IEC/IEEE 60802 Contribution – Annex D (Informative) Time Synchronization Informative Annex

Version 09 – October 2023

David McCall (Intel Corporation)

D.1 Overview

This document describes how a network of compliant devices can achieve a time synchronisation accuracy, at the application level, of ±1 µs, relative to the Clock Source at the Grandmaster, over 100 network hops. To achieve this, it allocates the overall error budget of 1 000 ns as described in Table D.1.

Table D.1: Time Synchronisation Error Budget

Network Aspect	Error Type	Network-Level Error Budget (ns)	Normative or Informative?
All PTP Instances	Constant Time Error	200	Normative
	Dynamic Time Error	600	
All PTP Links	Constant Time Error	200	Informative
	Dynamic Time Error	Negligible	

A chain of 1 Grandmaster PTP Instance, 99 PTP Relay Instances and 1 PTP End Instance (100 network hops) that all comply with the normative requirements of sections 6.2.2 and 6.2.3 will generate a network-level Time Error at or below the Error Budget for All PTP Instances.

~~Subclause~~ D.2 describes the principles of operation this document assumes.

~~Subclause~~ D.3 provides additional information on specific normative requirements.

The principles of operation include the use of crystal oscillators (XOs) as opposed to more accurate, stable and costly options such as temperature-compensated crystal oscillators (TCXOs).

~~Clause~~ D.4 describes a potential approach to testing the normative requirements. It is not a test specification but rather a high-level overview one potential approach that might be adopted by a full test specification.

The use of XOs means that some of the normative requirements are difficult or impossible to meet without employing algorithms that track Neighbor Rate Ratio drift and Rate Ratio drift and compensate for consequent errors in calculating Rate Ratio and Correction Field.

~~Clause~~ D.5 ~~of Annex D~~ provides examples of algorithms that can be used for this purpose, and which have been shown to enable compliance with the normative requirements.

Clock drift in neighboring PTP Instances that use XOs means implementations that employ TCXOs or other more accurate, stable oscillators can still find some of the normative requirements difficult or impossible to meet without employing algorithms to track and compensate for errors due to clock drift.

There is no normative requirement to use the algorithms described in ~~Clause~~ D.5; an implementation can employ alternative algorithms provided the normative requirements are met. ~~Clause~~ D.5 describes the potential risks of deploying a network whose instances employ a mix of different algorithms. It is the responsibility of implementers to mitigate the risks and ensure alternative algorithms deliver the network-level performance.

This document does not include normative requirements for PTP Links. ~~Annex~~ D.2.~~3~~ 4 describes PTP Link characteristics that influence achieving 1 µs time synchronisation accuracy. It includes some examples using common PTP Link characteristics.

This document's normative requirements regarding instance-level error generation are necessitated by the need to ensure not just an overall level of dTE generation at each node, but also the performance of drift tracking and error compensation algorithms and the amount of dTE generation due to timestamp error verses clock drift. The algorithms are employed to mitigate errors due to clock ~~drift, but~~drift but cannot mitigate timestamp errors. ~~Clause D.5 describes an example approach to testing the normative requirements. It is not a test specification nor the only viable approach.~~

D.2 Principles of Operation

D.2.1 General

Achieving ±1 µs time synchronisation accuracy across 100 network hops involves managing the accumulation of errors in the Precise Origin Timestamp + Correction Field and the Rate Ratio as they are passed, via Sync or Follow_Up messages, down the chain of PTP instances and are then used by the PTP End Instance to keep its ClockTarget in line with the ClockSource at the Grandmaster PTP Instance. The majority of significant errors can ultimately be traced back to one of three sources: timestamp error, clock drift, or path delay asymmetry. The selection of PTP protocol parameters often involves trading off one source of error against the other. This document requires specific PTP protocol configurations, and assumes the use of mechanisms (algorithms), that reduce dTE due to timestamp error but would also – without additional measures – increase dTE due to clock drift to the point where the latter exceeds the allocated error budget. However, this document also assumes additional measures to minimise some sources of dTE due to clock drift and mechanisms and to track and compensate for errors from other sources to a sufficient degree that the error budget is not exceeded.

The specific protocol configurations and other measures, along with their intended effects, are described in Table D.2.

Table D.2: Protocol configurations & other measures to achieve dTE budget

Configuration or Measure	Description and Intended Effect(s)
Sync Interval 125 ms	Effects: 1. Calibrate the balance between dTE from timestamp error vs error due to clock drift. Larger intervals lead to less timestamp error and more error due to clock drift.

	2. Keep below acceptable limits the impact of errors in Rate Ratio and Rate Ratio Drift estimation when keeping ClockTarget in line with ClockSource between arrival of Sync messages. Larger intervals increase the impact of any errors.

Drift_Tracking TLV - syncEgressTimestamp	Effect: Enables calculation of NRR using Sync message timestamps, which eliminates error due to NRR clock drift that would otherwise occur between calculation of NRR using Pdelay_Resp messages and use during Sync message processing (i.e. calculation of Rate Ratio and output Correction Field values)
NRR Smoothing	Description: Algorithm to use timestamps from multiple past Sync messages when estimating NRR. Effect: Reduce the amount of error in the estimate of NRR due to timestamp error while increasing the amount of error due to clock drift.
NRR Drift Tracking & Compensation	Description: Algorithm to use timestamps from multiple past Sync messages to estimate NRR drift then apply compensation to correct for consequent errors in NRR Smoothing calculation. Effect: Mitigate the effect of errors due to clock drift when calculating and using the estimated NRR.
Drift_Tracking TLV – rateRatioDrift	Description: Carries estimate of Rate Ratio drift rate from one node to the next. Effect: Allows each node to estimate its own Rate Ratio drift rate by combining the incoming Rate Ratio drift rate with the local estimate of NRR drift rate.
RR Drift Compensation	Description: Algorithm that uses the estimate of RR drift rate to compensate for that drift, adjusting the estimated RR over time according to the drift rate. Effect: For PTP Relay Instances, minimises errors in the Correction Field caused by Rate Ratio drift. For PTP End Instances, a similar approach can reduce errors in keeping ClockTarget in line with ClockSource between arrival of Sync messages, but is outside the scope of this document.
Pdelay Interval Consistency	Description: This document requires tighter control of the interval between Pdelay messages generated at the Grandmaster PTP Instance than the defaults in IEEE 802.1AS-2020. Effect: This document requires the use of Sync messages to calculate NRR (see above). However, when a sufficient number of Sync messages are not available, for example on startup or after a reconfiguration, Pdelay_Resp messages may be used instead. In such cases, errors due to clock drift at Relay Instances have a tendency to cancel out. A clock drift which generates a positive error in NRR measurement on receipt of a Pdelay_Resp message generates a negative error in NRR

	measurement at the next node. The degree of cancellation depends on the consistency of the intervals over which NRR is measured at neighboring nodes. Tighter control of the Pdelay Interval increases the consistency of the measurement interval and thus decreases the amount of error.
Mean Residence Time	Description: This document defines a mean Residence Time requirement, which is significantly lower than the default maximum Residence Time in IEEE 802.1AS-2020. Effect: The amount of error in the Correction Field at the PTP End Instance due to clock drift is proportional to the cumulative meanLinkDelay and residenceTime experienced by a Sync message during transit from the Grandmaster PTP Instance to the PTP End Instance. Specifying a lower mean residenceTime reduces this source of error.

D.2.2 Grandmaster PTP Instance Implementation

Depending on implementation, a Grandmaster PTP Instance can…

1. Contain a single oscillator used for both Local Clock and Clock Source,
2. Contain separate oscillators for Local Clock and Clock Source, or
3. Contain only an oscillator for Local Clock and accept an external input for Clock Source.

In some cases a Grandmaster PTP instance can support more than one mode of operation and transition between them depending on changes in network configuration (see Splitting, Joining and Aligning Time Domains).

In the first case the rateRatio and rateRatioDrift fields transmitted by the Grandmaster PTP Instance will be zero, reflecting the fact there is no difference between the Local Clock and Clock Source frequencies.

In the second and third cases there can be differences between the Local Clock and Clock Source frequencies. Any differences will be reflected in the rateRatio and rateRatioDrift fields transmitted by the Grandmaster PTP Instance. This means that Grandmaster PTP instances will track rateRatio over time in order to calculate rateRatioDrift, similarly to PTP Relay Instances and PTP End Instances. The exact implementation can vary.

D.2.3 Splitting, Joining and Aligning Time Domains

Modular machines or production cells can allow the splitting and combining of machines if this is required by the production process. When separate, the ClockSources of two machines run separately, each with its own time domain. If both ClockSources are traceable to the same PTP timescale, the difference between the ClockSources ~~may~~ might be relatively small. If traceable to different timescales, especially if one or both are ARB timescales, there can be a very large difference between the ClockSources.

When two machines are joined, the first machine's time domain remains unaffected, and it can continue operation without disruption. There are two typical approaches to how the second machine behaves. In the first case, at a time of the end user's choosing, the second machine's time domain ceases to exist,

with its PTP Instances becoming part of the first machine's time domain. In the second case, the second machine's time domain is gradually aligned with the first machine's time domain such that control loop cycles are coordinated. In the first case the second machine's time domain is unaffected, and it can continue production even if the machines are accidentally connected, until the end user chooses to join the time domains. In the second case the second machine can continue production while its time domain is being aligned.

D.2.3.1 Joining Machines with Single Time Domain

In the first case, where the second machine's time domain ceases to exist, a discontinuity in timing for the second machine's PTP Instances can occur, as they switch to use the first machine's Grandmaster. Some implementations implement measures to limit such timing discontinuities, but these measures are outside the scope of this document. Typically, in this case, the second machine is not operational while it is joined to the first. It resumes operation once its PTP Instances have synchronised with the first machine's Grandmaster.

D2.3.2 Joining Machines with Multiple Coordinated Time Domains

In the second case, where the second machine's time domain is gradually aligned with the first machine's time domain, this typically requires both machines to be implementing the same control loop cycle time. The goal is that, once coordinated, each control loop cycle of the first machine will be aligned with the start of a control loop cycle of the second machine, even though the two machines maintain separate time domains and there can be a large difference between their Clock Sources.

In this case, after being joined together, the first machine effectively drives the second machine's Clock Source faster or slower, during an alignment period, until coordination is achieved. During the alignment period, this drive from the first machine can result in the second ~~machines'~~ machine's Clock Source temporarily exceeding the usual normative requirement on range of fractional frequency offset relative to the nominal frequency of ±50 ppm. The usual normative requirement on range of rate of change of fractional frequency offset of ±1 ppm/s, applicable when split (i.e. independent) or coordinated (i.e. joined and stable, after the alignment period), may also be temporarily exceeded. However, if the value stays with the range ±1.5 ppm/s, the network-level performance of 1 µs time synchronisation accuracy can be maintained. For this reason, this document specifies a separate normative requirement for temporary, externally driven, rate of rate of change of fractional frequency offset.

Since the second machine experiences no time discontinuities and the network-level performance is maintained the second machine can, if desired, continue operation during the alignment period.

Once coordinated, the first machine continues to drive the Clock Source of the second machine to maintain coordination. In this stable, coordinated mode of operation the normal range of ±1 ppm/s is not exceeded.

The mechanism by which the first machine drives the Clock Source of the second machine is out of scope for this document.

D.2.3.3 Splitting Machines

In the first case, where the second machine's time domain ceased to exist while joined to the first, splitting machines means that the second machine must create it~~'~~s own time domain again. The second

machine's Clock Source typically starts at the PTP Grandmaster Instance's last, best estimate of the first machine's Clock Source. The goal is for no discontinuities in time sync to occur; however, depending on implementation, ~~It~~ it can take some time to before the time synchronisation accuracy of all the second machine's PTP Instances relative to its Grandmaster can be relied upon. For this reason, the second machine ~~is typically~~might not be operational during the split. Hot Standby can be employed to mitigate this transition time, but the details of how to do so are out of scope for this document.

In the second case, where the second machine maintains its time domain while joined to the first, splitting machines means that the first machine ceases driving the second machine's Clock Source to maintain coordination of control loop cycle times. Without this drive, the two time domains can drift relative to each other resulting in loss of coordination. Time synchronisation performance within the second machine is maintained during the split and the second machine can, if desired, continue operation throughout the process.

D.2.4 PTP Link Characteristics

A vast majority of time synchronisation error due PTP link characteristics is cTE due to asymmetrical path delay in one direction verses the other. The mechanism to measure path delay assumes the link is symmetrical and cannot detect asymmetry, thus asymmetry causes an error. The potential maximum asymmetry and thus error typically scales linearly with physical path length.

The error budget for cTE due to PTP link characteristics for an entire network is 200 ns. In any specific network this budget can be allocated as required with some links allocated a higher budget (typically longer length) than others.

A typical specified maximum delay skew for Category 6 Ethernet cables is 50 ns per 100 m. If such cables are used, a maximum total cable length between Clock Source and Clock Target with 99 PTP Relay Instances between them (i.e. 100 network hops) is 400m. Extending the cable length beyond 400 m without jeopardizing network-level performance would require the use of cables with less delay skew or asymmetry compensation for delay skew.

It is possible for the delay skew in one section of cable to cancel all or part of a delay skew in the opposite direction from prior section but, depending on how cables are manufactured and deployed, it is feasible for the delay skews of every cable segment between a Grandmaster PTP Instance and a PTP End Instance to be additive.

D.3 Notes on Normative Requirements

D.3.1 Oscillator Requirements

Clock drift at the Grandmaster PTP Instance causes greater dTE than the same amount of clock drift at a PTP Relay Instance or the PTP End Instance. This document therefore requires tighter limits on maximum fractional frequency offset for an oscillator at the Grandmaster PTP Instance than at other instances.

This document does not place requirements on operational temperature range or other environmental factors. The required oscillator behaviour is delivered for the operational conditions across which a device claims it is compliant. These conditions typically include temperature range but can also include rate of change of ambient temperature, supply voltage stability, amount of vibration and others.

D.3.2 Timestamp Granularity Error

Timestamp Granularity Error (TSGE) is the error in timestamping each incoming and outgoing message due to the maximum timestamp resolution of which an implementation is capable. It is typically directly related to an implementation's clock rate. For example, a clock rate of 125 MHz typically results in a maximum resolution of 8 ns while a clock rate of 500 MHz typically results in a maximum resolution of 2 ns.

It is assumed that TSGE varies between -4 ns and +4 ns with an average of 0 ns. Lower levels of TSGE are better. Implementations where TSGE is higher will find some of the normative requirements difficult or impossible to meet.

D.3.3 Dynamic Timestamp Error

Dynamic Timestamp Error (DTSE) is the, effectively random, error in timestamping each incoming and outgoing event message due to an implementation's inherent inaccuracies, excluding TSGE. It is assumed to vary between a minimum of -6 ns and a maximum of +6 ns with an average of 0 ns. Lower levesl of DTSE are better.

If an implementation timestamps an incoming or outgoing message at a point other than the PHY, any variability in delay between that point and the PHY (PHY delay) will translate to DTSE. Some common implementations were not designed to limit this variability. If care is not taken to avoid implementations with high variability, the assumed DTSE range is easily exceeded. Such implementations will find some of the normative requirements difficult or impossible to meet.

D.3.4 Grandmaster PTP Instance Error Generation Requirements

Table 12 sets normative requirements for error generation at a Grandmaster PTP Instance that ensure the relevant fields in the Sync and Sync_Followup messages it transmits are sufficiently accurate to deliver the network-level performance. Table D.3 describes how the normative requirements align with major sources of error.

Table D.3: Protocol configurations & other measures to achieve dTE budget

	Normative Requirement	**Main Sources of Error**
1	preciseOriginTimestamp + correctionField vs Direct measurement of Working Clock at Grandmaster (acting as a Clock Source)	Timestamp Error relative to Clock Source plus accuracy measuring any internal delay between generation of the preciseOriginTimestamp and Sync message transmission.
2	rateRatio vs Direct measurement of Rate Ratio of Clock Source vs Local Clock	Accuracy of internal mechanism to measure Rate Ratio of Clock Source vs. Local Clock, potentially including algorithms that track RateRatioDrift and modify Rate Ratio accordingly[1]
3	syncEgressTimestamp vs Direct measurement of Local Clock	Timestamp Error relative to Local Clock

1 – Only applicable if Clock Source and Local Clock are not locked to the same frequency by the implementation. If they are locked then rateRatio will be 0 ppm and rateRatioDrift will be 0 ppm/s.

Limits on error generation due to Clock Drift are defined via normative requirements in Table 9.

D.3.5 PTP Relay Instance Error Generation Requirements

Table 13 sets normative requirements for error generation at a PTP Relay Instance that ensure the relevant fields in the Sync and Sync_Followup messages it transmits as part of Sync processing are sufficiently accurate to deliver the network-level time sync performance. The requirements includes the ability to mitigate errors in rateRatio and rateRatio drift that would otherwise occur due to clock drift at the current PTP Relay Instance, an adjacent PTP Relay Instance, or the Grandmaster PTP Instance. Table D.4 describes how the normative requirements align with major sources of error.

Table D.4: Protocol configurations & other measures to achieve dTE budget

	Normative Requirement	**Clock Drifts**	**Main Sources of Error**
1	preciseOriginTimestamp + correctionField vs Direct measurement of Clock Source at Grandmaster PTP Instance	None	Timestamp Errors relative to Local Clock when measuring Residence Time, i.e. Sync message ingress and egress. Accuracy of meanLinkDelay measurement. Errors in Rate Ratio used when translating Residence Time measured in terms of Local Clock to Residence time in terms of Clock Source, although these are typicaly orders of magnitude smaller than those from Timestamp Errors.
2	rateRatio vs Direct measurement of Rate Ratio of Clock Source vs Local Clock	None	Timestamp Error affecting measurement of NRR when there is no NRR Drift. The effect should be low. This normative requirement is a baseline for the next two requirements.
3		Clock Source (RR Drift)	Accuracy of measurement of NRR when there is no NRR Drift (as above). Accuracy of calculation of rateRatio, including algorithms for RR Drift tracking & error compensation.
4		Clock Source and Local Clock at previous PTP Instance (RR Drift & NRR drift)	Accuracy of measurement of NRR when there is NRR Drift, including algorithms for NRR Drift tracking & error compensation Accuracy of calculation of rateRatio, including algorithms for RR Drift tracking & error compensation. Combined with test 3 this effectively requires a level of performance

			regarding NRR Drift tracking & error compensation, whether the source of the NRR drift is the Local Clock of the current PTP Instance or the previous PTP Instance.
5	rateRatioDrift vs Direct measurement of Rate Ratio Drift of Clock Source vs Local Clock	None	Timestamp Error affecting measurement of NRR Drift when there is no NRR Drift. The effect should be low. This normative requirement is a baseline for the next two requirements.
6		Clock Source (RR Drift)	Accuracy of measurement of NRR Drift when there is no NRR Drift (as above). Accuracy of calculation of rateRatioDrift, including algorithms for RR Drift tracking & error compensation.
7		Clock Source and Local Clock at previous PTP Instance (RR Drift & NRR drift)	Accuracy of measurement of NRR Drift when there is NRR Drift, including algorithms for NRR Drift tracking & error compensation. Accuracy of calculation of rateRatioDrift, including algorithms for RR Drift tracking & error compensation. Combined with test 6 this effectively requires a level of performance regarding NRR Drift tracking & error compensation, whether the source of the NRR drift is the Local Clock of the current PTP Instance or the previous PTP Instance.
8	syncEgressTimestamp vs Direct measurement of Local Clock	None	Timestamp Error relative to Local Clock

Limits on error generation due to Clock Drift are defined via normative requirements in Table 9.

D.3.6 PTP End Instance Error Generation Requirements

Table 14 sets normative requirements for error generation at a PTP End Instance that ensure the ClockTarget it generates from incoming Sync and Sync_Followup messages is sufficiently accurate to deliver the network-level time sync performance. Table D.5 describes how the normative requirements align with major sources of error.

Table D.5: Protocol configurations & other measures to achieve dTE budget

	Normative Requirement	Clock Drifts	Main Sources of Error
1	ClockTarget vs ClockSource	None	Timestamp Error affecting measurement of NRR Drift when there is no NRR Drift. The effect should be low. This normative requirement is a baseline for the next two tests.
2		Clock Source (RR Drift)	Accuracy of measurement of NRR Drift when there is no NRR Drift (as above). Accuracy of calculation of rateRatioDrift, including algorithms for RR Drift tracking & error compensation.
3		Clock Source and Local Clock at previous PTP Instance (RR Drift & NRR drift)	Accuracy of measurement of NRR Drift when there is NRR Drift, including algorithms for NRR Drift tracking & error compensation. Accuracy of calculation of rateRatioDrift, including algorithms for RR Drift tracking & error compensation. Combined with test 2 this effectively requires a level of performance regarding NRR Drift tracking & error compensation, whether the source of the NRR drift is the Local Clock of the current PTP Instance or the previous PTP Instance.

Limits on error generation due to Clock Drift are defined via normative requirements in Table 9.

D.4 Approach to Testing Normative Requirements

D.4.1 General

This document does not specify tests to ensure conformance with the normative requirements. However, it is important that the normative requirements are, in principle, testable. This clause describes, at a high level, approaches a test specification might take to testing conformance with some of the normative requirements related to time synchronisation.

It is assumed that test equipment can precisely measure the output of the ClockSource (at a Grandmaster PTP Instance), ClockTarget (at a PTP End Instance) and Local Clock (at any PTP Instance) to ensure conformance with frequency offset and frequency offset drift requirements. This might be via a Pulse per Second (PPS) plus Time of Day information or another mechanism.

It is also assumed that test equipment can generate sequences of PTP messages with precise timing and content (for testing PTP Relay Instances and PTP End Instances) and receive, log, and process sequences of PTP messages with precise timing measurement, e.g. of message arrival.

D.4.2 Testing Grandmaster PTP Instance

Figure D.1 illustrates an approach to testing the three normative requirements discussed in D.3.4.

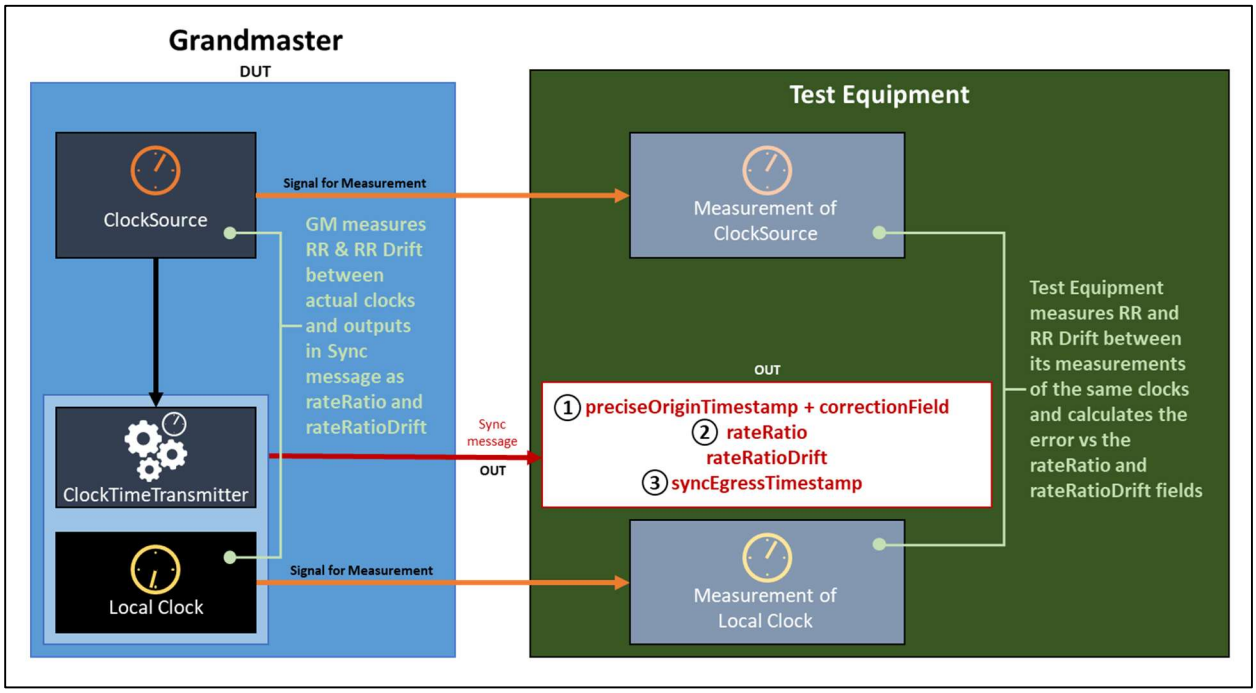

Figure D.1: Approach to Testing Normative Requirements for Grandmaster PTP Instance

The test equipment can calculate the time the Sync message is output at the DUT by subtracting the link delay from the measured arrival time at the test equipment.

For test 1, the test equipment compares the value of the preciseOriginTimestamp + correctionField against its measurement of the ClockSource.

For tests 2, the test equipment compares the value in the rateRatio field with its calculation of the equivalent value based on its measurement of the ClockSource.

For test 3, the test equipment compares the value of the syncEgressTimestamp against its measurement of the Local Clock.

D.4.3 Testing PTP Relay Instance

Figure D.~~8~~ 2 illustrates an approach to testing normative requirements 1, 2, 5 and 8 discussed in D.3.5.

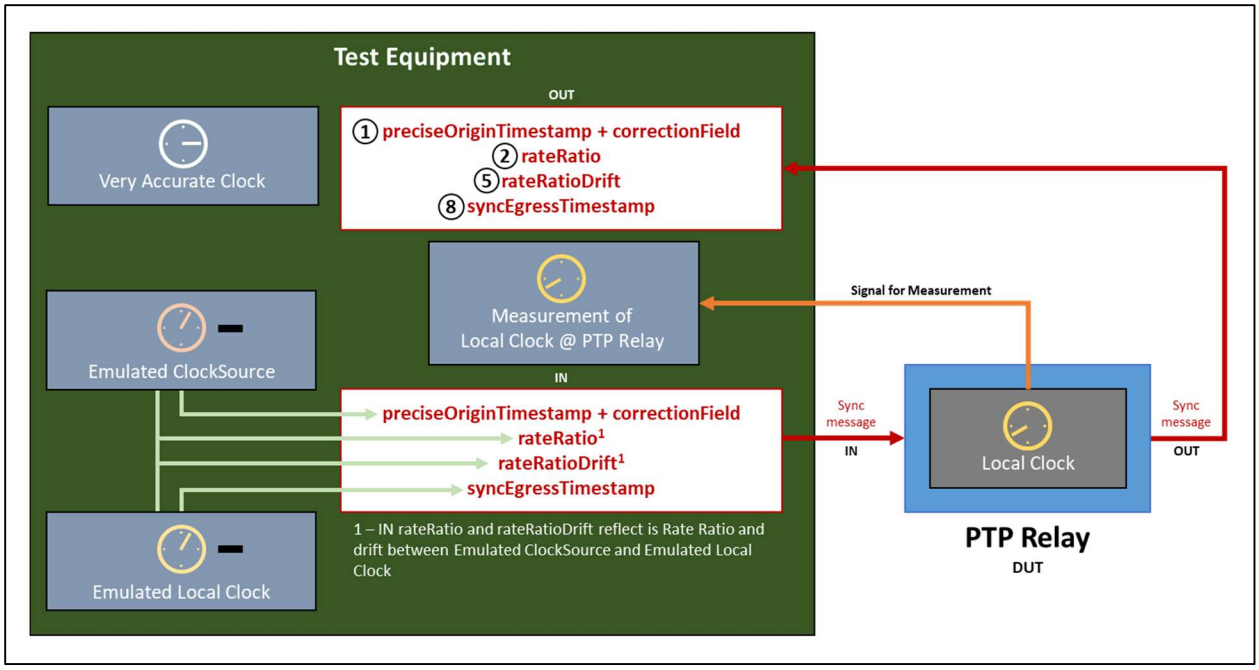

Figure D.2: Approach to Testing Normative Requirements for PTP Relay Instance - 1

The test equipment can compare the DUT's output Sync message to the expected result given the measurement of the Local Clock and the timing of the input Sync message transmission and output Sync message reception.

For these four tests, the Emulated ClockSource and Emulated Local Clock are stable and in sync. In practice, both can be equal to the test equipment's Very Accurate Clock. In the input Sync message, rateRatio will be 0 ppm and rateRatioDrift will be 0 ppm/s. If the Local Clock of the PTP Relay Instance is also stable, it will measure NRR of 0 ppm and NRR Drift of 0 ppm/s.

The test equipment can calculate the time the output Sync message is output at the DUT by subtracting the link delay from the measured arrival time at the test equipment.

For test 1, the test equipment can compare the increase in the value of the correctionField to the measured meanLinkDelay (from the test equipment to the DUT) plus residenceTime. The test equipment will need to account for the additional delay between the PTP Relay Instance's transmission of the input Sync message and its reception by the test equipment.

For tests 2 and 5, the test equipment can compare the rateRatio and rateRatioDrift fields in the output Sync message with the equivalent calculated values between the measured Local Clock and the Emulated ClockSource.

For test 8, the test equipment can compare syncEgressTimestamp value in the ouput Sync message with its measurement of the Local Clock.

Figure D.~~8~~ 3 illustrates an approach to testing normative requirements 3 and 6 discussed in D.3.5.

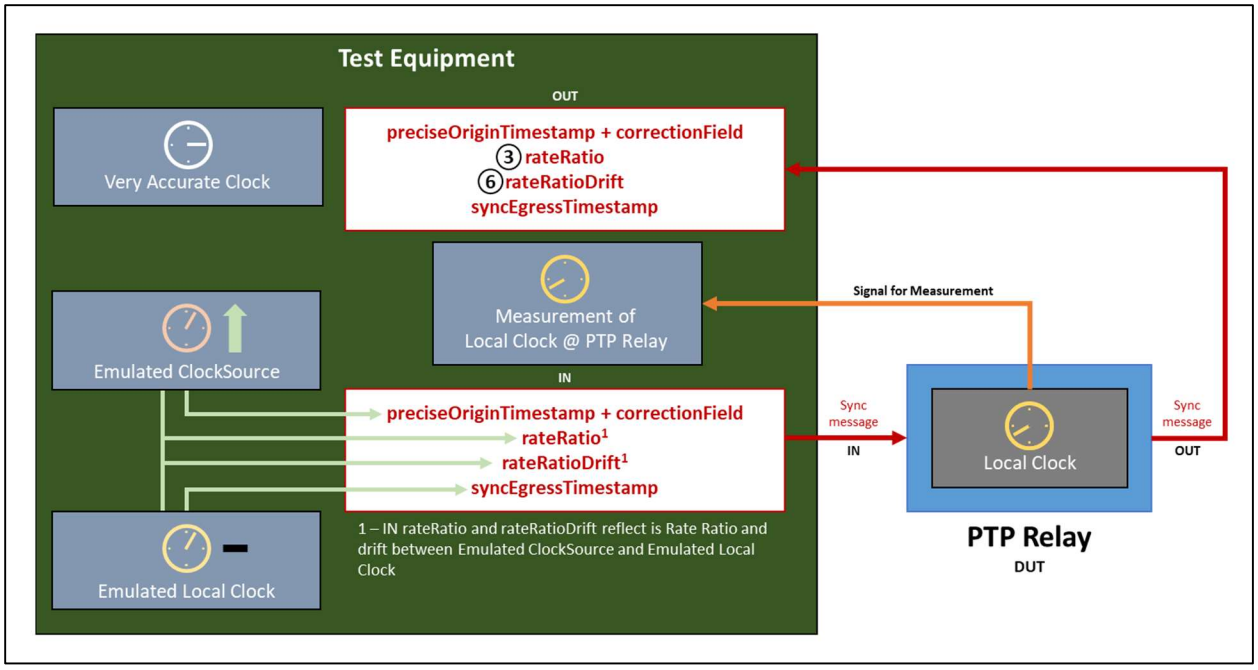

Figure D.3: Approach to Testing Normative Requirements for PTP Relay Instance - 2

For these two tests, the fractional frequency offset of the Emulated ClockSource is increasing at a defined ppm/s rate relative to the Very Accurate Clock. The Emulated Local Clock is stable; in practice, it can be equal to the test equipment's Very Accurate Clock. In the output Sync message, the rateRatio field will increase over time, and the rateRatioDrift field will maintain a matching positive value. If the Local Clock of the PTP Relay Instance is also stable, it will measure NRR of 0 ppm and NRR Drift of 0 ppm/s.

For tests 3 and 6, the test equipment can compare the rateRatio and rateRatioDrift fields in the output Sync message with the equivalent calculated values between the measured Local Clock and the Emulated ClockSource.

Figure D.~~8~~4 illustrates an approach to testing normative requirements 4 and 7 discussed in D.3.5.

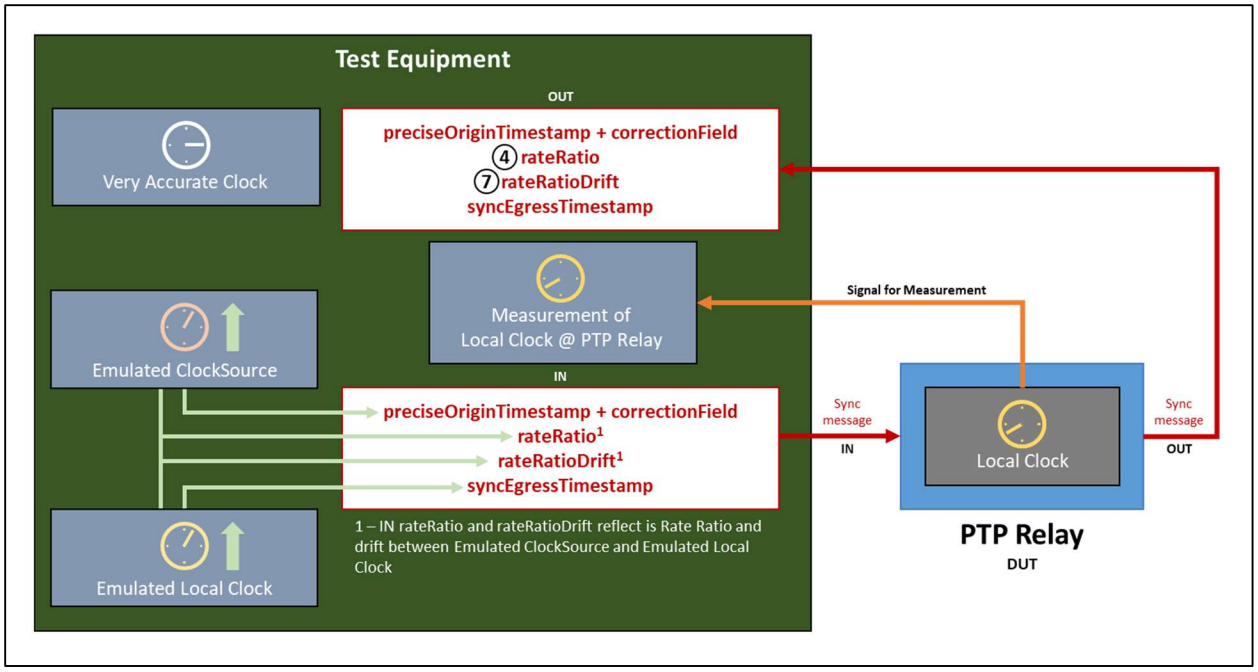

Figure D.4: Approach to Testing Normative Requirements for PTP Relay Instance - 3

For these two tests, the fractional frequency offsets of the Emulated ClockSource and the Emulated Local Clock are equal and increasing at a defined ppm/s rate relative to the Very Accurate Clock. In the output Sync message, the rateRatio field will be 0 ppm, and the rateRatioDrift field will be 0 ppm/s. If the Local Clock of the PTP Relay Instance is stable, the NRR it measures will increase over time and the NRR Drift it measures will maintain a matching positive value.

For tests 4 and 7, the test equipment can compare the rateRatio and rateRatioDrift fields in the output Sync message with the equivalent calculated values between the measured Local Clock and the Emulated ClockSource.

D.4.4 Testing PTP End Instance

Figure D.~~8~~ 5 illustrates an approach to testing the three normative requirements discussed in D.3.6.

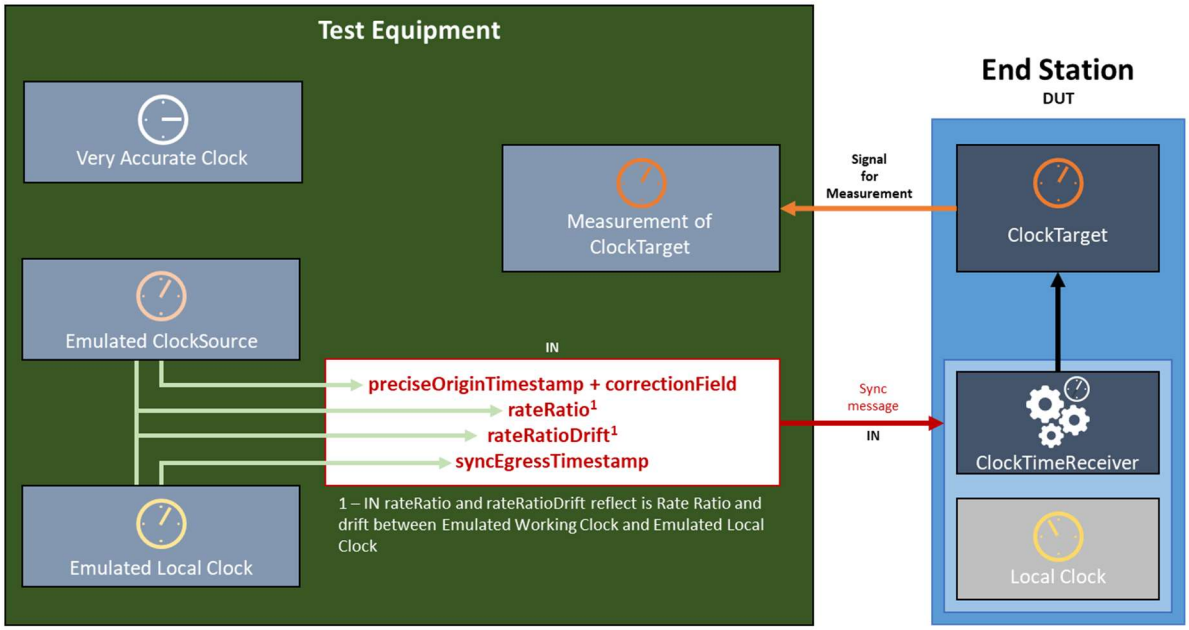

Figure D.5: Approach to Testing Normative Requirements for PTP End Instance

The test equipment can compare its measurement of the DUT's ClockTarget to the Emulated ClockSource. It will need to account for the additional delay between its transmission of the input Sync message and the reception of the message by the DUT.

For test 1, the Emulated ClockSource and Emulated Local Clock are stable and in sync. In practice, both can be equal to the test equipment's Very Accurate Clock. In the input Sync message, rateRatio will be 0 ppm and rateRatioDrift will be 0 ppm/s. If the Local Clock of the PTP End Instance is also stable, it will measure NRR of 0 ppm and NRR Drift of 0 ppm/s.

For test 2, the fractional frequency offset of the Emulated ClockSource is increasing at a defined ppm/s rate relative to the Very Accurate Clock. The Emulated Local Clock is stable; in practice, it can be equal to the test equipment's Very Accurate Clock. In the output Sync message, the rateRatio field will increase over time, and the rateRatioDrift field will maintain a matching positive value. If the Local Clock of the PTP Relay Instance is also stable, it will measure NRR of 0 ppm and NRR Drift of 0 ppm/s.

For test 3, the fractional frequency offsets of the Emulated ClockSource and the Emulated Local Clock are equal and increasing at a defined ppm/s rate relative to the Very Accurate Clock. In the output Sync message, the rateRatio field will be 0 ppm, and the rateRatioDrift field will be 0 ppm/s. If the Local Clock of the PTP Relay Instance is stable, the NRR it measures will increase over time and the NRR Drift it measures will maintain a matching positive value.

D.5 Example Algorithms

D.5.1 General

This document does not place normative requirements on the use of specific algorithms. However, the normative requirements assume the use of algorithms to reduce the effect of errors in meanLinkDelay

and to track clock drift and compensate for consequent errors. PTP instances that do not implement algorithms will find it difficult or impossible to meet the normative requirements.

This clause provides examples of algorithms that can be used for…

- Tracking NRR drift.
- Correcting for errors in measured NRR (mNRR) due to NRR drift.
- Calculating RR drift.
- Correcting for errors in measured RR (mRR) due to RR drift.
- Reducing the effect of errrors in meanLinkDelay

For measured NRR, measured RR, and meanLinkDelay, an option for how startup behaviour can be handled is provided.

D.5.2 Algorithm for Tracking NRR Drift

NRR Drift Tracking and Error Correction is carried out for each network hop, i.e. at every node other than the Grandmaster. It is based on pairs of timestamps with each pair associated with a Sync message transmitted from the previous node (n-1) to the current node (n)

- t_{s1outP} – Timestamp of the Sync message egress from the **previous** node (n-1), timestamped by that node's Local Clock. Unit: **ns**.
- t_{s2in} – Timestamp of the Sync message ingress to the current node (n), timestamped by that node's Local Clock. Unit: **ns**.

All timestamps are affected by Timestamp Errors.

The algorithm uses information from the 32 most recent Sync messages. However, a node need only keep track of the 9 most recent pairs of timestamps from the most recent (x) to the 9th most recent (x-8) Sync message. The algorithm generates one measurement of NRR using the prior 2 s of Sync message data (on average, based on a nominal Sync Interval of 125 ms), and a second measure based on the 2 s of Sync message data prior to that. It then uses the difference in the two measurements over the interval between the effective measurement points to calculate the NRR drift rate.

On arrival of a new ~~timestamp pair~~Sync message (x), a node executes a NRR calculation…

$$NRRcalc(x) = \left(\frac{t_{s1out}(x) - t_{s1outP}(x-8)}{t_{s2in}(x) - t_{s2i}(x-8)} - 1\right) \times 10^6 \quad \textbf{ppm}$$

…with an associated effective measurement point…

$$NRRcalcT(x) = \frac{t_{s2in}(x) + t_{s2in}(x-8)}{2} \quad \textbf{ns}$$

A node keeps track of the 24 most recent NRR calculations and effective measurement points, from the most recent (x) to the 24th most recent (x-23).

After of a new most-recent NNR calculation, a node calculates an NRR drift rate…

$$NRRaverageA = \sum_{i=x-7}^{x} \frac{mNRRcalc(i)}{8} \quad \textbf{ppm}$$

$$NRRaverageB = \sum_{i=x-2}^{x-16} \frac{mNRRcalc(i)}{8} \quad \textbf{ppm}$$

$$NRRdriftInterval = \sum_{i=x-7}^{x} \frac{mNRRcalcT(i)}{8} - \sum_{i=x-23}^{x-16} \frac{mNRRcalcT(i)}{8} \quad \textbf{ns}$$

$$NRRdriftRate(n) = \left(\frac{NRRaverageA - NRRaverageB}{NRRdriftInterval}\right) \times 10^9 \quad \textbf{ppm/s}$$

…where *NRRdriftRate(n)* is the NRR drift rate for the current Node n.

D.5.3 Algorithm to Compensate for Errors in measured NRR due to Clock Drift

The algorithm to measure NRR uses data from the previous 1 s of Sync message data, combined with the NRR drift estimate from the previous step. This smaller amount of data (vs. that used for either of the NRR measurements in the previous step) is employed as it improves responsiveness to sudden changes in NRR drift with minimal loss of accuracy.

On arrival of a ~~new timestamp~~Sync message ~~pair~~ (x), a node executes a NRR calculation…

$$mNRRcalc(x) = \left(\frac{t_{s1out}(x) - t_{s1outP}(x-4)}{t_{s2in}(x) - t_{s2in}(x-4)} - 1\right) \times 10^6 \quad \textbf{ppm}$$

…with an associated effective measurement point…

$$mNRRcalcT(x) = \frac{t_{s2in}(x) + t_{s2in}(x-4)}{2} \quad \textbf{ns}$$

A node keeps track of the 4 most recent mNRR calculations and effective measurement points, from the most recent (x) to the 4th most recent (x-3). (The mNRR calculations use information from the 5 most recent Sync messages, but the node is already keeping track of information from the 9 most recent Sync messages for the NRR drift tracking algorithm.)

The node then calculates an error-corrected measured NRR value.

$For\ i = x\ to\ (x-3)$

$$mNRRcorrected(i) = mNRRcalc(i) + \left(\text{~~m~~}NRRdriftRate(n) \times \frac{(t_{s2in}(x) - mNRRcalcT(i))}{10^9}\right) \quad \textbf{ppm}$$

$$mNRR(n) = \sum_{i=x-3}^{x} \frac{mNRRcorrected(i)}{4} \quad \textbf{ppm}$$

The result is a measured NRR value, error-corrected to the time when the most recent Sync message was received.

D.5.3.1 Measured NRR Algorithm – Startup Behaviour

NRR is used when calculating meanLinkDelay and output Sync message fields. The first NRR drift calculation will only be available after receipt of 32 Sync messages, i.e. after approximately 4 seconds of operation given the 125 ms Sync Interval. During this time meanLinkDelay and output Sync messages fields must still be calculated, so an alternative must be used, even if it can not deliver the same assurances regarding network-level performance.

If measured NRR from Sync message information is unavailable but equivalent information from Pdelay_Resp messages is available, it may be substituted for Sync message information. However, measuring NRR using Pdelay_Resp messages is vulnerable to additional error due to clock drift between the time NRR is measured, on receipt of the latest Pdelay_Resp message, and use of the measurement during Sync message processing. This is the reason using Sync message information is preferable. It also means that a switch to using Sync message information as soon as possible is desirable. It is technically possible to calculate a NRR using a combination of Pdelay_Resp and Sync messages but this can be risky due to the potential for very short intervals between messages and resulting high error due to timestamp errors, so it not recommended.

~~The following describes potential startup behaviour when using either Sync or Pdelay_Resp message information.~~ It is the responsibility of implementers to decide whether and when to use Pdelay_Resp message information and when to switch to using Sync message information~~. It is, however, a normative requirement that implementations use Sync message information when information from 32 or more timely Sync messages is available.~~The normative requirements in this document are for operation after 32 Sync messages have been received and assume use of the algorithms in D.5.2 and D.5.3, or more effective algorithms. Implementations that continue to use Pdelay_Resp message information to calculate NRR after 32 messages have been received can find some of the normative requirements difficult or impossible to meet.

The following describes potential startup behaviour applicable to either Sync or Pdelay_Resp message information.

At least two two messages must be received before calculating a NRR value.

Prior to two messages being received, NRR = 1 (i.e. 0 ppm) should be used.

Once two messages have been received, NRR should be calculated using the formula…

$$2^{nd}\ \text{message: } mNRR = \left(\left(\frac{t_3(x) - t_3(x-1)}{t_4(x) - t_4(x-1)}\right) - 1\right) \times 10^6 \quad \textbf{ppm}$$

Where…

- t_3 – Timestamp of the Pdelay_Resp message egress from the previous node (n-1), timestamped by that node's Local Clock. Unit: **ns**.
- t_4 – Timestamp of the Pdelay_Resp message ingress to the current node (n), timestamped by that node's Local Clock. Unit: **ns**.

When three to four messages have been received, NRR should be calculated using the following formulae…

3rd message: $mNRR = \left(\left(\frac{t_{1out}\ (x)-t_{1outP}(x-2)}{t_{2i}\ (x)-t_{2i}\ (x-2)}\right)-1\right)\times 10^6$ **ppm**

4th message: $mNRR = \left(\left(\frac{t_{1outP}(x)-t_{1out}\ (x-3)}{t_{2in}(x)-t_{2in}(x-3)}\right)-1\right)\times 10^6$ **ppm**

On arrival of the 5th Sync message the first mNRRcalc and mNRRcalcT calculations can take place and should be used for NRR…

$mNRRcalc(x) = \left(\frac{t_{s1outP}(x)-t_{s1outP}(x-4)}{t_{s2in}(x)-t_{s2in}(x-4)}-1\right)\times 10^6$ **ppm**

$mNRRcalcT(x) = \frac{t_{s2in}(x)+t_{s2in}(x-4)}{2}$ **ns**

5th Sync message: $mNRR = mNRRcalc(x)$ **ppm**

As the 6th, 7th and 8th messages arrive an average can be taken and used for NRR, so…

6th message: $mNRR = \sum_{i=x-1}^{x}\frac{mNRRcalc(i)}{2}$ **ppm**

7th message: $mNRR = \sum_{i=x-2}^{x}\frac{mNRRcalc(i)}{3}$ **ppm**

8th message: $mNRR = \sum_{i=x-3}^{x}\frac{mNRRcalc(i)}{4}$ **ppm**

For the 9th to the 31st message, the same equation as for the 8th message can be used.

Once the 32nd message arrives, the regular equations with NRR drift tracking and error correction can be used.

D.5.4 Algorithm for Tracking RR Drift

A Sync or Sync_Followup message carries the rateRatio field, which informs each node of the previous node's estimate of its (the previous node's) Rate Ratio. This document also requires support for the Drift_Tracking TLV that carries the rateRatioDrift field, which informs each node of the previous node's estimate of its (the previous node's) Rate Ratio Drift.

If the implementation of the Grandmaster PTP Instance means the ClockSource and Local Clock (at the Grandmaster PTP Instance) are linked such that the two are always operating at the same frequency, the rateRatio field received by the first node (Node 1) will always be 0 ppm and the rateRatioDrift field will always be 0 ppm/s. Thus, at Node 1, RR will equal NRR, RR Drift will equal NRR Drift, and therefore D. 5.2 and D.5.3 describe how to calculate RR and RR Drift at Node 1.

If the implementation of the Grandmaster PTP Instance means the ClockSource and Local Clock (at the Grandmaster PTP Instance) can operate at different frequencies, the implementation populates the rateRatio and rateRatioDrift field with values reflecting those differences.

In either case all PTP Instances, other than the Grandmaster PTP Instance, calculate an estimate of the local Rate Ratio Drift when the latest Sync Message is received, based on the received rateRatioDrift field and the local measure of NRR Drift. The Rate Ratio Drift Rate from the previous node is in ppm/s relative to the timebase of its Local Clock (i.e. the "s" in "ppm/s"). For highest precision, this ~~cam~~ can be converted to the timebase of the current node's Local Clock.

$$rateRatioDrift(n) = \frac{rateRatioDrift(n-1)}{\left(1+\frac{mNRR\epsilon(n)}{10^6}\right)} + NRRdriftRate(n) \qquad \textbf{ppm/s}$$

However, given that adding ppm/s already lacks the precision of multiplying actual ratios, this simplification delivers similarly accurate results.

$$rateRatioDrift(n) = rateRatioDrift(n-1) + NRRdriftRate(n) \qquad \textbf{ppm/s}$$

D.5.5 Algorithm to Compensate for Errors in measured RR due to Clock Drift

On receipt of a Sync or Sync_Folloup message, all PTP Relay Instances estimate a measured RR (mRR(n)) based on the received rateRatio field (mRR(n-1)) and the local measure of NRR (mNRR(n)). An mRR(n) value is used to translate the sum of meanLinkDelay and residenceTime from Local Clock timebase into Grandmaster timebase. An mRR(n) value is also passed in the transmitted Sync or Sync_Followup message's rateRatio field to the next node. Errors in these estimates due to clock drift can be reduced by taking account of RR Drift. Since the optimal point in time for each estimate is different, the amount of applicable RR Drift is different, and hence the estimates will be different.

(For discussion of how different Grandmaster PTP Implementations affect the behaviour of a PTP Relay Instance at Node 1 – or not – see D.5.4.)

A PTP End Instance is similar in that it estimates mRR(n) on receipt of a Sync message, subsequently uses an mRR(n) value, and errors in the latter due to clock drift can be mitigated by taking account of RR Drift. However, unlike a PTP Relay Instance, the mRR(n) value is used to keep the ClockTarget in line with the ClockSource and there is no need to transmit a rateRatio field to a subsequent node.

For a PTP Relay Instance there are three points in time of interest:

- Point A: Receipt of the Sync Message by the current node (Node n)
- Point B: Mid-point between transmission of the Sync message by the previous node (Node n-1) and transmission of the consequent Sync message by the current node (Node n)
- Point C: Transmission of the Sync Message by the current node (Node n)

Figure D.~~8~~6 illustrates these points and the associated calculations.

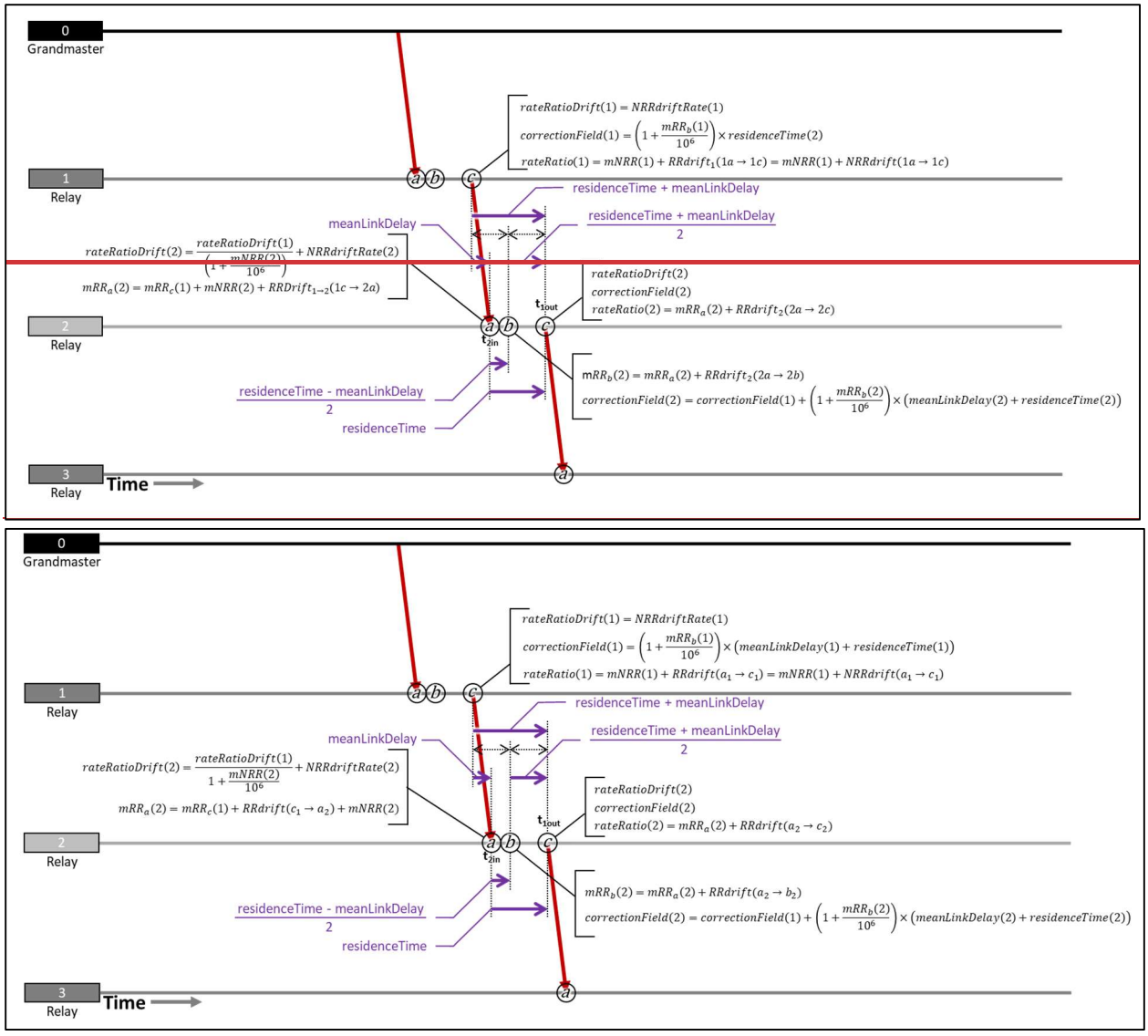

Figure D.6: RR Drift Tracking and Error Compensation Calculations – PTP Relay Instance

The estimate of RR when the Sync message arrives can be calculated as follows.

$$mRR_a(n) = rateRatio(n-1) + RRdrift\sout{RRdrift}_{n-1 \rightarrow n}((\sout{n-1})_c c_{n-1} \rightarrow \sout{n_a} a_n) \sout{+} + mNRR(n) \quad \textbf{ppm}$$

$$= rateRatio(n-1) + \left(rateRatioDrift(n-1) \times \left(1 + \frac{mNRR(n)}{10^6}\right) \times meanLinkDelay(n)\right) \sout{\left(\frac{rateRatioDrift(n-1)}{\left(1 + \frac{mNRR_c(n)}{10^6}\right)} \times meanLinkDelay(n)\right)} + mNRR(n) \quad \textbf{ppm}$$

Where $RRdrift\sout{RRdrift}_{n-1/n}((n-1)_c \rightarrow n_a)$ is amount $rateRatio(n-1)$ drifts between transmission of the Sync message at Node n-1 and reception at Node n. This is $rateRatioDrift(n-1)$ multipled by meanLinkDelay but, since meanLinkDelay is measured in terms of Node n's Local Clock and

rateRatioDrift is in terms of Node n-1's Local Clock the ~~latter~~ former should be ~~divided~~ multiplied by the NRR at Node n for the highest accuracy.

However, given that adding ppm/s already lacks the precision of multiplying actual ratios, this simplification delivers similarly accurate results.

$$mRR_a(n) = rateRatio(n-1) + \left(rateRatioDrift(n-1) \times meanLinkDelay(n)\right) + mNRR\text{~~e~~}(\text{~~2~~}n) \quad \textbf{ppm}$$

Once the time when Node n transmits the consequent Sync message is known, the correctionField value can be calculated.

$$mRR_b(n) = mRR_a(n) + RRdrift_n(a \rightarrow b) \quad \textbf{ppm}$$

$$= mRR_a(n) + \left(rateRatioDrift(n) \times \frac{residenceTime(n) - meanLinkDela\ \ (n)}{2}\right) \quad \textbf{ppm}$$

$$correctionField(n) = correctionField(n-1) + \left(1 + \frac{mRR_b(n)}{10^6}\right) \times \left(meanLinkDelay(2) + residenceTime(2)\right) \quad \textbf{ns}$$

And the rateRatio field.

$$rateRatio(n) = mRR_a(n) + RRdrift_n(a \rightarrow c) \quad \textbf{ppm}$$

$$= mRR_a(n) + \left(RRdriftRate(n) \times residenceTime(n)\right) \quad \textbf{ppm}$$

D.5.6 Algorithm to Compensate for Errors in measured RR due to Clock Drift at PTP End Instance

Figure D.~~8~~7 illustrates a possible approach to applying similar RR drift tracking and error compensation at a PTP End Instance.

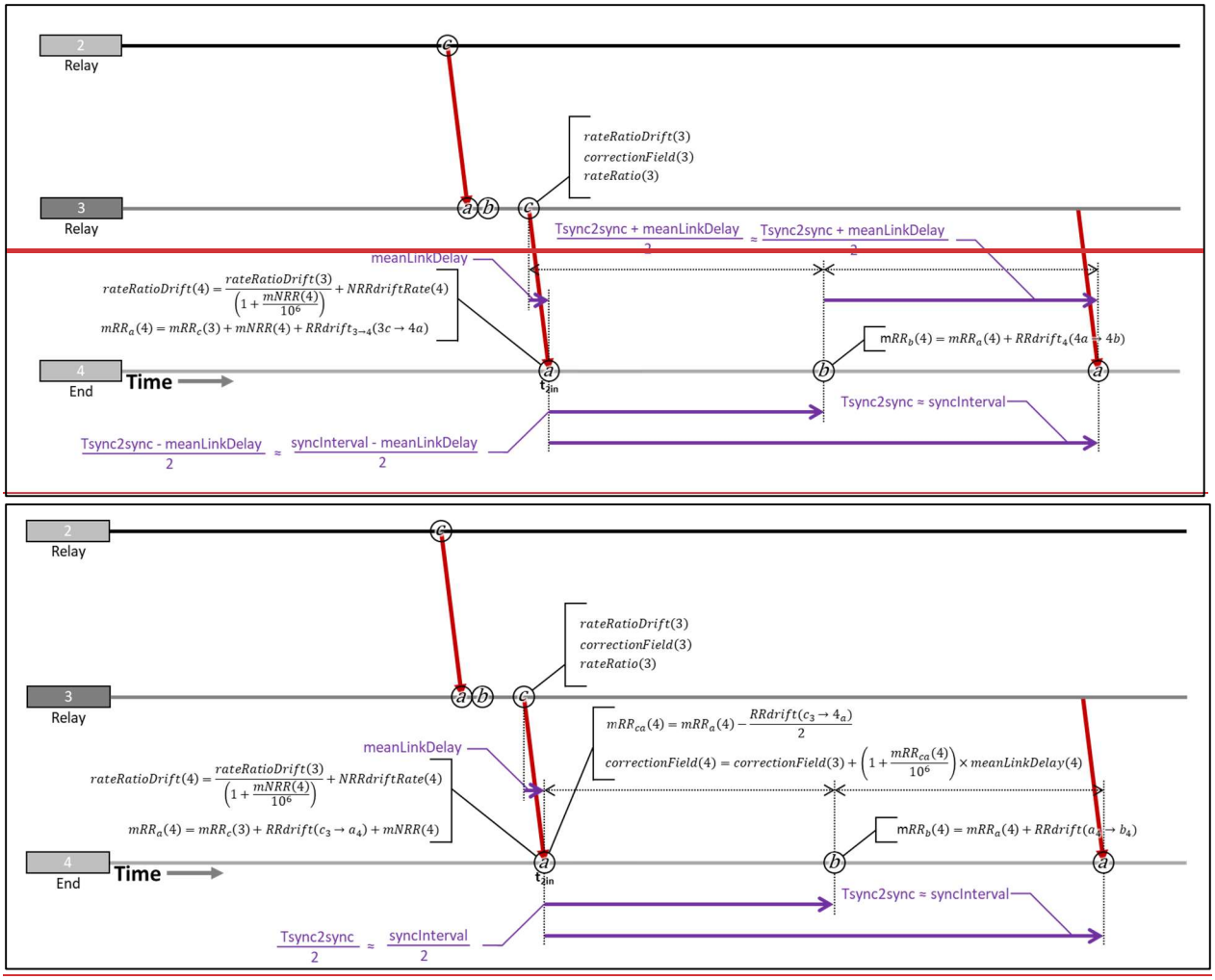

Figure D.7: RR Drift Tracking and Error Compensation Calculations – PTP End Instance

The initial calculations for rateRatioDrift and mRR_a are exactly the same as for a PTP Relay Instance. Instead of using RR to translate meanLinkDelay + residenceTime from the Local Clock timebase to the Grandmaster timebase – as is done at a PTP Relay Instance – a PTP End Instance uses mRR to translate meanLinkDelay to the Grandmaster timebase (there is no residenceTime at an End Instance), adding the result to the incoming correctionField to obtain an estimate of the ClockSource at the time the Sync message arrives, then uses mRR to keep its ClockTarget in line with the ClockSource until arrival of the next Sync message. The optimal mRR value for translating meanLinkDelay is half way between meanLinkDelay's transmission (at c_{n-1}, i.e. point C at the previous node) and reception (at a_n, i.e. point A at the current node); in the equations below, this value is referred to as mRR_{ca}.

$$mRR_{ca}(n) = mRR_a(n) - \frac{RRdrift(c_{n-1} \rightarrow a_n)}{2} \quad \text{ppm}$$

$$= mRR_a(n) - \left(rateRatioDrift(n) \times \frac{meanLinkDelay(n)}{2}\right) \quad \text{ppm}$$

$$correctionField(n) = correctionField(n-1) + mRR_{ca}(n) \times meanlinkDelay(n) \quad \text{ppm}$$

The optimal ~~If only one~~ value of mRR ~~is used for this purpose, the optimal value is not mRR_a; it~~for keeping the ClockTarget in line with the Clock Source is mRR_b, where Point B is half way between the most recently received Sync message and the next Sync message.

Of course, the exact interval until the next Sync message's arrival (Tsync2sync in Figure 106) can't be known before it happens, but the Rate Ratio value is required as soon as possible after arrival of the most recent Sync message. The solution is to use the nominal value of the interval, i.e. syncInterval, which is 125 ms.

$$mRR_b(n) = mRR_a(n) + RRdrift_n(a \rightarrow b) \quad \textbf{ppm}$$

$$= mRR_a(n) + \left(rateRatioDrift(n) \times \frac{syncInterval}{2}\right) \quad \textbf{ppm}$$

$$= mRR_a(n) + (rateRatioDrift(n) \times 0.0625) \quad \textbf{ppm}$$

It is also possible to use more complex algorithms that repeatedly or continuously adjust the mRR value between Sync messages, but such an approach is outside the scope of this document.

D.5.7 Mean Link Delay Averaging

The actual Path Delay from one node to the next – for a wired connection – is very stable and errors measuring it due to Timestamp Error average to zero. Thus, taking a long average or applying a low-pass filter with a low bandwidth is an effective way to reduce error in meanLinkDelay. Care needs to be taken during system startup or after any other initialisation of the algorithm, to quickly converge on a stable value.

The basic Pdelay calculation, used by the Common Mean Link Delay service, remains the same. Figure D.8 illustrates it.

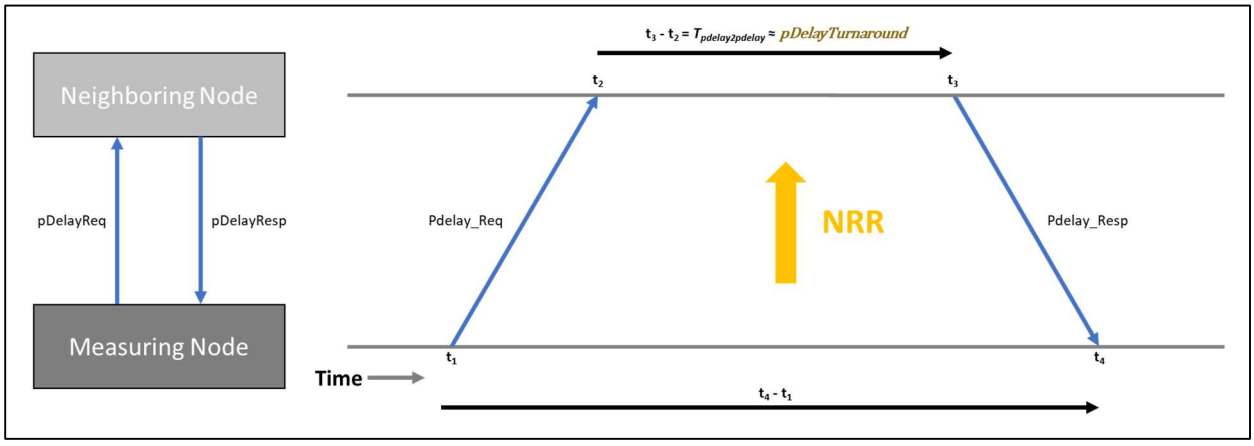

Figure D.8: Signals and timestamps to measure path delay

Following each Pdelay_Req – Pdelay_Resp exchange, the measured path delay (mPathDelay) is calculated.

For the x^{th} message after initialisation…

$$mPathDelay(x) = \frac{(t_4 - t_1) - \frac{(t_3 - t_2)}{NRR}}{2} \quad \textbf{ns}$$

The meanLinkDelay is then updated via an IIR (Infinite Impulse Response) filter. For the first couple of minutes after initialization the filter is in initialization mode.

If $x < 1000$ then $f = x$ else $f = 1000$

$$meanLinkDelay(x) = \frac{\big(meanLinkDelay(x-1)\times(f-1)\big)+pDelay(x)}{f}$$ **ns**

For example...

$$meanLinkDelay(100) = \frac{(meanLinkDelay(99)\times\cancel{(99)}99)+pDelay(x)}{100}$$ **ns**

$$meanLinkDelay(5836) = \frac{(meanLinkDelay(5835)\times\cancel{(999)}999)+pDelay(x)}{1000}$$ **ns**

It is possible to automatically reinitialise the algorithm if an $mPathDelay$ value, or series of values, deviates too much from the $meanLinkDelay$, but the details are outside the scope of this document.