
[Oct. 2022]

ETHERNOVIA

# Alternate Sync/pDelay Processing

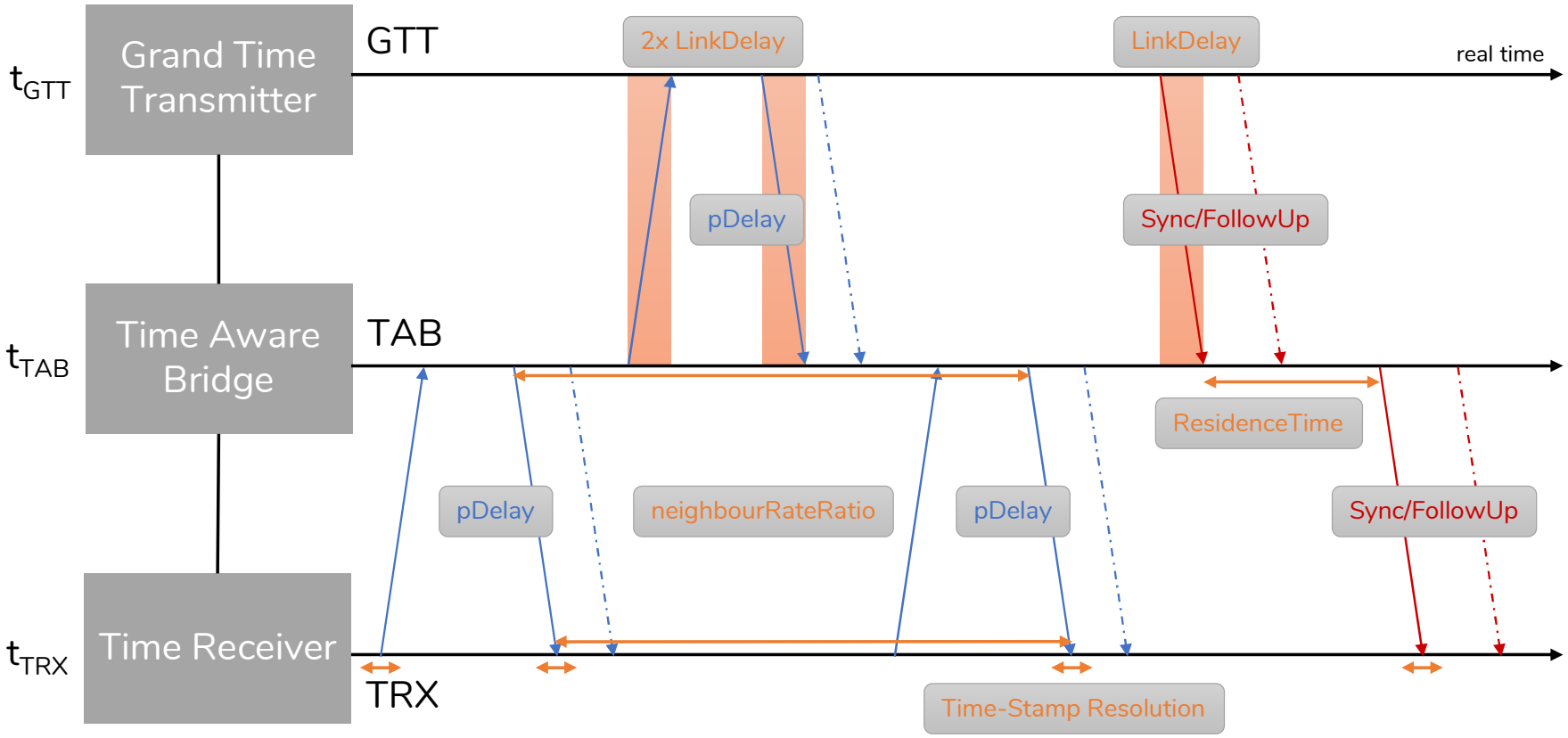
IEEE Contribution

VIRTUALIZING VEHICLE COMMUNICATION

# Setting the stage

- This presentation collects thoughts and ideas that were inspired by discussions mainly in IEEE/IEC 60802.
- The suggestions being made are preliminary and my require refinement.
- The goal if to generate a PTP Instance behaviour which is applicable to all TSN profiles (industrial, automotive, and aerospace), but is extensible to address the differences.

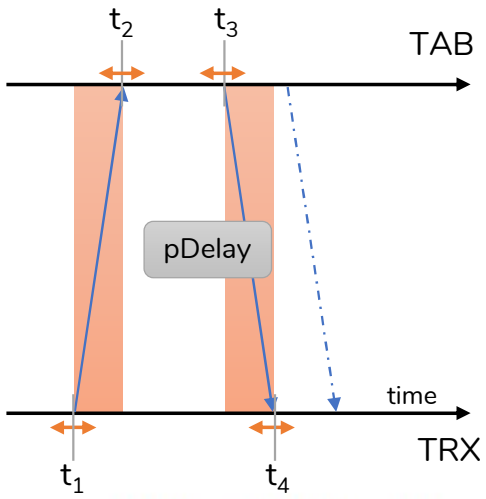
# Two Hop Example






ETHERNOVIA
VIRTUALIZING VEHICLE COMMUNICATION

# Nomenclature and abbreviations

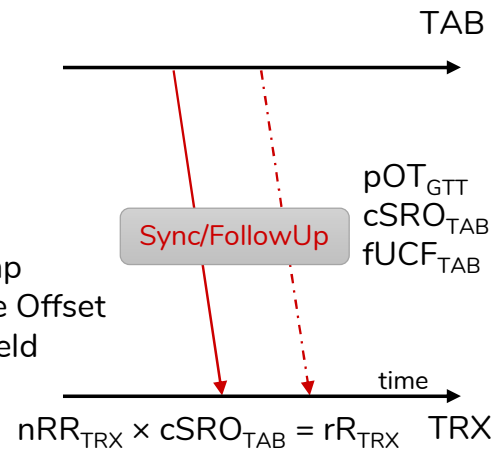



$$D = \frac{r \cdot (t_4 - t_1) - (t_3 - t_2)}{2}$$

neighbourRateRatio: nRR = $r_{TAB}$ / $r_{TRX}$ = r

GTT … Grand Time Transmitter
TAB … Time Aware Bridge
TRX … Time Receiver

carried in Sync/FollowUp:
pOT … precise Origin Timestamp
cSRO … cumulative Scaled Rate Offset
fUCF … follow Up Correction Field



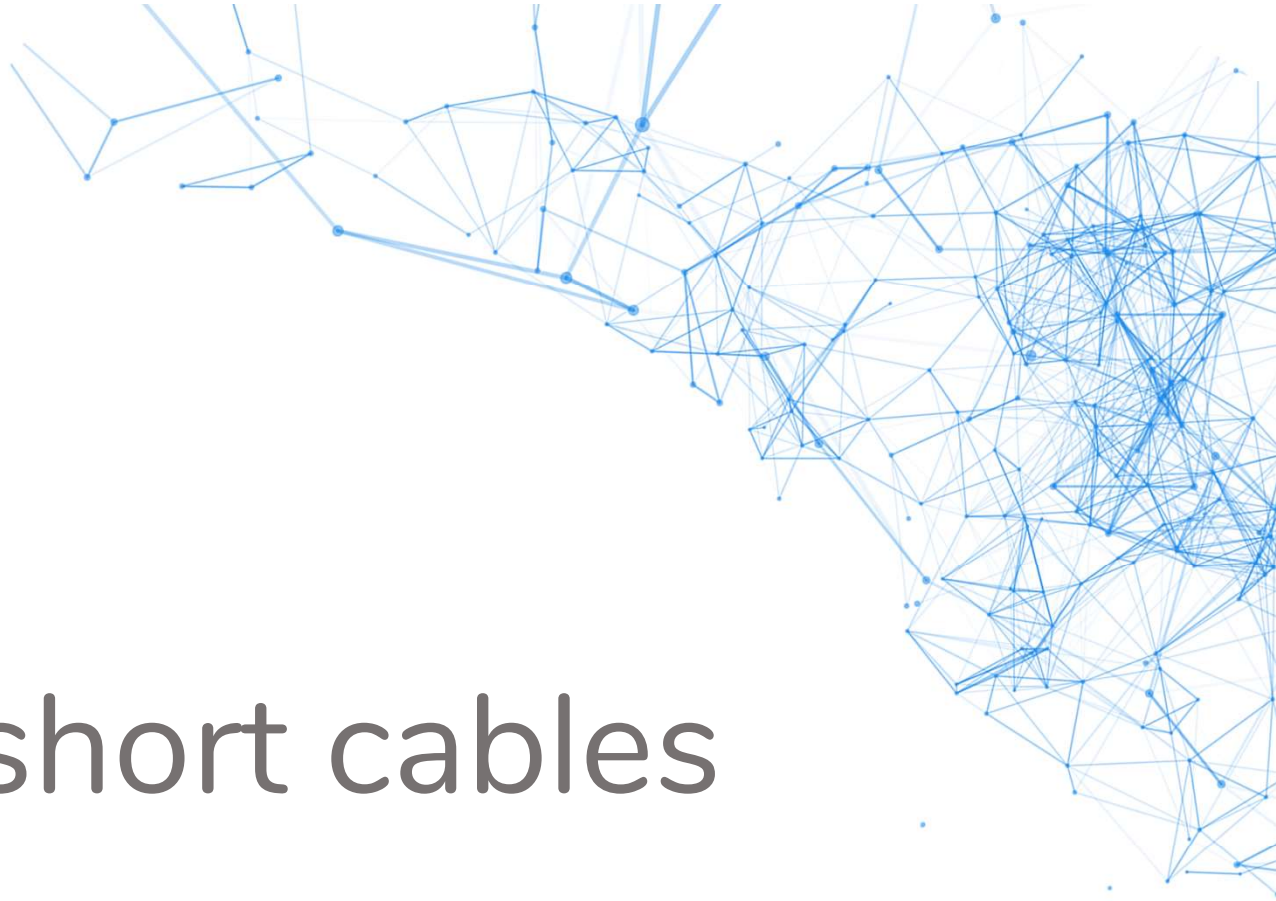

**10.2.5.7 neighborRateRatio:** The measured ratio of the frequency of the LocalClock entity of the time-aware system at the other end of the link attached to this port, to the frequency of the LocalClock entity of this time-aware system. The data type for neighborRateRatio is Float64. There is one instance of this variable for all the domains, i.e., all the PTP Instances (per port). The variable is accessible by all the domains.

**10.2.5.8 meanLinkDelay:** The measured mean propagation delay (see 8.3) on the link attached to this port, relative to the LocalClock entity of the time-aware system at the other end of the link (i.e., expressed in the time base of the time-aware system at the other end of the link). The data type for meanLinkDelay is UScaledNs. There is one instance of this variable for all the domains, i.e., all the PTP Instances (per port). The variable is accessible by all the domains.

rateRatio:
$cSRO_{TAB} = rR_{TAB} = r_{GTT} / r_{TAB}$
$rR_{TRX} = r_{GTT} / r_{TRX}$



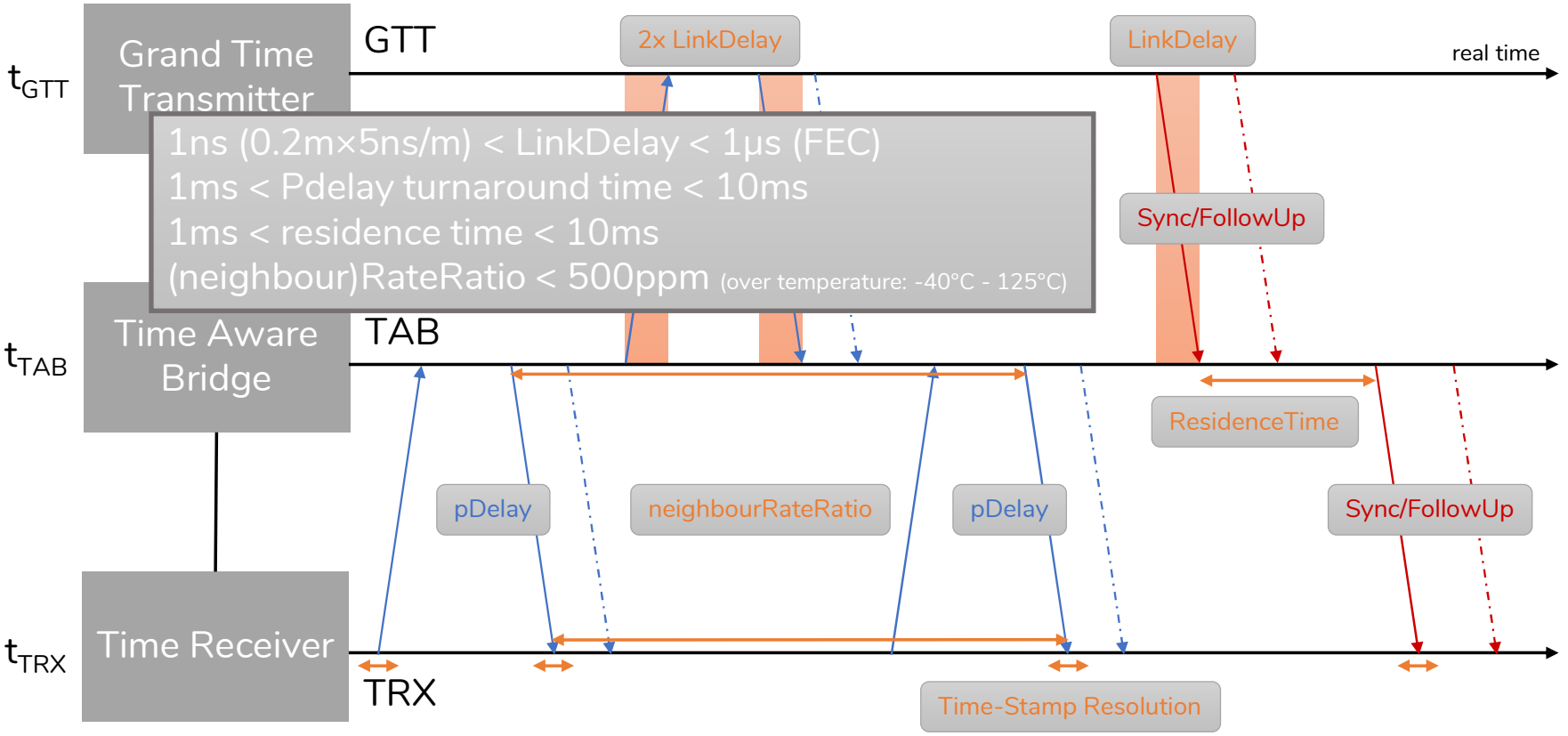
ETHERNOVIA
VIRTUALIZING VEHICLE COMMUNICATION



# Account for short cables

## automotive requirement

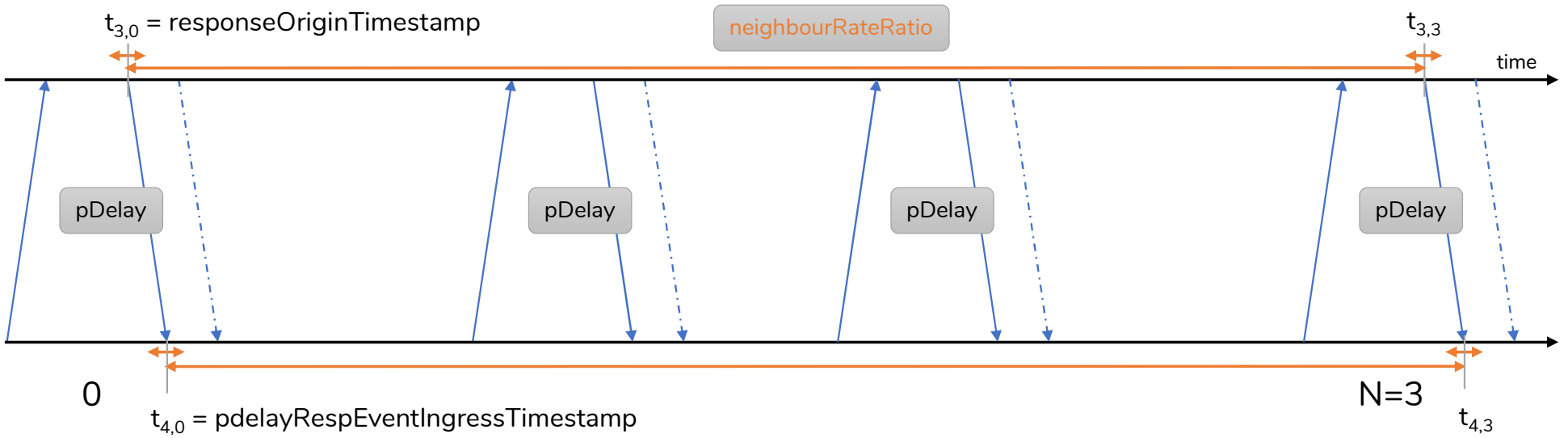
# Some basic value ranges






ETHERNOVIA
VIRTUALIZING VEHICLE COMMUNICATION

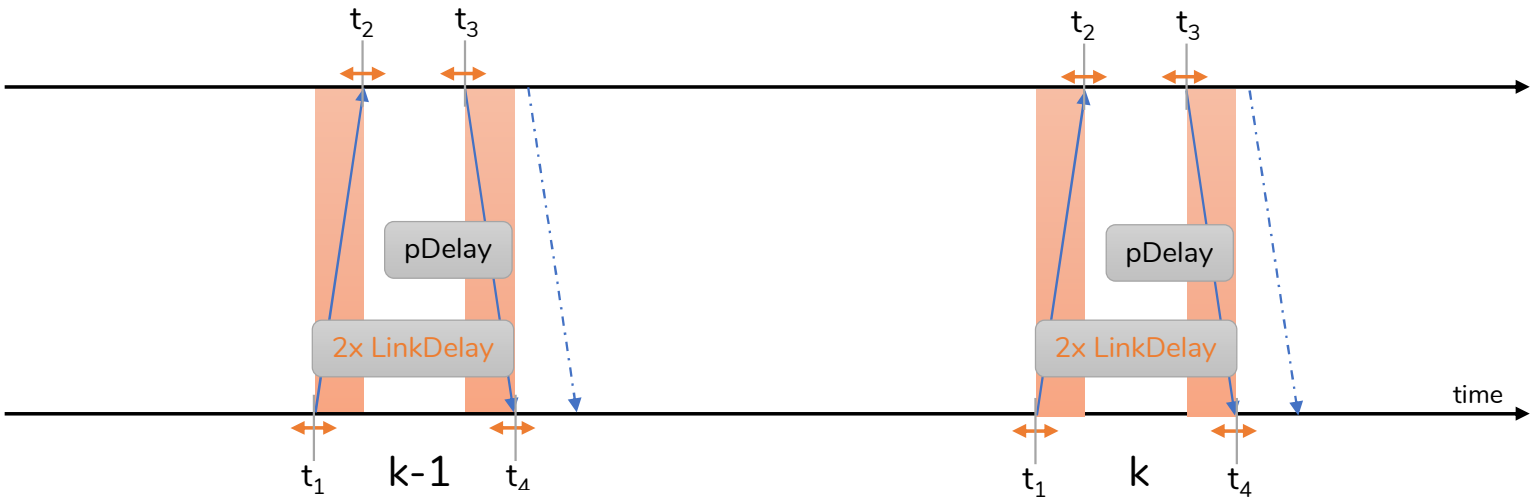
# Longer Interval for neighbourRateRatio




$$nRR = \frac{correctedResponderEventTimestamp_N - correctedResponderEventTimestamp_0}{pdelayRespEventIngressTimestamp_N - pdelayRespEventIngressTimestamp_0}$$

all in local time!

# Averaging propagationDelay Measurements




$$t_{ir} = t_2 - t_1$$

$$t_{ri} = t_4 - t_3$$

$$D = \frac{t_{ir} + t_{ri}}{2} = \frac{(t_4 - t_1) - (t_3 - t_2)}{2} \quad (11\text{-}1)$$

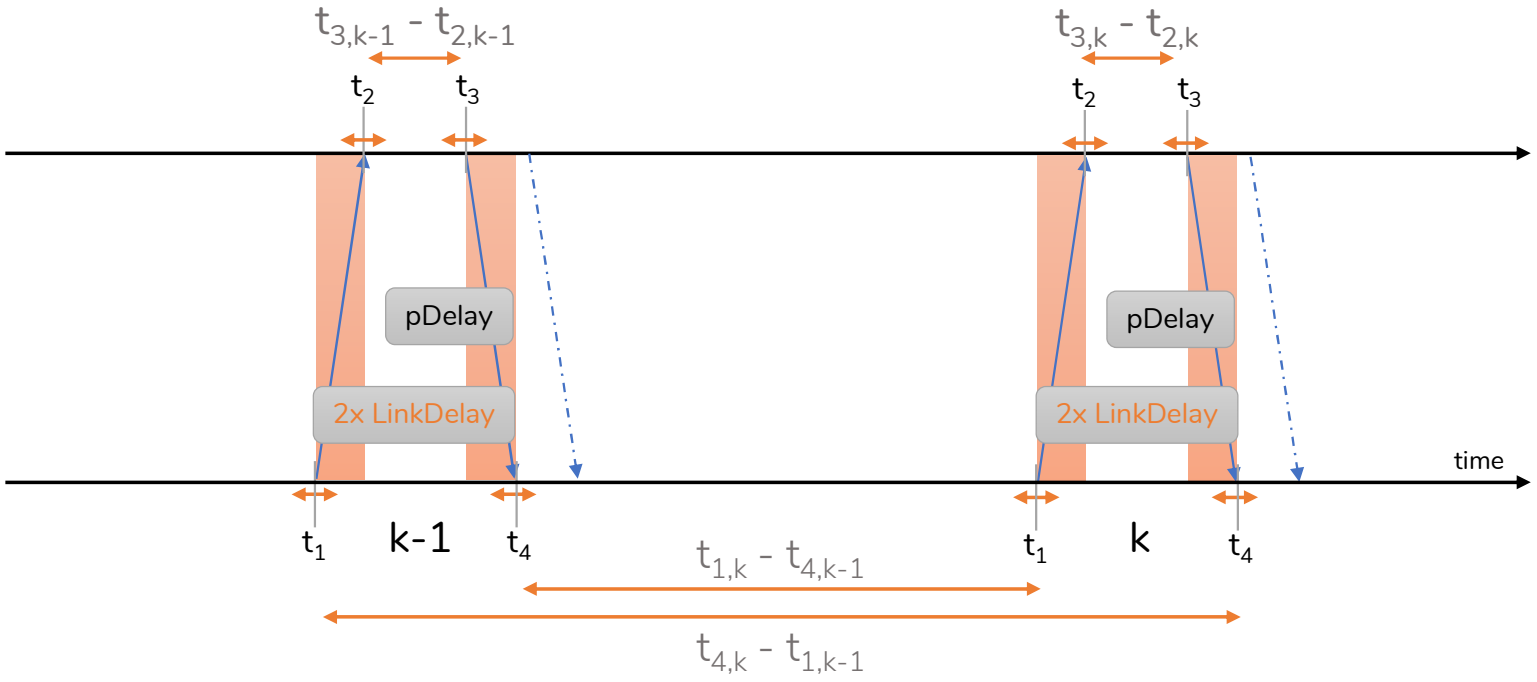
$$D = \frac{r \cdot (t_4 - t_1) - (t_3 - t_2)}{2} \quad (11\text{-}5)$$

$$D_{avg,k} = aD_{avg,k-1} + (1-a)D_{k-1} \quad (11\text{-}2)$$

$$D_{avg,k} = \frac{(k-1)D_{avg,k-1} + D_{k-1}}{k} \quad (11\text{-}4)$$

# Longer Interval for propagationDelay

$$t_{4,k} - t_{1,k-1} = 4 \times \text{propagationDelay} + (t_{3,k-1} - t_{2,k-1}) + (t_{1,k} - t_{4,k-1}) + (t_{3,k} - t_{2,k})$$




currently not foreseen in IEEE Std 802.1AS

# Different approaches to averaging

- Averaging over a longer interval is not the same as averaging multiple values measured over short intervals.
- If D happens to be measured as less than zero (D < 0) due to quantisation in the time stamping resolution
  - signal propagation speed: 5ns/m
  - in-vehicle data-line length: <20m
  - propagationDelay < 100ns ≈ 2 x Time-Stamp Resolution (40ns)
- there is no benefit in averaging those $D_k$ values.
- For time stamping in the MAC, propagationDelay will be dominated by FEC, not cable length for higher line rates.
- There is a minimum measurable nRR, which can be determined from the time stamp resolution (TSR) and the average time between pDelay exchanges (pDelayInterval):

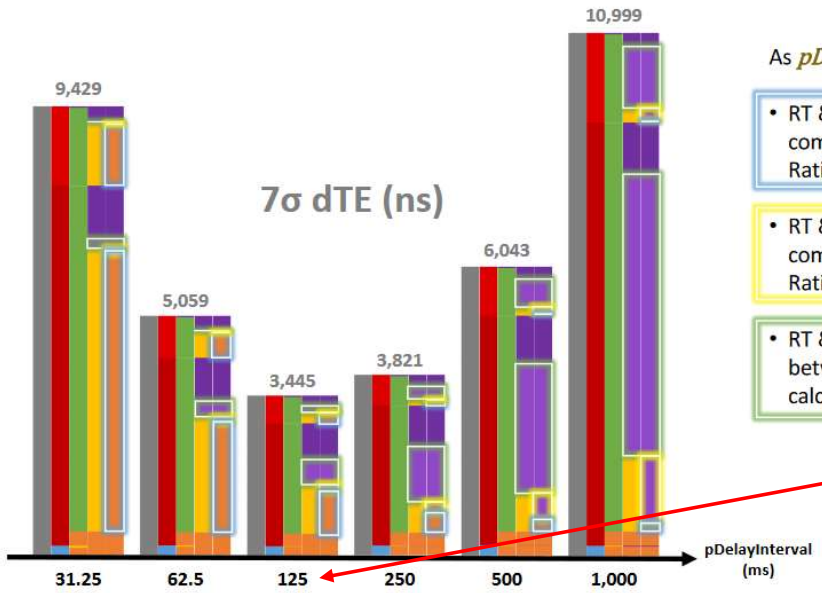
$$\frac{31.25\text{ms}+2\text{x}40\text{ns}}{31.25\text{ms}+2\text{x}40\text{ns}} = 1 + \mathbf{2{,}56ppm}$$

$$\approx 1 + (4 \text{ x } TSR / pDelayInterval) \leq \mathbf{neighbourRateRatio}$$

# Optimum pDealyinterval

## *pDelayInterval* Sensitivity Analysis

| Input Errors | | |
|---|---|---|
| Drift Type (Linear Temp Ramp) | 2 | |
| GM Clock Drift Max | +1.35 | ppm/s |
| GM Clock Drift Min | -1.35 | ppm/s |
| Fraction of GM nodes w/ Drift | 80% | |
| non-GM Clock Drift Max | +1.35 | ppm/s |
| non-GM Clock Drift Min | -1.35 | ppm/s |
| Fraction of non-GM Nodes w/ Drift | 80% | |
| Temp Max | +85. | °C |
| Temp Min | -40. | °C |
| Temp Ramp Rate | ±1 | °C/s |
| Temp Ramp Period | 125 | s |
| Temp Hold Period | 30 | s |
| GM Scaling Factor | 100% | |
| non-GM Scaling Factor | 100% | |
| Timestamp Granularity TX | ±4 | ns |
| Timestamp Granularity RX | ±4 | ns |
| Dynamic Time Stamp Error TX | ±4 | ns |
| Dynamic Time Stamp Error RX | ±4 | ns |
| **Input Parameters** | | |
| pDelay Interval | VAR | ms |
| Sync Interval | 125 | ms |
| pDelay Turnaround Time | 10 | ms |
| residenceTime | 10 | ms |
| **Input Correction Factors** | | |
| Mean Link Delay Averaging | 0% | |
| NRR Drift Rate Correction | 0% | |
| RR Drift Rate Error Correction | 0% | |
| pDelayResp → Sync Type (Uniform) | 1 | |
| pDelayResp → Sync Max | 100% | |
| pDelayResp → Sync Min | 0% | |
| pDelayResp → Sync Target | 10 | ms |
| mNRR Smoothing N | 1 | |
| mNRR Smoothing M | 1 | |
| **Configuration** | | |
| Hops | 100 | |
| Runs | 500,000 | |




As *pDelayInterval* is reduced...

- RT & ES errors due to **Timestamp** component of $mNRR_{error}$, via Rate Ratio **increase**.
- RT & ES errors due to **Clock Drift** component of $mNRR_{error}$, via Rate Ratio **decrease**.
- RT & ES errors due to **Clock Drift** between measurement of NRR and calculation of RR **decrease**.

Should be selectable based on:
- Time Stamp Accuracy
- Clock Drift

only.

IEEE 802.1 TSN / 60802 Time Sync Ad Hoc: Addressing Errors – Complexities & Tradeoffs – September 2022 David McCall p. 10

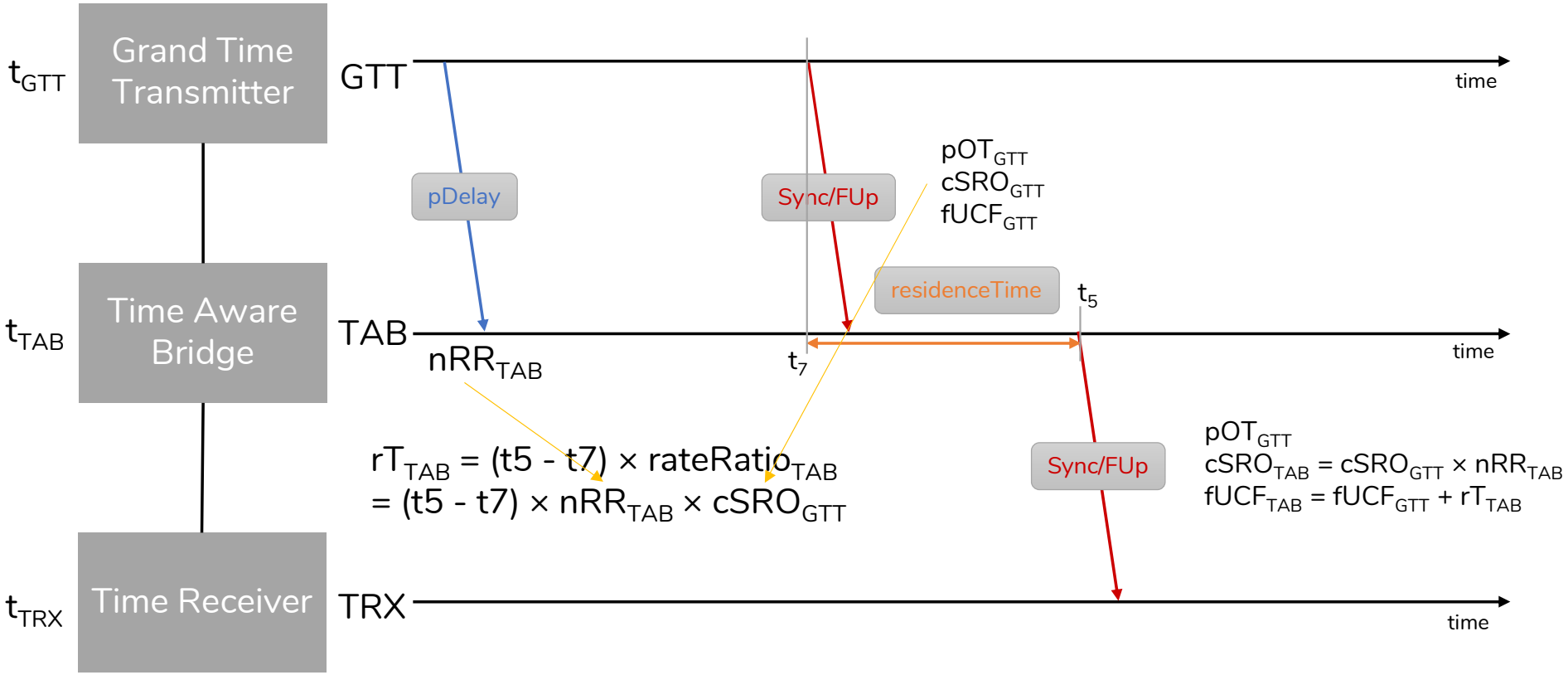


ETHERNOVIA
VIRTUALIZING VEHICLE COMMUNICATION

# What is nRR used for?

## Can we improve on this?

# Calculating residence time (local clocks)




not to scale!



ETHERNOVIA
VIRTUALIZING VEHICLE COMMUNICATION

# Synchronized Receiver Time

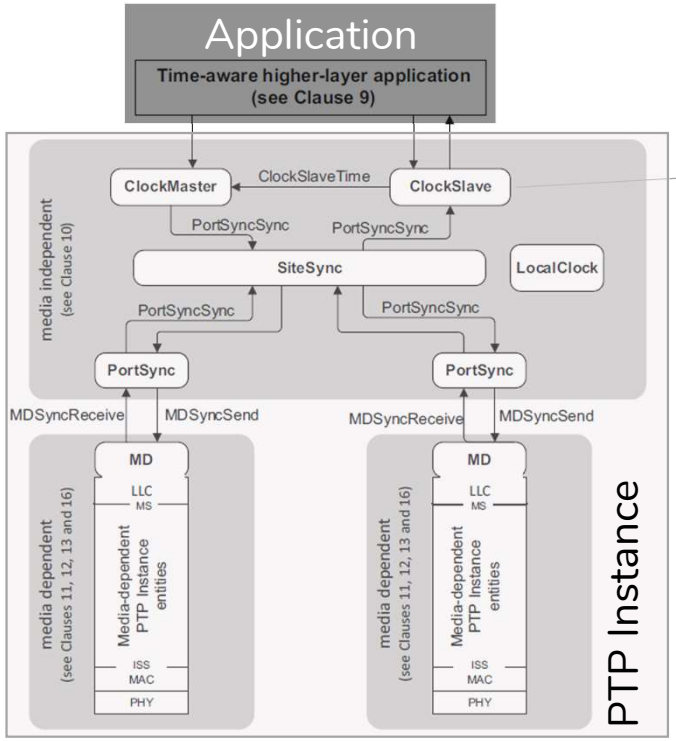



Figure 7-8—PTP Instance model

**10.2.4.3 clockSlaveTime:** The synchronized time maintained, at the slave, at the granularity of the LocalClock entity [i.e., a new value is computed every localClockTickInterval (see 10.2.4.18) by the ClockSlave entity]. The data type for clockSlaveTime is ExtendedTimestamp.

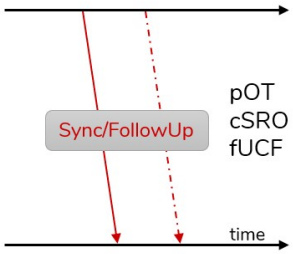
**10.2.13.2.1 updateSlaveTime():** Updates the global variable clockSlaveTime (see 10.2.4.3), based on information received from the SiteSync and LocalClock entities. It is the responsibility of the application to filter slave times appropriately (see B.3 and B.4 for examples). As one example, clockSlaveTime can be:

a) Set to syncReceiptTime at every LocalClock update immediately after a PortSyncSync structure is received, and

b) Incremented by localClockTickInterval (see 10.2.4.18) multiplied by the rateRatio member of the previously received PortSyncSync structure during all other LocalClock updates.

If no PTP Instance is grandmaster-capable, i.e., gmPresent is FALSE, then clockSlaveTime is set to the time provided by the LocalClock. This function is invoked when rcvdLocalClockTickCSS is TRUE.

$$sRT = pOT + fUCF + mLD \times (rR / nRR) + dA$$
$$rR = cSRO \times nRR$$

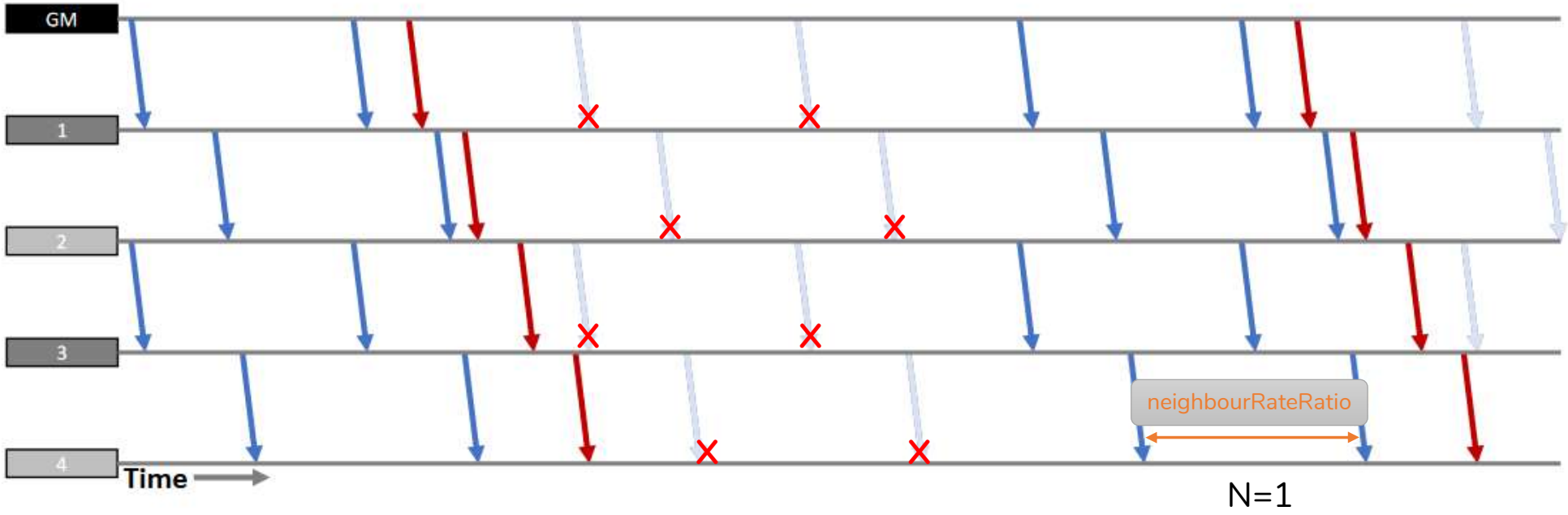





ETHERNOVIA
VIRTUALIZING VEHICLE COMMUNICATION

# pDelayInterval < Sync Interval

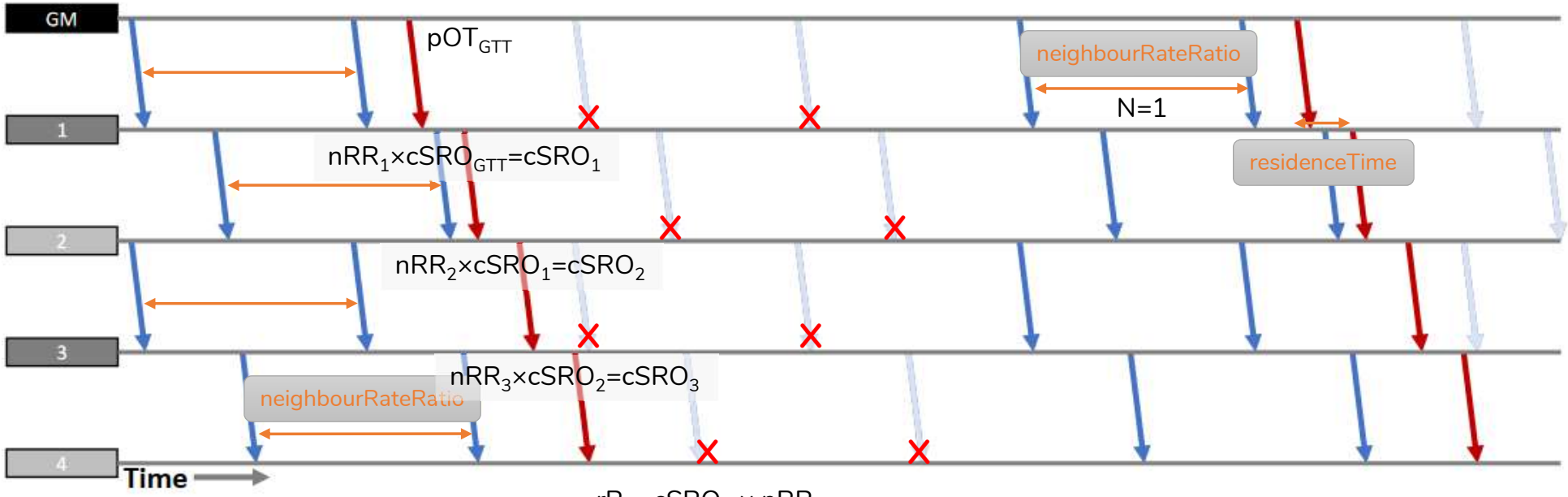
wasted information?




pDelayResp  Sync

- Error due to drift between receipt of one Sync message and the next. (**Node 4** to **GM**).
  - Interval is nominally the Sync Interval, but there is some variation.
  - Additional pDelayResp messages, updating mNRR (**Node 4** to **Node 3**) are not useful to update RR.

# When is synchronized time updated?



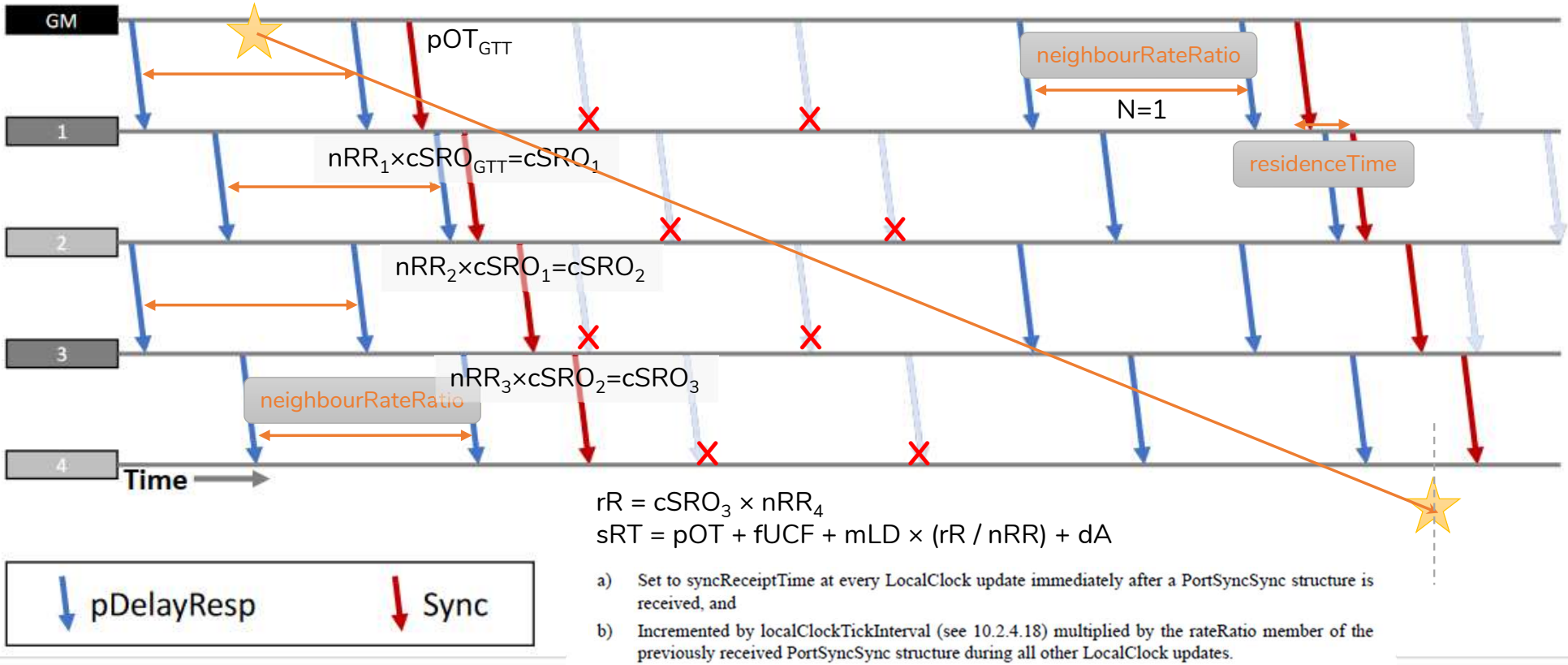

rR = cSRO$_3$ × nRR$_4$

sRT = pOT + fUCF + mLD × (rR / nRR) + dA

a) Set to syncReceiptTime at every LocalClock update immediately after a PortSyncSync structure is received, and

b) Incremented by localClockTickInterval (see 10.2.4.18) multiplied by the rateRatio member of the previously received PortSyncSync structure during all other LocalClock updates.

# The rateRatio during extrapolation is old!



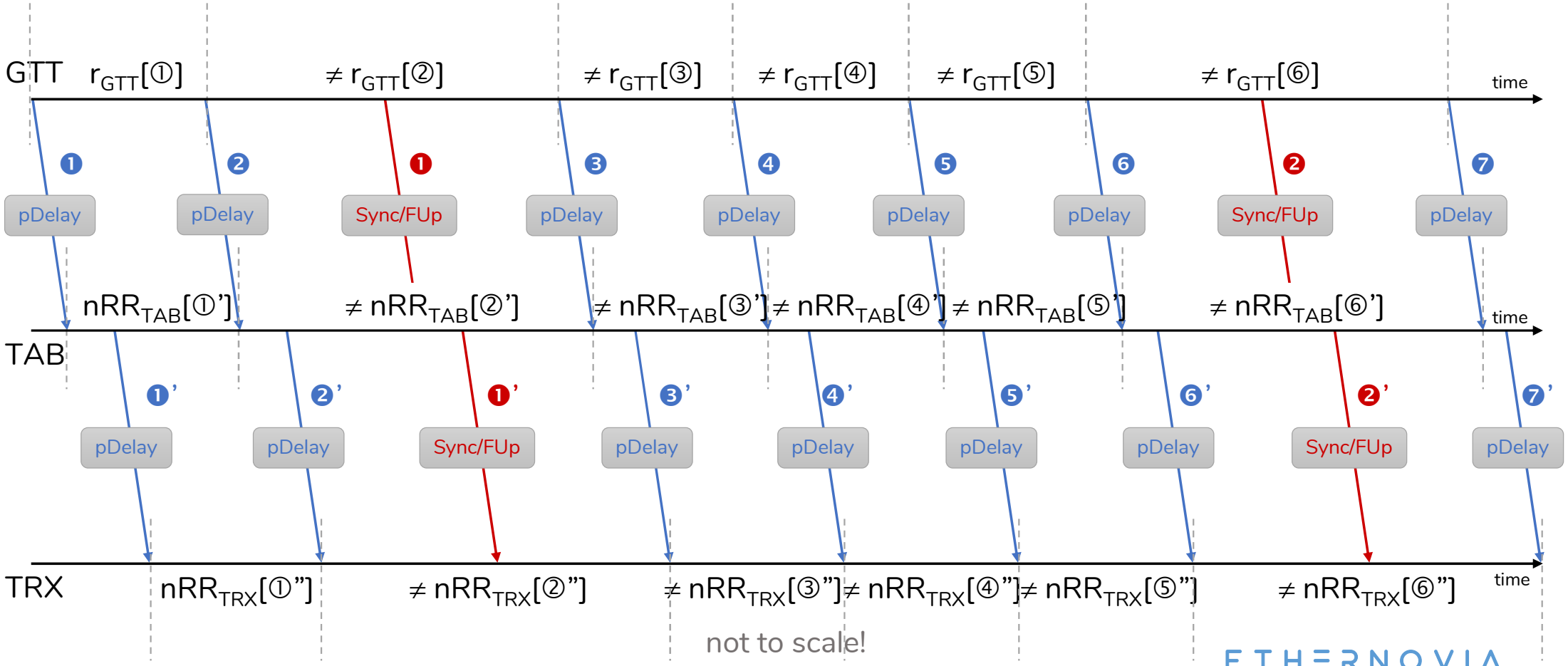

$rR = cSRO_3 \times nRR_4$

$sRT = pOT + fUCF + mLD \times (rR / nRR) + dA$

a) Set to syncReceiptTime at every LocalClock update immediately after a PortSyncSync structure is received, and

b) Incremented by localClockTickInterval (see 10.2.4.18) multiplied by the rateRatio member of the previously received PortSyncSync structure during all other LocalClock updates.

# Changing Rates, assuming ARB GTT

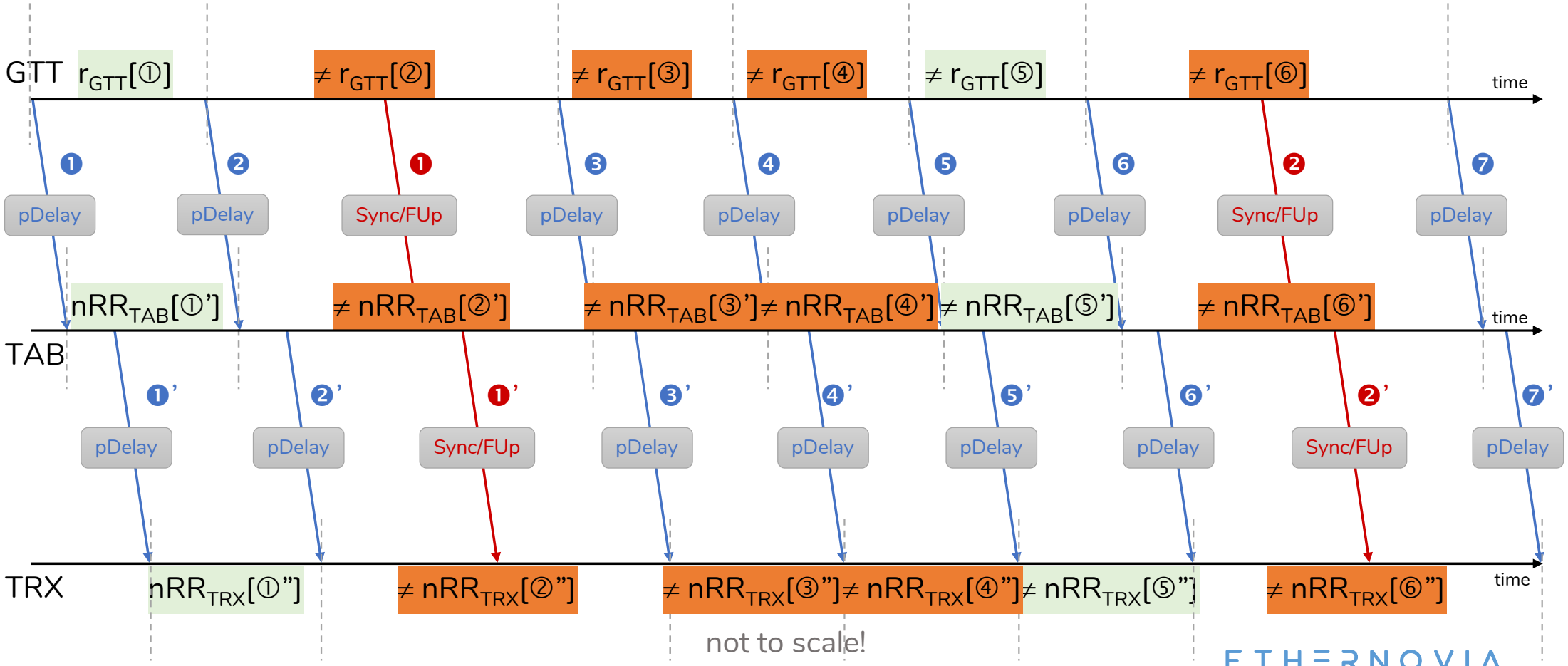



not to scale!



ETHERNOVIA
VIRTUALIZING VEHICLE COMMUNICATION

# Wasted Rate Information




not to scale!



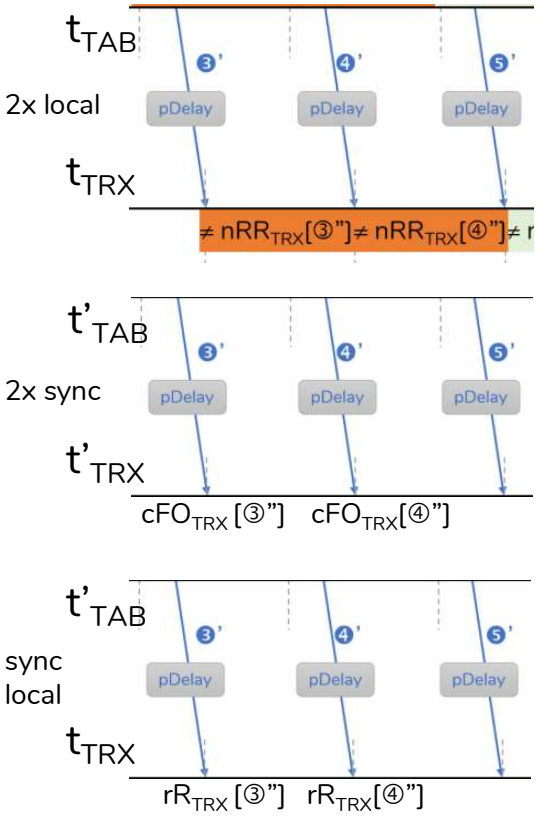
ETHERNOVIA
VIRTUALIZING VEHICLE COMMUNICATION

# How to make use of the “wasted” pDelay exchanges?

1. Measuring the neighbourRateRatio between any two nodes (where none is the GTT) more often is not useful as it does not give any indication what is happening at the GTT and in nodes further up the distribution tree.

2. Using (unfiltered) synchronizedTime on both sides, could deliver something like Frequency Offset relative to the GTT (or rather synchronized time on the transmitter node).

3. Using (unfiltered) synchronizedTime at the timeTransmitter and localClock at the timeReceiver can update rateRatio.






ETHERNOVIA
VIRTUALIZING VEHICLE COMMUNICATION

# Filtered ClockTarget Time

**10.2.13.2.1 updateSlaveTime():** Updates the global variable clockSlaveTime (see 10.2.4.3), based on information received from the SiteSync and LocalClock entities. It is the responsibility of the application to filter slave times appropriately (see B.3 and B.4 for examples). As one example, clockSlaveTime can be:

"*clockSlaveTime*" is the unfiltered synchronized time!?

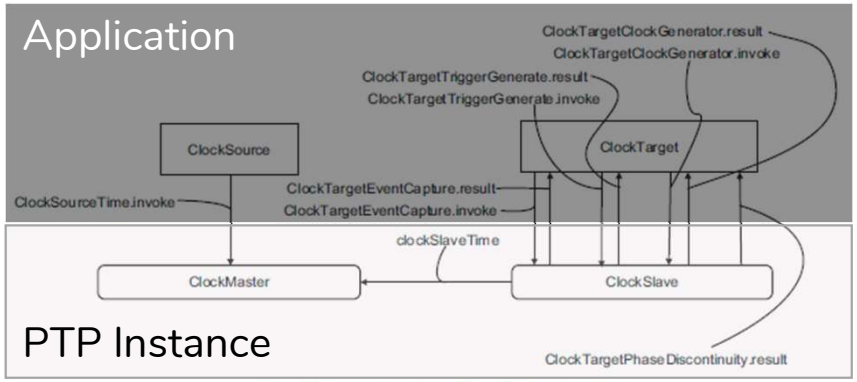



**Figure 9-1—Application interfaces**

B.4

NOTE—For example, the endpoint filter can be of the following form:

$$y_k = a_1 y_{k-1} + a_2 y_{k-2} + \ldots + a_n y_{k-n} + b_0 x_k + b_1 x_{k-1} + \ldots + b_n x_{k-n}$$

where the $x_k$ are the unfiltered synchronized time values, the $y_k$ are the filtered synchronized time values, and the $a_k$ and $b_k$ are filter coefficients. The $a_k$ and $b_k$ are chosen such that the filter has desired bandwidth and gain peaking that does not exceed 0.1 dB. The preceding equation is a general infinite impulse response (IIR) digital filter. Simplified forms, e.g., a second order IIR filter obtained by setting $n = 2$, or a finite impulse response (FIR) filter obtained by setting the $a_k$ to zero are possible.

# Synchronized Receiver Time

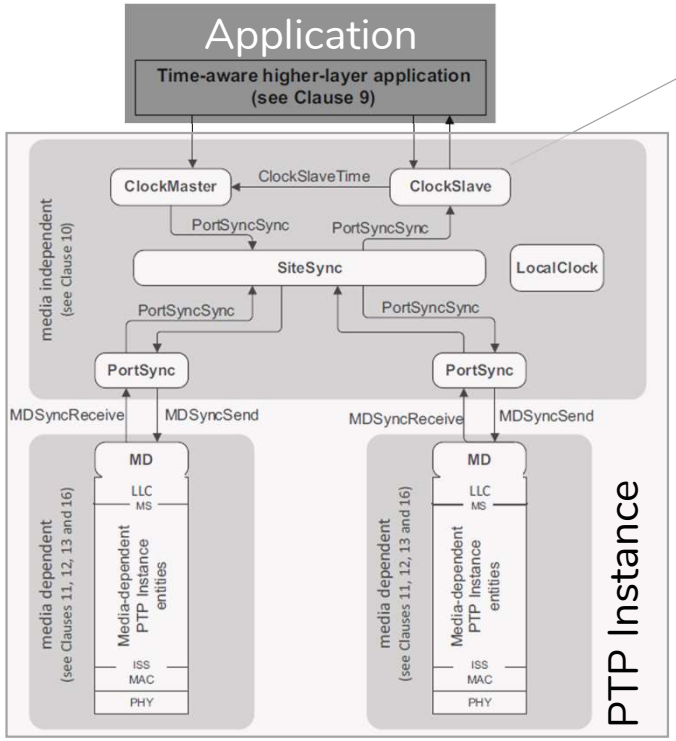



Figure 7-8—PTP Instance model

**10.2.13.2.1 updateSlaveTime():** Updates the global variable clockSlaveTime (see 10.2.4.3), based on information received from the SiteSync and LocalClock entities. It is the responsibility of the application to filter slave times appropriately (see B.3 and B.4 for examples). As one example, clockSlaveTime can be:

a) Set to syncReceiptTime at every LocalClock update immediately after a PortSyncSync structure is received, and

b) Incremented by localClockTickInterval (see 10.2.4.18) multiplied by the rateRatio member of the previously received PortSyncSync structure during all other LocalClock updates.

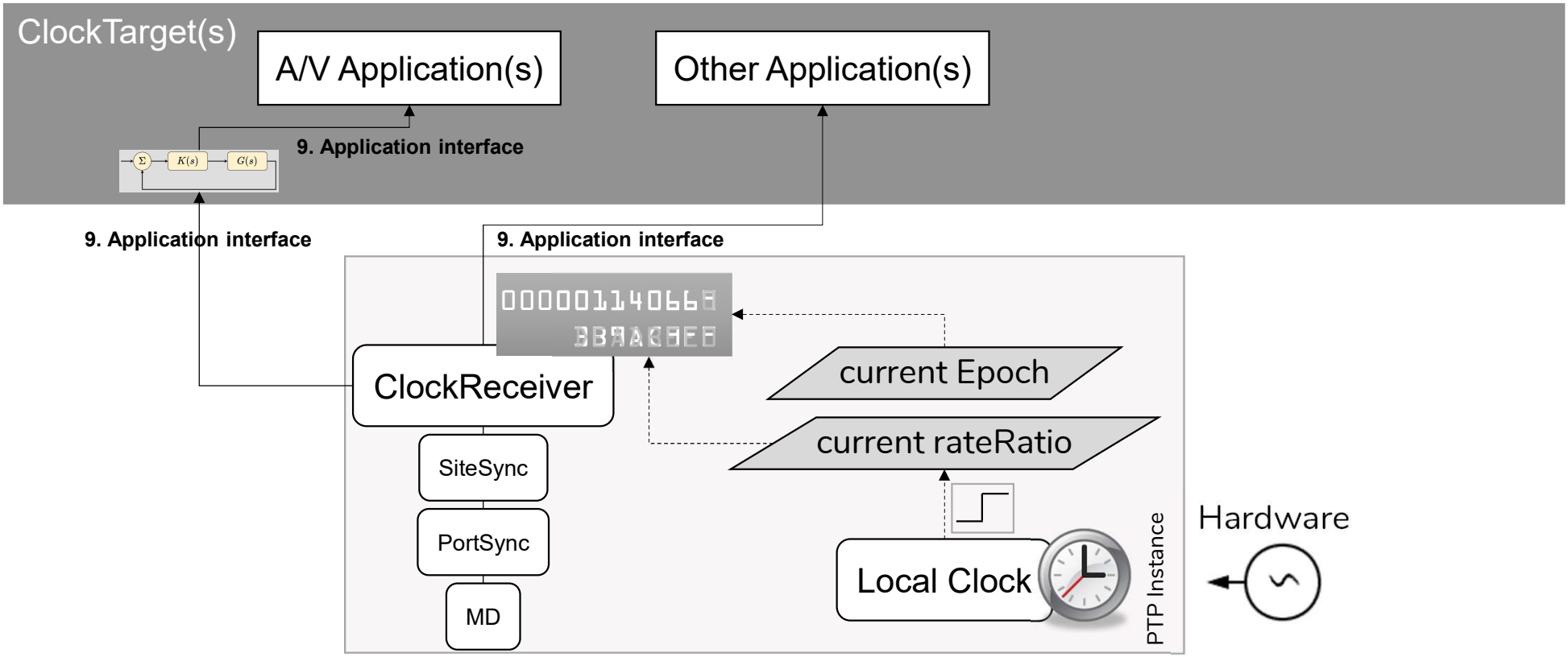
If no PTP Instance is grandmaster-capable, i.e., gmPresent is FALSE, then clockSlaveTime is set to the time provided by the LocalClock. This function is invoked when rcvdLocalClockTickCSS is TRUE.

Any other methods to set "*clockSlaveTime*" would well be within the specification.

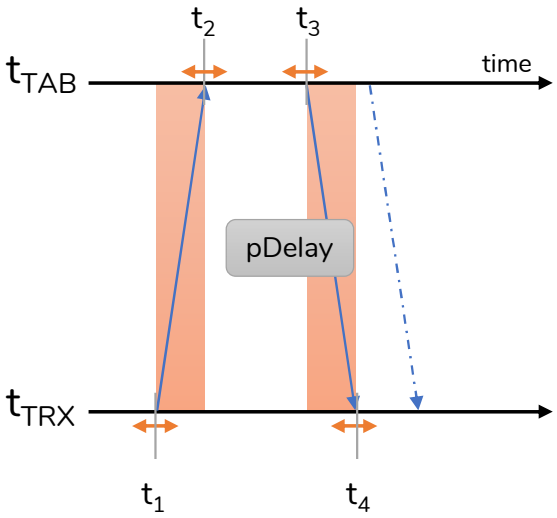
**10.2.4.3 clockSlaveTime:** The synchronized time maintained, at the slave, at the granularity of the LocalClock entity [i.e., a new value is computed every localClockTickInterval (see 10.2.4.18) by the ClockSlave entity]. The data type for clockSlaveTime is ExtendedTimestamp.

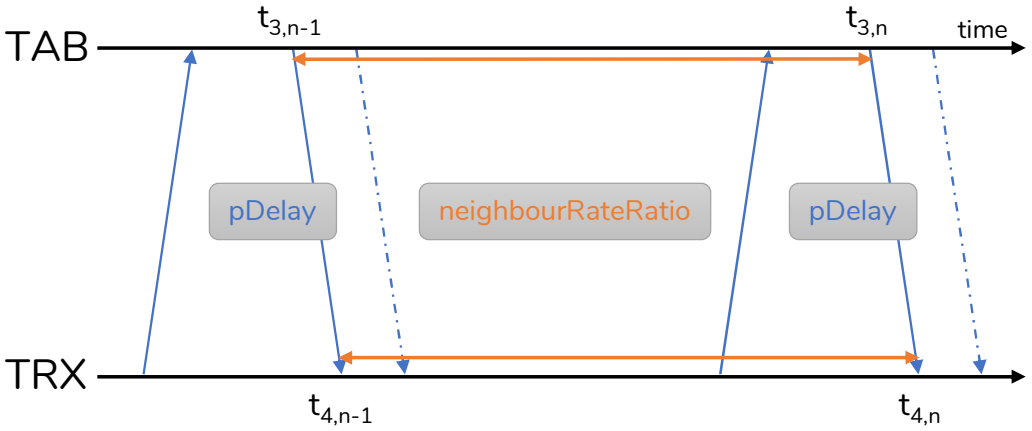
Since "*clockSlaveTime*" is global per PTP Instance, it is available to the MD entity(s)!

# Refining the 802.1AS model



# pDelay in Time Transmitter time




$mLD_{TAB} = ½ \times ((t_4 - t_1) \times nRR_{TRX} - (t_3 - t_2))$

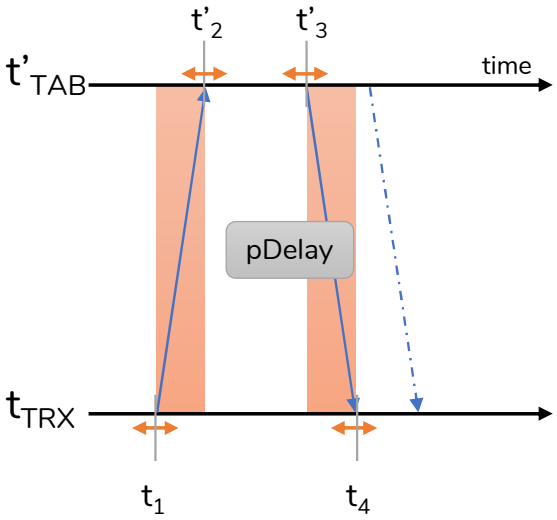
$nRR_{TRX} = (t_{3,n} - t_{3,n-1}) / (t_{4,n} - t_{4,n-1})$

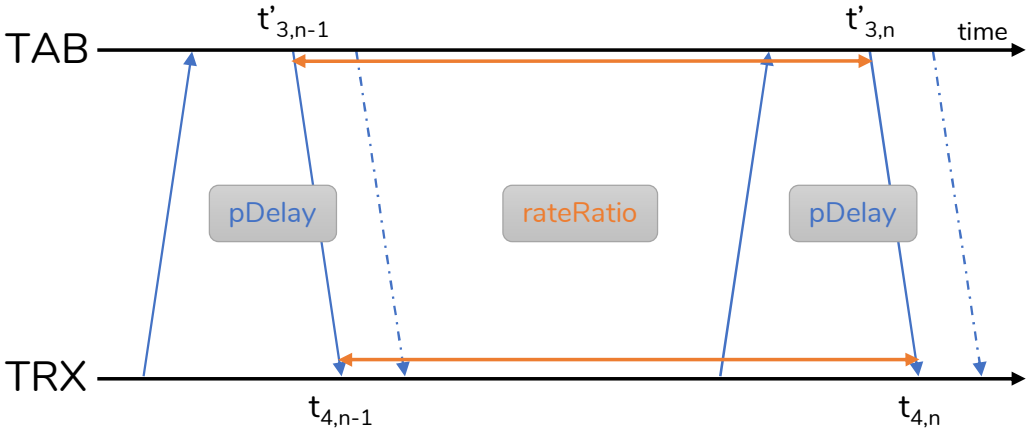
neighbourRateRatio: $nRR_{TRX} = r_{TAB} / r_{TRX}$

rateRatio: $rR_{TRX} = r_{GTT} / r_{TRX} = nRR_{TRX} \times cSRO_{TAB}$

$cSRO_{TAB} = rR_{TAB} = r_{GTT} / r_{TAB}$

# pDelay in Time Receiver time



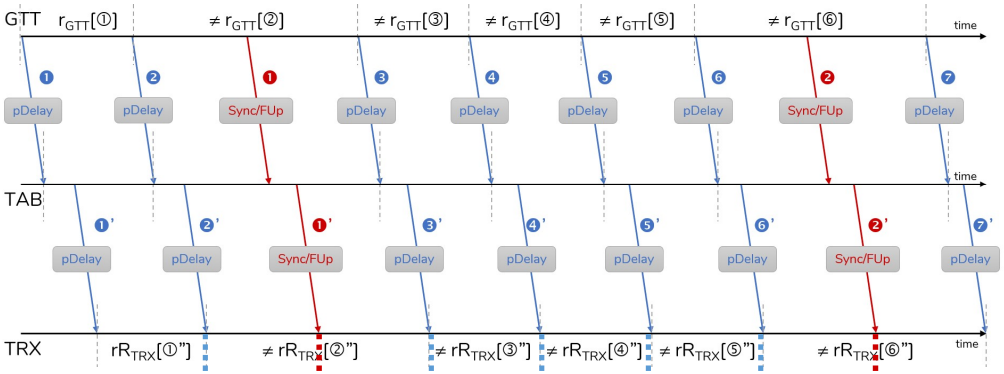

$mLD_{TRX} = ½ \times ((t_4 - t_1) \times cSRO_{TAB} - (t'_3 - t'_2))$

$rR_{TRX} = (t'_{3,n} - t'_{3,n-1}) / (t_{4,n} - t_{4,n-1})$

rateRatio: $rR_{TRX} = r_{GTT} / r_{TRX}$

$cSRO_{TAB} = rR_{TAB} = r_{GTT} / r_{TAB}$

# Update rateRatio more often




**10.2.13.2.1 updateSlaveTime():** Updates the global variable clockSlaveTime (see 10.2.4.3), based on information received from the SiteSync and LocalClock entities. It is the responsibility of the application to filter slave times appropriately (see B.3 and B.4 for examples). As one example, clockSlaveTime can be:

a) Set to syncReceiptTime at every LocalClock update immediately after a PortSyncSync structure is received, and

b) Incremented by localClockTickInterval (see 10.2.4.18) multiplied by the rateRatio member of the previously received PortSyncSync structure during all other LocalClock updates.

a) $sRT_{TRX}[❶'] = pOT_{TAB}[❶'] + fUCF_{TAB}[❶'] + mLD_{TRX}[❷'] + dA = SRT[❶']$

b) $sRT_{TRX} \mathrel{+}= sRT_{TRX} + ICTI_{TRX} \times rR_{TRX}[①”]$

b) $sRT_{TRX} \mathrel{+}= sRT_{TRX} + ICTI_{TRX} \times rR_{TRX}[②”]$

b) $sRT_{TRX} \mathrel{+}= sRT_{TRX} + ICTI_{TRX} \times rR_{TRX}[③”]$

b) $sRT_{TRX} \mathrel{+}= sRT_{TRX} + ICTI_{TRX} \times rR_{TRX}[④”]$

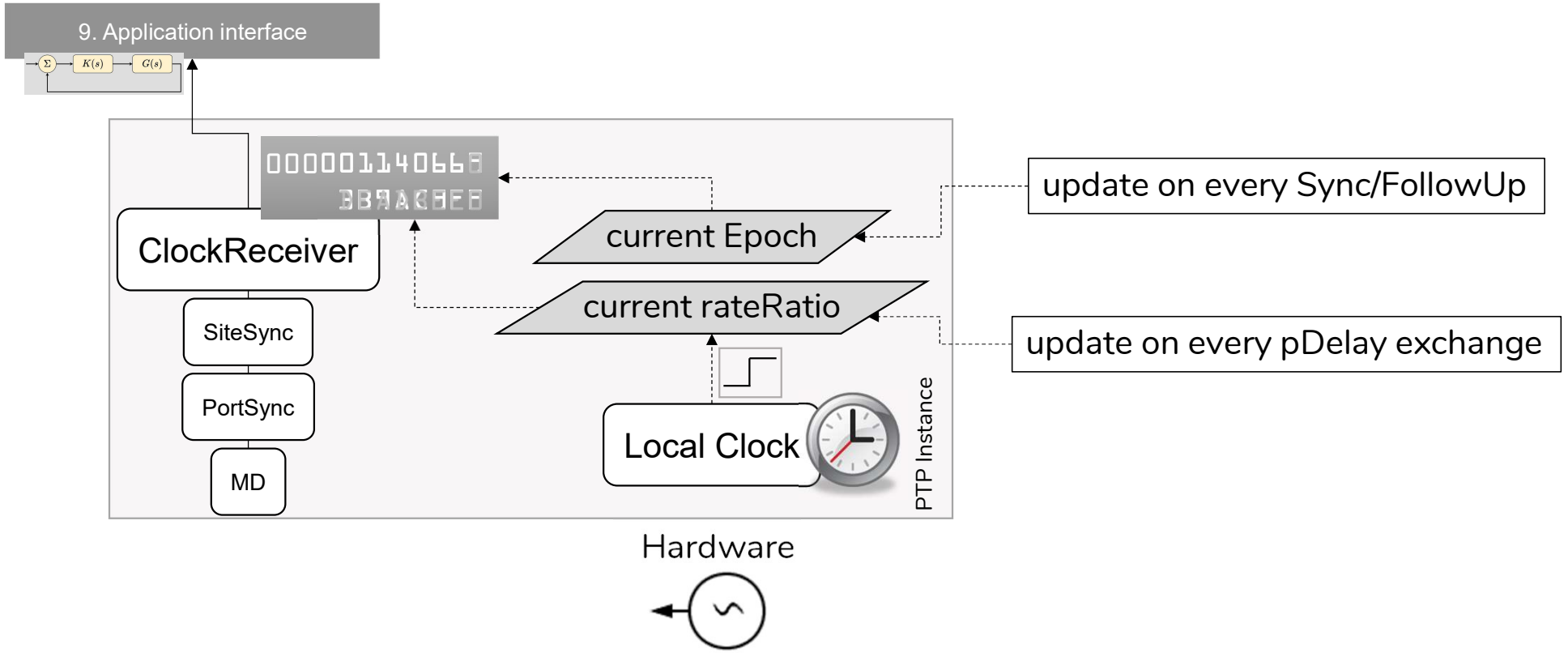
b) $sRT_{TRX} \mathrel{+}= sRT_{TRX} + ICTI_{TRX} \times rR_{TRX}[⑤”]$

a) $sRT_{TRX}[❷'] = pOT_{TRX}[❷'] + fUCF_{TRX}[❷'] + mLD_{TRX}[❻'] + dA = SRT[❷']$



ETHERNOVIA
VIRTUALIZING VEHICLE COMMUNICATION

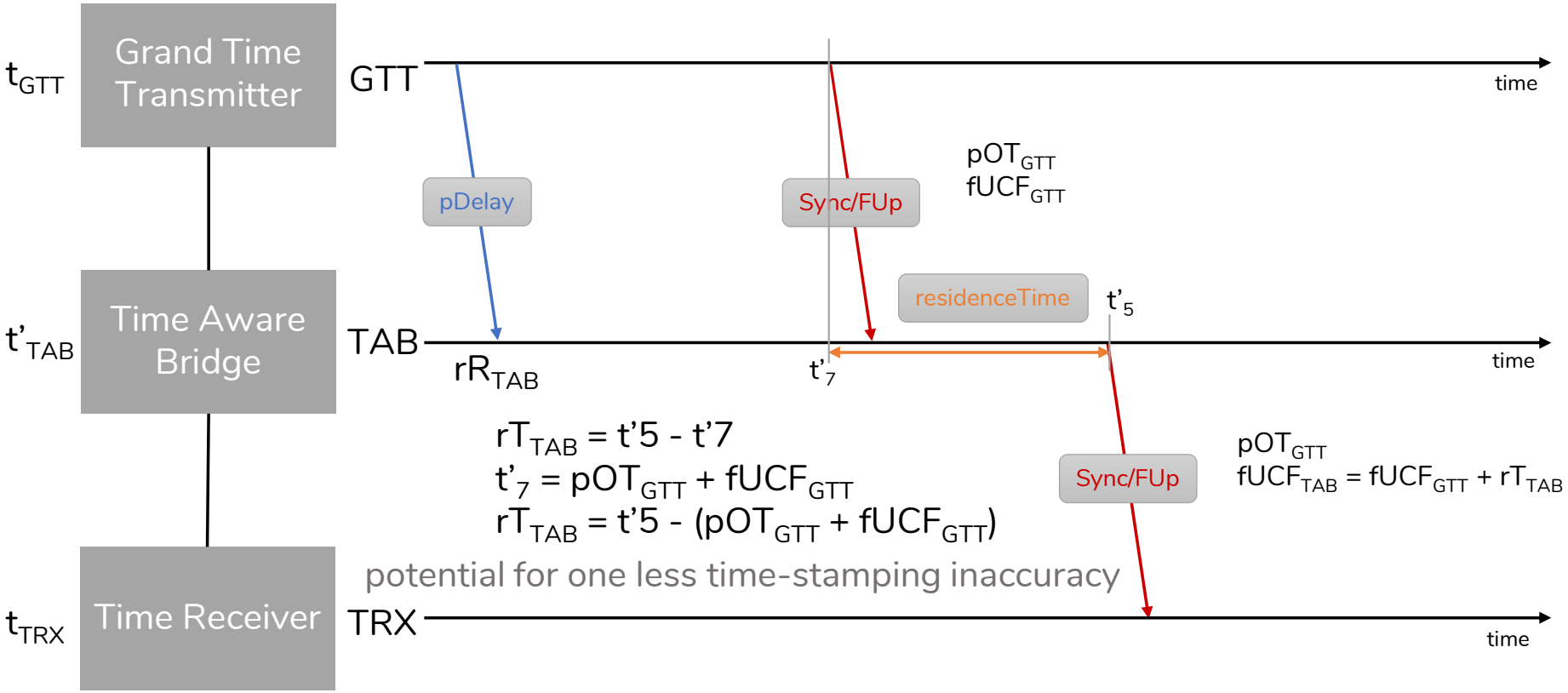
# Using rateRatio from pDealy






ETHERNOVIA
VIRTUALIZING VEHICLE COMMUNICATION

# Calculating residence time




not to scale!



ETHERNOVIA
VIRTUALIZING VEHICLE COMMUNICATION


THANK YOU
ETHERNOVIA
VIRTUALIZING VEHICLE COMMUNICATION