The 60802 challenge of meeting time accuracy goals across long daisy-chains using 802.1AS™-2020

An analysis and a proposed path forward

David McCall, Kevin Stanton, Greg Schlechter (Intel)
Jordan Woods, Tom Weingartner (Analog Devices)

September 2021 Interim of 802.1 TSN / 60802

IEEE 802.1 TSN / 60802 Interim, September 2021 Available at http://www.ieee802.org/1/files/public/docs2021 McCall, Stanton with Woods, Weingartner, and Schlechter

Abstract

Industrial Automation Systems require microsecond-accurate time across long daisy-chains of devices using IEEE Std. 802.1AS™-2020 as specified by IEEE/IEC 60802.

Simulated protocol and system parameters have thus far either been judged impractical or have failed to meet the time-accuracy requirement.

In this contribution we:

1. **Summarize the assumptions, requirements and results to date**
2. **Challenge certain assumptions**
3. **Propose an approach that we believe could lead to a solution**

IEEE 802.1 TSN / 60802 Interim, September 2021 Available at http://www.ieee802.org/1/files/public/docs2021 McCall, Stanton with Woods, Weingartner, and Schlechter

Some Requirements & Assumptions to Date

- **Requirements:**
  - Topology: 64-100 network hops (i.e. 2 endstations + 63-99 devices in a chain)
  - Time Accuracy: Not to exceed +/-1us with respect to the GM at any node
    - Measured at each Working Clock, e.g. used by 802.1Qbv (i.e. GCL) and the Application Clock
- **Assumptions:**
  - →Local Clock reference frequency (in each system) is an XO, i.e. a quartz crystal oscillator
  - →The 802.1AS GM does not have better PPM stability than the other nodes
  - → No averaging of the pDelay measurement is performed
  - Temperature gradients induce PPM gradients, resulting in time-duration-measurement errors
  - Various Ambient Temperature profiles ($T_{Ambient}$) have been used
    - With various transfer functions of PPM = f($T_{Crystal}$)]
  - → Zero thermal resistance between $T_{Ambient}$ and $T_{Crystal}$ ($0^{o}K/W$) ←

Note: Assumptions that we recommend revisiting are marked with "→"

IEEE 802.1 TSN / 60802 Interim, September 2021 Available at http://www.ieee802.org/1/files/public/docs2021 McCall, Stanton with Woods, Weingartner, and Schlechter

Other Principles

- **Initial 60802 specs should not trigger new silicon design / spins**
  - Should be met by most commodity switch and endpoint silicon today that were designed to support IEEE Std. 802.1AS™-2020
- **The solution should not unnecessarily limit the deployment of IEEE Std. 1588™-2019 security mechanisms**
  - With today's 802.1AS-2020-capable silicon
- **Once viable 802.1AS-2020 parameters are chosen, the worst-case PPM/sec allowed per time-aware system should be derived**
- **The function of PPM vs $T_{ambient}$ is a property of the electrical, mechanical and thermal design of the board / chassis / system**
  - Thus the PPM / $T_{Ambient}$ requirement should be specified, but not the method by which the PPM / $T_{Ambient}$ requirement must be met by the system design

Viable Parameter Selections

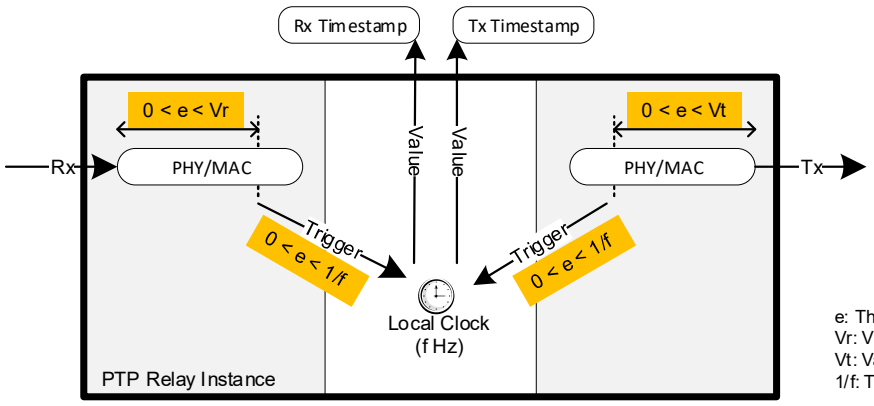
- **Sync Interval**
- **Sync-to-Follow-up delay**
- **pDelay Interval**
- **Timestamp Granularity**
- **Rx/Tx Timestamp Asymmetry:**
- **Residence Time = Sync-to-Followup-Delay + 10ms**

➔ **A simple analytical Model is needed to more quickly choose scenarios for more detailed time-series simulation**

Quickly iterate a Model over minutes, then invest the >1 week for the detailed simulation

IEEE 802.1 TSN / 60802 Interim, September 2021 Available at http://www.ieee802.org/1/files/public/docs2021 McCall, Stanton with Woods, Weingartner, and Schlechter

Variable Time Error (vTE): Timestamp Model



e: The time error
Vr: Variable Rx delay
Vt: Variable Tx delay
1/f: Timestamp granularity

Motivation & Results (so far)

- Develop a method of iterating faster
  - Input: key 802.1AS parameters
  - Output: time sync accuracy over 64 and 100hops
- Approach by modelling the errors and how they accumulate
  - Requires an analysis of the errors first
- Analysis of the errors has already suggested some assumptions inherent in the current time series simulation that should be revisited
- This presentation walks through the error analysis and the most important insights it reveals
  - Goal is to introduce the overall approach share the insights
  - There are some areas that require further analysis; will present in a later session
- Next steps:
  - Build the model
  - Complete the additional analysis

Before proceeding…

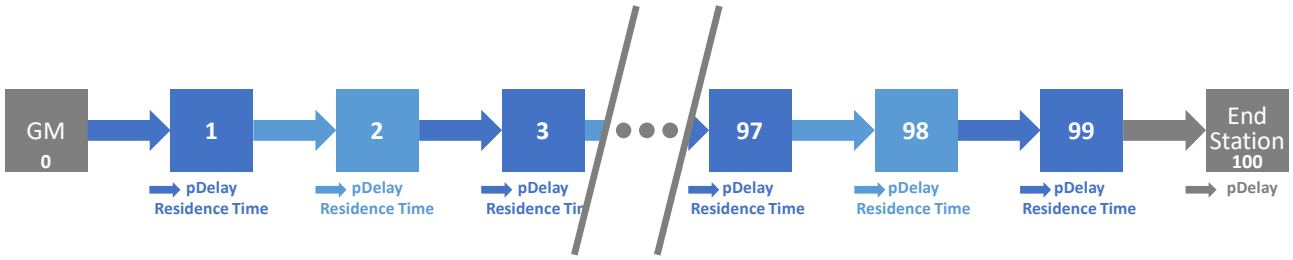
There is a lot of information in this presentation. It's presented in sections, each of which builds upon the previous sections. It's likely that if, during the section on how Timestamp Granularity and Dynamic Timestamp Error impacts pDelay Error you are thinking "What about Neighbor Rate Ratio?"…that your question will be answered during the later section on pDelay and Neighbor Rate Ratio. So can I ask that you limit your questions during the presentation to questions about the clarity of what is on the screen. I'd further ask that, if possible, you type your question into the chat window and, if he can, Kevin will answer it there…or interrupt the presentation in order to ask the question or make an important clarification clear to all attendees.

Thank you in advance for your help getting through everything in the time available.

Summary of Main Insights

- pDelay should be averaged.
  - Should nearly eliminate $pDelay_{TSerror}$
- NRR should not be averaged.
  - Provided (Timestamp Errors / pDelay Interval) is negligible in terms of ppm
- There are significant benefits to the GM having greater ppm stability (lower ppm/s).
- It is hard (impossible?) to eliminate Timestamp granularity and Dynamic Timestamp error effects from Residence Time
  - Need more work to see how far these elements can realistically be reduced
- Synchronising pDelayReq to arrive just before Sync would minimise the effect of Clock Drift.
- There may be opportunities to compensate for NRR_{error}
  - And therefore RR_{error}

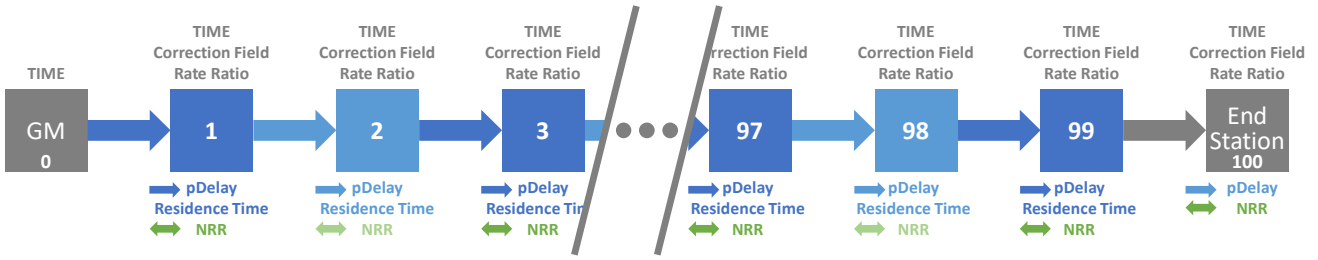
Time Sync



This is a simple view of how time sync works.

The basics are: measure path delays (pDelay) and residence times (residenceTime); add them to a Sync message as it is passed from the GM to the End Station.

Time Sync

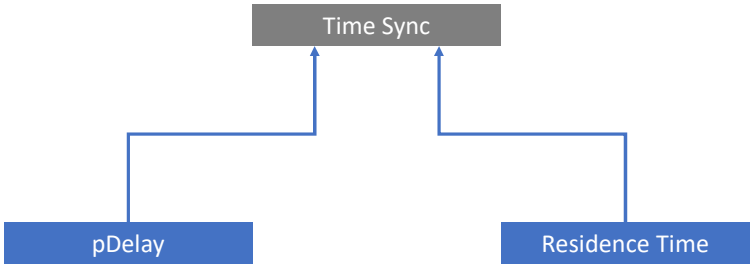


Adding a bit more detail, this is how that's done in practise.

The Sync message contains the TIME of transmission from the GM, a Correction Field of the delay that has accumulated since it was transmitted, and a measure of the Rate Ratio, i.e. the ratio between the current clock speed and the GM's clock speed.

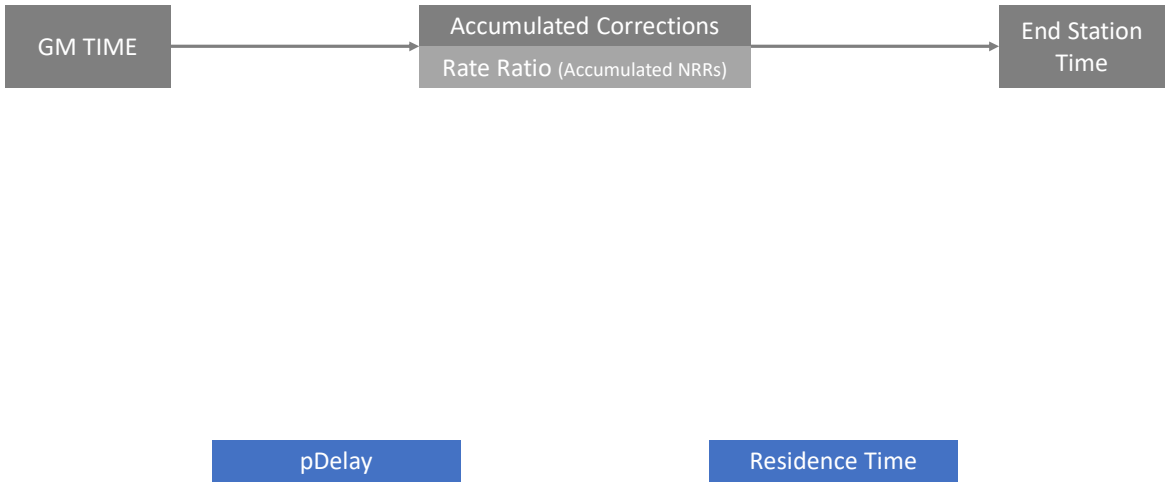
It's worth pointing out that each device is responsible for updating the Correction Field based on the **incoming** pDelay plus **it's own** Residence Time. Also a device measures the Neighbor Rate Ratio between itself and the upstream device. It then adds that to the incoming Rate Ratio. And finally uses the new Rate Radio to modify the measured pDelay and the Residence Time prior modifying the Correction Field, and passing on everything to the next device.

Time Sync – Elements & Relationships



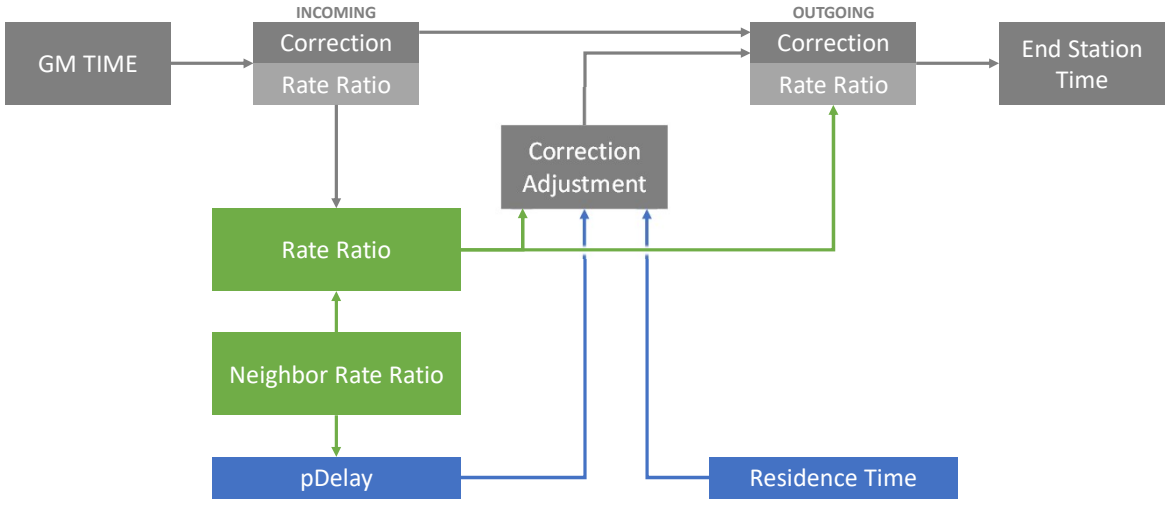
Building up the logical elements and connection of this process…the simple model looks like this…

Time Sync – Elements & Relationships



As mentioned, the time sync is actually accomplished by accumulating the delays (i.e. the corrections) and measuring the rate ratio between the GM and the End Station.

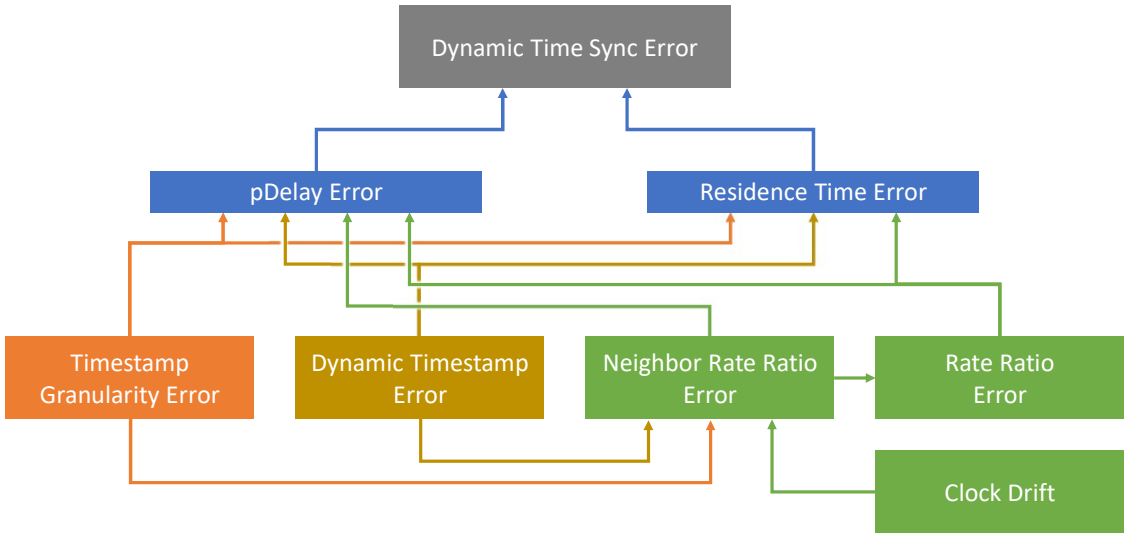
Time Sync – Elements & Relationships



At each bridge there is a process for taking the incoming Correction Field and Rate Ratio and adding effects of the local pDelay (between the local and upstream device) and Residence Time.

I'll be referring back to this diagram later, but when it comes to analysing the errors involved in this process, it's helpful to take a slightly different view…

Time Sync – Elements & Relationships



This is the start of the error analysis.

Timestamp Granularity, is related to the maximum resolution of Timestamps.

Dynamic Timestamp Error is in addition to Granularity and can be very implementation dependant.

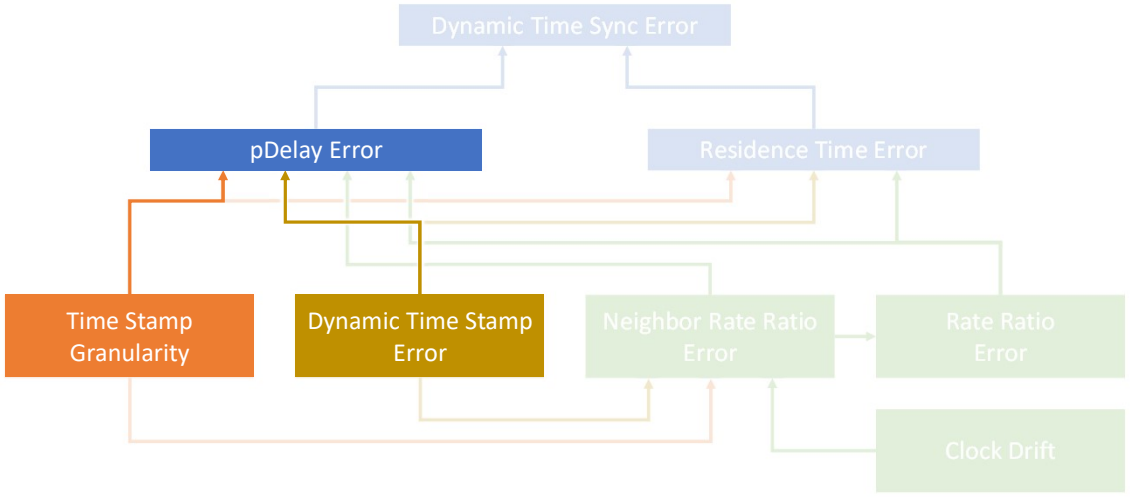
Clock Drift is just the way that each device has it's own local clock which is running a slightly different speed…and also changing that speed over time (mostly due to temperature changes; or at least those are the changes that have the greatest impact on our analysis).

The other errors are all rooted in those three.

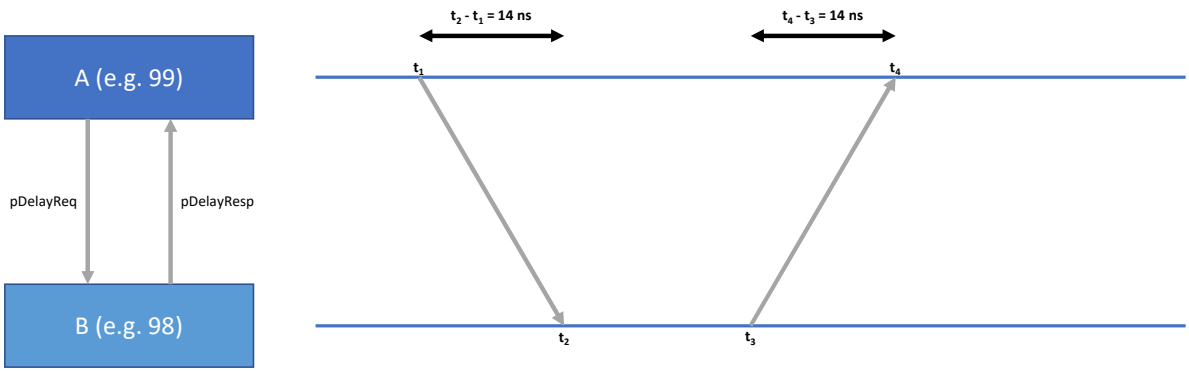
Not all of the relationships need to be modelled. Figuring out which can be ignored is part of the analysis.

pDelay – Timestamp Granularity & Dynamic Time Stamp Error

Time Sync – Elements & Relationships



To be clear, we're only looking at these three elements in this section. We aren't looking at any of the others.

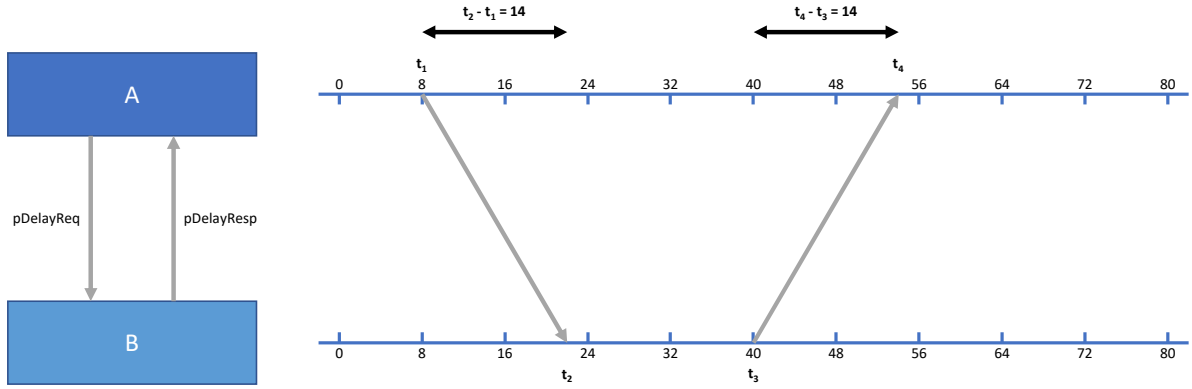


$$\text{pDelay} = \frac{(t_2 - t_1) + (t_4 - t_3)}{2}$$

This is the idealised view…the Platonic ideal of how pDelay is measured.

Note that "A" is device 99. It initiates a pDelayReq to the upstream device (98)…which responds with a pDelayResp…after the pDelay Response Time.

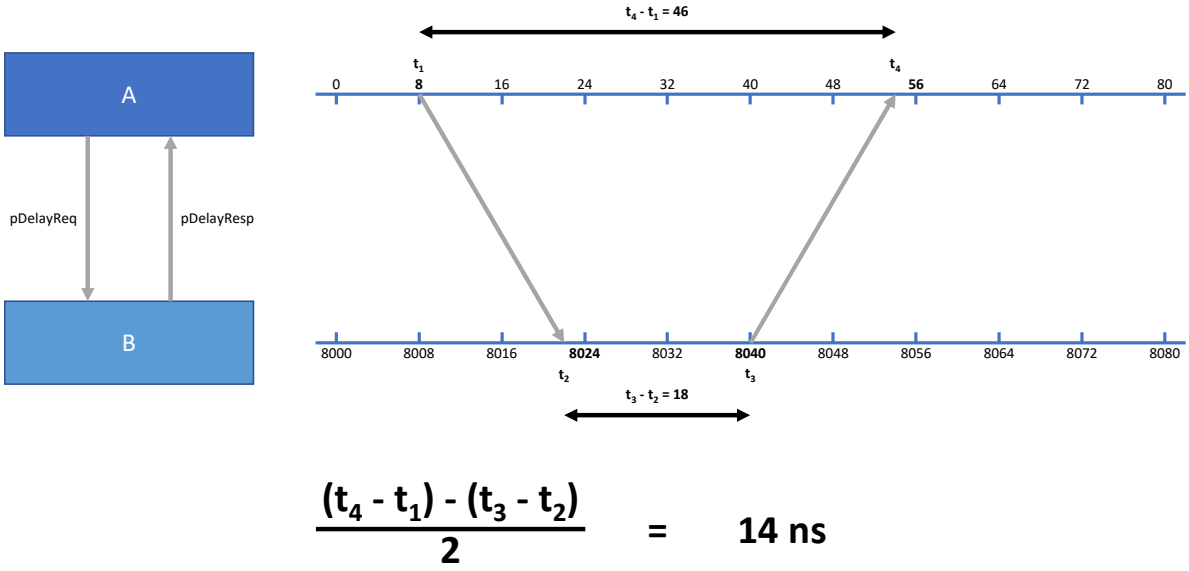
The default maximum for pDelay Response is 10ms, which is much longer than the pDelay which in this case is 14ns…indicating a cable length of 4.2m…but for the moment I'm going to model this shorter pDelay Response Time to make things legible and illustrate some principles…



$$\frac{(t_2 - t_1) + (t_4 - t_3)}{2} = 14 \text{ ns}$$

Let's put some numbers on the time line. We'll use a resolution of 8ns, which is what you'd get with a clock speed of 125MHz.

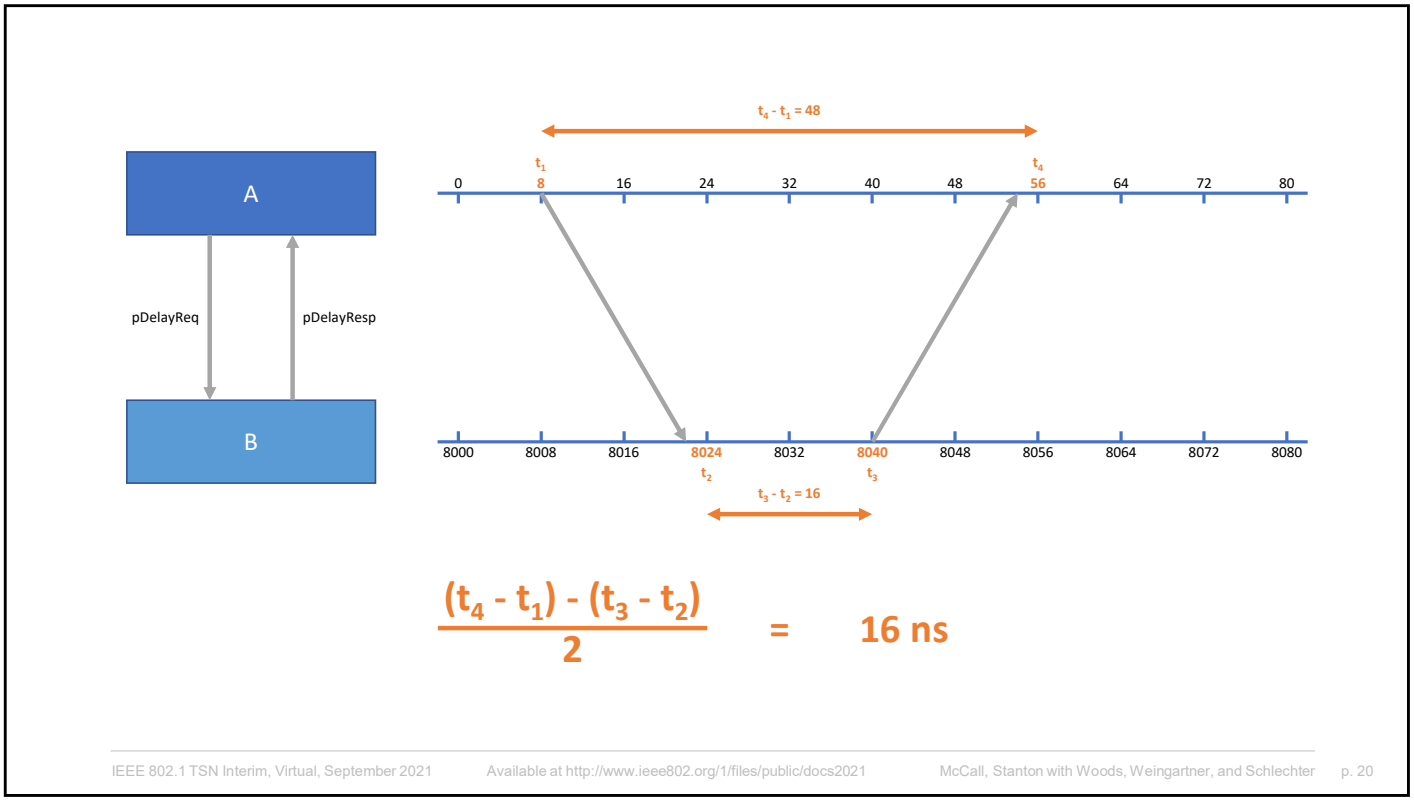
But…clocks A & B aren't the same…they aren't counting the same numbers at the same time.



$$\frac{(t_4 - t_1) - (t_3 - t_2)}{2} = 14 \text{ ns}$$

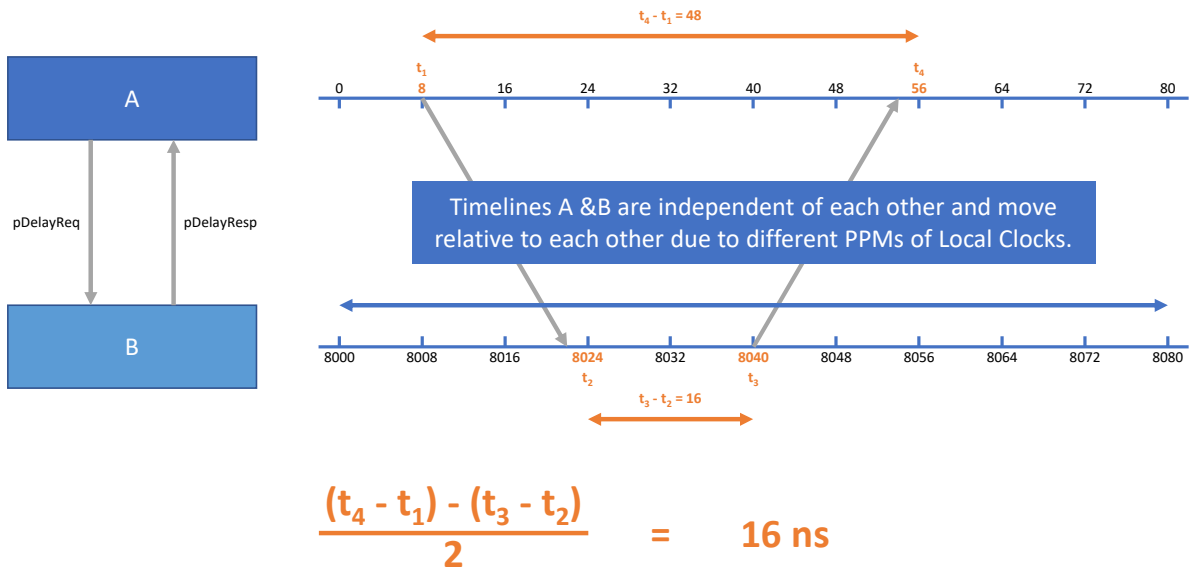
We'll model this by just adding a nice, round 8,000 to the lower line.

This requires changing the equation to subtract B's pDelay Response time (on the lower timeline) from the time between A sending the Request and receiving the Response. It's mathematically equivalent…and also allows compensation for differing clock speeds (via Neighbor Rate Ratio…which we are **not** considering in this section).

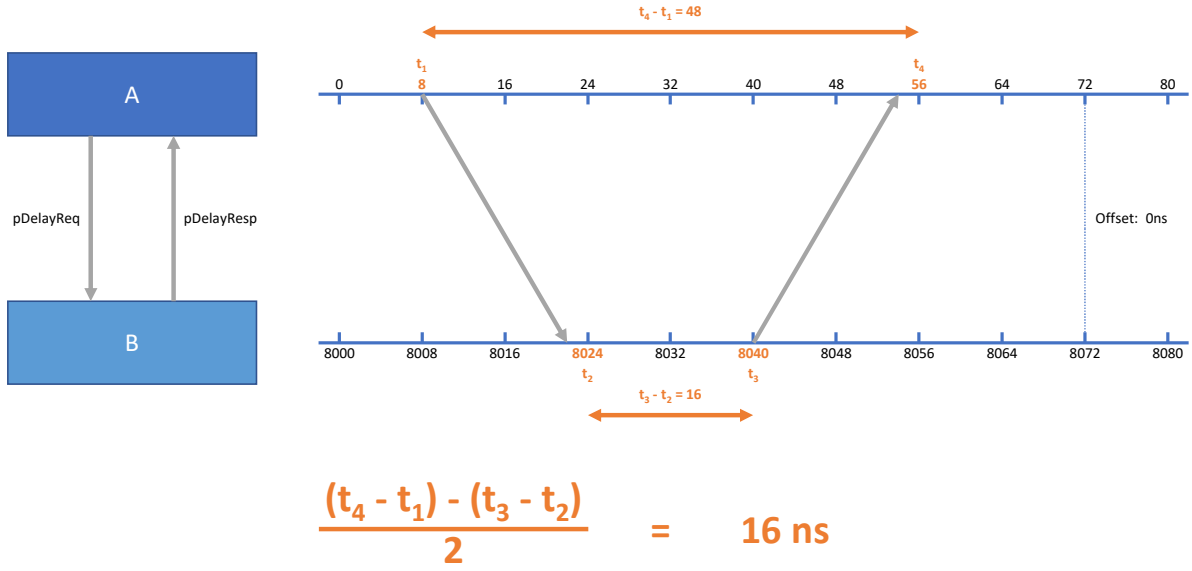


8ns is our Timestamp Granularity…which means the pDelay Req could arrive at any time between clock ticks and the Timestamp will actually reflect the next clock tick.

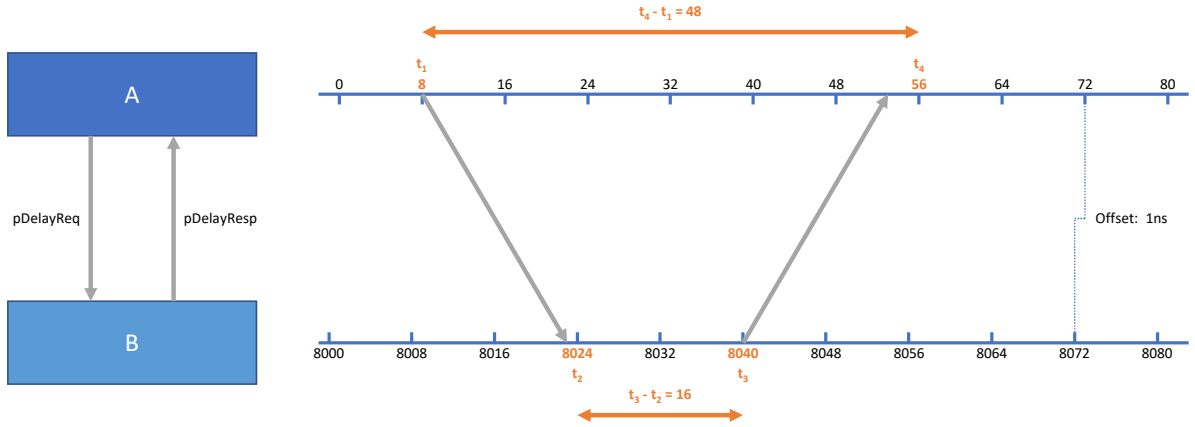
I'm keeping things simple here by assuming the TX time is exactly on a clock tick (and measured as such)….but you can see the effect this has. The measurement of pDelay is 16ns, not 14ns.



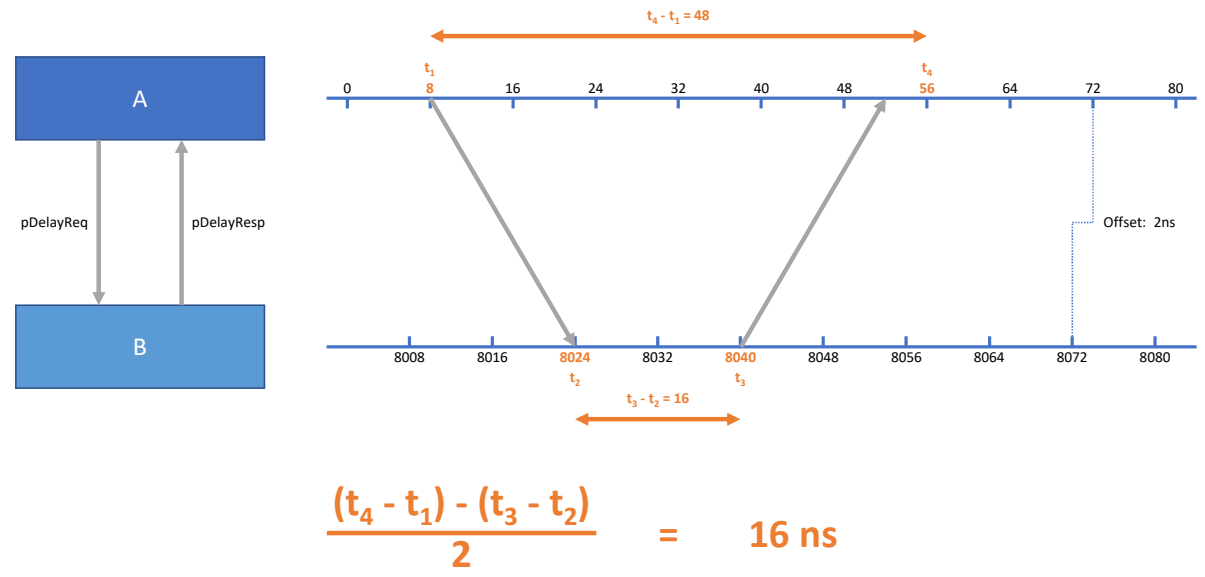
I'll deal with the movement relative to each in a moment. For now, let's just look at this measurement and the fact that the timelines are independent. This means the clock ticks don't necessarily line up exactly. They might…but it's more likely there will be an offset.



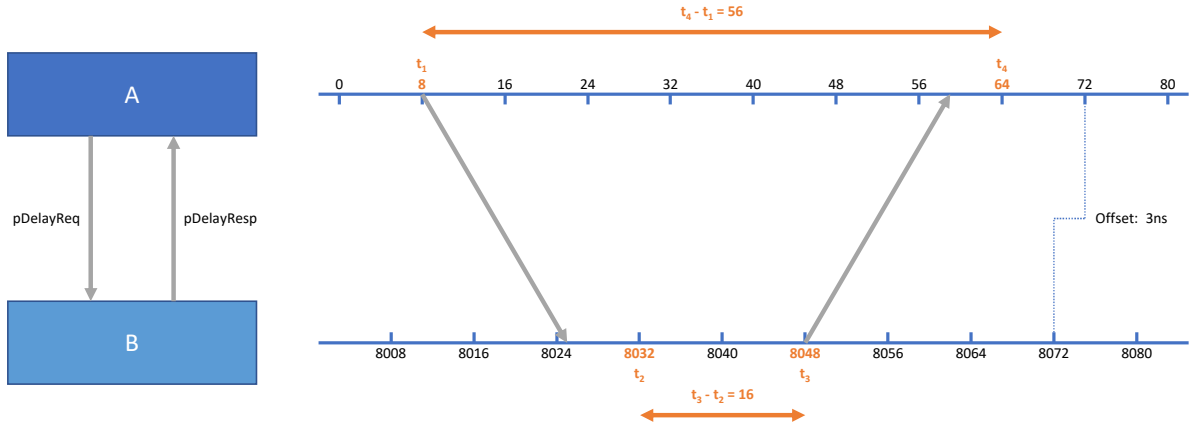
$$\frac{(t_4 - t_1) - (t_3 - t_2)}{2} = 16 \text{ ns}$$



$$\frac{(t_4 - t_1) - (t_3 - t_2)}{2} = 16 \text{ ns}$$

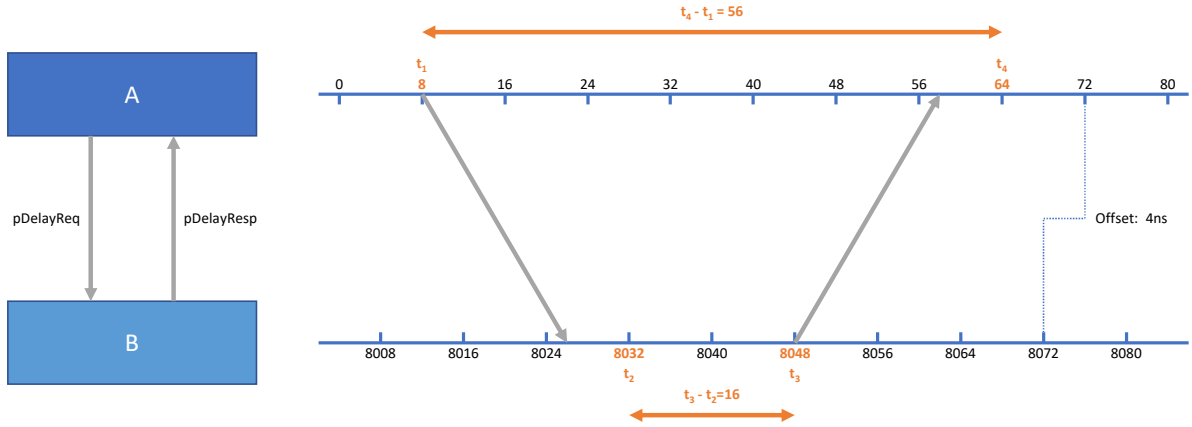


At an offset >2ns, the measurement is no longer 16ns…

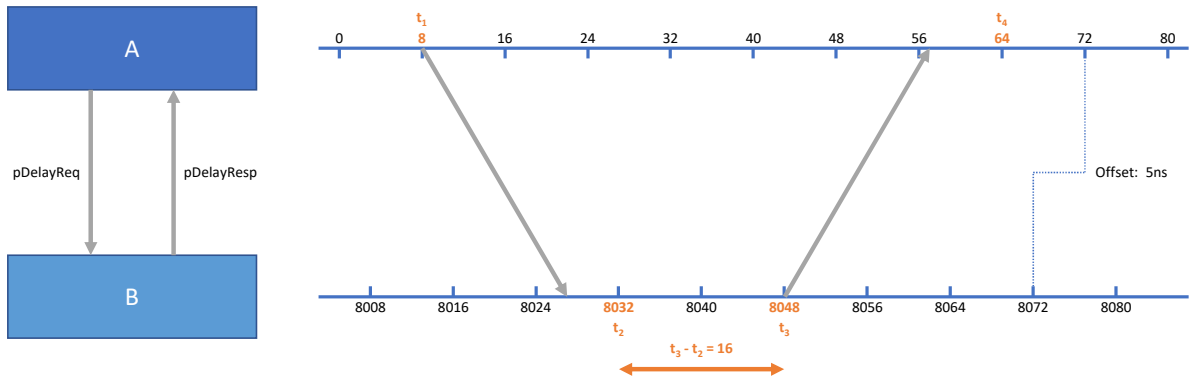


$$\frac{(t_4 - t_1) - (t_3 - t_2)}{2} = 20 \text{ ns}$$

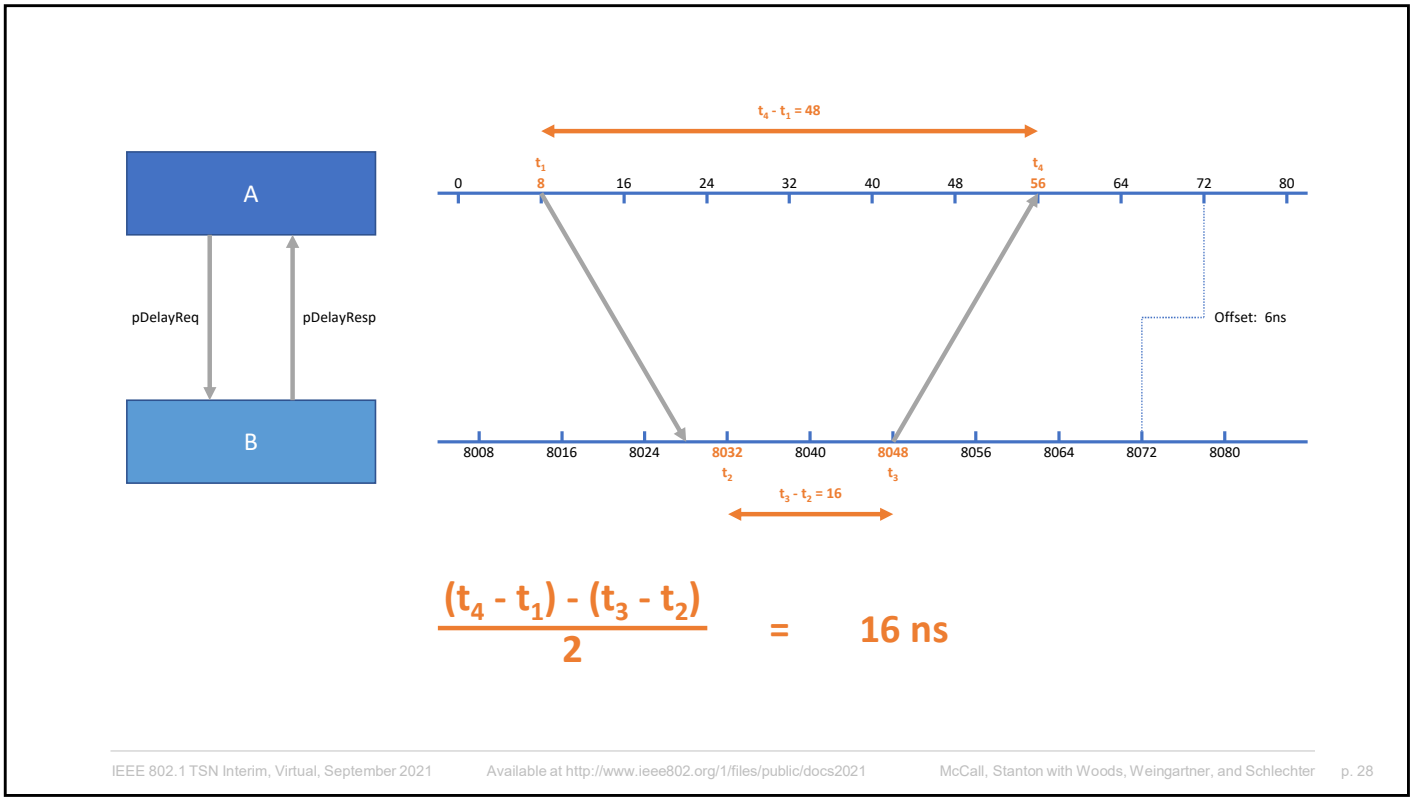
…it's worse. 20ns.



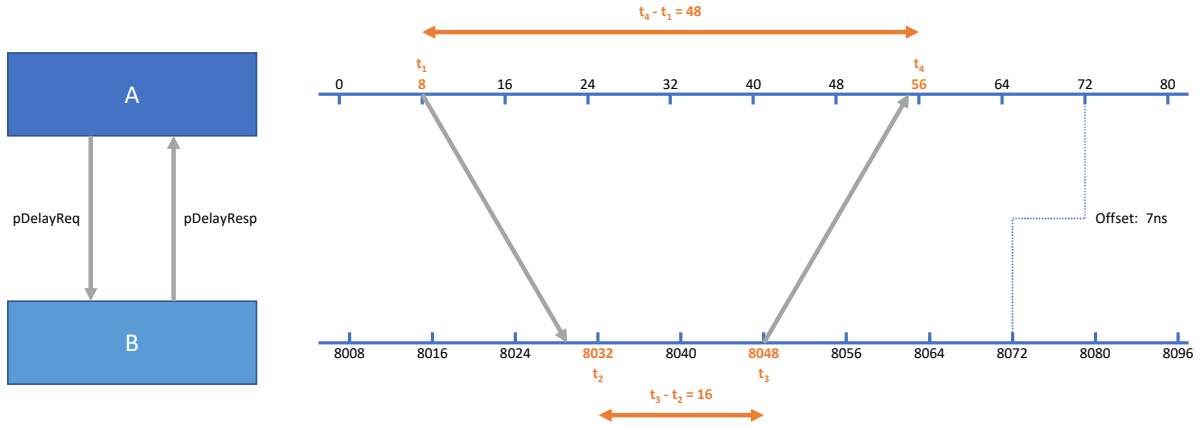
$$\frac{(t_4 - t_1) - (t_3 - t_2)}{2} = 20 \text{ ns}$$



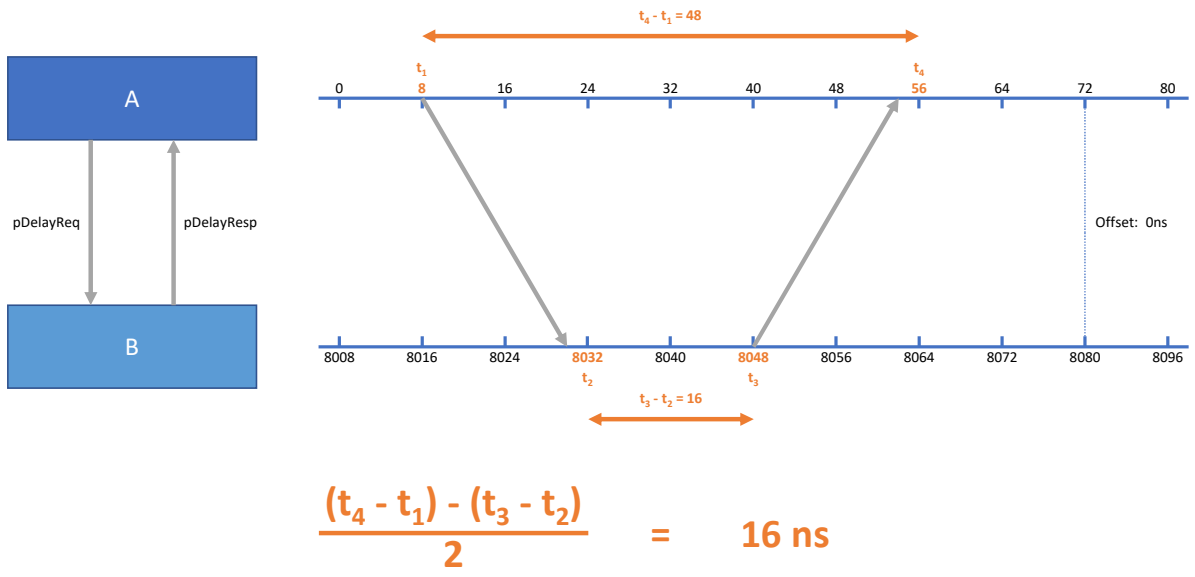
$$\frac{(t_4 - t_1) - (t_3 - t_2)}{2} = 20 \text{ ns}$$



At 6ns the measurement gets back to 16ns.



$$\frac{(t_4 - t_1) - (t_3 - t_2)}{2} = 16 \text{ ns}$$



$$\frac{(t_4 - t_1) - (t_3 - t_2)}{2} = 16 \text{ ns}$$

And then we're back to where we started.

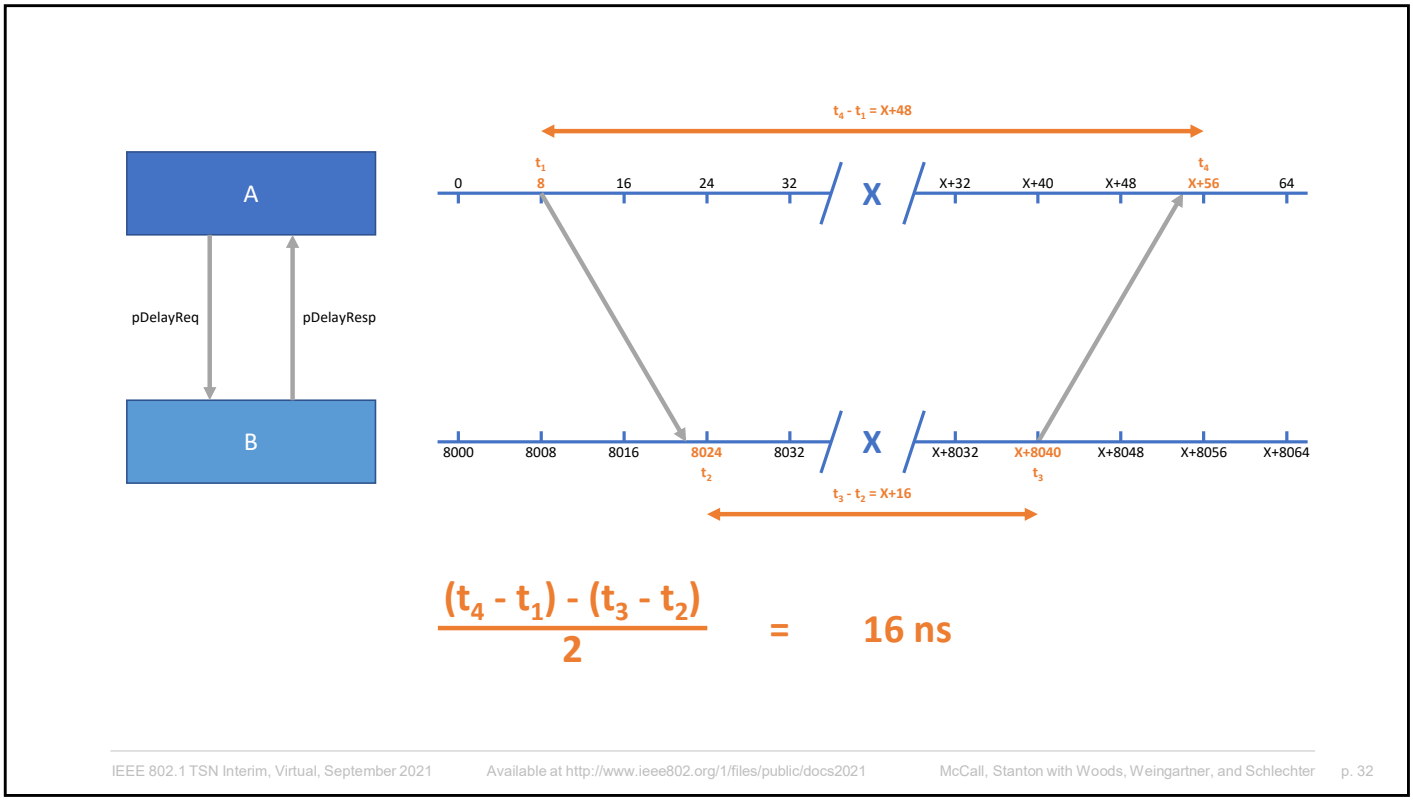
Offset / ns	Measured Path Delay / ns
0 – 2	16
>2 – 6	20
>6 – <8	16

Average = 18 ns ≠ 14 ns

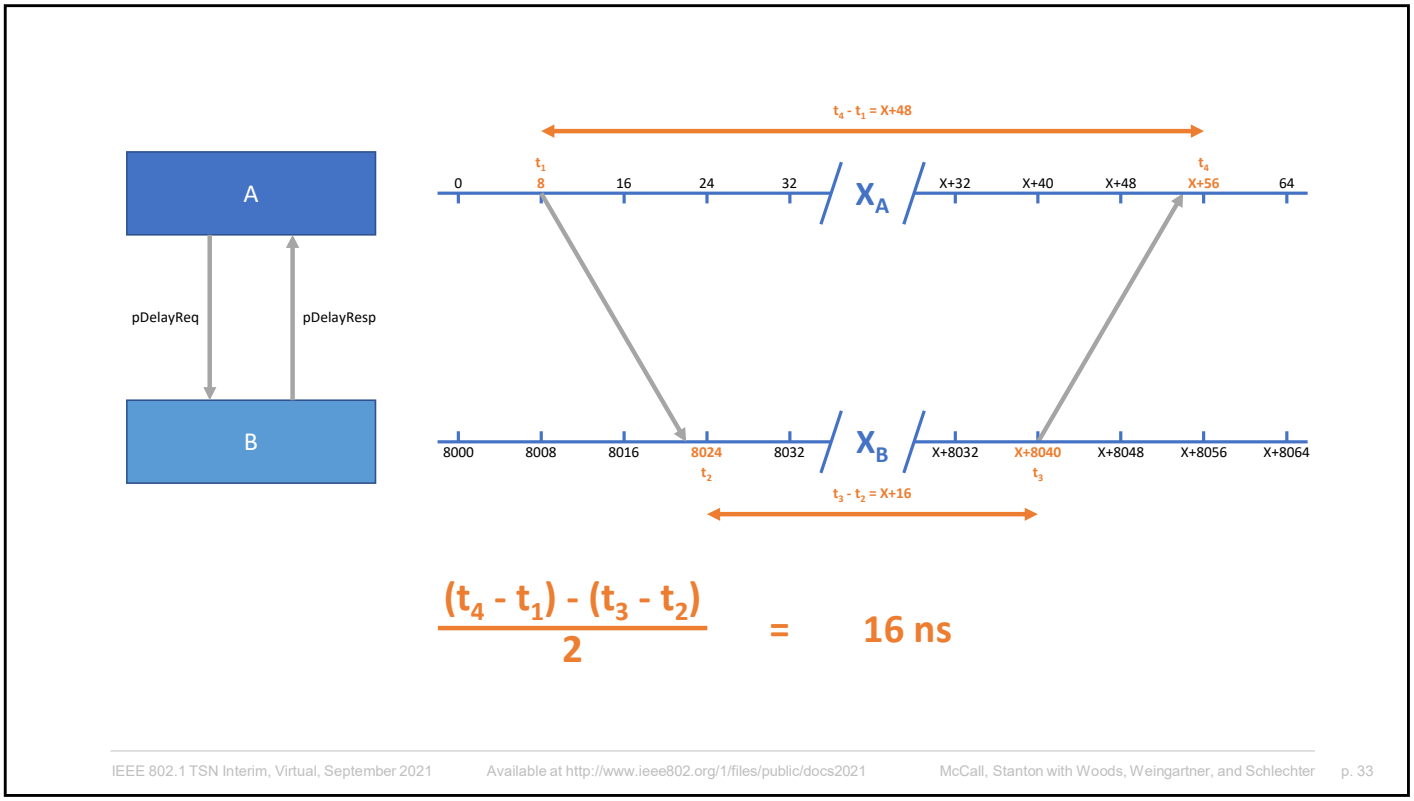
OK to average as, over multiple measurements, the timelines will move relative to each other.
But on it's own that doesn't help.

So…does this even work at all?

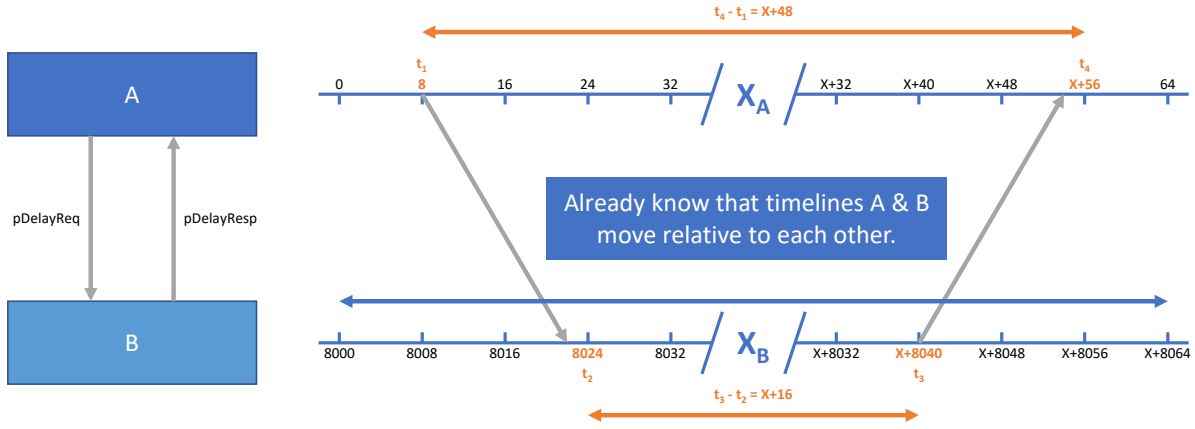
The good news is that yes, it does. There are two reasons…



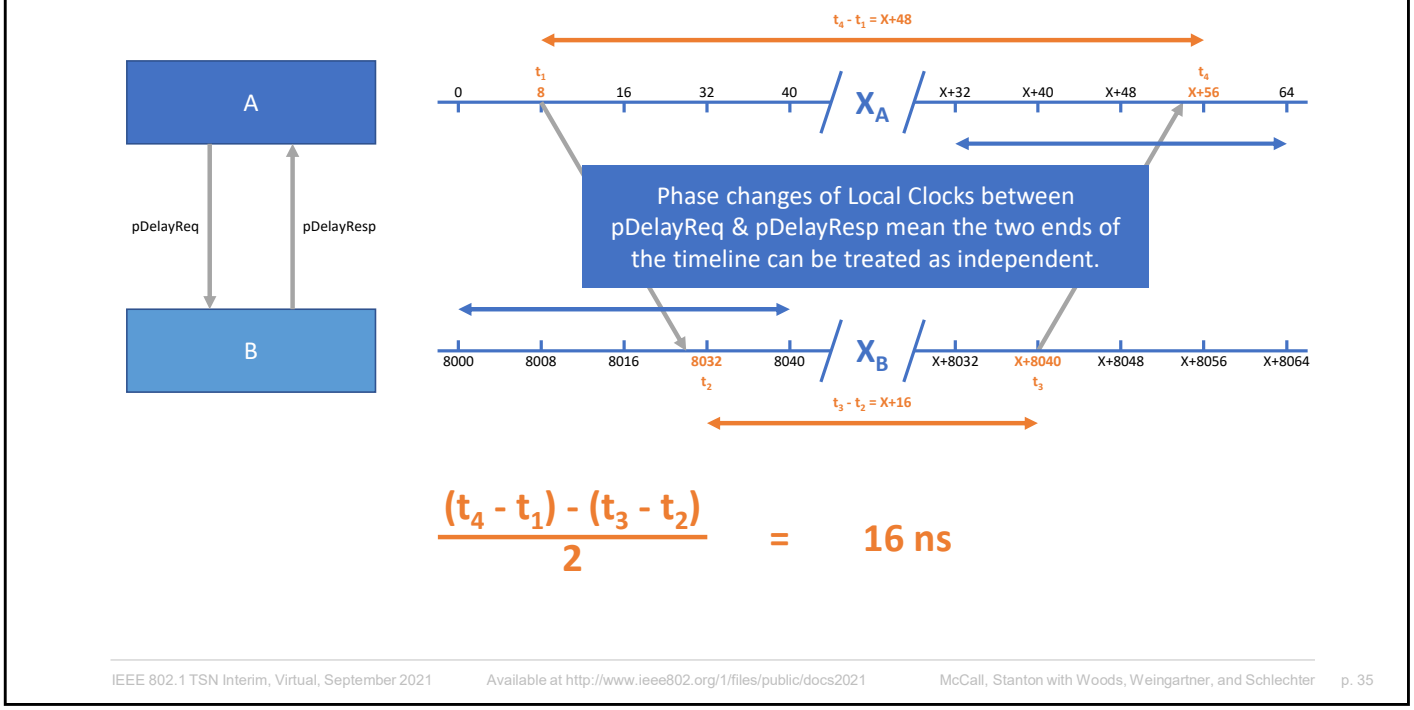
Firstly, there is a relatively large amount of time between the left and right ends of the timeline due to the fact that pDelay Response Time is much longer than pDelay…a fact I previously ignored.



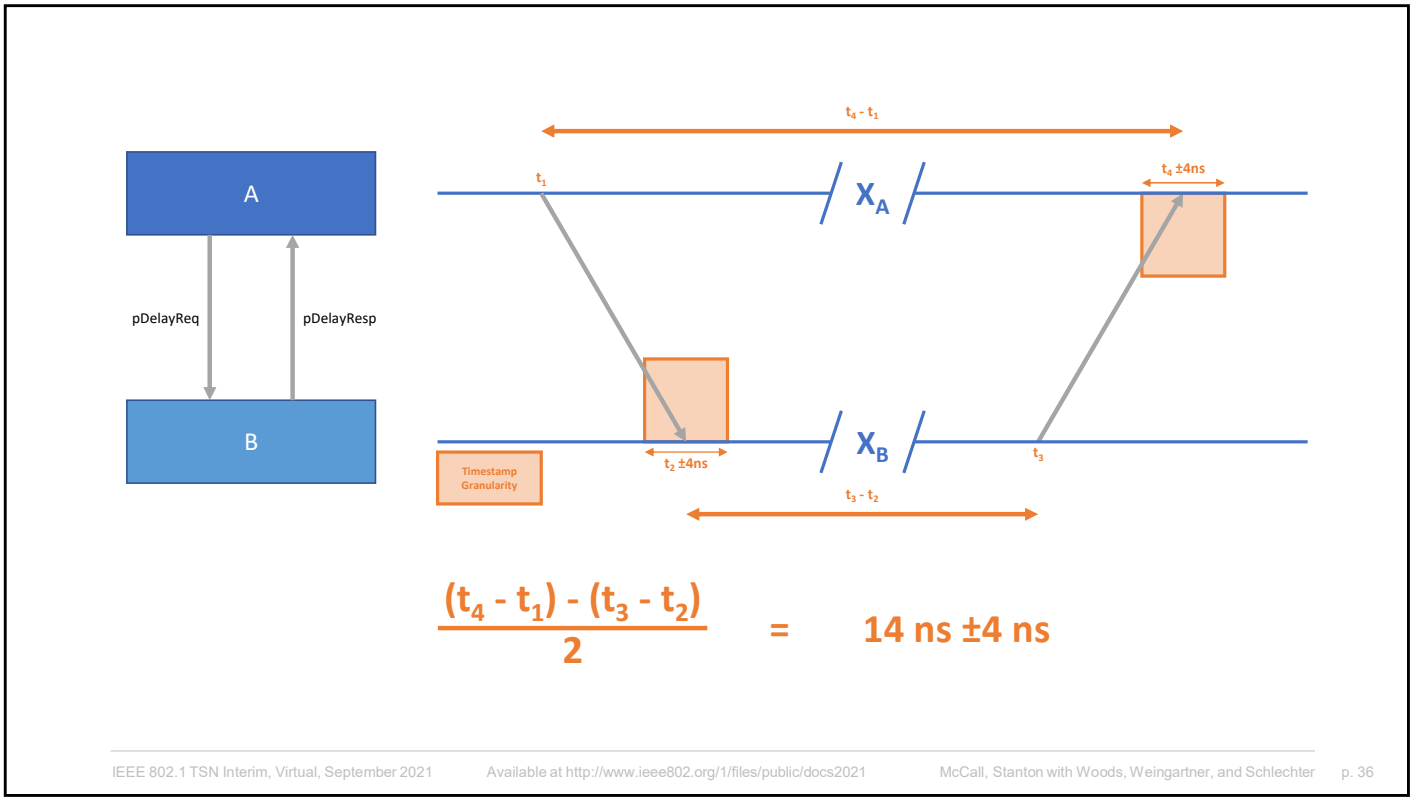
And secondly, that relatively long period of time is measured by two free running local clocks. Running at different speeds…and drifting slightly relative to each other.



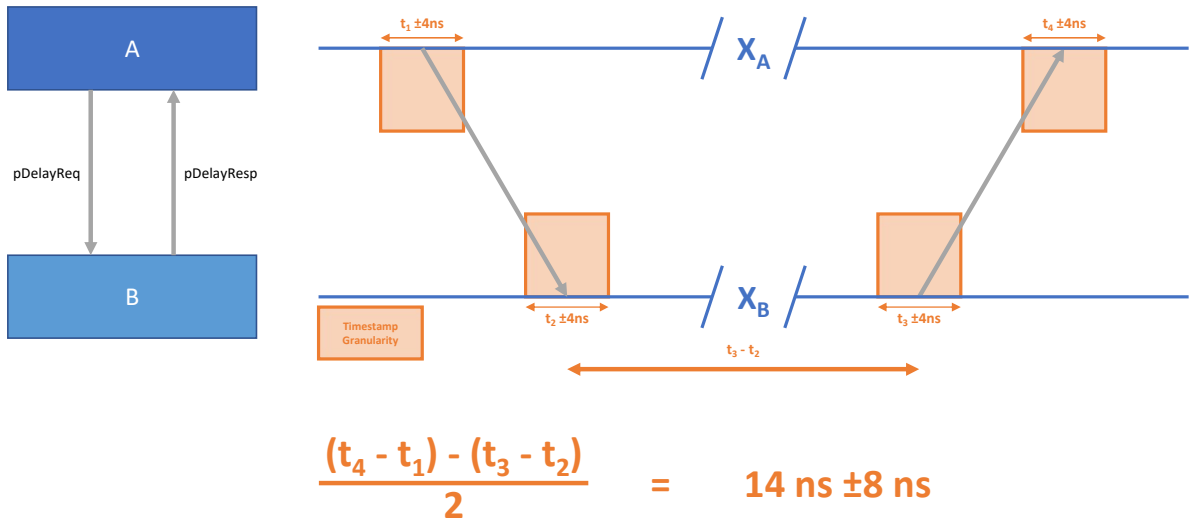
$$\frac{(t_4 - t_1) - (t_3 - t_2)}{2} = 16 \text{ ns}$$



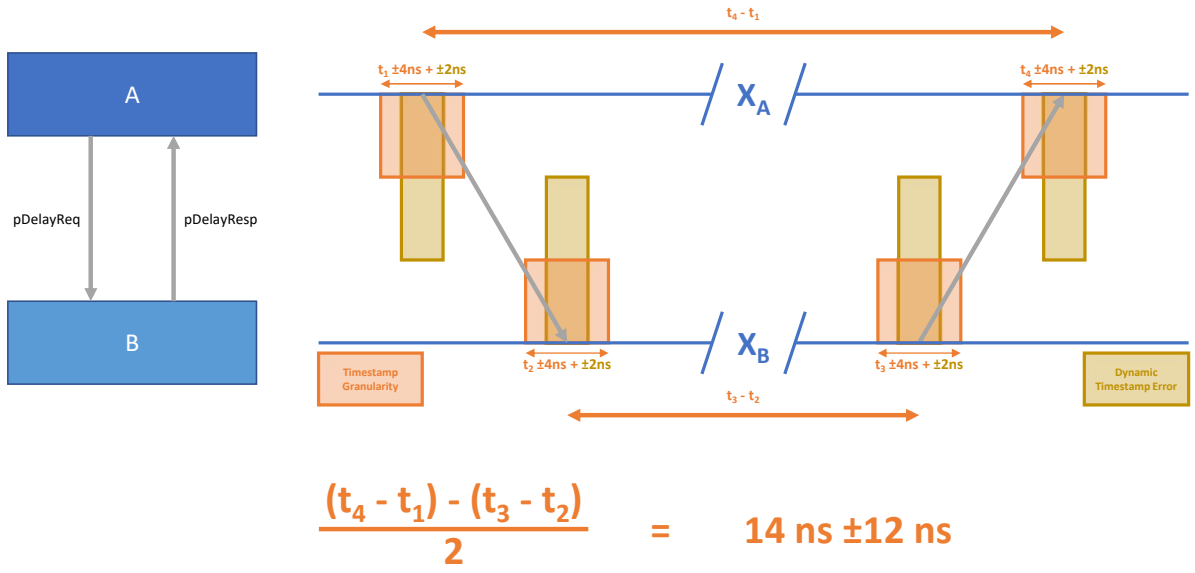
Because the left and right ends of the timeline can effectively be decoupled from each other…because there are different numbers of clock ticks in X_A and X_B (which is compensated for by NRR…which we aren't not currently considering)…you can look at the measurement probabilistically…which looks like this…



This shows only error due to Timestamp Granluarity…8ns…centred on the 14ns, i.e. ±4 ns



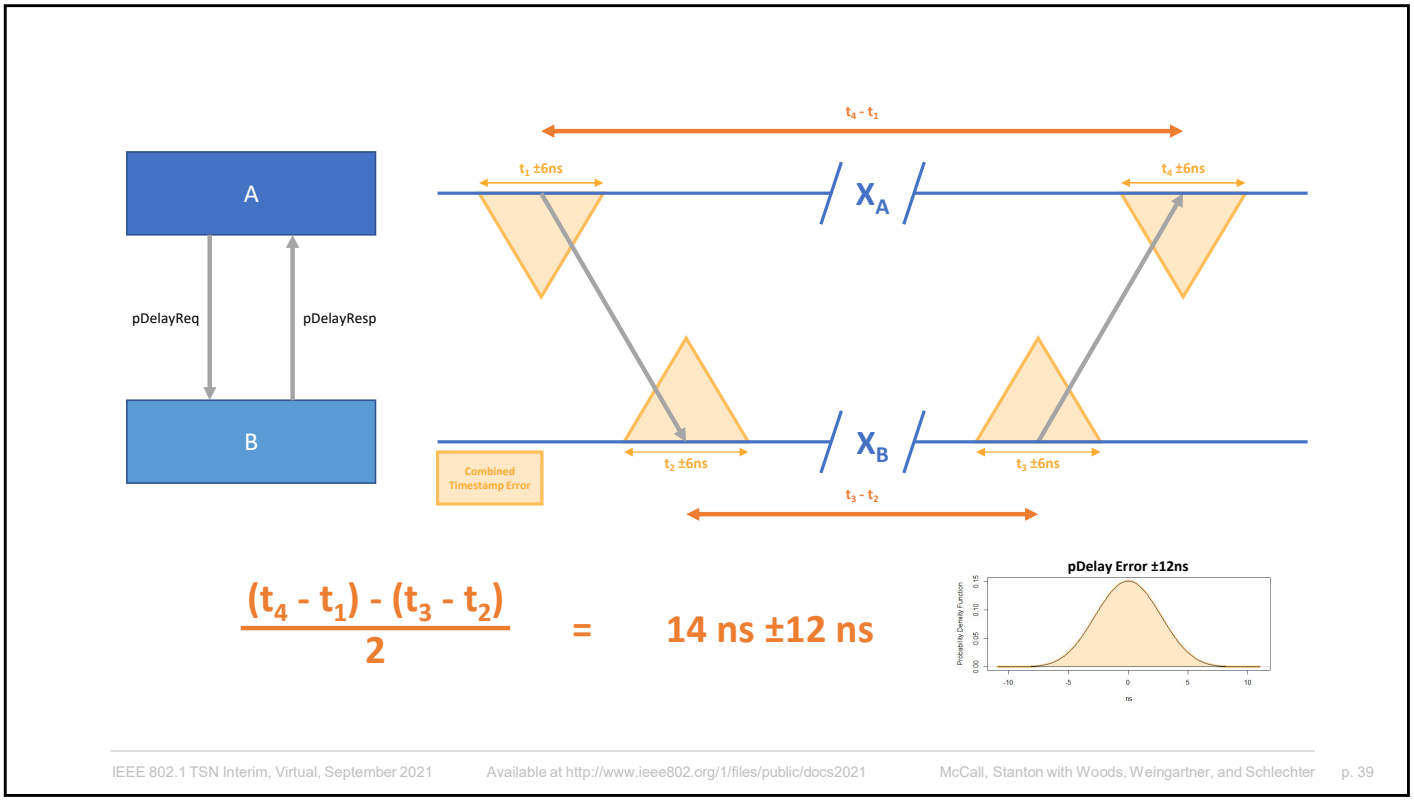
Add TX Timestamp Granularity.



Add Dynamic Time Stamp Error, ±2 ns.

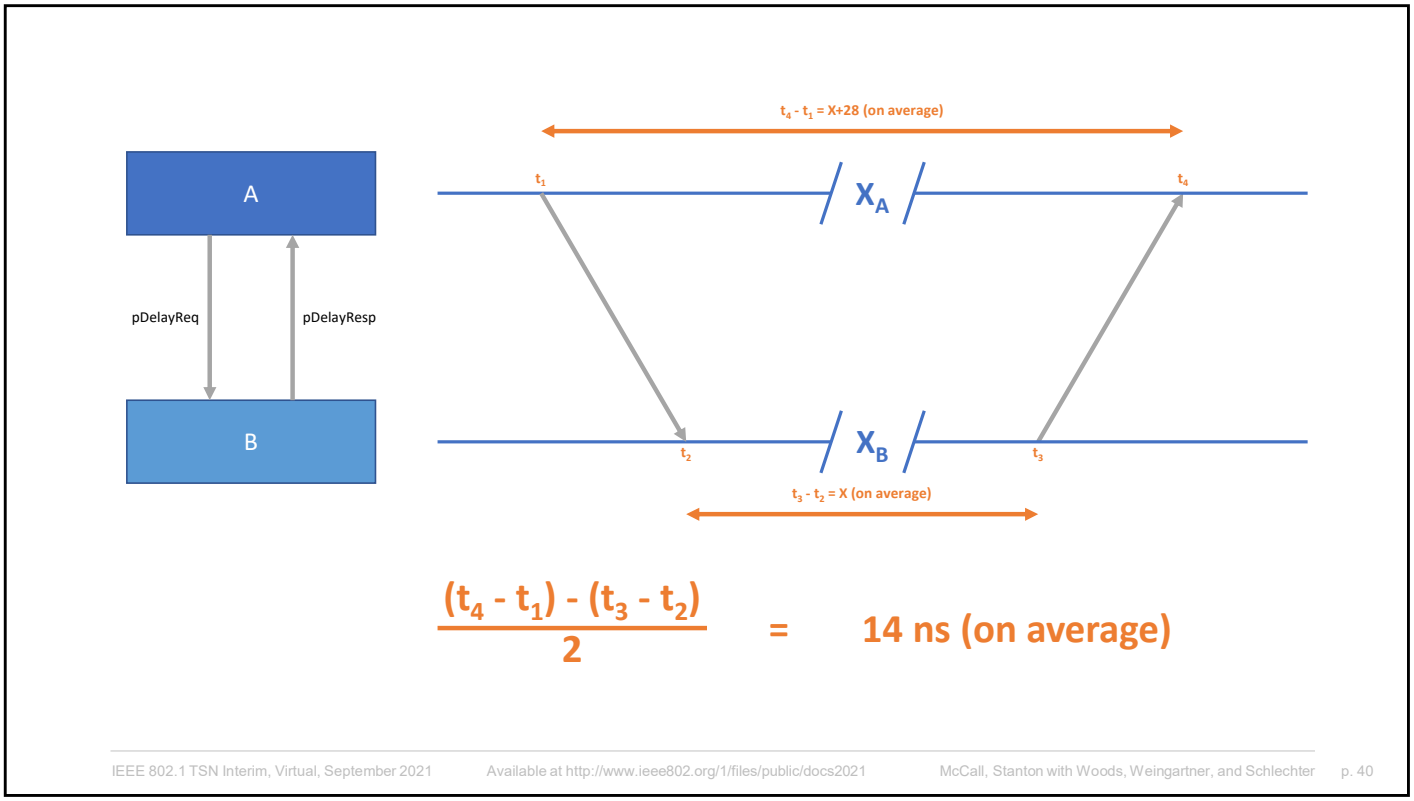
Indications are that this error should be less than Timestamp granularity. It varies with implementation and I'll let Kevin talk about that a bit more later…but for the moment let's just model it as this probability.

If you add together all 8 uniform probabilities…



You get this.

At each TX and RX point, two uniform probabilities combine to give a triangular probability…and those combine to an overall bell curve probability.

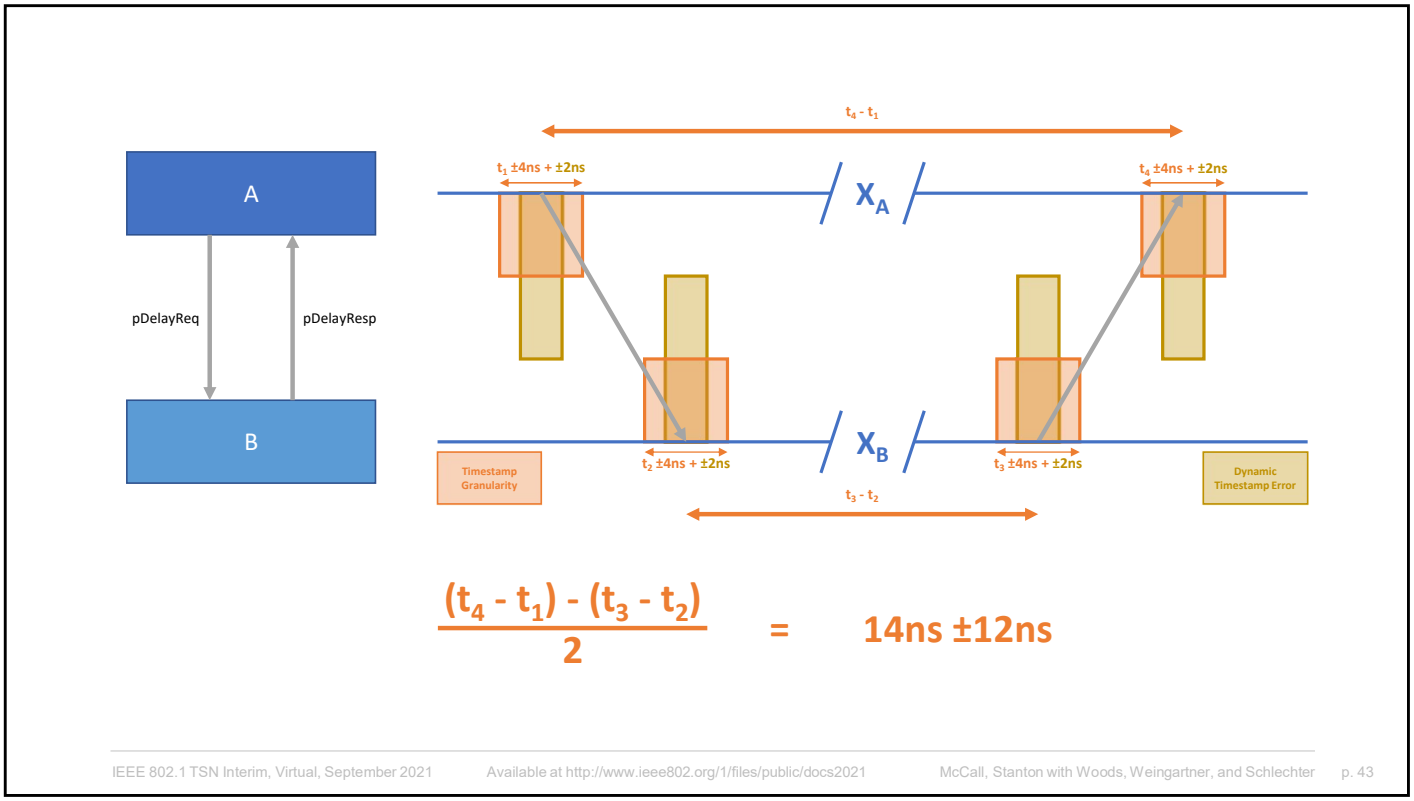


Which can be averaged out.

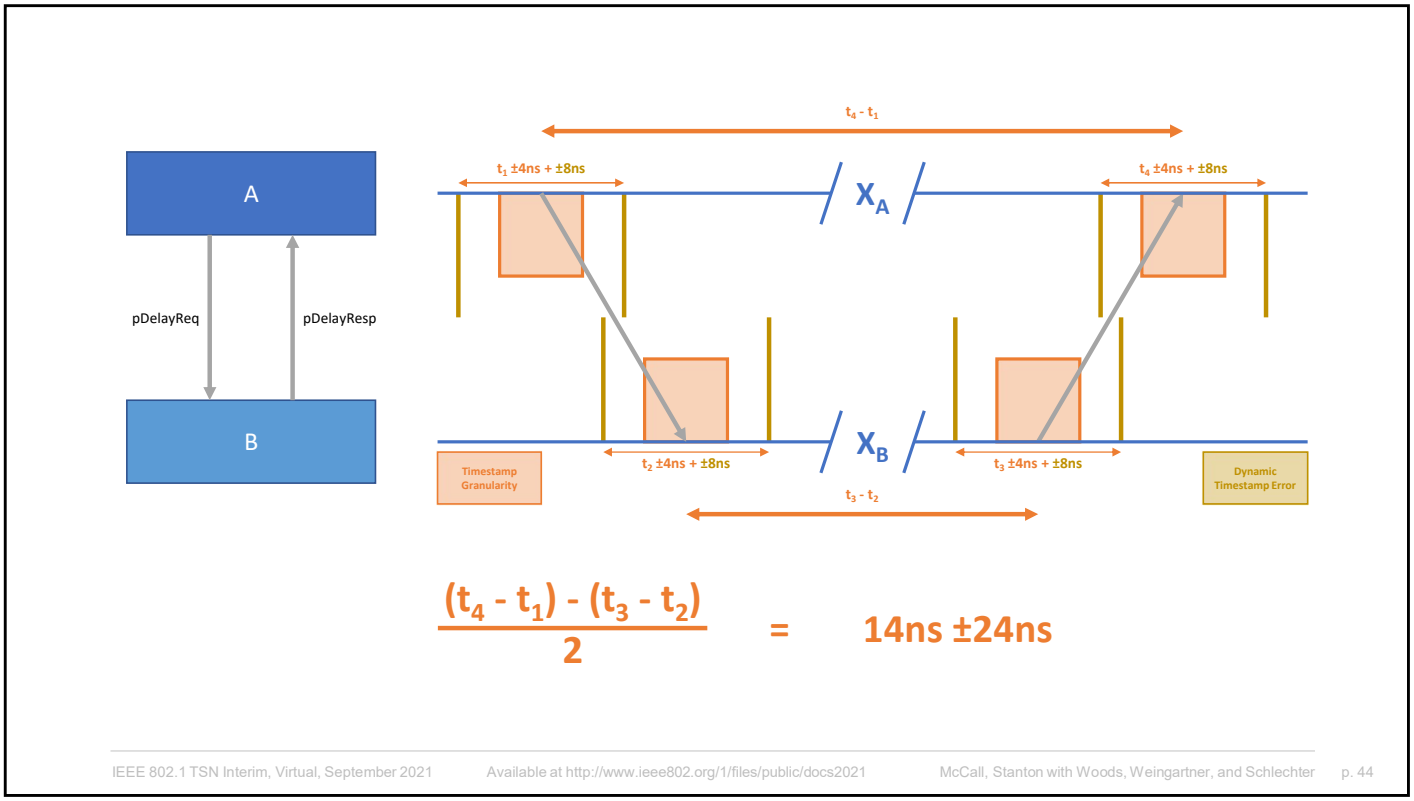
pDelay Error – Observations

- Averaging of pDelay measurements should be mandatory
  - Delivers pDelay estimate with much lower error
- Helpful if implementations exhibit error probability distributions that can be averaged out to zero
- For Ethernet…
  - The goal is to estimate a relatively fixed quantity
  - Very low bandwidth low-pass filter is appropriate due to stability of pDelay. (IIR?)
  - Some dedicated startup behaviour may be appropriate to arrive quickly at a useful average
- For wireless…
  - More complex due to multi-path and potential movement of devices, but…
  - Unlikely to have long chains of devices, or at least long chains of mobile devices.
  - Out of scope for the current discussion.

Current Simulation



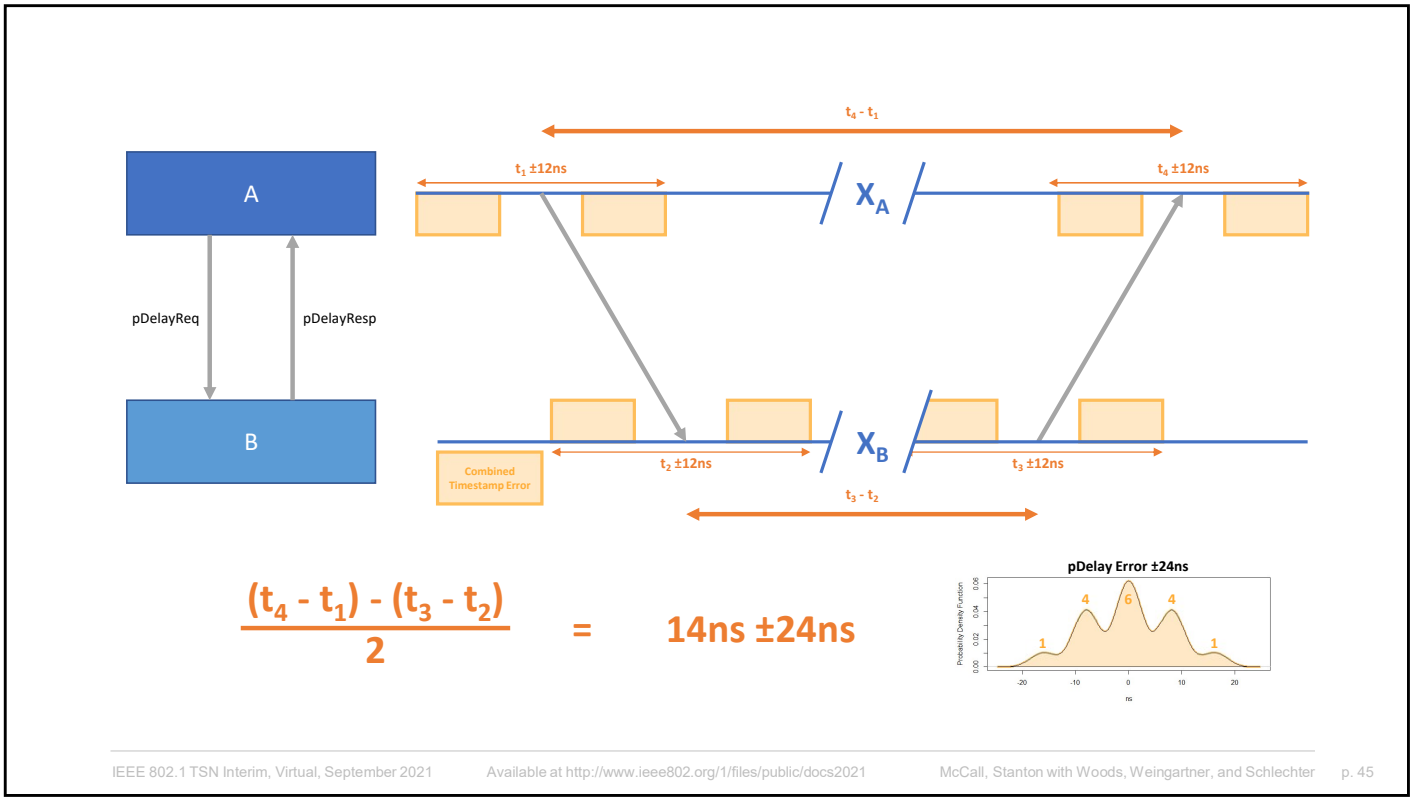
This is the model we just saw. The simulation does not do this. Instead it does…



This.

Dynamic Timestamp Error is ±8ns…but it's either +8 or -8 with a 50:50 change of either for each and every timestamp. That's represented by the two vertical lines…one at +8, one at -8.

Working out the probabilities…

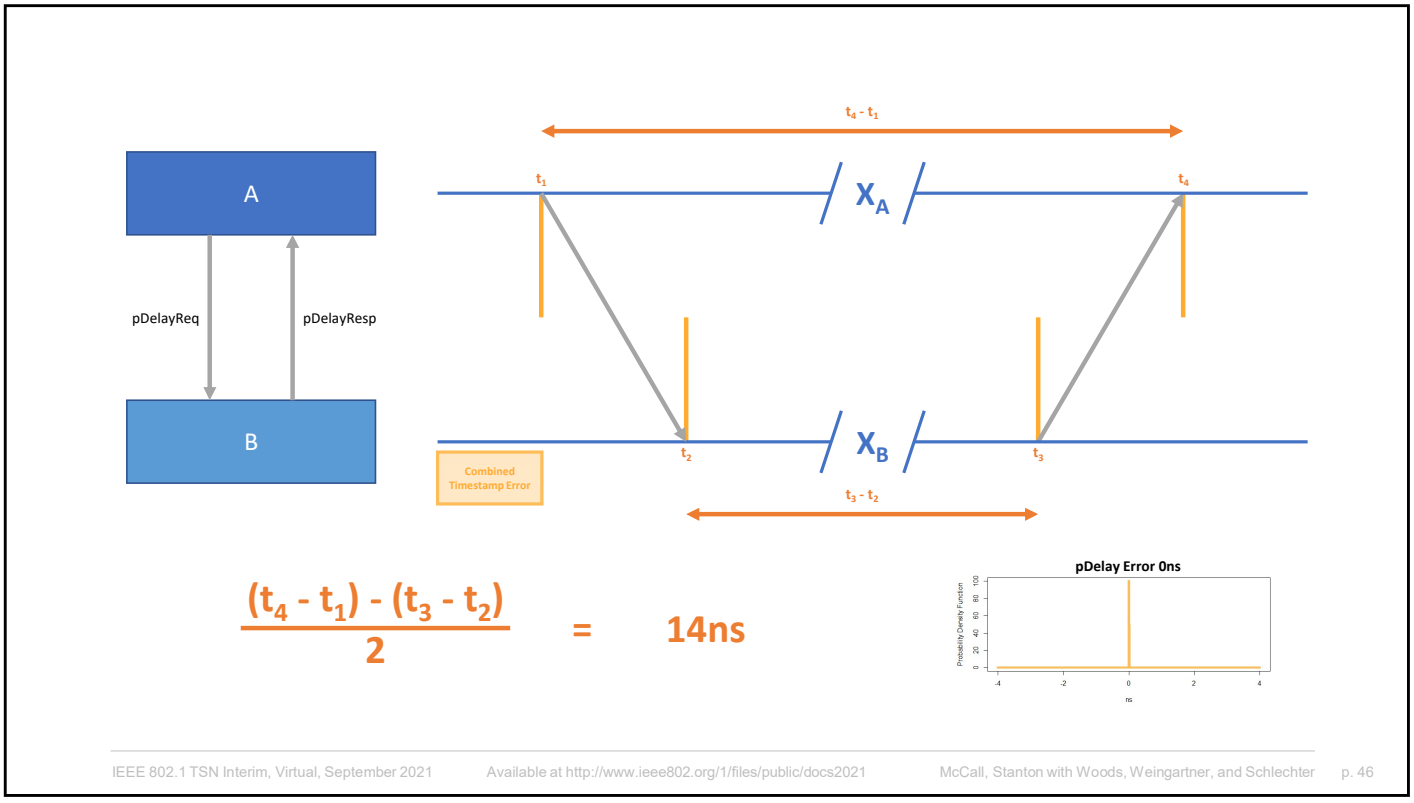


Case 1 in 60802-garner-new-simulation-results-new-freq-stab-model-0421-v02.pdf

And there is zero averaging of pDelay in the current simulation. The latest pDelay measurement is passed directly into the Correction Field.

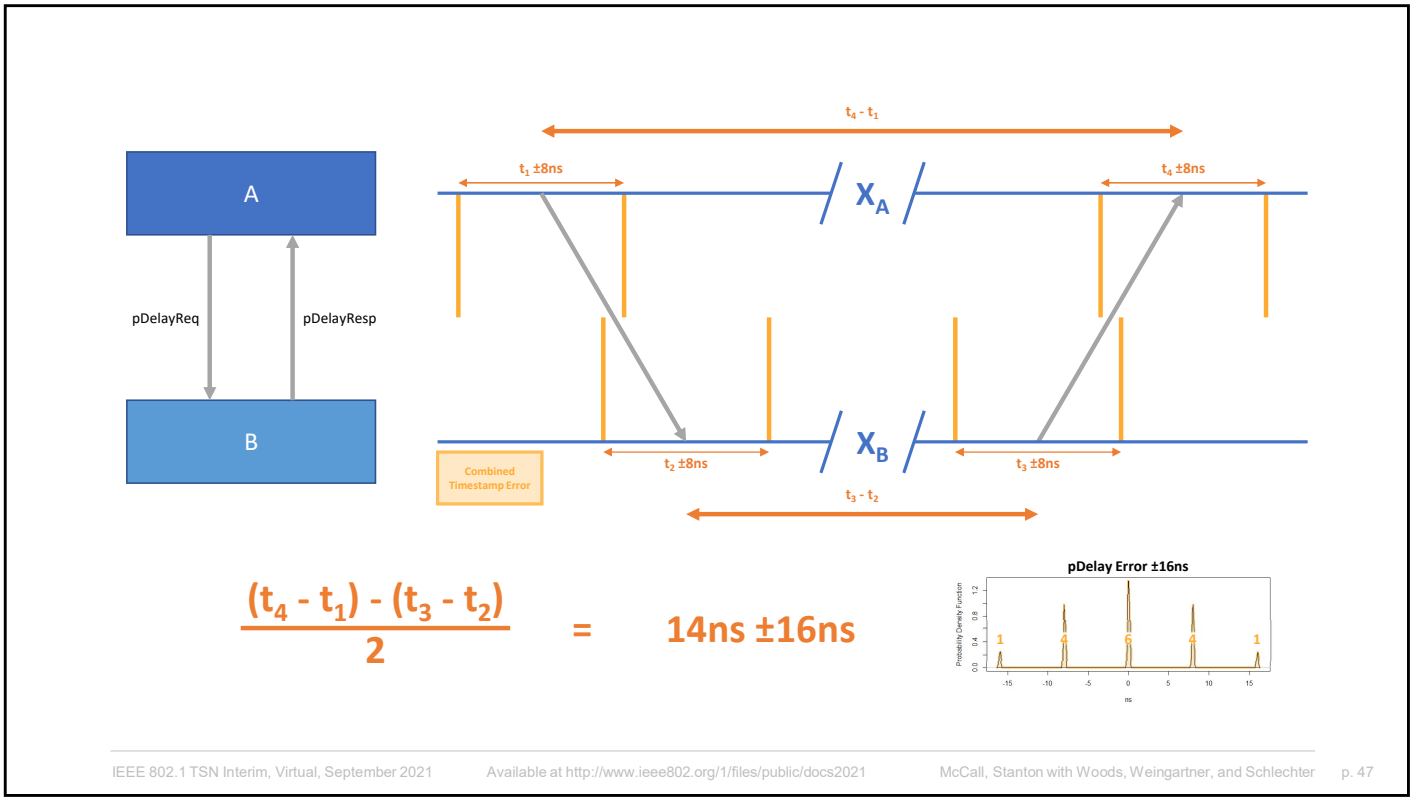
Oh…and if you are wondering if this could result in a negative pDelay, i.e. the simulation of time travel…**yes**, it could. But not with the simulation's pDelay value of 150ns.

There were also three other simulations of interest in that same run of simulations.



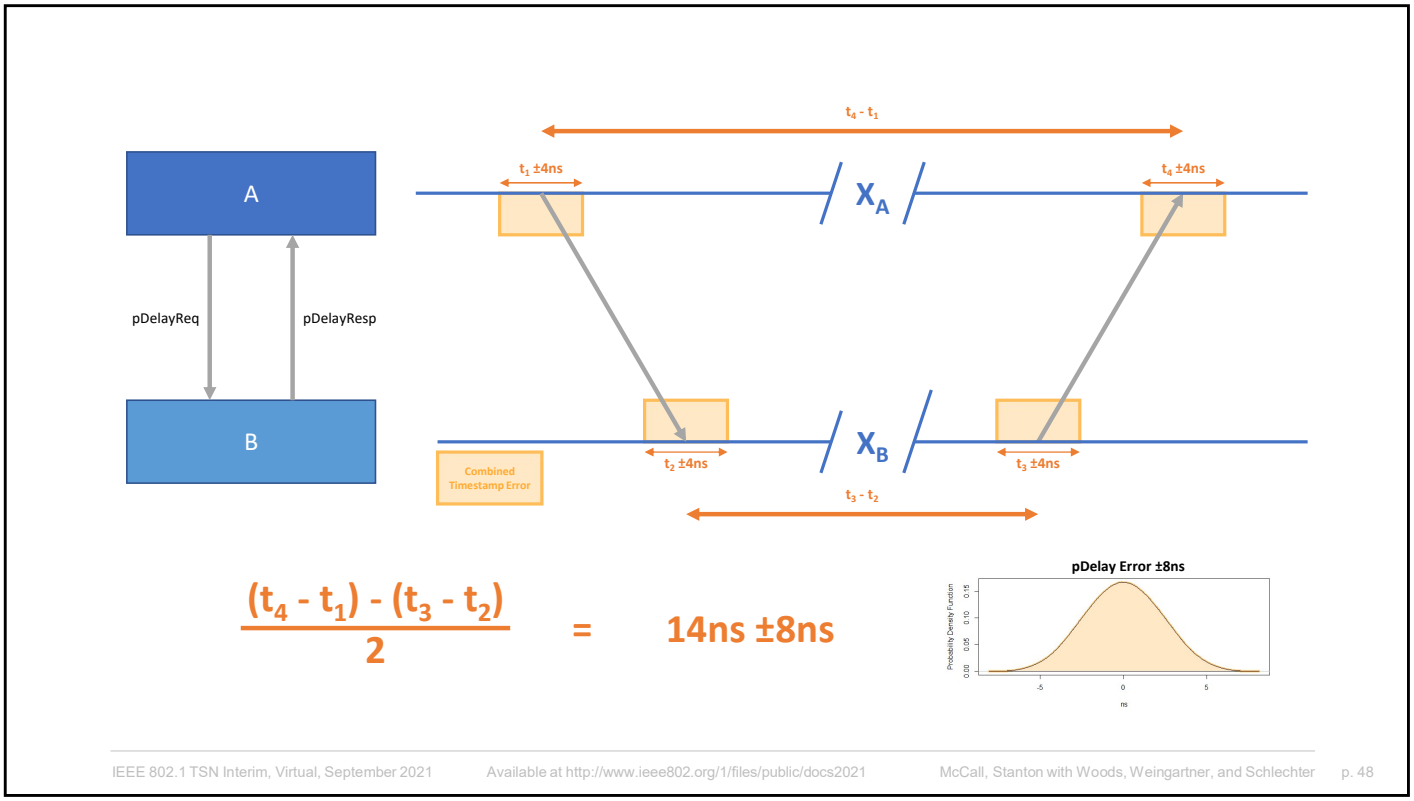
Case 13 in 60802-garner-new-simulation-results-new-freq-stab-model-0421-v02.pdf

This turned both errors to zero. Which, for pDelay is the equivalent of what averaging should accomplish.



Case 14 in 60802-garner-new-simulation-results-new-freq-stab-model-0421-v02.pdf

This removed Timestamp Granularity Error…but kept Dynamic Timestamp Error.

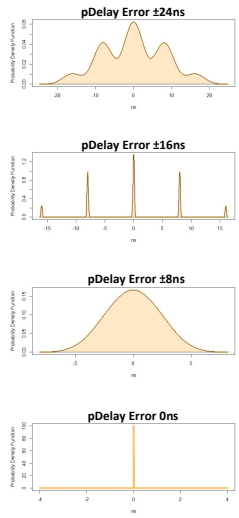


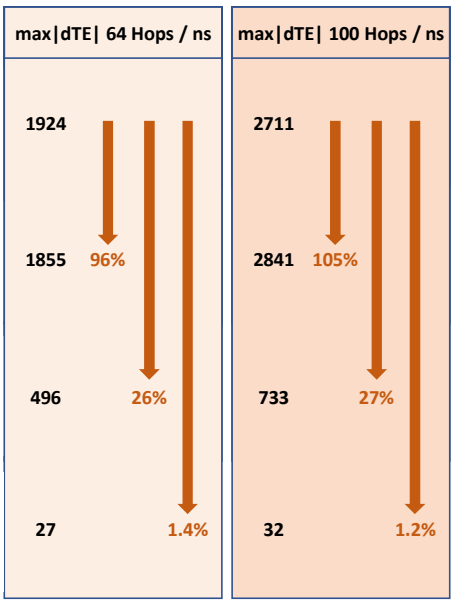
Case 15 in 60802-garner-new-simulation-results-new-freq-stab-model-0421-v02.pdf

And this did the opposite…kept Timestamp Granularity Error, but removed Dynamic Timestamp Error.

Comparison of Analysis & Latest Simulations

Case	Timestamp Granularity / ns	Dynamic Timestamp Error
1	8	±8
14	0	±8
15	8	0
13	0	0





Note that the simulation removed the effects of Timestamp Granularity and Dynamicstamp Error from both pDelay and Residence Time.

While pDelay could get close to this via averaging…you can't do that with Residence Time. So you'll get some big benefits from averaging…but not quite as much as shown here.

Modelling pDelay Error – Timestamp Granularity & Dynamic Timestamp Error

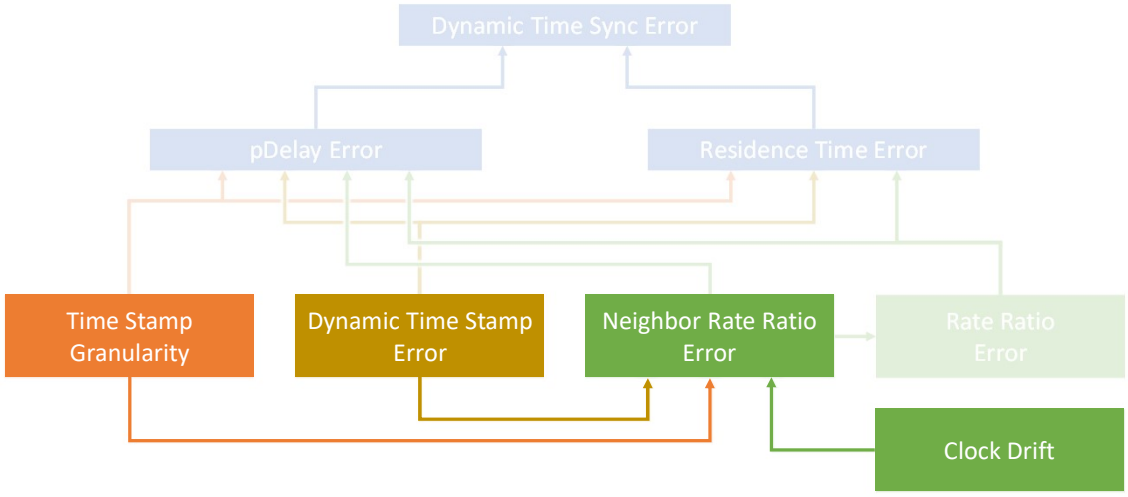
- Model linear error with uniform distribution for both factors.
- Allow separate RX & TX values for both, i.e. four factors…

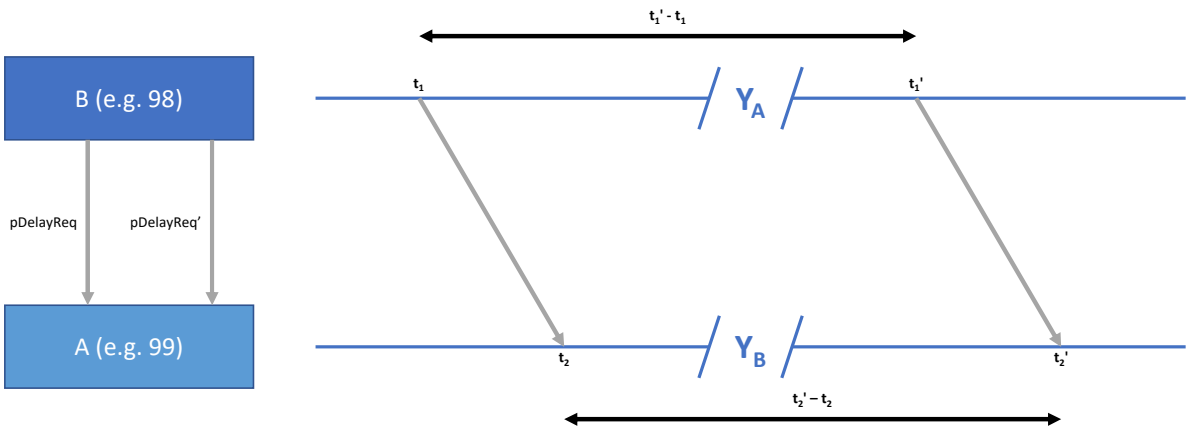
	Timestamp Granularity	Dynamic Time Stamp Error
RX	✓	✓
TX	✓	✓

- Add "Error Correction Factor" to account for averaging.
  - One value to cover all errors from all four factors.
  - 25% Effective → Removes 25% of the error.

Neighbor Rate Ratio

Time Sync – Elements & Relationships

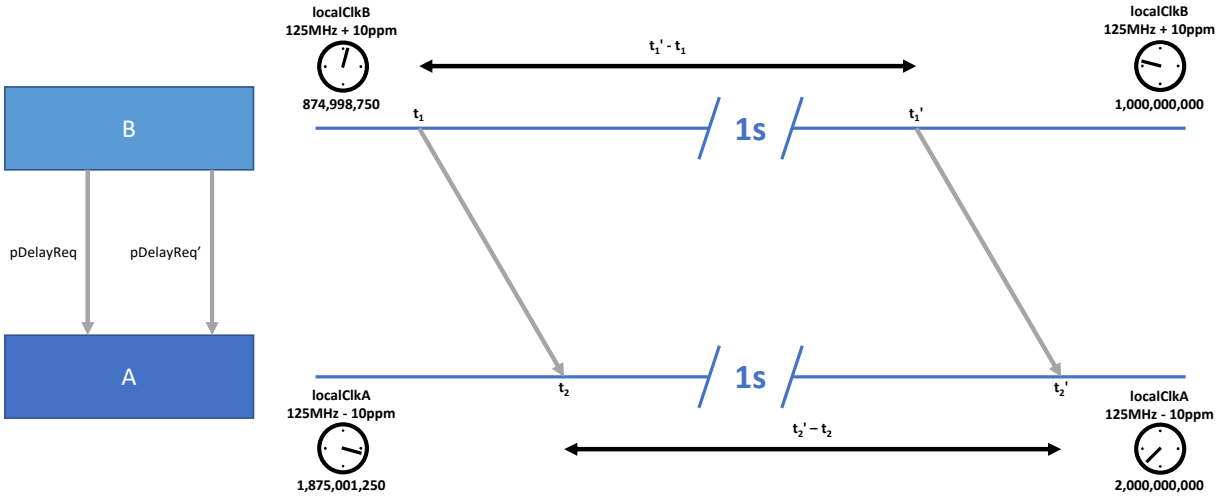




$$\text{Neighbor Rate Ratio @ } t_2' = \frac{(t_1' - t_1)}{(t_2' - t_2)}$$

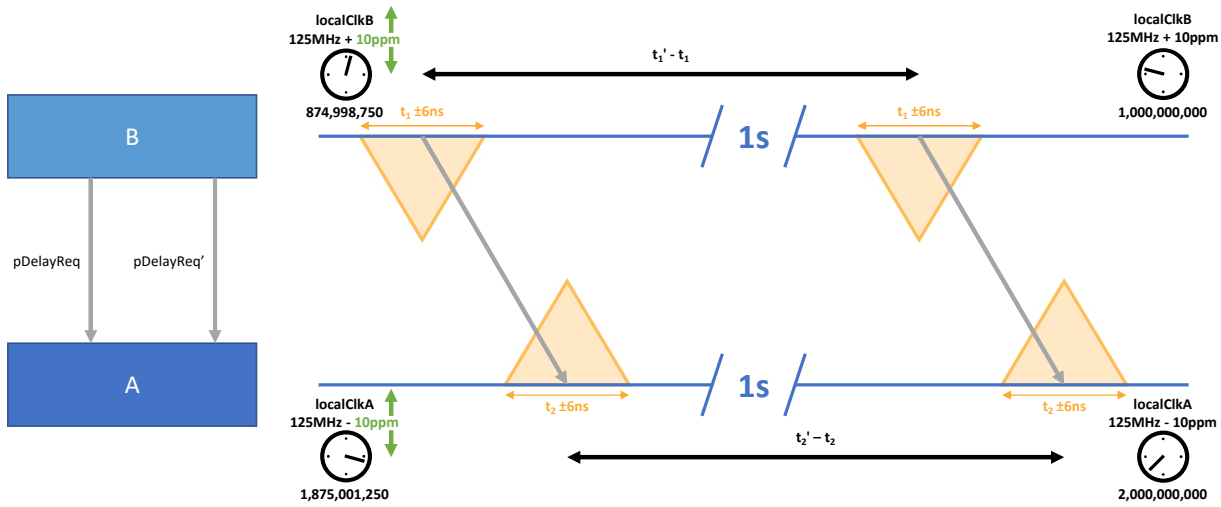
The Platonic ideal…

Note that A uses pDelayReq messages from B to measure Rate Ratio between the two. These are not the same messages that it uses to measure pDelay between A and B. For those exchanges the pDelayReq is issued by A.



$$\text{Neighbor Rate Ratio @ } t_2' = \frac{(t_1' - t_1)}{(t_2' - t_2)} = \frac{125{,}001{,}250}{124{,}998{,}750} = 0.002\% = 20\text{ppm}$$

Put some realistic number on it…



$$\text{Neighbor Rate Ratio @ } t_2' = \frac{(t_1' - t_1)}{(t_2' - t_2)} = \frac{125{,}001{,}250}{124{,}998{,}750} = 0.002\% = 20\text{ppm}$$

These are the errors that we could model.

Timestamp Granularity Error

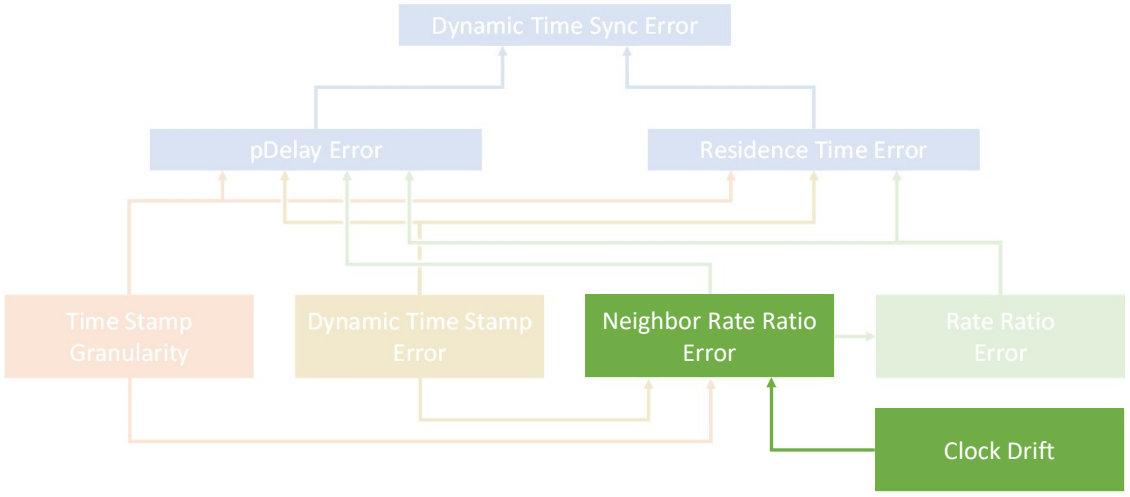
Dynamic Timestamp Error (shown combined)

Clock Drift (not the +10ppm and -10ppm…but the fact that those ppm values are changing over time, i.e. at the green points they **aren't** +10ppm and -10ppm)

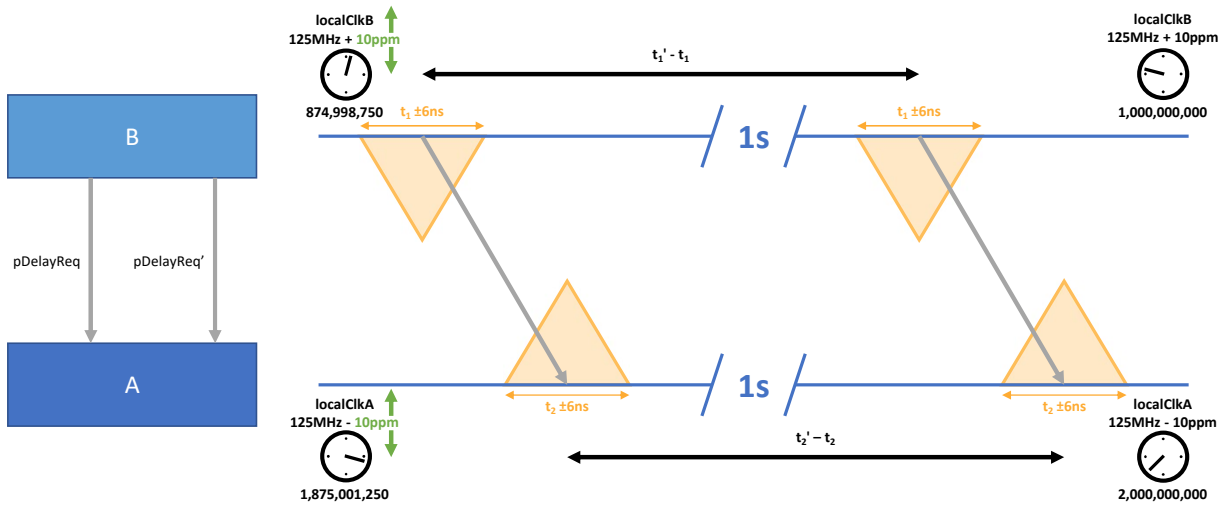
What factors to model?

Factor	YES / NO	
Time Stamp Granularity & Dynamic Time Stamp Error	NO	±20ns over 1s = ±0.02ppm Will be swamped by other errors.
Drift Rate of Local Clocks A & B (ppm/s) (Difference between Local Clock A ppm/s and Local Clock B ppm/s at start)	YES	For most recent temp profile, max possible difference between two clocks' rate of change of ppm is ±1.2ppm/s. ±1.2ppm/s over 1s = ±1.2ppm May be a significant contributor to overall dynamic time sync error.
Rate of change of Drift Rate of Local Clocks A&B (Difference between Local Clocks' ppm/s at start & finish.)	NO	For most recent temp profile, max rate of change of ppm/s (i.e. ppm/s per second) is ~±0.006ppm/s^2. Rounding up to ±0.01ppm/s^2, over 1s (or 5s) the change is not significant. (Note: ignores discontinuities at start and end of temperature ramp, which are not realistic.)

Sources of Dynamic Time Sync Error

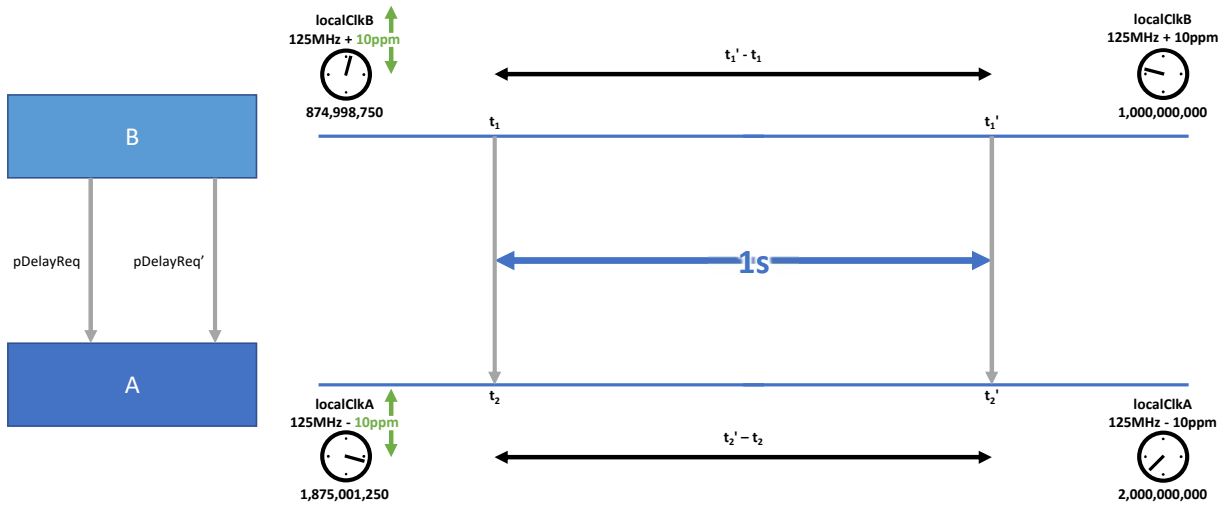


So, we only need to look at Clock Drift…



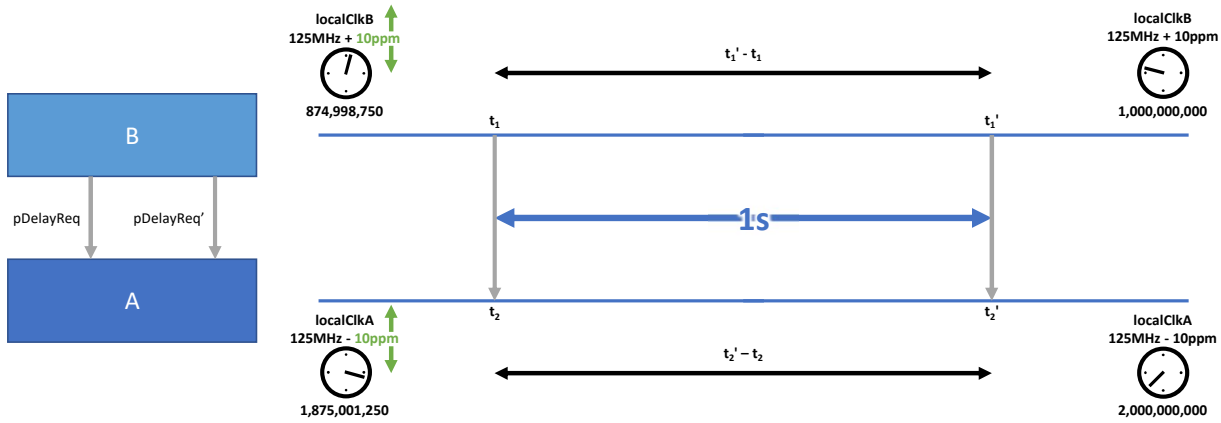
$$\text{Neighbor Rate Ratio @ } t_2' = \frac{(t_1' - t_1)}{(t_2' - t_2)} = \frac{125{,}001{,}250}{124{,}998{,}750} \rightarrow 0.002\% = 20\text{ppm}$$

So…remove the Timestamp Errors and modify the diagram so that we're looking at the appropriate scale…

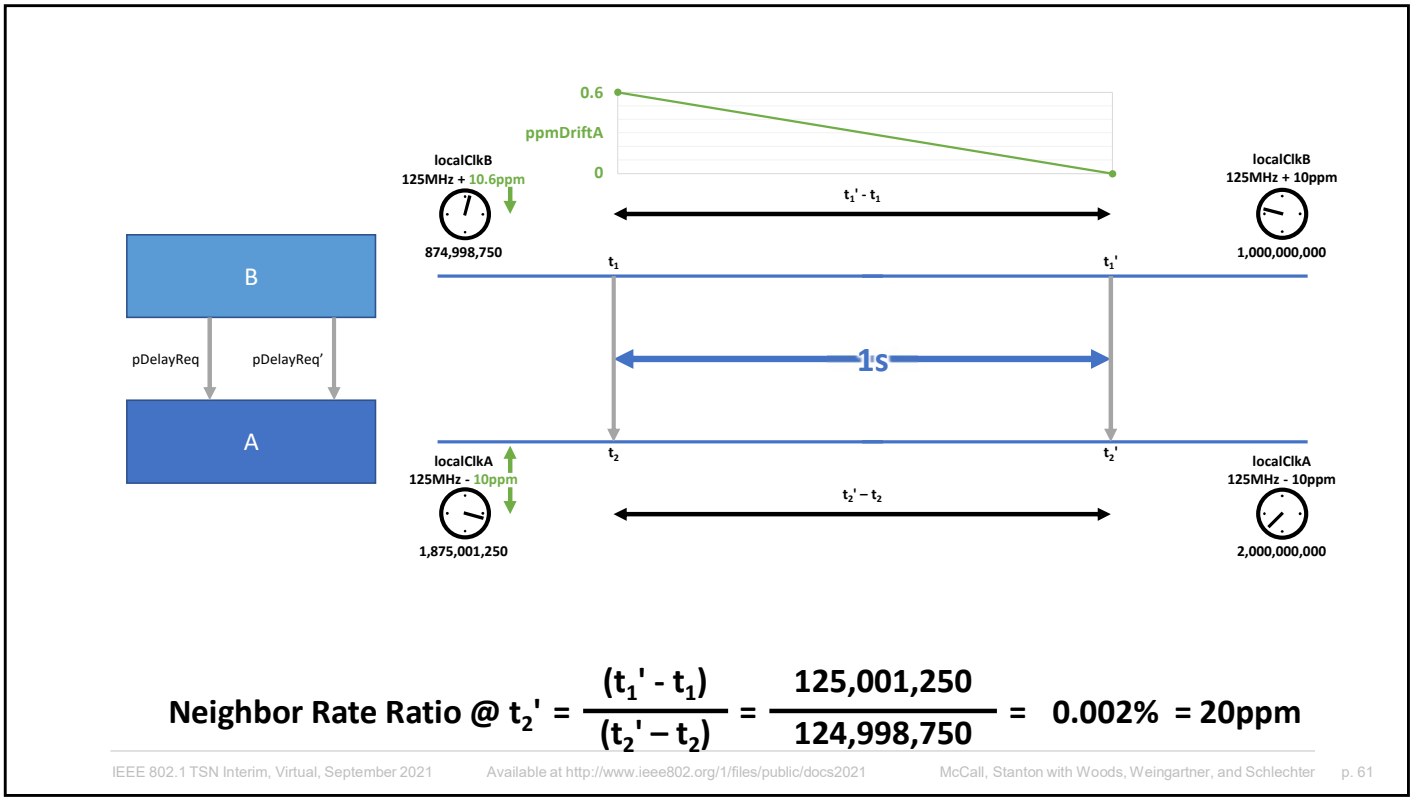


$$\text{Neighbor Rate Ratio @ } t_2' = \frac{(t_1' - t_1)}{(t_2' - t_2)} = \frac{125{,}001{,}250}{124{,}998{,}750} = 0.002\% = 20\text{ppm}$$

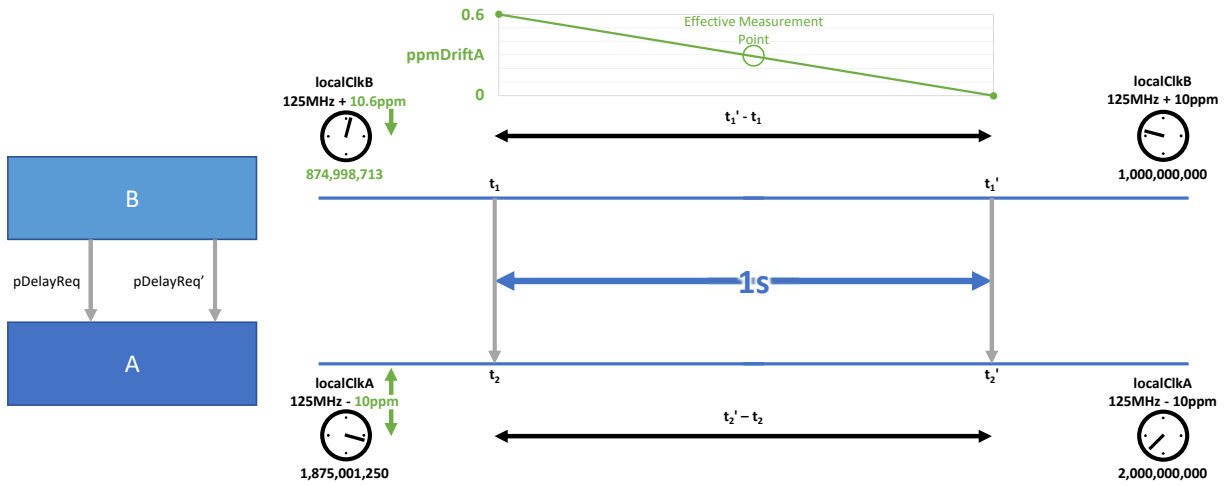
Then squeeze things together to make a bit of room…



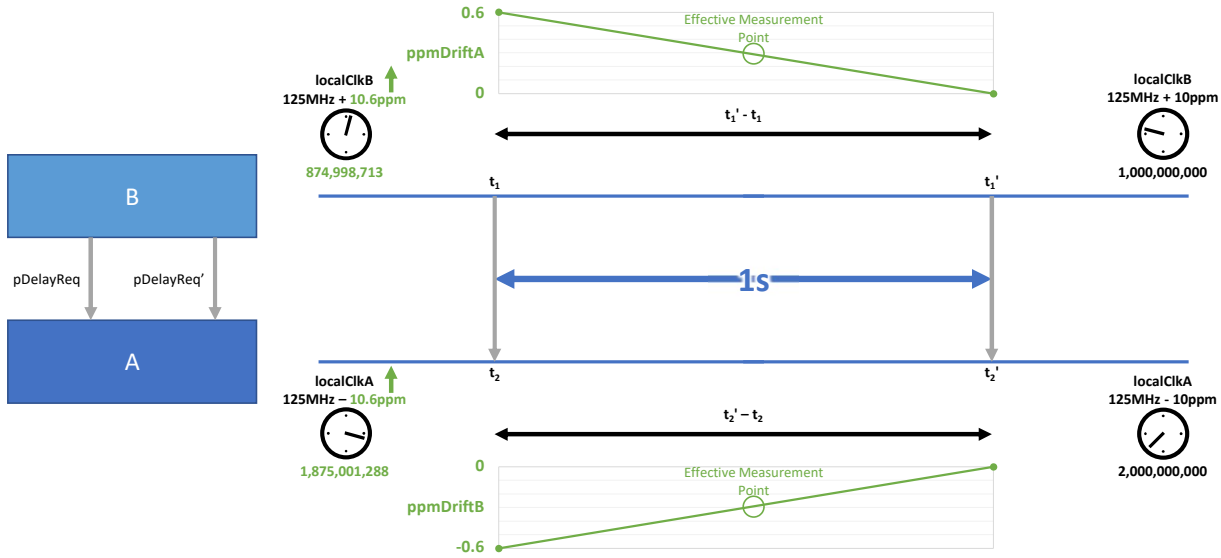
$$\text{Neighbor Rate Ratio @ } t_2' = \frac{(t_1' - t_1)}{(t_2' - t_2)} = \frac{125{,}001{,}250}{124{,}998{,}750} = 0.002\% = 20\text{ppm}$$



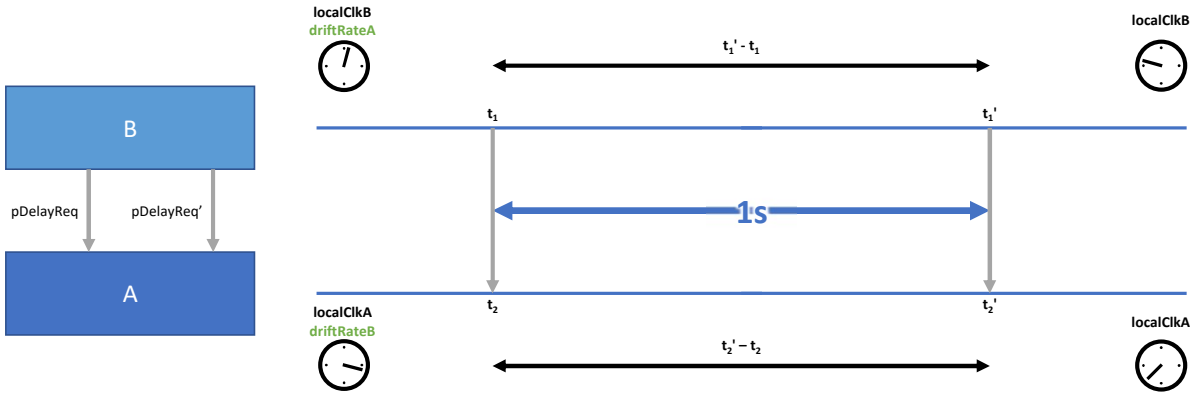
First…add a -0.6ppm drift on



$$\text{Neighbor Rate Ratio @ } t_2' = \frac{(t_1' - t_1)}{(t_2' - t_2)} = \frac{125{,}001{,}287}{124{,}998{,}750} = 0.00203\% = 20.3\text{ppm}$$



$$\text{Neighbor Rate Ratio @ } t_2' = \frac{(t_1' - t_1)}{(t_2' - t_2)} = \frac{125{,}001{,}287}{124{,}998{,}712} = 0.00206\% = 20.6\text{ppm}$$



$$\text{mNRR}_{\text{error}}\ (\text{ppm}) = \frac{(\text{driftRateA} - \text{driftRateB}) * (t_2' - t_2)}{2}$$

(Difference between measured and actual Neighbor Rate Ratio @ t_1)

$$\text{(For simulation)} = \frac{(\text{driftRateA} - \text{driftRateB}) * \text{pDelayInterval}}{2}$$

Neighbor Rate Ratio Error - Observations

- There is a difference between actual NRR and measured NRR (mNRR)
- When clock rates are stable mNRR is very accurate
  - Almost no difference between real NRR and mNRR
  - No large impact from timestamp granularity or errors
  - No benefit from averaging mNNR (or taking a median of previous values) if (Timestamp Errors / pDelay Interval) is negligible in terms of ppm (it actually makes things worse)
- When clock rates are drifting mNRR may have significant errors
  - Significant difference between real NRR and mNRR
  - Minimised when clock rates of neighboring devices drift in the same direction
  - Maximised when clock rates of neighboring devices drift in different directions
    - PPM vs. Temp curve for XO means the latter could be more common than expected
- There is the possibility of compensating for Neighbor Rate Ratio Error

Modelling Neighbor Rate Ratio – 1

- Use formula from earlier to model NRR_{error} at pDelayReq…

$$\mathbf{mNNR_{error}} = \frac{\mathbf{(driftRateA - driftRateB)*pDelayInterval}}{\mathbf{2}}$$

- Model driftRate as stable (0ppm) or varying ±0.6ppm (linear uniform distribution)
  - X% stable; 100-X% varying
- Add "Error Correction Factor" to account for modelling of drift (NRRdrift), correction of NRR (cNRR), and correction of other calculations via use of aNRR
  - One value to cover all errors related to NRR_{error} and drift.
  - 25% Effective → Removes 25% of the error.

Modelling Neighbor Rate Ratio – 2

- Actual error when NRR is used (NRR_{error}) will be larger than $mNRR_{error}$ if time has passed since measurement, allowing clocks to drift further apart. Relevant for Residence Time Error

$$\mathbf{NNR_{error} = mNRR_{error} + (driftRateA - driftRateB) * timeSincemNRR}$$

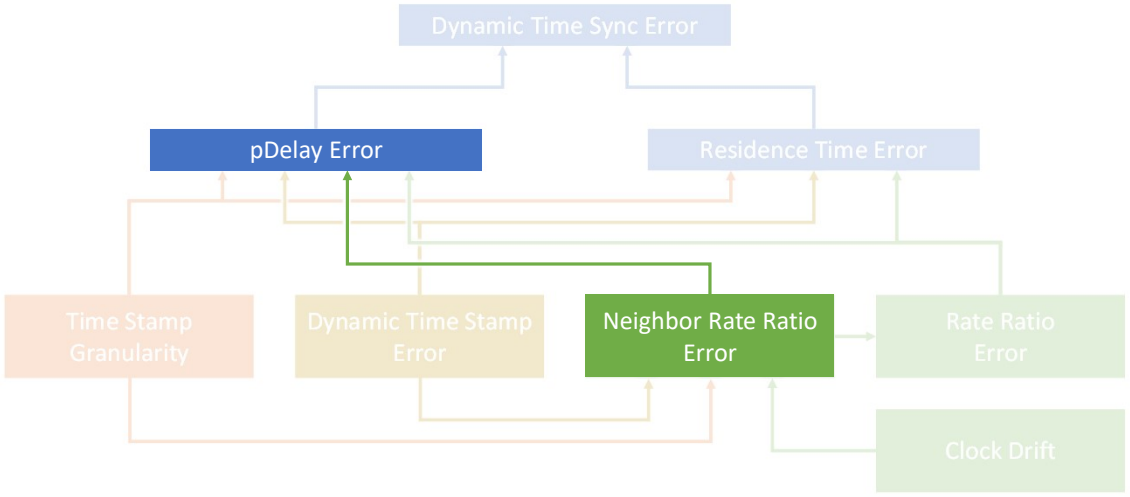
- Model timeSincemNRR as uniform distribution (min=0, max=pDelay).
- Apply same NRR_{error} correction factor to second term
  - 25% Effective → Removes 25% of the error.

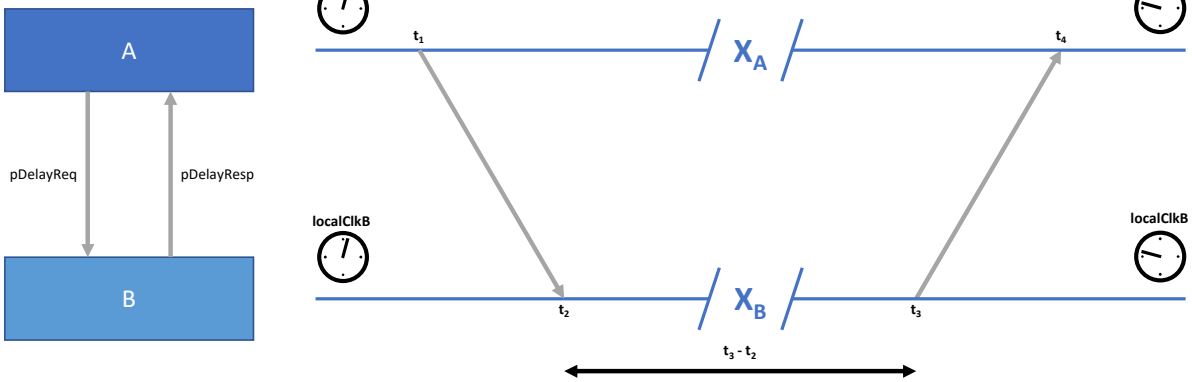
Current Simulation

- Mean pDelay Interval: 31.25ms
  - Would mean mNRRerror = ±0.019ppm, but…
- mNRR is measured based on t1 & t2 from most recent pDelay exchange…and the median of this plus 6 previous pDelay exchange (median of 7 exchanges)
  - As rate of change of clock drift is (ppm/s2) is low, clock drift can be treated as linear, so value from 3 pDelay intervals ago will be chosen every time.
  - This pushes the effective measurement time back by 3 pDelay intervals, i.e. makes $mNRR_{error}$ 7x worse.
  - Worst case $mNRR_{error}$ due to drift: ±0.13ppm
- pDelay interval is short enough that Timestamp errors have an appreciable effect
  - ±20ns over 31.25ms is ±0.64ppm
- Worst Case NRRerror: ±0.77ppm
- Effect of Clock drift on calculations made between NRR measurements is low due to low pDelay Interval
  - ±1.2ppm over 31.25ms is ±0.0375ns

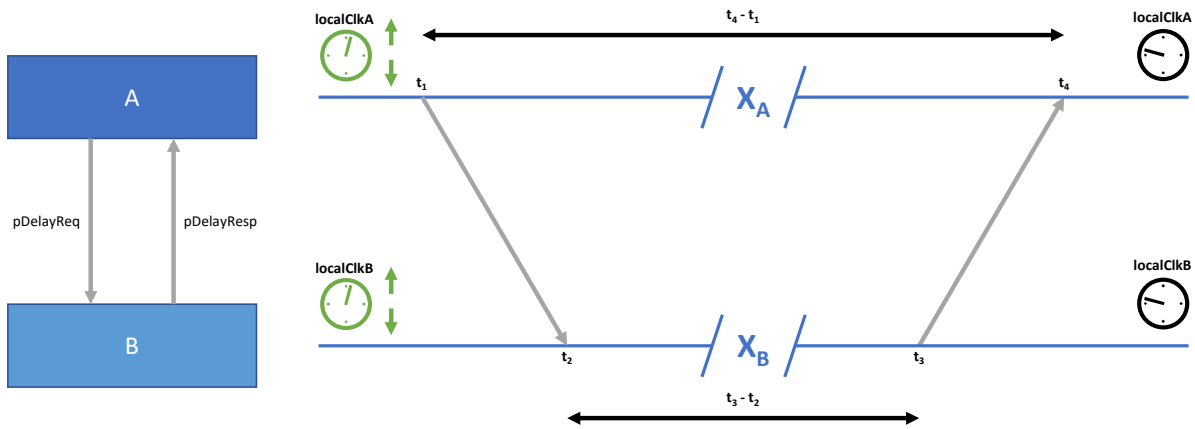
pDelay – Neighbor Rate Ratio Error

Time Sync – Elements & Relationships

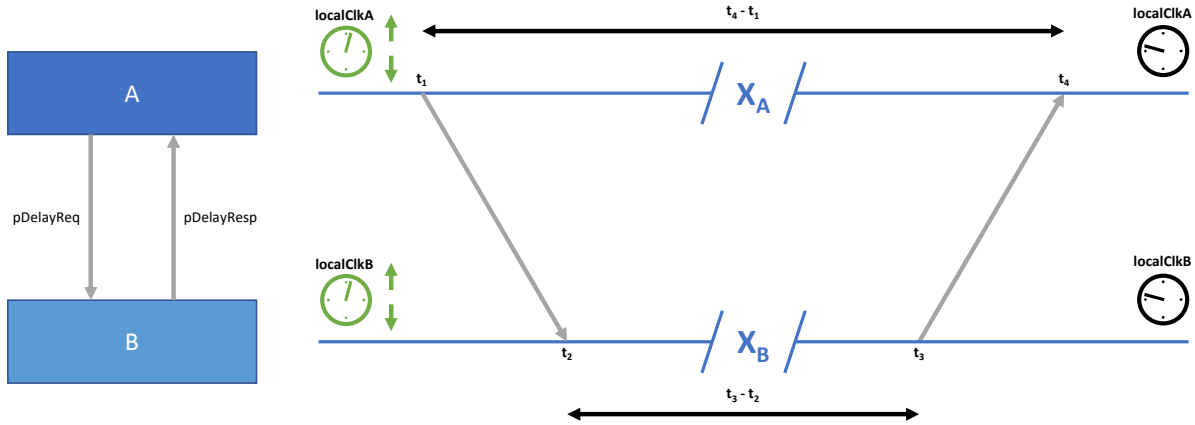




$$pDelay = \frac{(t_4 - t_1) - NRR(t_3 - t_2)}{2}$$



$$pDelay = \frac{(t_4 - t_1) - NRR(t_3 - t_2)}{2}$$



$$pDelay = \frac{(t_4 - t_1) - NRR(t_3 - t_2)}{2}$$

$$pDelay_{errorNRR}\ (ns) = (t_3 - t_2) * NRR_{error}$$

$$pDelay_{errorNRR}\ (ns) = (t_3 - t_2) * NRR_{Error}$$

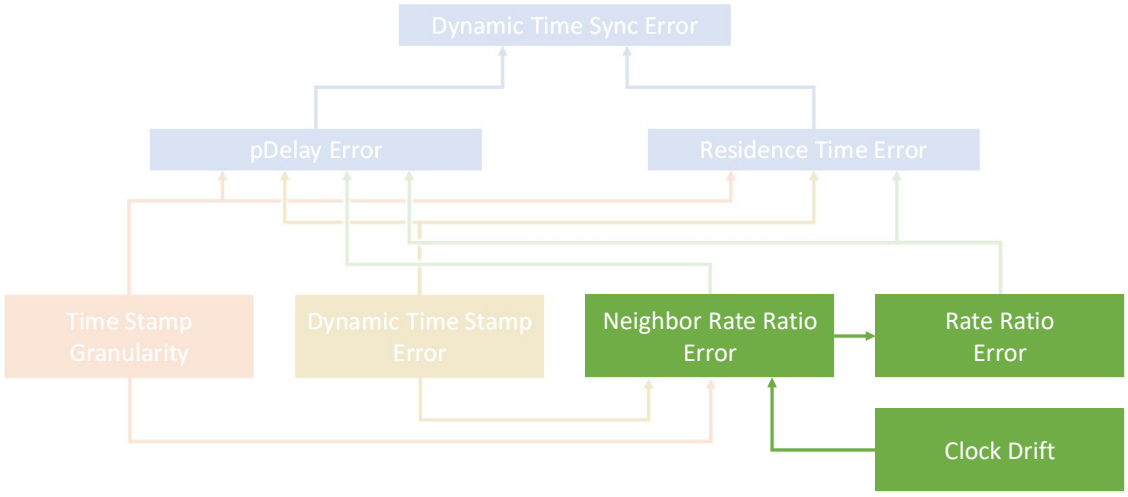
WORST CASE: $pDelay_{errorNRR}\ (ns) = 10ms * (0.6ppm)$

$= 6ns$

Conclusion: Model $pDelayError_{NNR}$

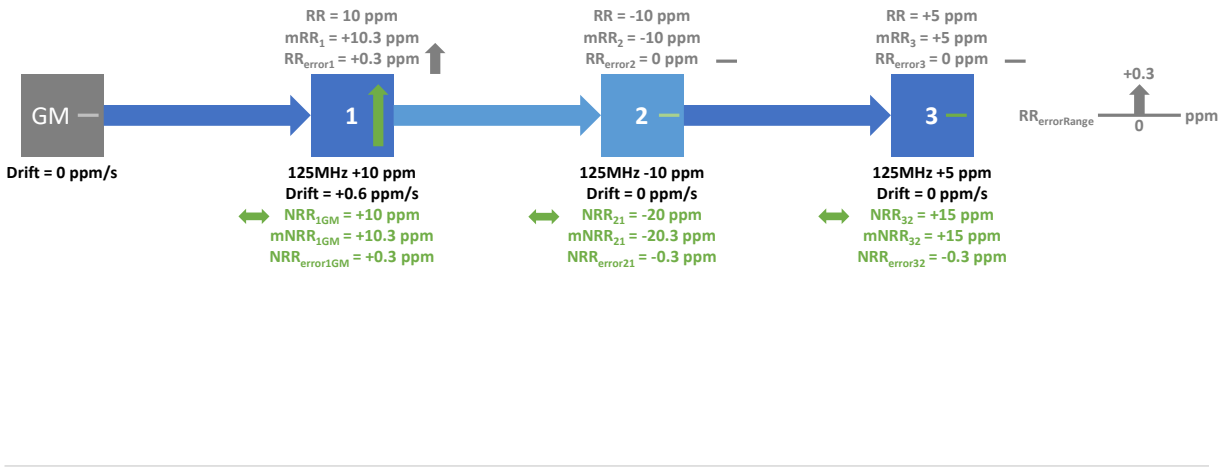
Rate Ratio Error

Time Sync – Elements & Relationships



$$RateRatio_n = RateRatio_{n-1} * NRR$$

Start with a “simple” analysis.



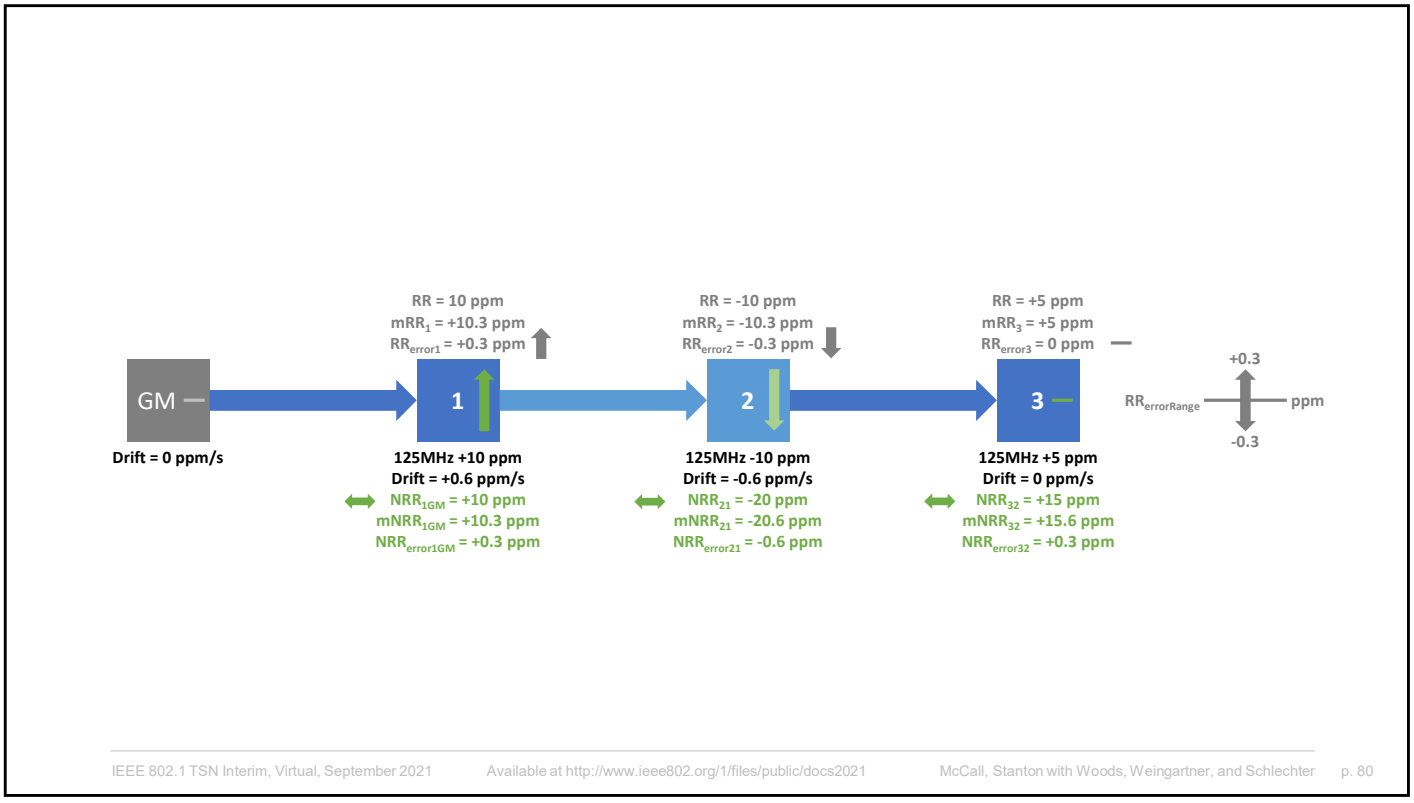
Simple in that…if you feed NRR directly into RR, and don't account for time between measuring mNRR and using mNRR, i.e. applying it to Rate Ratio.

For pDelay Interval = 1s

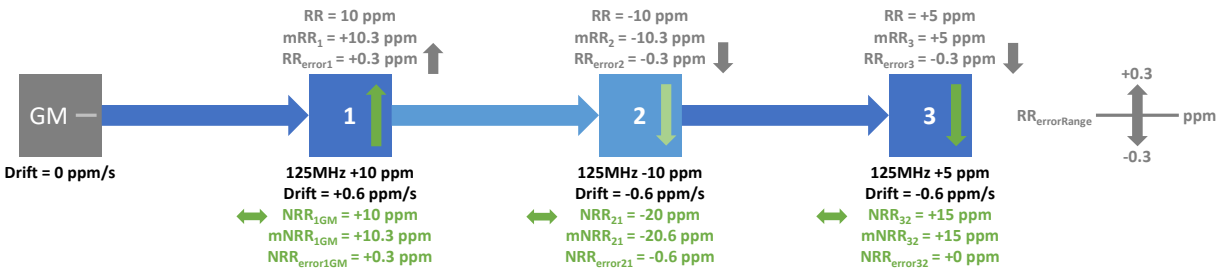
…which is the 802.1AS-2020 default value, and the value used in this presentation when looking at NRR error.

See earlier in the presentation for full analysis.

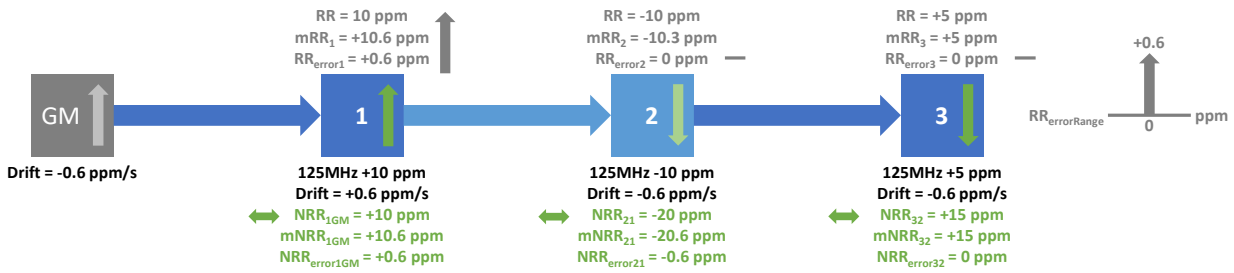
The RR_{error} at 1 is fully removed at 2. This makes sense as RR is trying to calculate the ratio of Local Clock to GM clock. If everything is working correctly, the error will depend on the drift between the two…not the drift of clocks in the transmission path.



This continues to line up as we add Clock Drift to 2…


RR = 10 ppm
mRR1 = +10.3 ppm
RRerror1 = +0.3 ppm
RR = -10 ppm
mRR2 = -10.3 ppm
RRerror2 = -0.3 ppm
RR = +5 ppm
mRR3 = +5 ppm
RRerror3 = -0.3 ppm
+0.3
RRerrorRange
ppm
-0.3
GM
1
2
3
Drift = 0 ppm/s
125MHz +10 ppm
Drift = +0.6 ppm/s
NRR1GM = +10 ppm
mNRR1GM = +10.3 ppm
NRRerror1GM = +0.3 ppm
125MHz -10 ppm
Drift = -0.6 ppm/s
NRR21 = -20 ppm
mNRR21 = -20.6 ppm
NRRerror21 = -0.6 ppm
125MHz +5 ppm
Drift = -0.6 ppm/s
NRR32 = +15 ppm
mNRR32 = +15 ppm
NRRerror32 = +0 ppm

…and 3.



Adding Clock Drift to the GM is very different because it affects the Rate Ratio Error at every single downstream device.

Worse, it pushes them all in one direction, which could have further implications…when you start adding in the next layer of Rate Ratio Error…

$$RR_{error}\ (ppm) = \frac{(driftRate_{local} - driftRate_{GM}) * pDelayInterval}{2} \quad ?$$

Which looks a lot like the NRRerror equation, but with $driftRate_{GM}$ instead of $driftRate_{n-1}$

Note: Effect of GMdrift skews RR_{error} for all downstream instances in the same direction. This will have implications for how errors accumulate.

Therefore, model GM clock separately, i.e. separate ppm/s range vs. other clocks in the sysem.

(A more accurate GM clock could address up to 50% of basic Rate Ratio related errors in the system.)

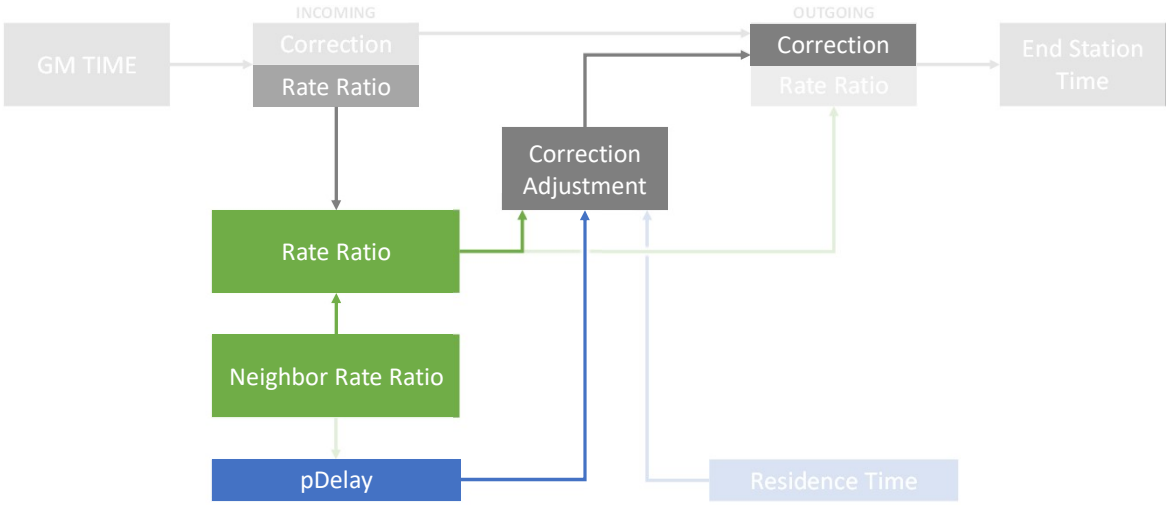
However, there is an additional source of error.

RateRatio is used during Time Sync processing.

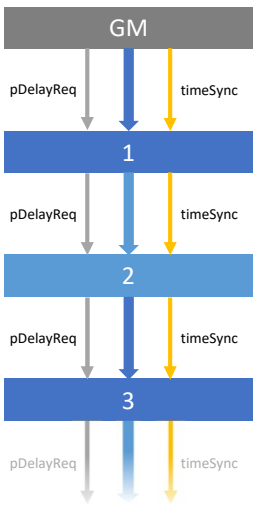
RateRatio is effectively measured at the same time as NRR. There is a time delay between measurement and use…which is different at each node.

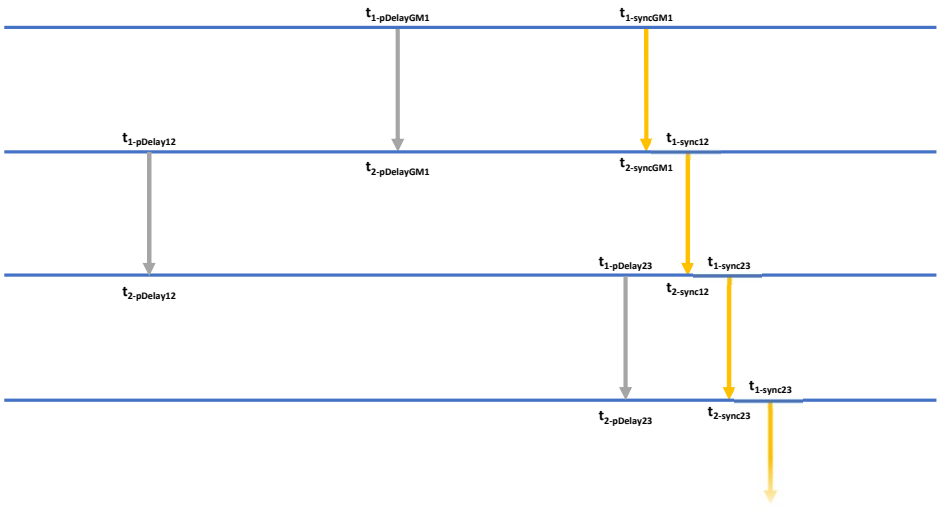
(NRR is measured as part of pDelay exchange.)

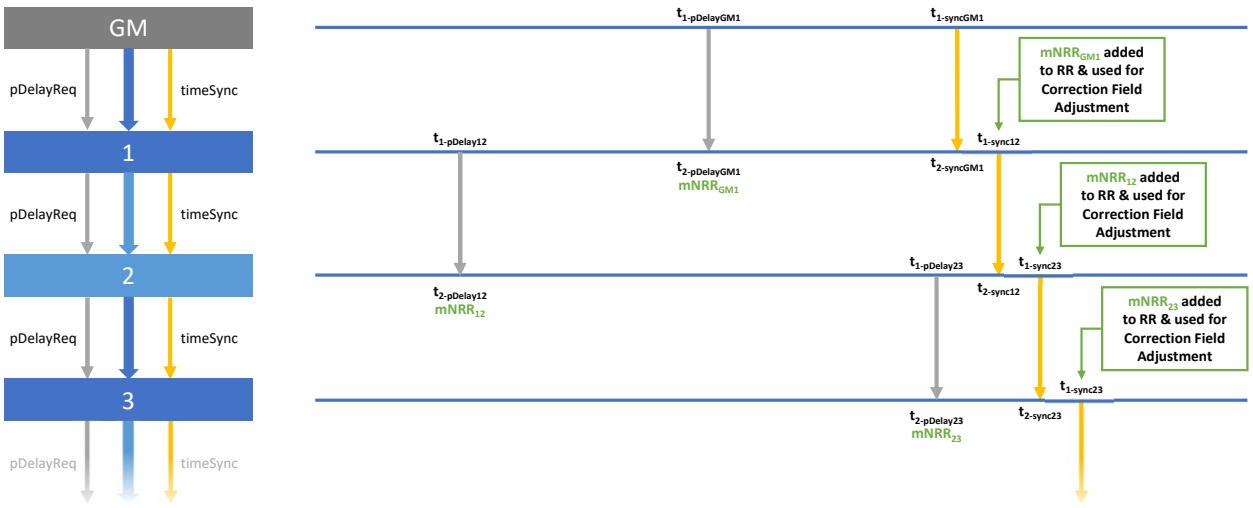
Time Sync – Elements & Relationships



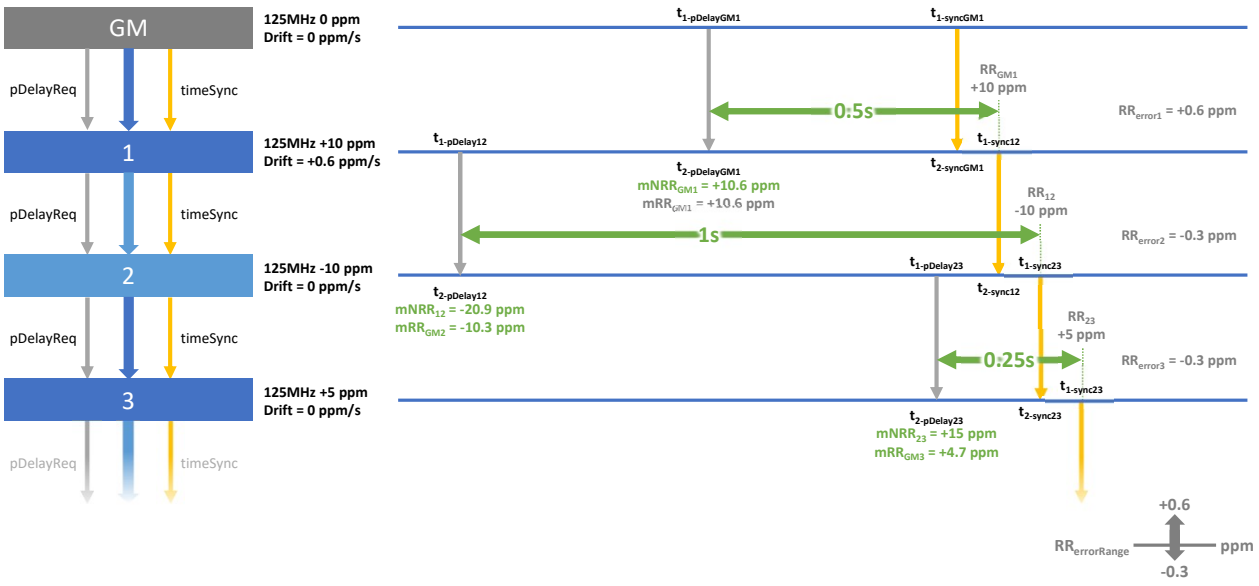

GM
1
2
3


GM
pDelayReq
timeSync
1
pDelayReq
timeSync
2
pDelayReq
timeSync
3
pDelayReq
timeSync


$t_{1\text{-pDelayGM1}}$
$t_{1\text{-syncGM1}}$
$t_{1\text{-pDelay12}}$
$t_{2\text{-pDelayGM1}}$
$t_{1\text{-sync12}}$
$t_{2\text{-syncGM1}}$
$t_{2\text{-pDelay12}}$
$t_{1\text{-pDelay23}}$
$t_{1\text{-sync23}}$
$t_{2\text{-sync12}}$
$t_{1\text{-sync23}}$
$t_{2\text{-pDelay23}}$
$t_{2\text{-sync23}}$

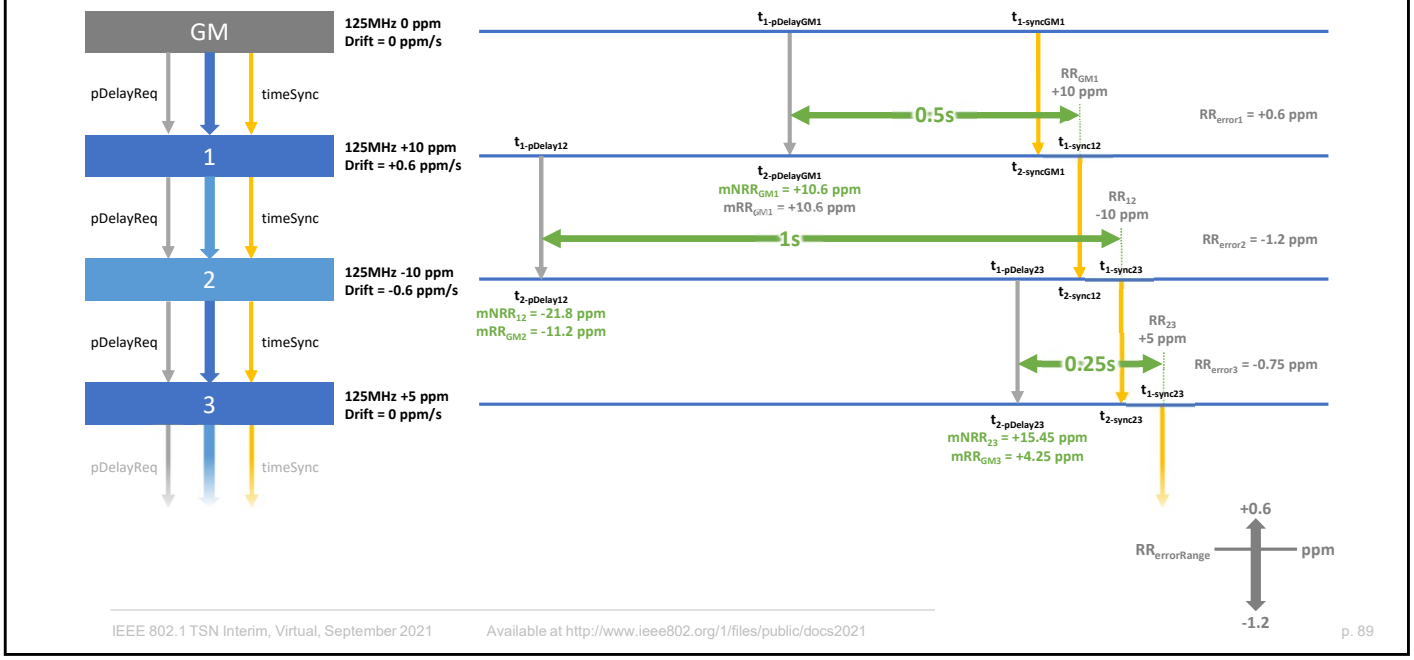


NRR is measured at last pDelay message (mNRR)…but added to Rate Ratio at end of Residence Time (Time Sync message TX). This means that the amount of adjustment to Rate Ratio (i.e. NRR) is effectively measured at mNRR.



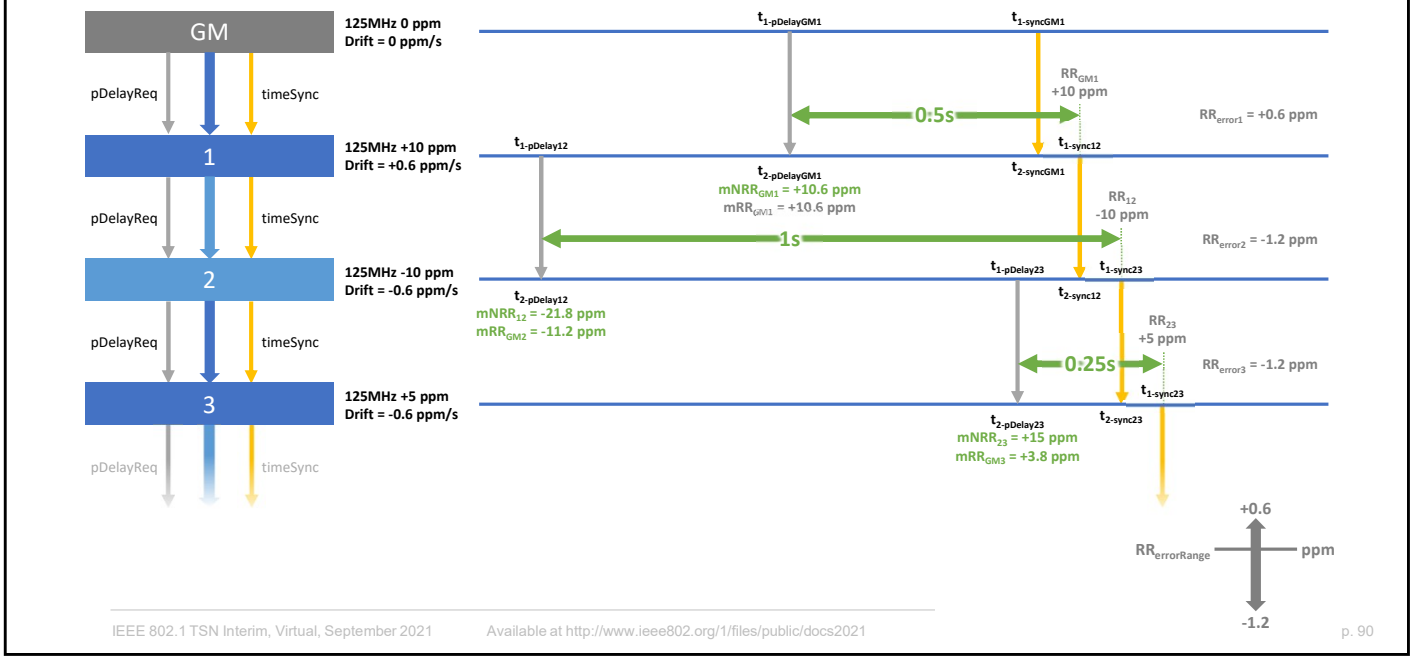
The basic mNRR errors cancel out, because pDelay interval is constant (or close enough; if you want to be really accurate, you'd need to model variation in pDelay interval, but for the purpose of this error model, we'll assume there is no variation).

But the errors due to drift can remain in the system. This is the same inputs as for the simple analysis…but with added delay. In the simple analysis the



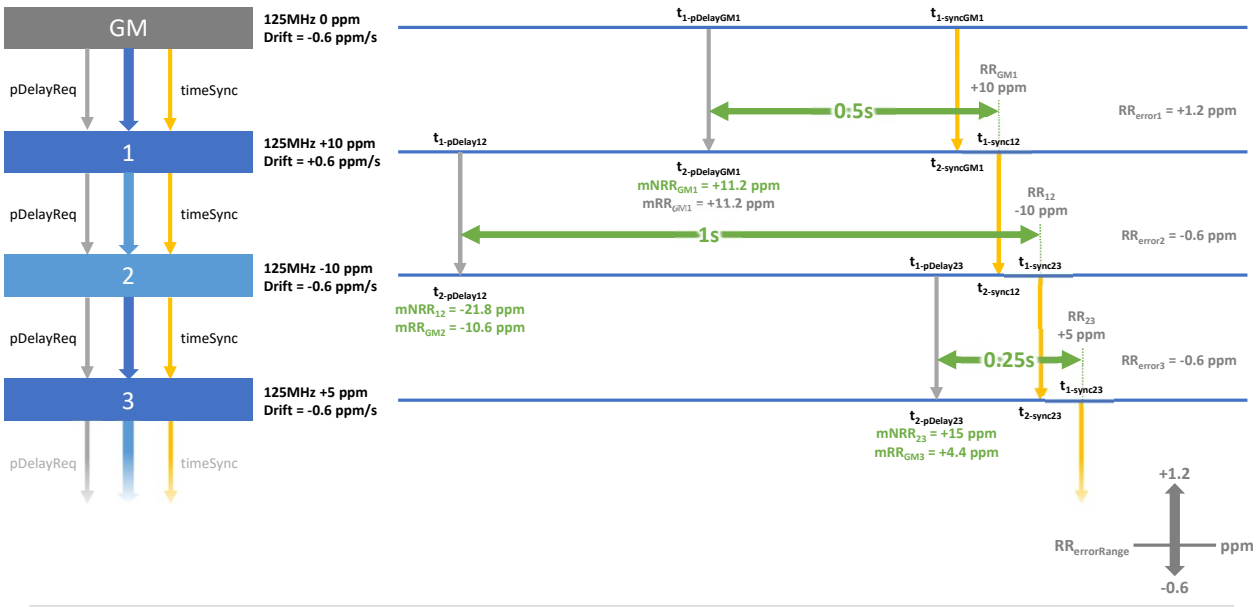
The basic mNRR errors cancel out, because pDelay interval is constant (or close enough; if you want to be really accurate, you'd need to model variation in pDelay interval, but for the purpose of this error model, we'll assume there is no variation).

But the errors due to drift can remain in the system. This is the same inputs as for the simple analysis…but with added delay. In the simple analysis the



The basic mNRR errors cancel out, because pDelay interval is constant (or close enough; if you want to be really accurate, you'd need to model variation in pDelay interval, but for the purpose of this error model, we'll assume there is no variation).

But the errors due to drift can remain in the system. This is the same inputs as for the simple analysis…but with added delay. In the simple analysis the



The basic mNRR errors cancel out, because pDelay interval is constant (or close enough; if you want to be really accurate, you'd need to model variation in pDelay interval, but for the purpose of this error model, we'll assume there is no variation).

But the errors due to drift can remain in the system. This is the same inputs as for the simple analysis…but with added delay. In the simple analysis the

Observations

- Effect of RR_{error} can be significant
  - See section on effect of RR_{error} Residence Time Error for details
- RR_{error} due to $mNRR_{error}$ cancels out at the next node
- RR_{error} due to Clock Drift does not necessarily cancel out.
  - Can accumulate to some degree. Should average to zero, unless…
- GM clock drift affects every device in the chain, and in the same direction, i.e. affects the average
- Less time between pDelay message (used to measure NRR) and TimeSync is better
  - Lower pDelay Interval is one method
    - If pDelay Interval is < Sync Interval many pDelay messages are “wasted”
  - Setting pDelay to a multiple of Sync Interval and synchronising to send pDelay just before Sync would be more efficient
- Anything that can be done to compensate for NRR_{error} will flow through to RR_{error}

Modelling Rate Ratio Error

- Model two aspects…
  - Error from $mNNR_{error}$
    - Errors out along the chain, node-to-node-to-node.
    - $RR_{errorNRR}$ never deviates by more than $mNRR_{errorRange}$
    - See earlier for equation
  - Error from drift between time of mNNR measurement and Time Sync
    - Should average out eventually…but with potentially large variances as the errors won't cancel out node-to-node-to-node.
    - RRerrorDrift may vary by a lot more than mNRRerrorRange
      - Need more work to model and quantify this.

Rate Ratio Error Forumla

$$RR_{error}\ (ppm) = RR_{errorNRR} + RR_{errorDrift}$$

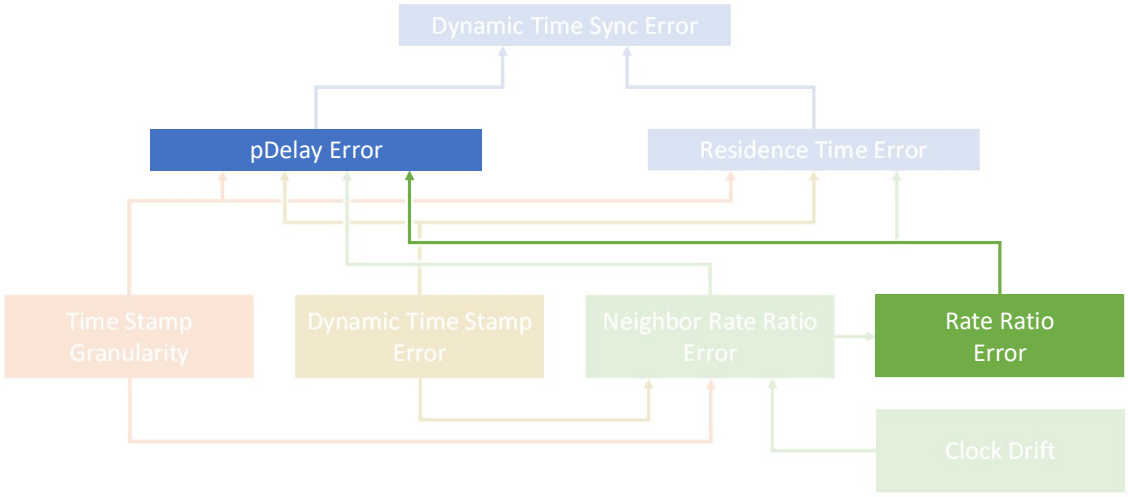
$$RR_{errorNRR}\ (ppm) = \frac{(driftRate_{local} - driftRate_{GM}) * pDelayInterval}{2}$$

$$RR_{errorDriftn}\ (ppm) = RR_{errorDriftn-1} + \frac{(driftRate_{local} - driftRate_{GM}) * Time\ since\ mNRR}{2}$$

Model Time since mNRR as random with uniform distribution between 0 and pDelay Interval.

pDelay –Rate Ratio Error

Time Sync – Elements & Relationships



$pDelayError_{RR}$ (ns) = pDelay * RR_{Error}

WORST CASE: RR_{Error} (ppm) = ±10

$pDelayError_{RR}$ (ns) = 150ns * (±10 ppm)

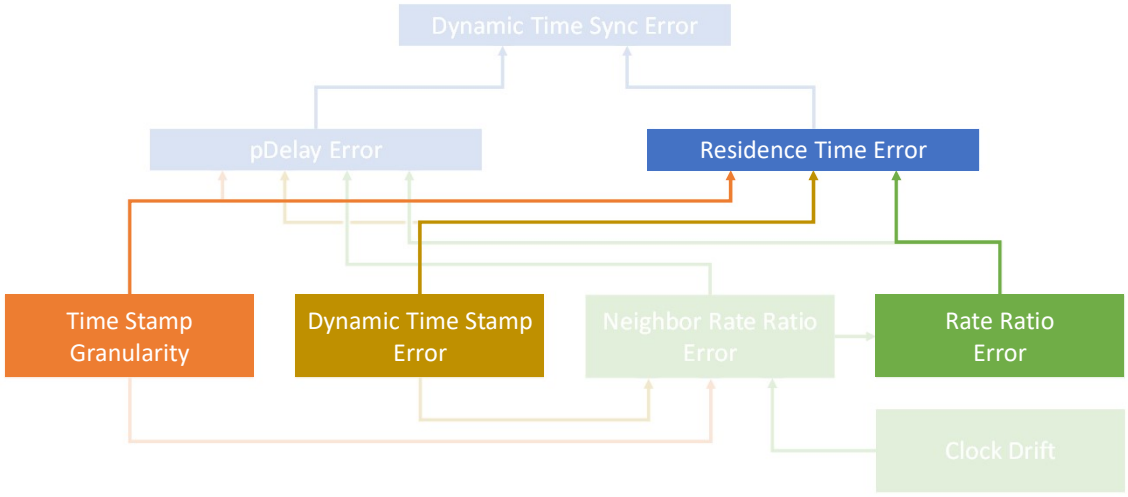
= 0.0015ns (i.e. 1.5ps)

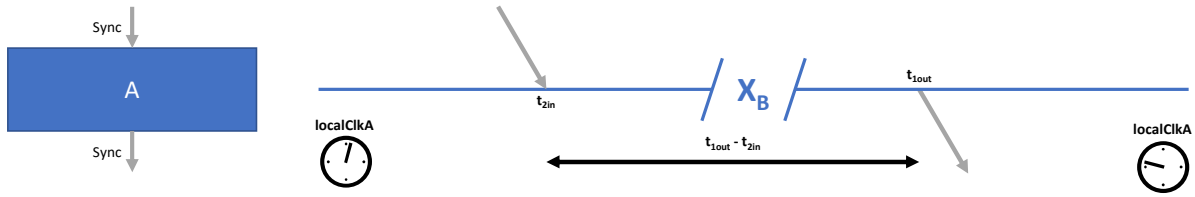
150ns pDelay → 45m

Conclusion: Do Not Model $pDelayError_{RR}$

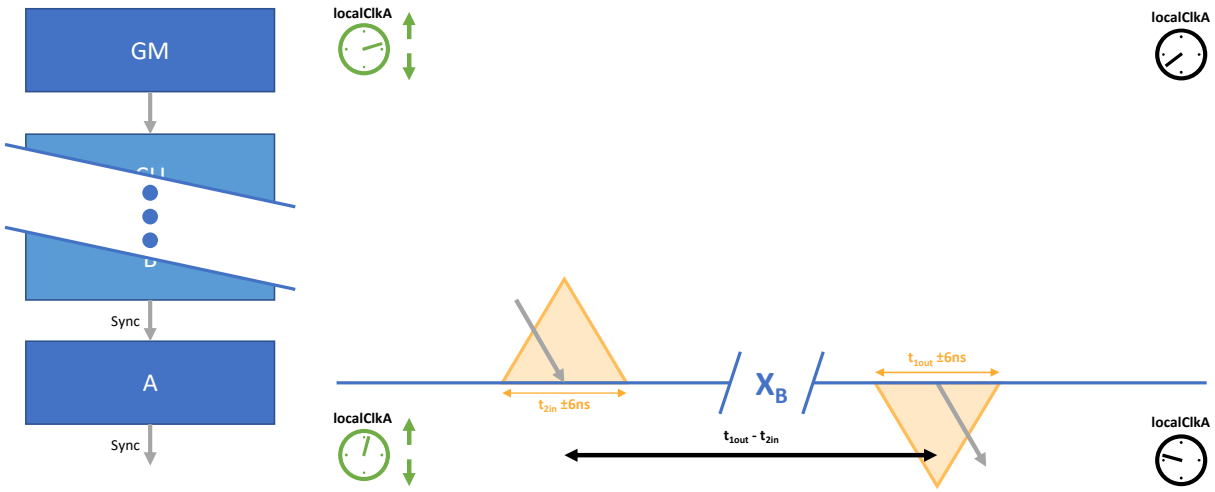
Residence Time Error

Time Sync – Elements & Relationships





residenceTime = RateRatio (t_{1out} - t_{2in})



$$residenceTime = RateRatio\ (t_{1out} - t_{2in})$$

$$residenceTime_{error} = t_{2inerror} + t_{1outerror} + RR_{error}\ (t_{1out} - t_{2in})$$

Conclusion: Model $residenceTime_{errort2in}$ and $residenceTime_{errort2out}$

$residenceTimeError_{RR}$ (ns) = $(t_{1out} - t_{2in}) * RR_{Error}$

WORST CASE: RR_{Error} (ppm) = ± 10

$residenceTimeError_{RR}$ (ns) = $(t_{1out} - t_{2in}) * RR_{Error}$

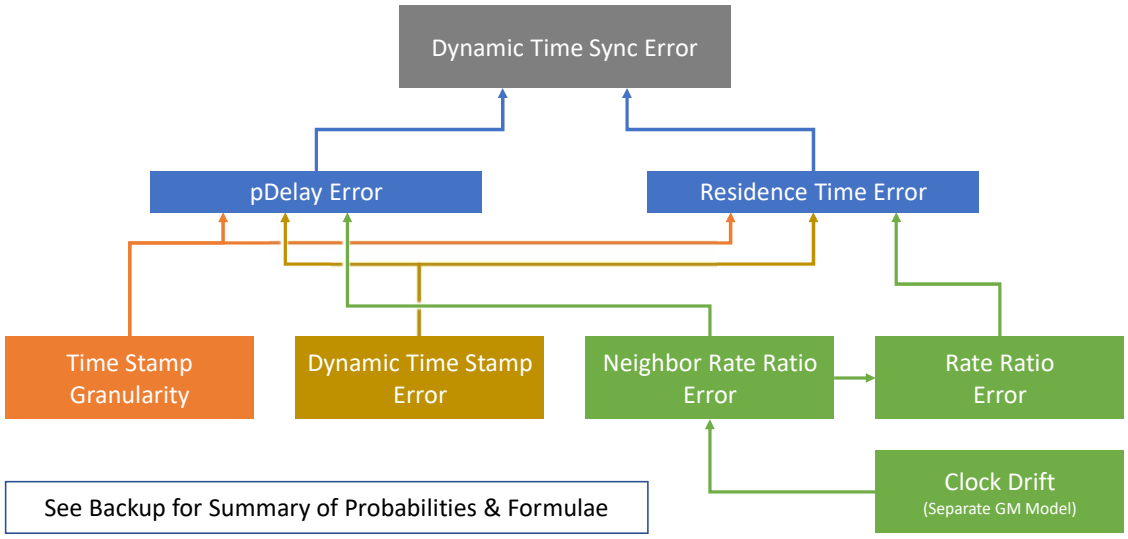
= (10ms) * 10ppm

= 100ns

150ns pDelay → 45m

Conclusion: Model $residenceTimeError_{RR}$

Time Sync – Errors to Model



Summary of Main Insights

- pDelay should be averaged.
  - Should nearly eliminate $pDelay_{TSerror}$
- NRR should not be averaged.
  - Provided (Timestamp Errors / pDelay Interval) is negligible in terms of ppm
- There are significant benefits to the GM having greater ppm stability (lower ppm/s).
- It is hard (impossible?) to eliminate Timestamp granularity and Dynamic Timestamp error effects from Residence Time
  - Need more work to see how far these elements can realistically be reduced
- Synchronising pDelayReq to arrive just before Sync would minimise the effect of Clock Drift.
- There may be opportunities to compensate for NRR_{error}
  - And therefore RR_{error}

Recommended Next Steps

- **Complete the Model**
- **Validate the Model using existing simulation(s) from Geoff**
- **Report back with a specific proposal containing 802.1AS parameters (e.g. rates etc.) and other recommendations that would meet the time accuracy goals for JP 60802 review**

Backup

Input Errors

Error	Distribution	Default Min	Default Max	Unit
$clockDrift_{GM}$	X% Stable (zero) (X-1)% Uniform	-0.6	+0.6	ppm/s
clockDrift	X% Stable (zero) (X-1)% Uniform	-0.6	+0.6	ppm/s
$TSGE_{TX}$	Uniform	-4	+4	ns
$TSGE_{RX}$	Uniform	-4	+4	ns
$DTSE_{TX}$	Uniform	-2	+2	ns
$DTSE_{RX}$	Uniform	-1	+1	ns

Other Variable Inputs

Error	Default Value	Unit
pDelay	150	ns
pDelayInterval	1	s
pDelayResponse	10	ms
residenceTime	10	ms

Formulae

$$DTE\ (ns) = pDelay_{error\_acc} + ResidenceTime_{error\_acc}$$

$$pDelay_{error\_acc} = pDelay_{errorTS\_acc} + pDelay_{errorNRR\_acc}$$

$$ResidenceTime_{error\_acc} = ResidenceTime_{errorTS\_acc} + ResidenceTime_{errorRR\_acc}$$

Rest of Formulae

Formal definition will be in subsequent submission.