CONSIDERATIONS FOR DEPLOYING IEEE 1588 V2 IN NETWORK-CENTRIC DATA ACQUISITION AND TELEMETRY SYSTEMS

Todd Newton, Evan Grim, Myron Moodie

Southwest Research Institute®
Automation and Data Systems Division
San Antonio, TX USA
todd.newton@swri.org; evan.grim@swri.org; myron.moodie@swri.org

ABSTRACT

Network-centric architectures continue to gain acceptance in data acquisition and telemetry systems. Though networks by nature impose non-deterministic transit time of data through a given link, the IEEE 1588 standard provides a means to remove this jitter by distributing time messages to the data acquisition units themselves. But like all standards, they evolve over time. The same is true with IEEE 1588, which is releasing its second version later this year. This paper discusses the challenges of the first version of the IEEE 1588 standard that Version 2 set out to address, potential challenges with Version 2, and interoperability issues that may arise when incorporating a mixture of Version 1 and Version 2 devices.

KEY WORDS

IEEE-1588, Time Synchronization, Networks, Network-Centric, Data Acquisition, Flight Test

INTRODUCTION

Networks and network-centric systems seem to be everywhere these days. Whether made up of a large corporation's IT infrastructure or one's personal home office, networks are used on a daily basis for communication and data transport. Yet even with the mass familiarity of networks, the telemetering community has only recently begun to incorporate networks into their telemetry systems. One of these reasons is because until the past few years, networks were not equipped to meet the timing requirements of data acquisition and telemetry systems. It was not until the introduction of the IEEE 1588 standard that defines the Precision Time Protocol (PTP) in late 2002 that began to breathe new life on the concept of incorporating networks into the world of flight test. The use of networks in telemetry systems is now a viable alternative to the traditional IRIG 106 Chapter 4 standard for PCM-based test and telemetry systems.

However, like all standards, there grows a need to revise the standard in order to incorporate newer technology and to address different usage scenarios that may not have been considered in a previous version. The same is true for the IEEE 1588 standard. While version 2 (v2) of the PTP standard seeks to answer the questions and challenges left by its predecessor, it brings about a few challenges of its own in regards to system integration and interoperability with version 1 (v1) devices.

Southwest Research Institute has been involved in developing, testing, and integrating a data acquisition and telemetry flight test system that employs the use of IEEE 1588 v1 over the past few years. At the time of writing this paper, the IEEE 1588 v2 standard has not been officially released. Even so, a look at the v2 Draft 2.2 standard through the lens of our v1 expertise has given a platform for identifying possible issues when meshing v1 and v2 devices. This paper will provide guidelines to consider when choosing to deploy an IEEE 1588 v2 system. Though the standard is not exclusive to IEEE 802.3 (Ethernet), this paper will only discuss those network systems realizing the PTP protocol over Ethernet.

PTP BACKGROUND

Before attempting to comprehend the possible interoperability issues that may be encountered, one must have a decent grasp at what the IEEE 1588 standard is all about and a general understanding of how it works. As the name "Precision Time Protocol" suggests, the standard defines the protocol for synchronizing different clocks to a high degree of precision. The protocol is able to provide sub-microsecond accuracy between devices distributed across the network. In its simplistic form, there is one clock in the network that synchronizes all other clocks. The clock that is the source of time on the network is referred to as the Grandmaster clock. All devices that synchronize to this clock are referred to as Slave clocks. Some devices may have multiple ports, such as a network switch that implements a boundary clock. A boundary clock will place all of its ports in a Master state if it is acting as the Grandmaster clock. Otherwise, it will place the port that receives the timing packets in the Slave state while the rest of the ports remain in the Master state. In this case, the boundary clock synchronizes its main clock to the Grandmaster clock with the timing information received over its Slave port, but it generates timing packets over its ports in the Master state in order to synchronize other Slaves clocks which could be end devices or other boundary clocks

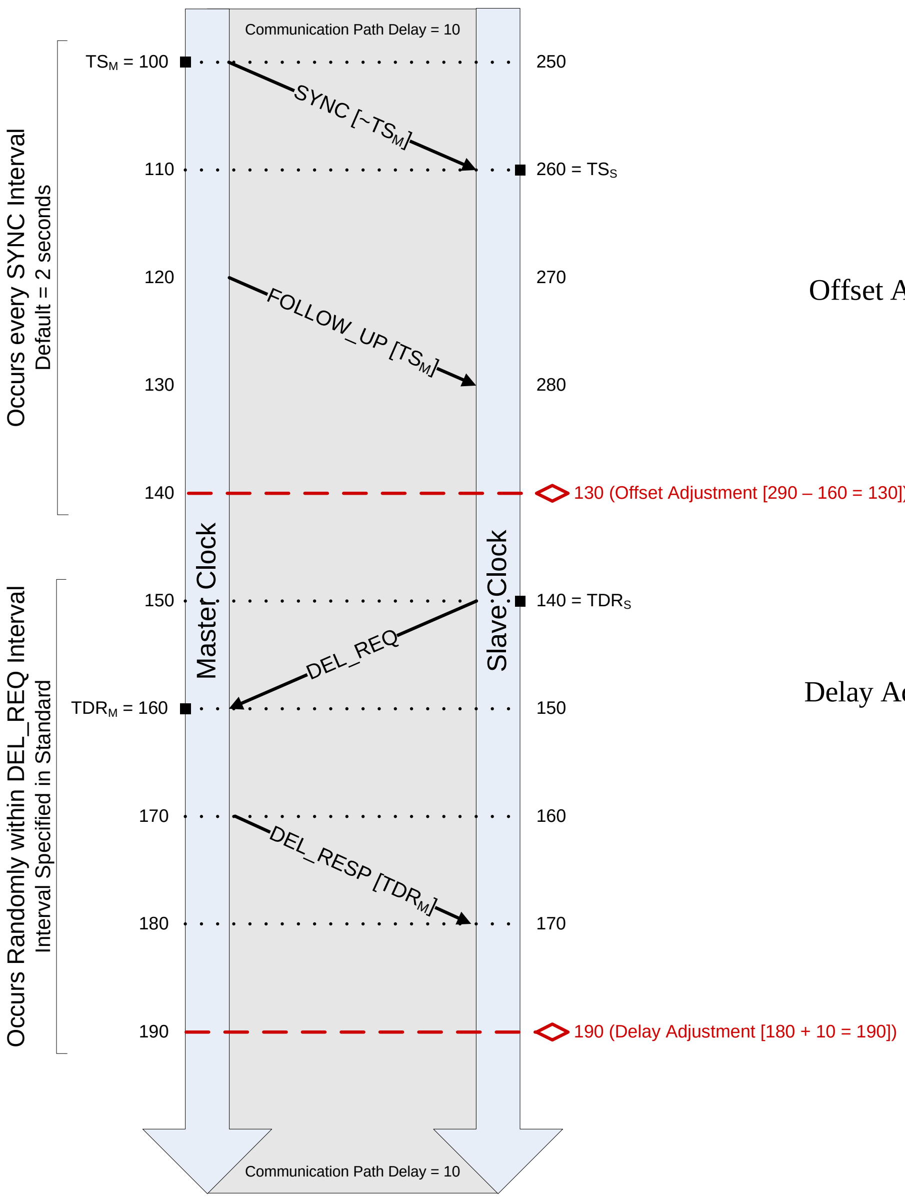
The method in which a Slave is synchronized to its Master under PTP v2 follows the same structure and principles as in PTP v1 even though the latest revision has added a few new message types and has shuffled up the formats of the existing message types. These differences will be discussed later in this paper. But before one can synchronize two or more devices, one must first elect one clock to act as the Grandmaster to which the rest of the devices can synchronize. This election can be done manually by configuring certain clock properties on a given device, or it can be done automatically by way of the Best Master Clock (BMC) algorithm. Version 2 introduces the ANNOUNCE message type to carry the pertinent information needed to make proper decisions with respect to which clock is the best master. Information about the clock included in this message includes attributes relating to the clock's time source reference, accuracy, and a user-specified priority level.

Once a Grandmaster has been determined, it begins by generating SYNC and FOLLOW_UP messages at a configurable sync interval. Though the FOLLOW_UP messages are optional in the standard, they are required in order to achieve high precision synchronization which is needed for real-time data acquisition systems. When a SYNC message is assembled and transmitted from a Master port, the embedded timestamp in the message is approximated while the precise time that the message was transmitted onto the communication medium is captured. The captured time is then embedded into the FOLLOW_UP message. When the Slave device receives the SYNC message, it takes a snapshot of its local time when the SYNC message is first received. The slave then compares the precise time it received the SYNC message with the embedded timestamp in the FOLLOW_UP message stating the precise time when the SYNC message was transmitted from the Master. A simple subtraction operation would give you the time adjustment needed on the Slave in order to be synchronized with the Master if this were an ideal world and there was not a transit delay. But the facts are, this is not an ideal world, and transit delays do exist. To determine the transit delays, there are two mechanisms that can be employed.

The delay request-response mechanism is a direct carry-over from v1 of the standard. This mechanism contains two messages: DELAY_REQUEST and DELAY_RESPONSE. A DELAY_REQUEST message is issued by the Slave clock. When the message leaves the Slave, the Slave records its precise timestamp. The Master records the precise timestamp that it received the DELAY_REQUEST. The Master then embeds the message receipt time into a DELAY_RESPONSE message and then sends it to the Slave that sent the request. Once the slave has these values, it is able to determine the path delay. The path delay is assumed to be symmetrical though in practice it may not be. This delay is then used in conjunction with the SYNC and FOLLOW_UP messages to achieve synchronization.

The second path delay mechanism is the peer delay mechanism. It has no v1 counterpart. This mechanism brings with it three new message types: PDELAY_REQUEST, PDELAY_RESPONSE, and PDELAY_RESPONSE_FOLLOW_UP. Unlike the delay request-response mechanism, the requesting node does not have to be the Slave. With the peer delay mechanism, both ports on a link are able to learn the path delay. This allows for a rapid recovery from a change in network topology. For discussion, Node A will be the requesting node; Node B will be the responding node. To determine the mean path delay, Node A issues a PDELAY_REQUEST message. The time at which Node A transmits the message is time $TREQ_A$. Once Node B receives the message, it stores the reception time, $TREQ_B$. Node B then issues a PDELAY_RESPONSE message back to Node A. Embedded into this message is the PDELAY_REQUEST reception time $TREQ_B$ and the approximate PDELAY_RESPONSE transmit time, $TRESP_B$. Node A stores the time that it receives the PDELAY_RESPONSE message, $TRESP_A$. If greater accuracy is required, Node B can issue a PDELAY_RESPONSE_FOLLOW_UP message after it sends the PDELAY_RESPONSE message in order to provide Node A with the precise transmit time of the PDELAY_RESPONSE message. Once Node A has received the response message(s), it can then proceed to use the four timestamps that it has to compute the mean path delay.

To better understand the basic principles of the protocol, it may be best to walk through an example. Figure 1 describes the example scenario. The figure shows two clocks over time. The one on the left is the Master clock, and the one on the right is the Slave clock. The time scale referenced in the figure is arbitrary, and the units referenced are in generic ticks. The example assumes a symmetrical communication path delay of 10 ticks. At the onset of the example, the Slave clock is free running, not synchronized to the Master. The master has an initial time value of 100 ticks while the free running slave has an initial time value of 250 ticks.



$$\begin{aligned}\text{Offset Adjustment} &= TS_S - TS_M \\ &= 260 - 100 \\ &= 160\end{aligned}$$

$$\begin{aligned}\text{Delay Adjustment} &= \frac{TDR_M - TDR_S}{2} \\ &= \frac{160-140}{2} \\ &= \frac{20}{2} \\ &= 10\end{aligned}$$

Figure 1. Example SYNC/FOLLOW_UP with Delay Request Mechanism

The Master clock issues a SYNC message at its local time 100 (TS_M). The actual SYNC message will not arrive at the Slave until 10 ticks later due to the link delay, making the reception time 260 (TS_S)on the local Slave clock. The Master then generates a FOLLOW_UP message that includes the precise origin timestamp of the SYNC message. Once the Slave receives this message, it is able to determine its time offset from the Master. Simple subtraction reveals that the Slave clock is 160 ticks ahead of the Master, and the Slave responds to this offset by adjusting its local clock by the difference. However, synchronization is not achieved until the link delay is accounted for. As can be seen in Figure 1, after the first adjustment the Slave is running 10 ticks behind the Master. The figure illustrates the delay request-response mechanism as it accounts for the path delay.

At the local Slave time of 140 (TDR_S), the Slave issues a DELAY_REQUEST message to the Master. The communication path delays the message for 10 ticks before the Master receives the message, pushing the time on the Master to 160 (TDR_M). The TDR_M is returned to the Slave in a DELAY_RESPONSE message. The difference between the TDR_M and the TDR_S represents the delay incurred by two path traversals, one in each direction. The first path traversal is inherited from the SYNC message. Dividing the difference by two yields the path delay for a single path traversal in either direction. This method assumes the path delay is relatively the same from Master to Slave as it is from Slave to Master.

Suppose the network supports the peer delay mechanism. The DELAY_REQUEST and DELAY_RESPONSE messages are replaced by PDELAY_REQUEST, PDELAY_RESPONSE, and an optional PDELAY_RESPONSE_FOLLOW_UP. The main difference between the two mechanisms is that the peer delay method measures time delays in both directions, getting a round-trip measurement. The delay request-response method only measures the delay from the requestor to the responder. Figure 2 illustrates an example of the peer delay mechanism at work. In this example, the path delay is not symmetrical. Traffic flowing one direction takes 10 ticks of time while traffic flowing in the other direction takes 20 ticks of time. Keep in mind that with this method, the requestor may be either the Master or the Slave.

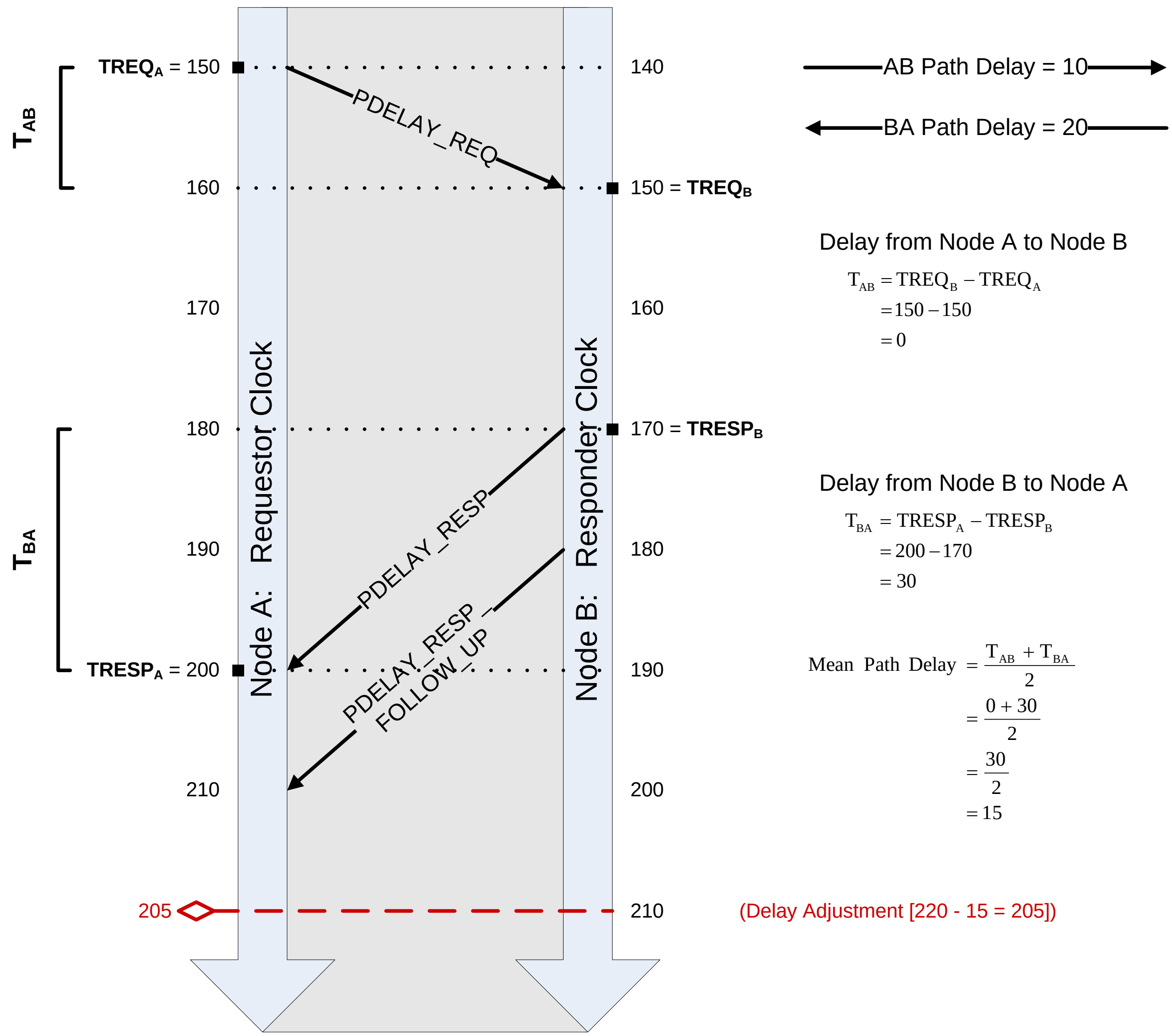


Figure 2. Example Peer Delay Mechanism

For ports utilizing the peer delay mechanism, the first step to determining the mean path delay is for Node A, the requestor, to issue a PDELAY_REQUEST message. In this example Node A begins with a clock time 10 ticks ahead of Node B. Node A transmits the request at time 150 ($TREQ_A$). After the 10 tick path delay, Node B receives the message at local time 150 ($TREQ_B$). Node B delivers a PDELAY_RESPONSE message at time 170 ($TRESP_B$) that includes time $TREQ_B$. The time that Node A receives the PDELAY_RESPONSE message is 200 ($TRESP_A$). Node B then sends a PDELAY_RESPONSE_FOLLOW_UP message that includes time $TRESP_B$. Once Node A contains all four timestamps, it is able to compute the measured two-way delay. Division by two yields the mean path delay. A Slave clock can then use the known link delay in its next synchronization computation. The reason for the final computation to still be off by 5 ticks has to do with the asymmetry of the path delay and not on the method itself. If the delay request-response method were used, similar results would have been obtained. The method by which the directional asymmetry is determined is out of scope for this paper.

Both methods for determining the mean path delay are good ones. One benefit from the peer delay mechanism is the quick recovery time should the network topology change. This benefit is available because the path delay of a link is known on both ports, unlike the delay request-response method in which only the Slave clock knows the delay. However, before the decision to use one delay mechanism instead of the other is made, a few more new features need to be mentioned.

NEW FEATURES

In addition to the new message types and formats, a couple of new PTP devices have also been added, namely the end-to-end (E2E) transparent clock and the peer-to-peer (P2P) transparent clock. These clocks take advantage of the correction field in the PTP header, another new parameter added in this version of the standard. The correction field is used to provide a receiving device with specific timing adjustments that can be applied to the timestamp prior to processing. The correction field may contain fractional nanoseconds.

Transparent clocks are attractive because they are able to account for their own internal latency, something that is common to all network switching devices. This internal latency is referred to as the residence time. A transparent clock predicts the residence time of a PTP packet by subtracting the ingress timestamp from the predicted egress timestamp. If the transparent clock is a two-step clock, it can then use the precise egress timestamp instead of the predicted egress timestamp. The residence time is added to the correction field of the PTP packet before it is forwarded to the next port. The residence time is dynamic, and it varies based on network load on a port by port basis. The presence of transparent clocks removes the need for boundary clocks.

Figure 3 provides a visual representation of transparent clock operation, specifically that of an E2E transparent clock. For the sake of simplicity, the example consists of one-step clocks, meaning that FOLLOW_UP messages will not be sent. One-step clocks place all timing information in the SYNC message, and the transparent clock adds its residence time to the correction field before sending the message to the slave. If two-step clocks were in use, the FOLLOW_UP messages would contain the transparent switch's residence time in the correction field. The example assumes that the path delay between the Master and transparent clocks is symmetrical and takes 10 ticks of time to traverse the path one way. The path delay between the transparent and Slave clocks is also symmetrical, but this delay consumes 15 ticks of time to traverse the path one way.

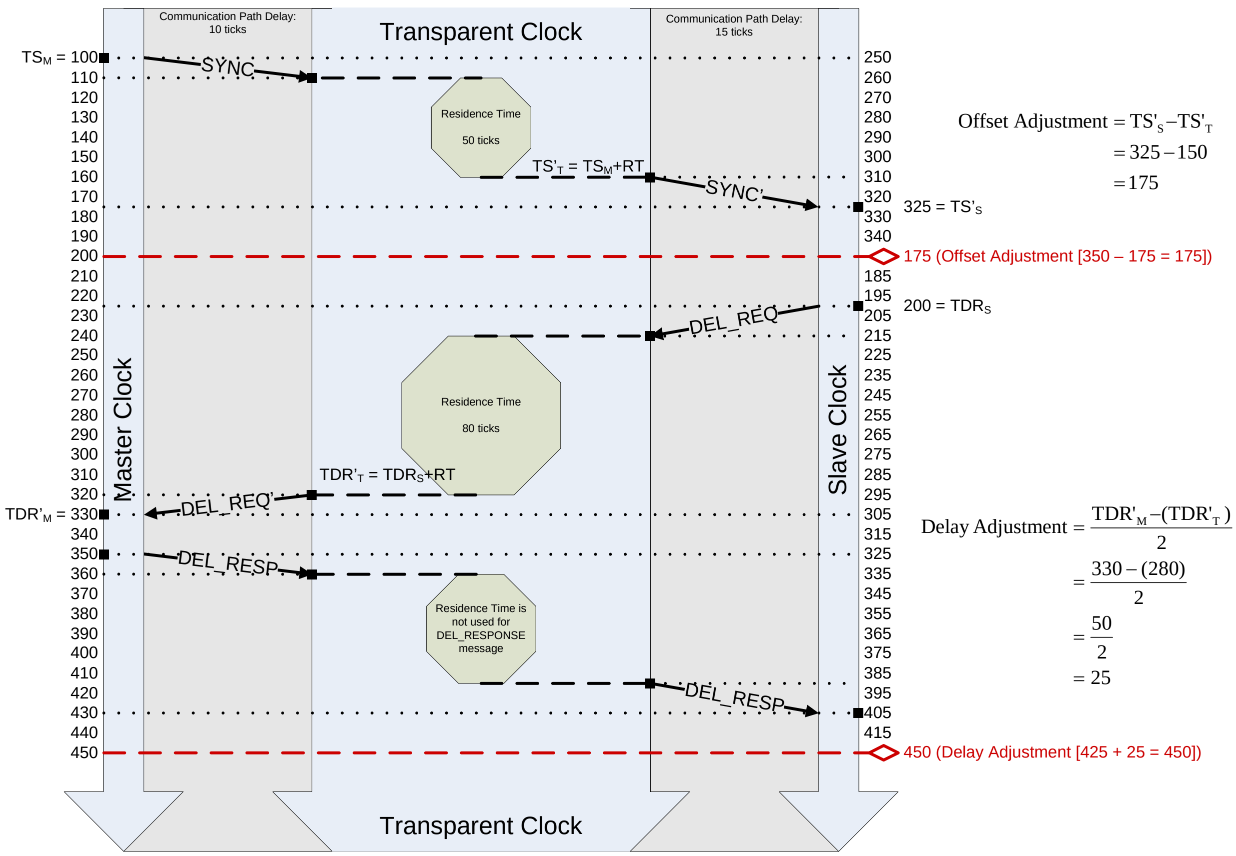


Figure 3. Example of E2E Transparent Clock Operation

In very similar fashion to the first example, the time on the Master clock begins at its local time 100 while the Slave clock is running at its own local time 250. The Master transmits the SYNC message with an embedded timestamp of 100, TS_M. It arrives at the transparent clock 10 ticks later. At the time of arrival, the transparent clock projects a residence time of 50 ticks. It modifies the original SYNC message by adding the residence time to the correction field in the PTP header of the message. The new message, SYNC', is then delivered to the Slave. After a brief 15 tick path delay, the updated SYNC' message arrives at the Slave. SYNC' reports the Master clock is at time 150 when it arrives at the Slave at time 325, TS'_S. A quick subtraction operation gives the offset from the Master being +175 ticks. This is subtracted from the Slave clock in order to make the appropriate adjustment. A DELAY_REQUEST message is issued from the Slave at time 200, TDR_S. After the 15 tick path delay, the transparent clock receives the packet. Upon reception, the current residence time for the egress port towards the Master is 80 ticks. The residence time is added to the correction field of the DEL_REQUEST packet before it is transmitted to the Master. The updated request, DEL_REQ', arrives at the Master at time 330, TDR'_M. This value is returned to the Slave in the subsequent DELAY_RESPONSE message. Like all packets traversing through the transparent device, there is a packet delay, but no modifications are made to the DELAY_RESPONSE packet regarding residence time since its transit time is not critical, only the timing data contained therein. The Slave uses this time to compute the mean path delay, which is found to be 25 ticks from Master to Slave. The slave clock is adjusted by this amount, and synchronization through a transparent clock is complete.

One thing to note from Figure 3 and its example is that the time of the transparent clock is not known nor is it required for any computation. The key to transparent clocks is not time synchronization but frequency synchronization of the clock, known as syntonization. Transparent clocks need to synchronize to the frequency of the Grandmaster in order have an accurate residence time prediction. The manner by which transparent clocks syntonize to the Grandmaster is not discussed in this paper.

The basic principles of transparent operation apply to both E2E and P2P transparent clocks. However, E2E transparent clocks only use the delay request-response mechanism, and P2P transparent clocks only use the peer delay mechanism. If both types of transparent clocks are present in the same system, they must be separated by a boundary clock. Boundary clocks do not need to be used if the system only uses one of the two types of transparent clocks.

PTP DEPLOYMENT CONSIDERATIONS

With the evolution of any standard, every system integrator wants to know one thing: how will this revision affect my system that is already working? This question is certainly no stranger to those that have already successfully deployed a PTP v1 system as they see v2 approaching on the horizon. The hope is always for a seamless migration from one version to the next in such a way as to protect the investment already poured into the previous version. This is as true for product companies as it is for system designers. As such is the case, the discussion now turns to examine the built-in backward compatibility features of v2 in order to get a grasp of the possible issues one could run into when meshing v1-compliant devices with v2-compliant devices.

While the general concept of how synchronization is achieved remains unchanged, the message formats themselves have changed rather significantly. The format of the PTP header has changed to the point that a v1 clock, Master or Slave, would not be able to interpret a v2 PTP message on its own. The same is true of a v2 clock receiving a v1 PTP message. These changes alone are enough to indicate that the meshing of v1 and v2 clocks will not be as simple and easy as one would hope. But this fact does not lend itself to say that there shall not be any meshing of v1 and v2 devices either. The standard does specify translation procedures in order to convert a message type from one version to the other.

Though translating between the two versions may be the only approach, it is not without its quirks and limitations either. Just as language translators often run into problems trying to communicate concepts from one language to another, there are some pieces of data that PTP cannot translate. These non-transferrable concepts consist of data that has been added in v2 or has been made obsolete from v1. For example, the error correction field that is introduced into the PTP header in v2 has no corresponding v1 field. Also, the three message types introduced in the peer delay mechanism of v2 cannot be translated into v1. The important thing to remember, though, is that timing information pertinent to device synchronization can be successfully translated between v1 and v2 clocks.

It must be pointed out that version translation does not happen automatically. A device running the v2 protocol is not required to support the notion of translating into and out of v1. In order for version translation to occur, a system designer will need to ensure that a PTP translator is in the path between v1 and v2 clocks. A boundary clock with translation capabilities should be used to translate between the two versions. Because transparent clocks are not defined in v1, transparent clock translation falls outside of the scope of the standard.

A good rule of thumb for designing and integrating a meshed network is that boundary clocks should be placed on the boundaries of the different protocol versions and features. A boundary clock that supports v1-v2 translation should be placed between a region of v1 devices and a region of v2 devices. A boundary clock should also be used to join regions of E2E transparent clocks with P2P transparent clocks if both types are used. Though some extra cost may be incurred to incorporate a boundary clock device with translation support, it is the key component used to tie the two versions together. Without translation, failure is inevitable.

CONCLUSION

When all is said and done, PTP v2 is capable of providing more precise time synchronization than in the initial version thanks to the support of fractional nanoseconds. The introduction of transparent clocks into the standard provides more flexibility and choices for system designers. With these additional benefits, it is hard to imagine v2 not surpassing the popularity of v1. But before jumping in with both feet, it will be wise to recognize the potential integration issues that are likely to surface if the mesh network of v1 and v2 devices are not designed appropriately. A well thought out network with the proper v1-v2 translating boundary clocks should mitigate the major hurdles in deploying a meshed network.

ACKNOWLEDGEMENTS

I would like to thank my colleagues at SwRI for their support, encouragement, commitment to excellence, and sense of humor, all of which keep the world of research exciting and rewarding.

REFERENCES

Grim, Evan, “Achieving High-Accuracy Time Distribution in Network-Centric Data Acquisition and Telemetry Systems with IEEE 1588,” Proceedings of the International Telemetering Conference, San Diego, California, 2006.

IEEE Instrumentation and Measurement Society, IEEE P1588 D2.2: Draft Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, Jan 2008.