will use the following six IEEE 1588 messages:

a) Sync
b) Follow_Up
c) ~~Peer_D~~Pdelay_Req (formerly called ADelay_Req)
d) ~~Peer_D~~Pdelay_Resp (formerly called ADelay_Resp)
e) ~~Peer_D~~Pdelay_Resp_Follow_Up (formerly called ADelay_Resp_~~FollowUp~~Follow_Up)
f) Announce

Each message will have a standard Ethernet header (with or without 802.1Q tags) that precedes the PTP (i.e., IEEE 1588) payload. The PTP payload consists of a common header, i.e., a portion common to all PTP messages, followed by a portion specific to each PTP message. The common header is not a header in the sense of a protocol layer, i.e., it is not true that the PTP layer alters the common header while leaving the portion specific to each message unaltered. Rather, the PTP layer can alter any portion of the PTP payload (in this sense, the use of the term “payload” is different from the normal use, as normally a layer alters only the header information and transports the payload unaltered). The main reason for distinguishing the common header is convenience; since this portion is the same in all PTP messages, we can describe it once rather than repeat the fields for each message type.

–Note that AVB will not use Delay_Req and Delay_Resp messages; all AVB nodes will be required to process the ~~Peer_D~~Pdelay messages.
[***Author’s Note: There are two additional PTP message types: Transport Message and Management Message (there are multiple Management Messages). The AVB TG must examine these messages and determine which, if any, are needed for AVB.***]

The message formats are shown in the following subsections. The material reflects the latest agreements (as of the date of this document) of the IEEE 1588 Short Frames Subcommittee, documented in References ~~is taken mostly from~~ [5], ~~and~~ [6], [12], [13], [14], and [15]. One change from [5] and [6] is the fact that the Ethertype is indicated as m_0m_1 m_2m_3; this is intended to denote whatever Ethertype is assigned to frames that must be timestamped [8]. The Ethernet header, PTP common header, and PTP message specific portions ~~payloads~~ are shown separately for conciseness (i.e., to avoid repeating the header and common fields for each message). The PTP common header and PTP message specific portion together form the PTP payload. The PTP common header immediately follows the Ethernet header; the PTP message specific portion immediately follows the PTP common header; and ~~Note that~~ the 4-octet frame check sequence (FCS) follows the PTP payload (i.e., the PTP message specific portion) of each message (the FCS is not shown).

Note that it is possible that names of fields will change as the material is further discussed in the IEEE 1588 committee and subcommittees.

3.1 Ethernet header (without 802.1Q tags)

N	Octet N	Octet N+1	Octet N+2	Octet N+3	Type (informative)	Field name
0	h_0h_1	h_2h_3	h_4h_5	h_6h_7	Octet[6]	destination MAC address
4	h_8h_9	$h_{10}h_{11}$	k_0k_1	k_2k_3	Octet[6] (cont) | Octet[6]	destination MAC address (cont) | source MAC address
8	k_4k_5	k_6k_7	k_8k_9	$k_{10}k_{11}$	Octet[6] (cont)	source MAC address (cont)
12	m_0m_1	m_2m_3	N/A	N/A	UInteger16	type (this will be whatever Ethertype is assigned to frames that must be time stamped)

3.2 Ethernet header (with 802.1Q tags)

N	Octet N	Octet N+1	Octet N+2	Octet N+3	Type (informative)	Field name
0	H_0h_1	h_2h_3	h_4h_5	h_6h_7	Octet[6]	destination MAC address
4	H_8h_9	$h_{10}h_{11}$	k_0k_1	k_2k_3	Octet[6] (cont) | Octet[6]	destination MAC address (cont) | source MAC address
8	K_4k_5	k_6k_7	k_8k_9	$k_{10}k_{11}$	Octet[6] (cont)	source MAC address (cont)
12	0x81	00	j_0j_1	j_2j_3	UInteger16 | UInteger16	type (0x8100 = tagged MAC frame) | tag control information
16	m_0m_1	m_2m_3	N/A	N/A	UInteger16	type (this will be whatever Ethertype is assigned to frames that must be

							time stamped)

3.3 PTP Common header

X = 14 for Ethernet without 802.1Q tags
X = 18 for Ethernet with 802.1Q tags

SOF	**N**	**Octet N**	**Octet N+1**	**Octet N+2**	**Octet N+3**	**Type (informative)**	**Field name**
X+0	0	h_0h_1	h_0h_1	k_0k_1	k_2k_3	UInteger4 | UInteger4 | UInteger8 | UInteger16	transportSpecific | messageID |versionPTP | messageLength
X+4	4	h_0h_1	h_2h_3	h_4h_5	k_1k_2	Octet[3] | UInteger8	reserved | subdomain
X+8	8	h_0h_1	h_2h_3	h_4h_5	k_1k_2	Octet[3] | Integer8	flags | logSyncPeriod
X+12	12	k_0k_1	k_2k_3	k_4k_5	k_6k_7	Integer48	correctionNs
X+16	16	k_8k_9	$k_{10}k_{11}$	h_0h_1	h_2h_3	Integer48(cont) | UInteger16	correctionNs (cont) | correctionSubNs
X+20	20	k_0k_1	k_2k_3	k_4k_5	k_6k_7	|Octet[6]	sourceUuid
X+24	24	k_8k_9	$k_{10}k_{11}$	h_0h_1	h_2h_3	Octet[6](cont) | UInteger16	sourceUuid(cont) | sourcePortId
X+28	28	h_0h_1	h_0h_1	k_0k_1	k_2k_3	Octet | Octet | UInteger16	sourceCommunicationTechnology | reserved | sequenceId
X+32	32	h_0h_1	h_0h_1	N/A	N/A	UInteger8 | Octet	control | reserved

a) transportSpecific – not used by AVB networks (likely will be set to all zeros in AVB)
b) messageID – four-bit subtype that indicates the PTP message as indicated in Table 1 below. Note that not all the message types are used in AVB networks. Note also that all the messages that must be timestamped (i.e., the event messages) have the first bit of messageID set to zero (and therefore this bit can be used as an indicator to timestamping hardware of which messages must be timestamped [8].

Table 1. messageID for each PTP message

Category of message	**Message**	**messageID value (hex)**
Event	Sync	0
Event	Delay_Req	1
Event	Pdelay_Req	2
Event	Pdelay_Resp	3
Event	reserved	4-7
General	Follow_Up	8
General	Delay_Resp	9
General	Pdelay_Resp_Follow_Up	A
General	Announce	B
General	Transport	C
General	PTP Management Message	D
General	reserved	E-F

c) VersionPTP – version of the PTP standard implemented (Version 2 for this version of IEEE 1588)

d) messageLength – length of the PTP message payload in octets, from the first octet of the the common header to the last octet of the message specific portion for each respective message given below (note: for more general (non-AVB) applications, messageLength also includes optional extensions).

e) subdomain – AVB will use the default sub domain, i.e., sub domain 0. In more general PTP applications, there can be 3 alternate subdomains (numbered 1 – 3) and additional subdomains (numbered 4 – 255) set by the node manager of each node.

f) flags – The table below is taken from [15]. All unused flags shall be transmitted as zero and ignored by the receiver. Note that not all the defined flags will be used by AVB networks. [***Author's note: It must be determined which flags AVB networks will use.***]

Bit	**Contained in Message Type**	**Name**	**Description**
7	All	SECURE	True if the message is suffixed by a security hash code.
6	Sync, Follow-up	PTP_SYNC_BURST	TRUE if this message is part of a burst of messages (and can thus be ignored by clocks not requesting it).
6	Delay_Req	PTP_SYNC_BURST_REQ	In a Delay_Req message, TRUE if the sender is requesting a burst of multicast Sync messages. The number of messages transmitted in response and the mean period between those messages is determined by the transmitting node. While the requesting node is requesting the master to transmit a burst of sync messages, it shall set the flag on every Delay_Req message it transmits to that master. The requesting node may set the flag to FALSE at any time and it shall do so when it receives a Sync message with a TRUE PTP_SYNC_BURST flag.
5	Sync, Announce, Delay Response	TimeScaleAccurate	The value of the time stamps generated by this node is considered accurate. When TimeScaleAccurate is true, the value of the timestamp contained in this message is aligned to the beginning of the PTP epoch. When TimeScaleAccurate is false, the value of the timestamp contained in this message will increment with the n accuracy indicated by the clockIdentifier but need not be aligned to the start of the PTP epoch. [GMG Comment: I think I understand the intent, but is the notation "n accuracy" a typo (it is not clear what "n" is)? Can this be clarified, e.g., could we say "… will increment by an amount that reflects the frequency accuracy indicated by the clockIdentifier…"?
4	Sync, Announce	UTC_REASONABLE	The value of the UTC Offset in the transmitted message is considered correct.
3	Sync, Announce, Delay Request, Path_Delay_Resp	TIME_APPROXIMATE	The value of originTimeStamp is approximate.

Bit	Contained in Message Type	Name	Description
2	All	Standby master	True if the node is not the best master but is transmitting SYNC and ANNOUNCE messages. Otherwise false.
1	Sync, Announce	PTP_LI_59	Value of leap_59 of global time properties data set
0	Sync, Announce	PTP_LI_61	Value of leap_61 of global time properties data set

g) logSyncPeriod – the logarithm to base 2 of the current sync interval in seconds, for a port in the PTP_MASTER state.

h) correctionNs and correctionSubNs – together, correctionNs and the correctionSubNs hold the correction value that must be applied to the time information contained in this message. When no time information is in the message, the correctionNs and correctionSubNs fields should be transmitted as zero. The correction value, in nanoseconds, shall be interpreted as $correctionNs + \frac{correctionSubNs}{2^{16}}$. The correctionNs field is a signed integer. If correctionNs is negative, the unsigned correctionSubNs field shall also be interpreted as negative. A value of one in all bits of the correctionNs and correctionSubNs fields, except the most significant bit of correctionNs, indicates that the delay is too big to be represented.

i) sourceUuid – for AVB, Ethernet MAC address of the source of the message

j) sourcePortId – the ID of the port that is the source of the message. The ports on a network element with *N* ports are numbered from 1 to *N*.

k) sourceCommunicationTechnology – indicates the communication medium and technology for the port that issues the PTP message. Initially AVB will focus on Ethernet, for which the value of this field is 1 (see Table 2 of [7] for a list of the various communications technologies recognized in IEEE 1588 Version 1)

l) sequenceId – (i) If the message is a PDelay Request message, the value shall be one greater than the sequenceId of the previous PDelay Request message issued by the port. (ii) If the message is a PDelay Response message, the value shall be one greater than the sequenceId of the previous PDelay Response message issued by the port. (iii) If the message is a multicast event message and not a PDelay Request or PDelay Response message, the value shall be one greater than the sequenceId of the previous such message. (iv) If the message is a multicast general message, the value shall be one greater than the sequenceId of the previous such message. (v) If the message is a unicast event message, the value shall be one greater than the sequenceId of the previous unicast event message sent to the same destination address (not relevant for AVB). (vi) If the message is a unicast general message, the value shall be one greater than the sequenceId of the previous unicast general message sent to the same destination address (not relevant for AVB).

m) control – not used in AVB (retained for backward compatibility with IEEE 1588 Version 1; indicates the Version 1 message type (see Table 28 of [7] for allowable values).

3.4 Sync ~~payload~~message specific portion

X = ~~14~~ 48 for Ethernet without 802.1Q tags
X = ~~18~~ 52 for Ethernet with 802.1Q tags

SOF	N	Octet N	Octet N+1	Octet N+2	Octet N+3	Type (informative)	Field name
~~X+0~~	~~0~~	~~h_0h_1~~	~~h_2h_3~~	~~k_0k_1~~	~~k_2k_3~~	~~UInteger4 |~~	~~transportSpecific | messageID~~

SOF	N	Octet N	Octet N+1	Octet N+2	Octet N+3	Type (informative)	Field name
						~~UInteger4 | UInteger8 | UInteger16~~	~~|versionPTP | reserved~~
~~X+4~~	~~4~~	~~h_0h_1~~	~~h_2h_3~~	~~k_0k_1~~	~~k_2k_3~~	~~UInteger16 | UInteger16~~	~~totalMessageLength | subdomain~~
~~X+8~~	~~8~~	~~h_0h_1~~	~~h_2h_3~~	~~h_4h_5~~	~~h_6h_7~~	~~Octet[4]~~	~~flags~~
~~X+12~~	~~12~~	~~k_0k_1~~	~~k_2k_3~~	~~k_4k_5~~	~~k_6k_7~~	~~Integer64~~	~~correctionField~~
~~X+16~~	~~16~~	~~k_8k_9~~	~~$k_{10}k_{11}$~~	~~$k_{12}k_{13}$~~	~~$k_{14}k_{15}$~~	~~Integer64(cont)~~	~~correctionField(cont)~~
~~X+20~~	~~20~~	~~k_0k_1~~	~~j_0j_1~~	~~h_0h_1~~	~~h_2h_3~~	~~UInteger8 | UInteger8 |Octet[6]~~	~~reserved | sourceCommunicationTechnology | sourceUuid~~
~~X+24~~	~~24~~	~~h_4h_5~~	~~h_6h_7~~	~~h_8h_9~~	~~$h_{10}h_{11}$~~	~~Octet[6](cont)~~	~~sourceUuid(cont)~~
~~X+28~~	~~28~~	~~h_0h_1~~	~~h_2h_3~~	~~k_0k_1~~	~~k_2k_3~~	~~UInteger16 | UInteger16~~	~~sourcePortId | sequenceId~~
X+~~32~~ 0	~~32~~ 0	h_0h_1~~j_0j_1~~	h_2h_3~~00~~	h_0h_1	h_2h_3	~~UInteger8~~ UInteger16 | UInteger32 ~~Octet | UInteger16~~	~~control | reserved |~~ epochNumber | originTimestamp (seconds)
X+~~36~~ 4	~~36~~ 4	h_4h_5~~h_0h_1~~	~~H~~h_6~~$_2$~~h_7~~$_3$~~	h_0h_1~~h_4h_5~~	h_2h_3~~h_6h_7~~	UInteger32 (cont) | Uinteger32	originTimestamp (seconds) (cont) | originTimestamp (nanoseconds)
X+~~40~~ 8	~~40~~ 8	h_4h_5~~h_0h_1~~	h_6h_7~~h_2h_3~~	~~h_4h_5~~h_0h_1	~~h_6h_7~~h_2h_3	UInteger32 (cont) | Integer16	originTimestamp (nanoseconds) (cont) | currentUTCOffset
~~X+44~~	~~44~~	~~k_0k_1~~	~~k_2k_3~~	~~N/A~~	~~N/A~~	~~Integer16~~	~~currentUTCOffset~~

a) epochNumber – when the epoch is the PTP epoch, the epochNumber is the total number of times the 32-bit seconds counter has rolled over since the PTP epoch.[8] More generally, the epochNumber may be treated as the most significant part of the total number of seconds since the epoch (the least significant part is the 32-bit integer seconds portion of the PTP timestamp). See Section 6.2.5.7 and Appendix B of [7] for more detail.

b) originTimestamp (seconds) – the seconds portion of the timestamp that carries any timestamp measurement made on-the-fly by the master BC or OC that issues the Sync message. AVB is not required to make on-the-fly measurements and, if it does make them, there is no requirement that they be precise (i.e., AVB may still use Follow_Up messages in this case). However, AVB is allowed to make precise timestamp measurements on-the-fly and not use Follow_Up.

c) originTimestamp (nanoseconds) - the nanoseconds portion of the timestamp that carries any timestamp measurement made on-the-fly by the master BC or OC that issues the Sync message. AVB is not required to make on-the-fly measurements and, if it does make them, there is no requirement that they be precise (i.e., AVB may still use Follow_Up messages in this case). However, AVB is allowed to make precise timestamp measurements on-the-fly and not use Follow_Up.

d) CurrentUTCOffset – the offset between the UTC and TAI timescales at the master BC or OC that issues the Sync or Followup message.

[8] In IEEE 1588, the term *epoch* is defined as the reference time that defines the origin of a timescale. For PTP, the epoch is 0:00:00 on 1 January 1970 (see Appendix B, Table B.2 of [7]).

3.~~4~~ 5 Follow_Up message specific portion~~payload~~

X = ~~14~~ 48 for Ethernet without 802.1Q tags
X = ~~18~~ 52 for Ethernet with 802.1Q tags

Note that the Follow_Up message specific portion~~payload~~ differs from the Sync ~~payload~~ message specific portion in that:

~~a)currentUTCOffset is not repeated in Follow_Up (it is only in Sync) [*Author's Note: epochNumber must be added to the table below. It also must be decided whether to add currentUTCOffset to Followup (for the same reason it was added to Sync, namely that in theory it could change between an initial, less precise timestamp measurement when Sync is sent and the more precise later measurement.*]~~

~~b)~~a)Follow_Up has the associatedSequenceId of the corresponding Sync

~~c)~~b)Follow_Up has a preciseOriginTimestamp instead of an originTimestamp.

SOF	N	Octet N	Octet N+1	Octet N+2	Octet N+3	Type (informative)	Field name
~~X+0~~	~~0~~	~~h_0h_1~~	~~h_2h_3~~	~~k_0k_1~~	~~k_2k_3~~	~~UInteger4 | UInteger4 | UInteger8 | UInteger16~~	~~transportSpecific | messageID|versionPTP | reserved~~
~~X+4~~	~~4~~	~~h_0h_1~~	~~h_2h_3~~	~~k_0k_1~~	~~k_2k_3~~	~~UInteger16 | UInteger16~~	~~totalMessageLength | subdomain~~
~~X+8~~	~~8~~	~~h_0h_1~~	~~h_2h_3~~	~~h_4h_5~~	~~h_6h_7~~	~~Octet[4]~~	~~flags~~
~~X+12~~	~~12~~	~~k_0k_1~~	~~k_2k_3~~	~~k_4k_5~~	~~k_6k_7~~	~~Integer64~~	~~correctionField~~
~~X+16~~	~~16~~	~~k_8k_9~~	~~$k_{10}k_{11}$~~	~~$k_{12}k_{13}$~~	~~$k_{14}k_{15}$~~	~~Integer64(cont)~~	~~correctionField(cont)~~
~~X+20~~	~~20~~	~~k_0k_1~~	~~j_0j_1~~	~~h_0h_1~~	~~h_2h_3~~	~~UInteger8 | UInteger8 |Octet[6]~~	~~reserved | sourceCommunicationTechnology | sourceUuid~~
~~X+24~~	~~24~~	~~h_4h_5~~	~~h_6h_7~~	~~h_8h_9~~	~~$h_{10}h_{11}$~~	~~Octet[6](cont)~~	~~sourceUuid(cont)~~
X+0 ~~X+28~~	0~~28~~	h_0h_1~~h_0h_1~~	h_2h_3~~h_2h_3~~	h_0h_1~~k_0k_1~~	h_2h_3~~k_2k_3~~	UInteger16 | UInteger32 ~~UInteger16 | UInteger16~~	epochNumber | originTimestamp (seconds)~~sourcePortId | sequenceId~~
X+4 ~~X+32~~	4~~32~~	h_4h_5~~j_0j_1~~	h_6h_7~~00~~	h_0h_1~~h_0h_1~~	h_2h_3~~h_2h_3~~	UInteger32 (cont) | Uinteger32~~UInteger8 | Octet | UInteger16~~	preciseOriginTimestamp (seconds) (cont) | originTimestamp (nanoseconds) ~~control | reserved | associatedSequenceId~~
X+8 ~~X+36~~	8~~36~~	h_4h_5~~h_0h_1~~	h_6h_7~~h_2h_3~~	h_0h_1~~h_4h_5~~	h_2h_3~~h_6h_7~~	UInteger32 (cont) | Integer16~~UInteger32~~	preciseOriginTimestamp (nanoseconds) (cont) | currentUTCOffset~~preciseOriginTimestamp (seconds)~~
~~X+40~~ X+12	~~40~~ 12	h_0h_1~~h_0h_1~~	h_2h_3~~h_2h_3~~	~~h_4h_5~~N/A	~~h_6h_7~~N/A	~~Integer32~~UInteger16	~~preciseOriginTimestamp (nanoseconds)~~associatedSequenceId

a) epochNumber – when the epoch is the PTP epoch, the epochNumber is the total number of times the 32-bit seconds counter has rolled over since the PTP epoch.[9] More generally, the epochNumber may be treated as the most significant part of the total number of seconds since the epoch (the least significant part is the 32-bit integer seconds portion of the PTP timestamp). See Section 6.2.5.7 and Appendix B of [7] for more detail.
b) preciseOriginTimestamp (seconds) – the seconds portion of the more precise timestamp measurement carried in a Follow_Up message.
c) preciseOriginTimestamp (nanoseconds) - the nanoseconds portion of the more precise timestamp measurement carried in a Follow_Up message.
d) CurrentUTCOffset – the offset between the UTC and TAI timescales at the master BC or OC that issues the Sync or Followup message.
e) associatedSequenceId – the sequenceId of the Sync message that corresponds to this Follow_Up message.

3.5 ~~Peer_D~~Pdelay_Req ~~payload~~ message specific portion

~~To be supplied.~~
X = 48 for Ethernet without 802.1Q tags
X = 52 for Ethernet with 802.1Q tags

Note that the Pdelay_Req message specific portion consists of 28 reserved octets so that its length is the same as the length of the Pdelay_Resp message.

SOF	**N**	**Octet N**	**Octet N+1**	**Octet N+2**	**Octet N+3**	**Type (informative)**	**Field name**
X+0	0	h_0h_1	h_0h_1	h_0h_1	h_0h_1	Octet[4]	reserved
X+4	4	h_0h_1	h_0h_1	h_0h_1	h_0h_1	Octet[4]	reserved
X+8	8	h_0h_1	h_0h_1	h_0h_1	h_0h_1	Octet[4]	reserved
X+12	12	h_0h_1	h_0h_1	h_0h_1	h_0h_1	Octet[4]	reserved
X+16	16	h_0h_1	h_0h_1	h_0h_1	h_0h_1	Octet[4]	reserved
X+20	20	h_0h_1	h_0h_1	h_0h_1	h_0h_1	Octet[4]	reserved
X+24	24	h_0h_1	h_0h_1	h_0h_1	h_0h_1	Octet[4]	reserved

3.6 ~~Peer_D~~Pdelay_Resp message specific portion~~payload~~

~~To be supplied.~~
X = 48 for Ethernet without 802.1Q tags
X = 52 for Ethernet with 802.1Q tags

SOF	**N**	**Octet N**	**Octet N+1**	**Octet N+2**	**Octet N+3**	**Type (informative)**	**Field name**
X+0	0	h_0h_1	h_2h_3	h_0h_1	h_2h_3	UInteger16 | UInteger32	requestingSequenceId | originTimestamp (seconds)
X+4	4	h_4h_5	h_6h_7	h_0h_1	h_2h_3	UInteger32 (cont) | Uinteger32	originTimestamp (seconds) (cont) | originTimestamp (nanoseconds)
X+8	8	h_4h_5	h_6h_7	h_0h_1	h_2h_3	UInteger32 (cont)	originTimestamp (nanoseconds) (cont) |

[9] In IEEE 1588, the term *epoch* is defined as the reference time that defines the origin of a timescale. For PTP, the epoch is 0:00:00 on 1 January 1970 (see Appendix B, Table B.2 of [7]).

SOF	**N**	**Octet N**	**Octet N+1**	**Octet N+2**	**Octet N+3**	**Type (informative)**	**Field name**
						| UInteger32	requestReceiptTimestamp (seconds)
X+12	12	h_4h_5	h_6h_7	h_0h_1	h_2h_3	UInteger32 (cont) | Uinteger32	requestReceiptTimestamp (seconds) (cont) | requestReceiptTimestamp (nanoseconds)
X+16	16	h_4h_5	h_6h_7	h_0h_1	h_2h_3	UInteger32 (cont) | Octet[6]	requestReceiptTimestamp (nanoseconds) (cont) | requestingPortUuid
X+20	20	h_4h_5	h_6h_7	h_8h_9	$h_{10}h_{11}$	Octet[6] (cont)	requestingPortUuid (cont)
X+24	24	h_0h_1	h_2h_3	h_0h_1	h_0h_1	UInteger16 | Octet | Octet	requestingPortPortId | requestingPortCommunicationTechnology | reserved

a) requestingSequenceId – the sequenceId of the Pdelay_Req message that corresponds to this Pdelay_Resp message.

e) originTimestamp (seconds) – the seconds portion of the timestamp that carries any timestamp measurement made on-the-fly by the P2P TC when it issues the Pdelay_Resp message. AVB is not required to make on-the-fly measurements and, if it does make them, there is no requirement that they be precise (i.e., AVB may still use Pdelay_Resp_Follow_Up messages in this case). However, AVB is allowed to make precise timestamp measurements on-the-fly and not use Pdelay_Resp_Follow_Up.

f) originTimestamp (nanoseconds) - the nanoseconds portion of the timestamp that carries any timestamp measurement made on-the-fly by the P2P TC when it issues the Pdelay_Resp message. AVB is not required to make on-the-fly measurements and, if it does make them, there is no requirement that they be precise (i.e., AVB may still use Pdelay_Resp_Follow_Up messages in this case). However, AVB is allowed to make precise timestamp measurements on-the-fly and not use Pdelay_Resp_Follow_Up.

b) requestReceiptTimestamp (seconds) and requestReceiptTimestamp (nanoseconds) – the seconds and nanoseconds portions, respectively, of the timestamp for the receipt of the corresponding Pdelay_Req message

c) requestingPortUuid – for AVB, the Ethernet MAC address of the Port that issued the corresponding Pdelay_Req message

n) requestingPortPortId – the ID of the port that issued the corresponding Pdelay_Req message. The ports on a network element with *N* ports are numbered from 1 to *N*.

o) requestingPortCommunicationTechnology – indicates the communication medium and technology for the port that issued the corresponding Pdelay_Req message. Initially AVB will focus on Ethernet, for which the value of this field is 1 (see Table 2 of [7] for a list of the various communications technologies recognized in IEEE 1588 Version 1)

NOTE: The requestingPortUuid, requestingPortPortId, and requestingPortCommunicationTechnology fields are included in the Pdelay_Resp message to support any future solution to the 1:N problem in IEEE 1588. This is not relevant for AVB networks because the AVB cloud will not contain E2E TCs nor non-PTP bridges.

3.7 ~~Peer_D~~Pdelay_Resp_~~FollowUp~~Follow_Up message specific portion~~payload~~

~~To be supplied~~.
X = 48 for Ethernet without 802.1Q tags
X = 52 for Ethernet with 802.1Q tags

SOF	N	Octet N	Octet N+1	Octet N+2	Octet N+3	Type (informative)	Field name
X+0	0	h_0h_1	h_2h_3	h_0h_1	h_2h_3	UInteger16 | UInteger32	associatedSequenceId | preciseOriginTimestamp (seconds)
X+4	4	h_4h_5	h_6h_7	h_0h_1	h_2h_3	UInteger32 (cont) | Uinteger32	preciseOriginTimestamp (seconds) (cont) | preciseOriginTimestamp (nanoseconds)
X+8	8	h_4h_5	h_6h_7	N/A	N/A	UInteger32 (cont)	preciseOriginTimestamp (nanoseconds) (cont)

f) associatedSequenceId – the sequenceId of the Pdelay_Resp message that corresponds to this Pdelay_Resp_Follow_Up message.
g) preciseOriginTimestamp (seconds) – the seconds portion of the more precise timestamp measurement carried in a Pdelay_Resp_Follow_Up message.
h) preciseOriginTimestamp (nanoseconds) - the nanoseconds portion of the more precise timestamp measurement carried in a Pdelay_Resp_Follow_Up message.

3.8 Announce message specific portion~~payload~~

~~To be supplied.~~
X = 48 for Ethernet without 802.1Q tags
X = 52 for Ethernet with 802.1Q tags

SOF	N	Octet N	Octet N+1	Octet N+2	Octet N+3	Type (informative)	Field name
X+0	0	h_0h_1	h_2h_3	h_0h_1	h_2h_3	UInteger16 | UInteger32	epochNumber | originTimestamp (seconds)
X+4	4	h_4h_5	h_6h_7	h_0h_1	h_2h_3	UInteger32 (cont) | Uinteger32	originTimestamp (seconds) (cont) | originTimestamp (nanoseconds)
X+8	8	h_4h_5	h_6h_7	h_0h_1	h_2h_3	UInteger32 (cont) | Integer16	originTimestamp (nanoseconds) (cont) | currentUTCOffset
X+12	12	h_0h_1	h_2h_3	h_0h_1	h_2h_3	Octet[2] | UInteger16	announceFlags | localStepsRemoved
X+16	16	h_0h_1	h_2h_3	h_4h_5	h_6h_7	Octet[6]	grandmasterUuid
X+20	20	h_8h_9	$h_{10}h_{11}$	h_0h_1	h_2h_3	Octet[6] (cont) | UInteger16	grandmasterUuid (cont)| grandmasterPortId
X+24	24	h_0h_1	h_0h_1	h_0h_1	h_0h_1	Octer | Octer | UInteger8 | Octet	grandmasterCommunicationTechnology | reserved | grandmasterStratum | grandmasterIdentifier
X+28	28	h_0h_1	h_2h_3	h_0h_1	h_2h_3	Integer16 |	grandmasterVariance |

SOF	**N**	**Octet N**	**Octet N+1**	**Octet N+2**	**Octet N+3**	**Type (informative)**	**Field name**
						UInteger16	grandmasterSequenceId
X+32	32	h_0h_1	h_2h_3	h_4h_5	h_6h_7	Octet[6]	masterUuid
X+36	36	h_8h_9	$h_{10}h_{11}$	h_0h_1	h_2h_3	Octet[6] (cont) | UInteger16	masterUuid (cont) | masterPortId
X+40	40	h_0h_1	h_0h_1	h_0h_1	h_0h_1	Octer | Octer | UInteger8 | Octet	masterCommunicationTechnology | reserved | localClockStratum | localClockIdentifier
X+44	44	h_0h_1	h_2h_3	h_0h_1	h_2h_3	Integer16 | Integer32	localClockVariance | estimatedMasterClockPhaseChangeRate
X+48	48	h_4h_5	h_6h_7	h_0h_1	h_2h_3	Integer32 (cont) | Integer16	estimatedMasterClockPhaseChangeRate (cont) | estimatedMasterVariance
X+52	52	h_0h_1	N/A	N/A	N/A	Integer8	logAnnouncePeriod

[***Author's note: The definitions of these fields will be supplied, may be found in IEEE 1588, Version 1 [7]. AVB networks may not use all the fields.***]

3.9 Definitions of selected payload fields

~~The definitions below are taken from [3], [5], and [7] and augmented by discussions in the February 22 – 24 IEEE 1588 face-to-face meeting.~~

~~***Author's Note:** The definitions below are for Sync and Follow_Up message fields; modified versions may be necessary for the Peer_DPdelay messages when they are supplied.*~~

~~a)transportSpecific – not used by AVB networks (likely will be set to all zeros in AVB)~~
~~b)messageID – four-bit subtype that indicates the PTP message as indicated in Table 1 below. Note that not all the message types are used in AVB networks. Note also that all the messages that must be timestamped (i.e., the event messages) have the first bit of messageID set to zero (and therefore this bit can be used as an indicator to timestamping hardware of which messages must be timestamped [8].~~

~~**Table 1. messageID for each PTP message**~~

~~**Category of message**~~	~~**Message**~~	~~**messageID value**~~
~~Event~~	~~Sync~~	~~0~~
~~Event~~	~~Delay_Req~~	~~1~~
~~Event~~	~~Peer_DPdelay_Req~~	~~2~~
~~Event~~	~~Peer_DPdelay_Resp~~	~~3~~
~~Event~~	~~reserved~~	~~4-7~~
~~General~~	~~Follow_Up~~	~~8~~
~~General~~	~~Delay_Resp~~	~~9~~
~~General~~	~~Peer_DPdelay_Resp_FollowUpFollow_Up~~	~~10~~
~~General~~	~~Announce~~	~~11~~
~~General~~	~~PTP Management Message~~	~~12~~
~~General~~	~~To be completed~~	~~13-15~~

~~c)VersionPTP – version of the PTP standard implemented~~
~~d)TotalMessageLength – length of the PTP message payload~~
~~e)subdomain – the specific value for AVB is to be defined; note that an AVB network will consist of a single PTP subdomain.~~
~~f)flags – To be supplied (an incomplete set of flags is contained in Table 3 of [5])~~
~~g)correctionField – the field that the TCs use to accumulate the residence time. This field is 8 bytes, and represents time in units of 2^{-16} ns (approximately 15.26 fs). Therefore, the format of the this field is ns (represented by the most significant 6 bytes) and fraction of a ns (represented by the least significant 2 bytes)~~
~~h)sourceCommunicationTechnology – indicates the communication medium and technology for the port that issues the PTP message. Initially AVB will focus on Ethernet, for which the value of this field is 1 (see Table 2 of [7] for a list of the various communications technologies recognized in IEEE 1588 Version 1)~~
~~i)sourceUuid – for AVB, Ethernet MAC address of the source of the message~~
~~j)sourcePortId – the ID of the port that is the source of the message. The ports on a network element with *N* ports are numbered from 1 to *N*.~~
~~k)sequenceId – an ID assigned to Sync and Follow_Up messages as they are transmitted on a BC or OC port. IDs are assigned sequentially on each transmitting port for event messages (i.e., Sync), and separately (but also sequentially) for general messages (i.e., Follow_Up and Announce). [***Author's Note:*** *Peer_DPdelay messages will be covered by this definition when their formats are supplied.*]~~
~~l)control – not used in AVB (retained for backward compatibility with IEEE 1588 Version 1; indicates the Version 1 message type (see Table 28 of [7] for allowable values).~~
~~m)epochNumber – when the epoch is the PTP epoch, the epochNumber is the total number of times the 32-bit seconds counter has rolled over since the PTP epoch.[10] More generally, the epochNumber may be treated as the most significant part of the total number of seconds since the epoch (the least significant part is the 32-bit integer seconds portion of the PTP timestamp). See Section 6.2.5.7 and Appendix B of [7] for more detail.~~
~~n)originTimestamp (seconds) – the seconds portion of the timestamp that carries any timestamp measurement made on-the-fly by the master BC or OC that issues the Sync message. AVB is not required to make on-the-fly measurements and, if it does make them, there is no requirement that they be precise (i.e., AVB may still use Follow_Up messages in this case). However, AVB is allowed to make precise timestamp measurements on-the-fly and not use Follow_Up.~~
~~o)originTimestamp (nanoseconds) – the nanoseconds portion of the timestamp that carries any timestamp measurement made on-the-fly by the master BC or OC that issues the Sync message. AVB is not required to make on-the-fly measurements and, if it does make them, there is no requirement that they be precise (i.e., AVB may still use Follow_Up messages in this case). However, AVB is allowed to make precise timestamp measurements on-the-fly and not use Follow_Up.~~
~~p)~~d) CurrentUTCOffset – the offset between the UTC and TAI timescales at the master BC or OC that issues the Sync or Followup message.
~~q)~~e) AssociatedSequenceId – the sequenceId of the Sync message that corresponds to this Follow_Up message.
~~r)preciseOriginTimestamp (seconds) – the seconds portion of the more precise timestamp measurement carried in a Follow_Up message.~~
~~s)preciseOriginTimestamp (nanoseconds) – the nanoseconds portion of the more precise timestamp measurement carried in a Follow_Up message.~~

[10] In IEEE 1588, the term *epoch* is defined as the reference time that defines the origin of a timescale. For PTP, the epoch is 0:00:00 on 1 January 1970 (see Appendix B, Table B.2 of [7]).

4. Processing of PTP messages

4.1 Sync and Follow_Up messages

4.1.1 Description based on OC and P2P TC functions

The processing rules below for Sync and Follow_Up messages do not make any simplification based on the fact that P2P TC and OC functions are collocated at every AVB node. Rather, the architecture on the left hand side of Figure 1 is assumed, and P2P TC functions and OC functions are assumed to be separate entities, with explicit reference to the logical link between the two. The rules are taken from [3] and [4].

When a Sync or Follow_Up message arrives at a TC, it is terminated at Layer 2 (Ethernet) and passed to the PTP layer for processing. A new Sync or Follow_Up message is generated for each respective port that it needs to be transmitted on to reach each slave clock without looping. The new Sync or Follow_Up messages are similar to the arriving message, except that the correction field has possibly been altered by adding the residence time. As a result, the checksum also must be altered when the message is transmitted. Note that the Sync or Follow_Up messages transmitted on each port will not necessarily be the same, because they won't necessarily be transmitted on the different ports at exactly the same time (and therefore the residence times will differ). Therefore, from the point of view of Layer 2, the Sync or Follow_Up message transmitted by each successive TC is a new Sync or Follow_Up message compare to the arriving Sync or Follow_Up message that it corresponds to. Nonetheless, we loosely refer to the Sync or Follow_Up message originally transmitted by the GM as being transmitted to all the slaves as a multicast message, even though relative to Layer 2 at each intermediate TC we really have termination, processing, and generation of new messages.

When we refer to Sync or Follow_Up originating at a port, we mean that Sync or Follow_Up are transmitted on this port not as a result of a corresponding Sync or Follow_Up having been received on another port of the same node. This would happen at a GM port. Similarly, when we refer to Sync or Follow_Up egressing the network, we mean that the message is terminated an no new corresponding Sync or Follow_Up is generated. This would happen at a slave port.

1) As a result of executing the BMC algorithm, exactly one OC in the network will be the GM. All the other OCs will be slaves. The GM port is in the master state with respect to Sync and Follow_Up (and Announce) messages. Each port of each slave is in the slave state with respect to Sync and Follow_Up (and Announce) messages.
2) Sync and Follow_Up messages can originate only at a master port (i.e., only a GM port), and can egress the network only at a slave port.

~~If Follow_Up is transmitted by an OC only if the OC is in the master state (and its link is therefore in the master state).~~

3) If the GM is on-the-fly, it sends Sync on its port with the follow-up flag (i.e., the PTP_ASSIST flag) not set and the correction field initialized to zero. It measures the departure time of the Sync message on the port and writes this time in the originTimestamp field. It does not send Follow_Up.
4) If the GM is follow-up, it sends Sync on its port with the follow-up flag set and the correction field initialized to zero. It measures the departure time of the Sync message, and

writes this in the preciseOriginTimestamp of a Follow_Up message that it generates and sends after the Sync message.

~~The GM sends Sync on its port with the follow-up flag (i.e., the PTP_ASSIST flag) set and the correction field initialized to zero, and measures the time of departure of Sync on each port~~

~~The GM sends Follow_Up on its port with the correction field initialized to zero and the preciseOrigin timestamp equal to the measured time of departure of the Sync message on that port~~

5) Each P2P TC port that receives a Sync message measures its arrival time. This arrival time is used for computation of the residence time ~~for the case where the node is not the destination~~.

6) Each slave OC that receives a Sync message measures its arrival time. This arrival time is used fr~~o~~or computation of the slave offset ~~for the case where the node is the destination~~.

7) When a slave OC receives a Sync message with the follow-up flag not set, it adds the correction field of the Sync message and the propagation time on the link on which the Sync arrived to the origin Timestamp in the Sync message. The result is subtracted from the arrival time of the Sync message to obtain the slave offset.

8) When a slave OC receives a Sync message with the follow-up flag set, it waits for the corresponding Follow_Up message.~~, which corresponds to a Sync message for which this slave is its destination, it~~ It adds the correction fields in the Sync and Follow_Up messages and the propagation time on the link on which the Sync arrived to the preciseOrigin Timestamp in the Follow_Up message. The result is subtracted from the arrival time of the Sync message to obtain the slave offset.

9) When a Sync message arrives at an on-the-fly P2P TC node, the residence time is measured and added to the correction field of the Sync message as it is transmitted on each respective port. The propagation time for the link on which the Sync message arrived is added to the correction field of the Sync message. Any corresponding Follow_Up message (which may be due either to the GM or one or more upstream P2P TCs being follow-up) is transmitted without its correction field being altered. [***Author's note: There is a question of whether an on-the-fly P2P TC should hold Sync for any follow-up messages just as a follow-up TC would. It seemed that the Sync message should not be held; the main reason for holding Sync until Follow_Up arrives in follow-up TCs is that the follow-up TC may not be able to process Follow_Up as quickly as Sync, and a number of consecutive Sync messages may arrive at a downstream node before the Follow_Up message corresponding to the first Sync message. This should not be a problem for the case of an on-the-fly TC because (1) it presumably has faster hardware, and (2) it doesn't have to process the Follow_Up, but merely send it on. Nonetheless, this point was not explicitly discussed in the AVB group, and needs to be confirmed.***]

10) When a Sync message with the follow-up flag set arrives at a follow-up P2P TC node, it is held until the corresponding Follow_Up message arrives. On arrival or generation of the Follow_Up message corresponding to the Sync message, the correction field of the Follow_Up Message is added to the correction field of the Sync message. After performing this computation, the Sync message is sent on the respective ports and its departure time on each port is measured.

11) When a Sync message with the follow-up flag not set arrives at a follow-up P2P TC node, the follow-up flag is set. The Sync message is sent on the respective ports and its departure time is measured.

12) ~~On arrival of the Follow_Up message corresponding to the Sync message in (9), the correction field of the Follow_Up Message is added to the correction field of the Sync message~~

13) ~~After performing the computation in (10), the Sync message is sent on the respective ports indicated by the forwarding data base for multicast PTP messages, and its departure time on each port is measured.~~
14) The residence time for the Sync message is computed on each port, and is placed in the correction field of a respective new Follow_Up message generated for each port.
15) The propagation time for the link on which the Sync message arrived is added to the correction field of the Follow_Up message.
16) Each new Follow_Up message is sent on the respective port for which it was generated.

4.1.2 Specialization of rules for processing Sync and Follow_Up to AVB node, which always has collocated P2P TC and OC function

The rules below were obtained by modifying the rules in Section 4.1.1 (taken from [3] and [4]) to apply to the case of a P2P TC ~~colocated~~collocated with a GM or slave OC. ~~We could have alternatively left the rules in [4] as is by defining a logical link, internal to a node, between the OC and PTP TC functions in the node.~~ The rules below capture the behavior external to the node without the need to define an internal, logical link. The discussion in the second and third paragraphs of Section 4.1.1, where we describe how we loosely refer to the Sync or Follow_Up message originally transmitted by the GM as being transmitted to all the slaves as a multicast message, applies here also.

1) As a result of executing the BMC algorithm, exactly one OC in the network will be the GM. All the other OCs will be slaves. All the ports of the GM are in the master state with respect to Sync and Follow_Up (and Announce) messages. All the ports of the slaves are in the slave state with respect to Sync and Follow_Up (and Announce) messages.[11]
2) Sync can originate only at a master port (i.e., only a GM port), and can egress the network only at a slave port.
3) If Follow_Up is transmitted on a port without a corresponding ~~Sync and~~ Follow_Up (and also a corresponding Sync) having been received on another port, then the port Follow_Up is transmitted on must be a master port.
4) If the GM is on-the-fly, it sends Sync on each respective port with the follow-up flag (i.e., the PTP_ASSIST flag) not set and the correction field initialized to zero. It measures the departure time of the Sync message on the port and writes this time in the originTimestamp field. It does not send Follow_Up.
5) If the GM is follow-up, it sends Sync on each respective port with the follow-up flag set and the correction field initialized to zero. It measures the departure time of the Sync message, and writes this in the preciseOriginTimestamp of a Follow_Up message that it generates and sends after the Sync message.

~~4)The GM sends Sync on each respective port (i.e., on all ports indicated by the forwarding data base such that the multicast Sync reaches all slaves) with the follow-up flag (i.e., the PTP_ASSIST flag) set and the correction field initialized to zero, and measures the time of departure of Sync on each port~~

~~5)The GM sends Follow_Up on each respective port with the correction field initialized to zero and the preciseOrigin timestamp equal to the measured time of departure of the Sync message on that port~~

[11] The master and slave states of ports are meaningful to Sync, Follow_Up, and Announce messages. This is because an AVB node can be decomposed functionally into an OC function and a P2P TC function, as in the left-hand illustration of Figure 1. In that illustration, the link connecting the OC and P2P TC is in either the master or slave state. The master and slave states are not meaningful to the ~~Peer_D~~Pdelay messages, because P2P TC ports are stateless. In the left-hand illustration of Figure 1, the ~~Peer_D~~Pdelay messages are transmitted or received on the ports that emanate horizontally from the P2P TC, but not on the link to the OC.

6) Each port that receives a Sync message measures its arrival time. This arrival time is used for both (a) computation of the residence time for the case where the node is not the destination, and (b) computation of the slave offset for the case where the node is the destination.

7) When a slave (i.e., port in the slave state) receives a Sync message with the follow-up flag not set, it adds the correction field of the Sync message and the propagation time on the link on which the Sync arrived to the origin Timestamp in the Sync message. The result is subtracted from the arrival time of the Sync message to obtain the slave offset, which is made available to the slave OC function in the node.

8) When a slave (i.e., port in the slave state) receives a Sync message with the follow-up flag set, it waits for the corresponding Follow_Up message. It adds the correction fields in the Sync and Follow_Up messages and the propagation time on the link on which the Sync arrived to the preciseOrigin Timestamp in the Follow_Up message. The result is subtracted from the arrival time of the Sync message to obtain the slave offset, which is made available to the slave OC function in the node..

9) When a Sync message arrives at an on-the-fly P2P TC node, the residence time is measured and added to the correction field of the Sync message as it is transmitted on each respective port. The propagation time for the link on which the Sync message arrived is added to the correction field of the Sync message. Any corresponding Follow_Up message (which may be due either to the GM or one or more upstream P2P TCs being follow-up) is transmitted without its correction field being altered. [***Author's note: There is a question of whether an on-the-fly P2P TC should hold Sync for any follow-up messages just as a follow-up TC would. It seemed that the Sync message should not be held; the main reason for holding Sync until Follow_Up arrives in follow-up TCs is that the follow-up TC may not be able to process Follow_Up as quickly as Sync, and a number of consecutive Sync messages may arrive at a downstream node before the Follow_Up message corresponding to the first Sync message. This should not be a problem for the case of an on-the-fly TC because (1) it presumably has faster hardware, and (2) it doesn't have to process the Follow_Up, but merely send it on. Nonetheless, this point was not explicitly discussed in the AVB group, and needs to be confirmed.***]

10) When a Sync message with the follow-up flag set arrives at a follow-up P2P TC node, it is held until the corresponding Follow_Up message arrives. On arrival or generation of the Follow_Up message corresponding to the Sync message, the correction field of the Follow_Up Message is added to the correction field of the Sync message. After performing this computation, the Sync message is sent on the respective ports and its departure time on each port is measured.

11) When a Sync message with the follow-up flag not set arrives at a follow-up P2P TC node, the follow-up flag is set. The Sync message is sent on the respective ports and its departure time is measured.

12) The residence time for the Sync message is computed on each port, and is placed in the correction field of a respective new Follow_Up message generated for each port.

13) The propagation time for the link on which the Sync message arrived is added to the correction field of the Follow_Up message.

14) Each new Follow_Up message is sent on the respective port for which it was generated.

~~7)When a slave (i.e., port in the slave state) receives the Follow_Uup message corresponding to a Sync message for which this slave is its destination, it adds the correction fields in the Sync and Follow_Uup messages and the propagation time on the link on which the Sync arrived to the preciseOrigin Timestamp in the Follow_Uup message. The result is subtracted from the arrival time of the Sync message to obtain the slave offset.~~

~~8)When a Sync message arrives at a node that is not its destination, it is held until the corresponding Follow_Up message arrives.~~

~~9)On arrival of the Follow_Uup message corresponding to the Sync message in (8), the correction field of the Follow_Uup Message is added to the correction field of the Sync message~~

~~10)After performing the computation in (9), the Sync message is sent on the respective ports indicated by the forwarding data base for multicast PTP messages, and its departure time on each port is measured.~~

~~11)The residence time for the Sync message is computed on each port, and is placed in the correction field of a respective new Follow_Up message generated for each port.~~

~~12)The propagation time for the link on which the Sync message arrived is added to the correction field of the Follow_Up message.~~

~~13)~~15) ~~Each new Follow_Uup message is sent on the respective port for which it was generated~~

4.2 ~~Peer_D~~Pdelay messages

~~To be supplied.~~

The processing rules below describe the actions taken during the exchange of Pdelay messages between a P2P TC delay requestor and a P2P TC delay responder. Note that the exchange occurs separately and independently in both directions, i.e., each TC initiates sending Pdelay_Req to the other TC independently of the other TC, and each TC responds to the Pdelay_Req received from the other TC. In this manner, propagation time is measured by (and known to) both ends of the link. Note that any consideration of intermediate E2E TCs between the delay requestor and delay responder is omitted, as AVB networks will not contain E2E TCs (E2E TCs would need to be included in a description of more general IEEE 1588 systems).

1) The P2P TC delay requestor sends Pdelay_Req with the correction field initialized to zero, and notes the time t_1 that it sends the message
2) The delay responder measures the arrival time of the Pdelay_Req message, t_2.
3) The delay responder copies the correction field of the Pdelay_Req message to a Pdelay_Resp message correction field, and places the time t_2 (when the Pdelay_Req message arrived) in the Pdelay_Resp message. (***Author's note: Since the correction fields in Pdelay messages are not used in AVB networks (because there are no E2E TCs in AVB networks), it actually is not necessary to set them to zero or copy the contents of the Pdelay_Req correction field to the Pdelay_Resp correction field (as a means of setting it to zero). The correction fields can simply be ignored.***)
4) The delay responder sends the Pdelay_Resp message, and measures the time t_3 that it sends the message. If it is on-the-fly, it places t_3 in the Pdelay_Resp message as it goes out. If it is followup, it also sets the followup flag in the Pdelay_Resp message; it then generates a Pdelay_Resp_Follow_Up message and places t_3 in that message.
5) The delay requestor notes the time t_4 when the Pdelay_Resp message arrives
6) The propagation time is estimated by the delay requestor as $\{ t_4 - t_1 - (t_3 - t_2)\}/2$. This formula assumes the delays are symmetric.
7) Neither P2P TC is master or slave; they are symmetric relative to each other. Each acts as a delay requestor by sending Pdelay_Req to the other, and each acts as a responder by responding to Pdelay_Req from the other. In this manner, each measures propagation time (i.e., propagation time is known to both).

4.3 Announce message

To be supplied

4.4 Frequency compensation

This subsection describes how the information in the Sync and Follow_Up messages received by a P2P TC may be used to (1) measure frequency offset of the GM relative to the local free-running oscillator and (2) synthesize the syntonized timing signal.

4.4.1 Measurement of the frequency offset of the GM relative to the local free-running oscillator

The steps below are equivalent to Eq. (2-6). Note that this is not the only way to implement Eq. (2-6); any implementation equivalent to Eq. (2-6) is acceptable. Note that on sync intervals when the frequency offset measurement is made, it is made after the computation of the syntonized time in Section 4.4.2 (i.e., the steps in Section 4.4.2 are done first).

1) On receipt of a Sync message, increment a counter. The purpose of this is to count *M* sync intervals, because the frequency offset measurement is done every *M* sync intervals. At present, *M* = 10 (though the value may change based on jitter/wander simulation results).
2) If the counter equals *M*, add the value of the timestamp for the time at which the Sync was sent by the grandmaster (contained in the originTimestamp field if the follow-up flag is not set or the preciseOriginTimestamp field if it is set) plus the correction fields in the Sync and Follow_Up messages plus the measured propagation delay on the link that the Sync arrived on. Save this sum.
3) Measure the local oscillator time when the Sync message arrives and save this value.
4) Subtract the sum indicated by step (2) calculated and saved at the previous frequency offset measurement from the sum just calculated.
5) Subtract the local oscillator time measured when the Sync message at the previous frequency offset measurement arrived (this value was saved) from the measurement in step (3).
6) Divide the result of step (4) by the result of step (5)
7) Subtract 1 from the result of step (6), and save this value. This is the measured frequency offset of the GM relative to the local free-running oscillator.
8) Discard the values saved on the previous frequency offset measurement as these are no longer needed. Replace them by the corresponding values obtained for the current sync interval in steps (2) and (3).

4.4.2 Synthesis of the syntonized timing signal

The steps below are equivalent to Eq. (2-9). Note that this is not the only way to implement Eq. (2-9); any implementation equivalent to Eq. (2-8) or (2-9) is acceptable. Note that on sync intervals when the frequency offset measurement is made, it is made after the computation of the syntonized time in Section 4.4.2 (i.e., the steps in Section 4.4.1 are done after the steps here).

1) At each sync interval, measure the time of the local oscillator when the sync message arrives. Save this value.
2) Subtract from the value measured in step (1) the corresponding value measured and saved from the previous sync interval.
3) Multiply the value computed in step (2) by 1 plus the current measured frequency offset of the GM relative to the local free-running oscillator. (Note: it is probably advantageous to save 1 plus the measured frequency offset rather than the frequency offset, but this is a matter of implementation.)

4) Add the value computed in step (3) to the value of syntonized time computed when the previous Sync message arrived.
5) Save the value computed in (4). This is the current value of syntonized time. Discard the syntonized time value computed at the previous sync interval.

5. References

[1] IEEE Standard for Local and Metropolitan Area Networks – Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks, Draft 802.1as PAR forwarded by IEEE 802 EC to NesCom on March 10, 2006.

[2] IEEE 1588/802.1 AVB Design Meeting, February 21, 2006 (see meeting minutes).

[3] *Transparent Clock – Working Technical Description*, Revision 13, IEEE 1588 TC Subcommittee, prepared during February 22 – 24, 2006 IEEE 1588 Face-to-Face Meeting.

[4] Geoffrey M. Garner, *Initial Description of Possible Use of Sync and Followup Messages with Peer-to-Peer Transparent Clocks in AVB*, Samsung Contribution to IEEE 1588 and IEEE 802.1 AVB TG, Revision 1.0, March 1, 2006 (plus accompanying VG presentation dated March 6, 2006 and presented at March, 2006 802.1 meeting).

[5] Dave Tonks, *Variable Length Unified Frames, Formats and Contents*, Version 1.3, IEEE 1588 Short Frames Subcommittee, February 22, 2006.

[6] Ron Cohen and Silvana Rodrigues, *Unifying Short and Long Messages*, Contribution to Precise Networked Clock Synchronization Working Group – IEEE 1588 revision, November 30, 2005.

[7] IEEE 1588, *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and control Systems*, September 12, 2002.

[8] IEEE 1588 Timestamp Subcommittee calls through March 21, 2006.

[9] John Eidson and Bruce Hamilton, *PTP Clock Synchronization Model*, Draft material provided to IEEE 1588 Rewrite Subcommittee for Clause 6 of IEEE 1588, ~~Version 2, March, 2006~~April 10, 2006 version.

[10] Geoffrey M. Garner, *End-to-End Jitter and Wander Requirements for ResE Applications*, Samsung Presentation for IEEE 802.3 ResE SG, May 16, 2005.

[11] *Residential Ethernet(RE) (a working paper)*, Draft 0.142, maintained by David V. James and based on work by him and other contributors, November 16, 2005.

[12] David Roe, *Unified PTPv2 Frame Format Proposal*, Revision 1, Short Frames – Proposal, IEEE 1588 Short Frames Subcommittee, May 3, 2006.

[13] David Roe, *Unified PTPv2 Frame Format Proposal*, Revision 2, Short Frames – Proposal, IEEE 1588 Short Frames Subcommittee, May 4, 2006.

[14] David Roe, *Unified PTPv2 Frame Format Proposal*, Revision 3, Short Frames – Proposal, IEEE 1588 Short Frames Subcommittee, May 10, 2006.

[15] David Roe, *Unified PTPv2 Frame Format Proposal*, Revision 4, Short Frames – Proposal, IEEE 1588 Short Frames Subcommittee, May 12, 2006.