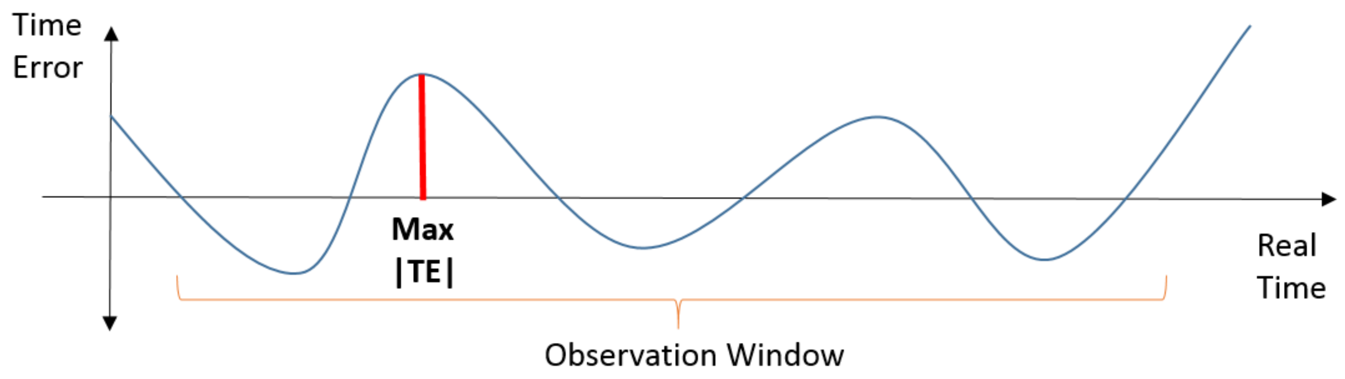
Measuring recovered clock quality

Bogdan Tenea, 7/18/2016

Measuring recovered clock quality



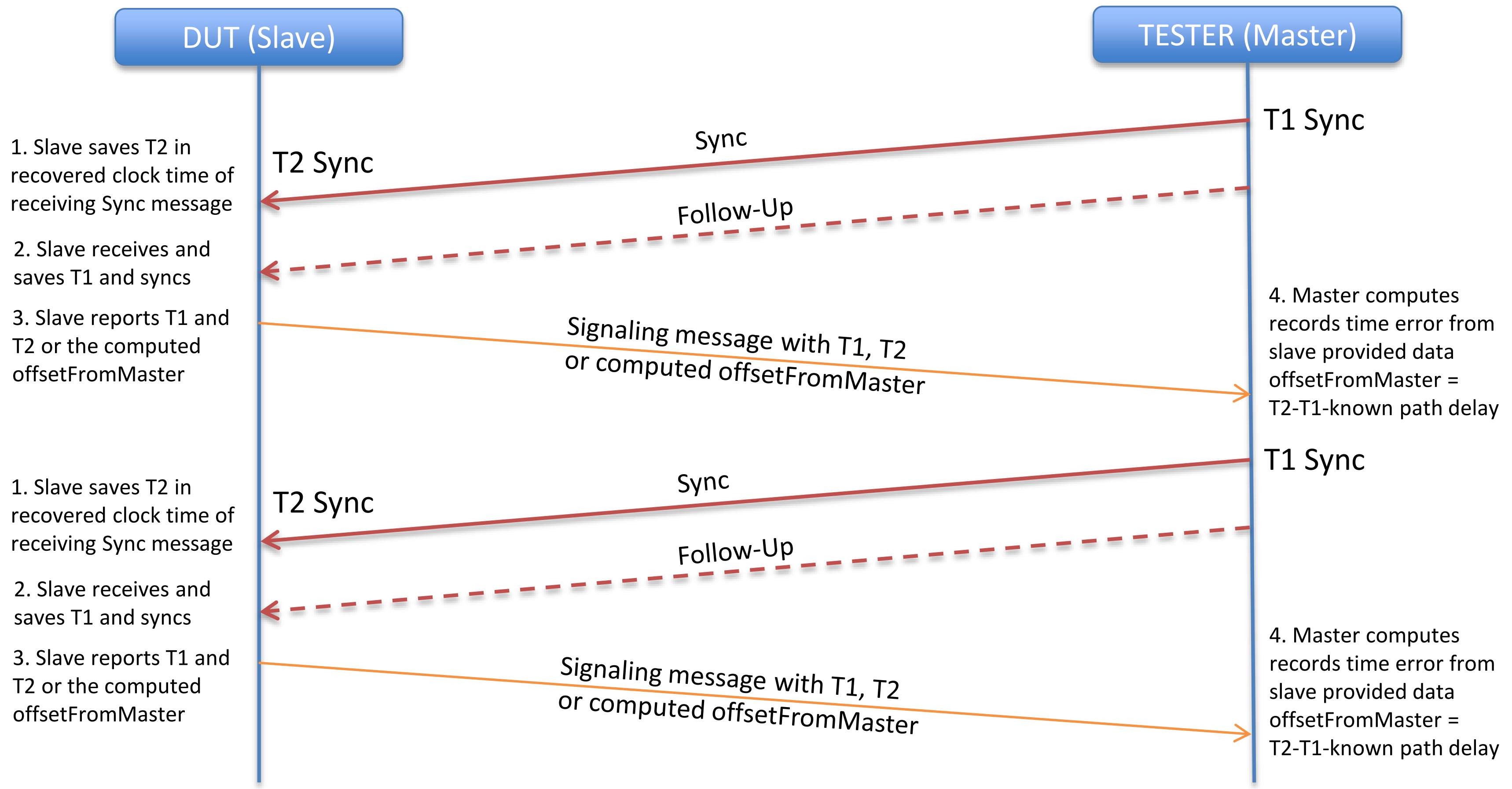
Objective – measure the maximum absolute error from a reference, within an observation window

Problem definition – some gPTP devices don't have a 1PPS output (traditional way to measure time error), but the requirement to validate clock phase & frequency accuracy remains

Discussion on how to solve this started in AVnu, moved to 802.1AS group, and then to 1588
Based on this, 1588-rev is defining two methods of measuring the time error:

1) **Ingress method** – relies on **Slave** clock to **report** the time error it is experiencing
2) **Egress method** – a **Tester** directly connected **evaluates** the time error of the Slave

Ingress method



DUT (Slave)
TESTER (Master)
T1 Sync
Sync
T2 Sync
Follow-Up
1. Slave saves T2 in recovered clock time of receiving Sync message
2. Slave receives and saves T1 and syncs
3. Slave reports T1 and T2 or the computed offsetFromMaster
Signaling message with T1, T2 or computed offsetFromMaster
4. Master computes records time error from slave provided data offsetFromMaster = T2-T1-known path delay
T1 Sync
Sync
T2 Sync
Follow-Up
1. Slave saves T2 in recovered clock time of receiving Sync message
2. Slave receives and saves T1 and syncs
3. Slave reports T1 and T2 or the computed offsetFromMaster
Signaling message with T1, T2 or computed offsetFromMaster
4. Master computes records time error from slave provided data offsetFromMaster = T2-T1-known path delay

Ingress method relies on Slave to declare it's error at the moment of receiving a Sync message
(Example: *"When you sent me the last Sync message, my recovered clock when receiving it was X"*)
Two flavors defined in 1588, one that reports error, and one that reports T1 and T2 – **same principle!**

Ingress method pros/cons

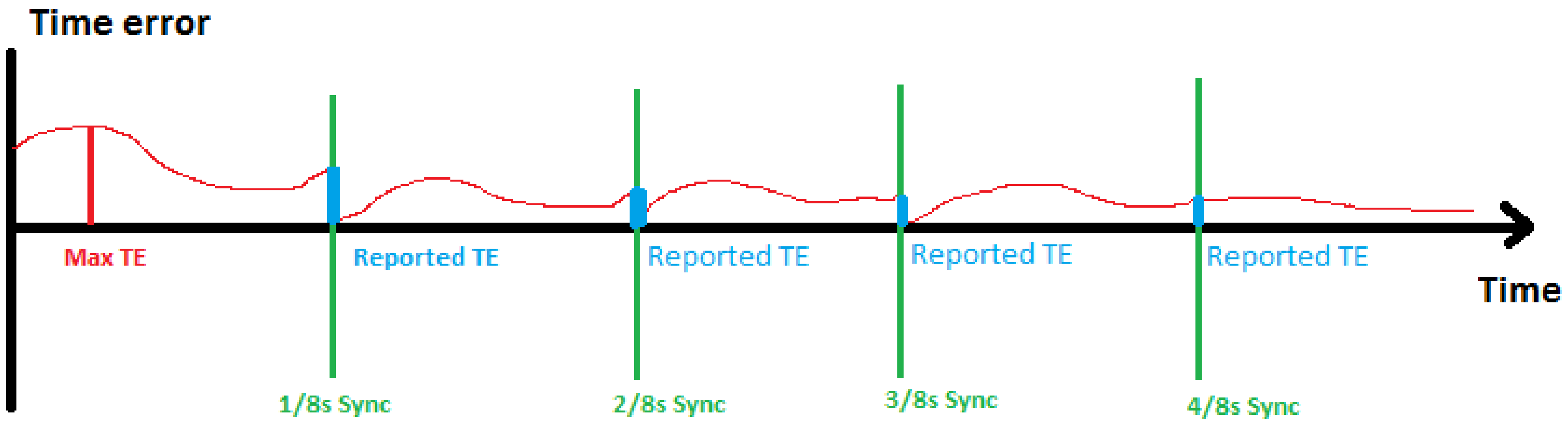
Advantages of Ingress method:

- Reporting of can be done in-band with 1588 proposed TLVs attached on signaling message or simply locally logged/stored to be accessed remotely through YANG (ideal for live deployments)
- Does not require any extra capability on the Slave except sending TLV/storing the data

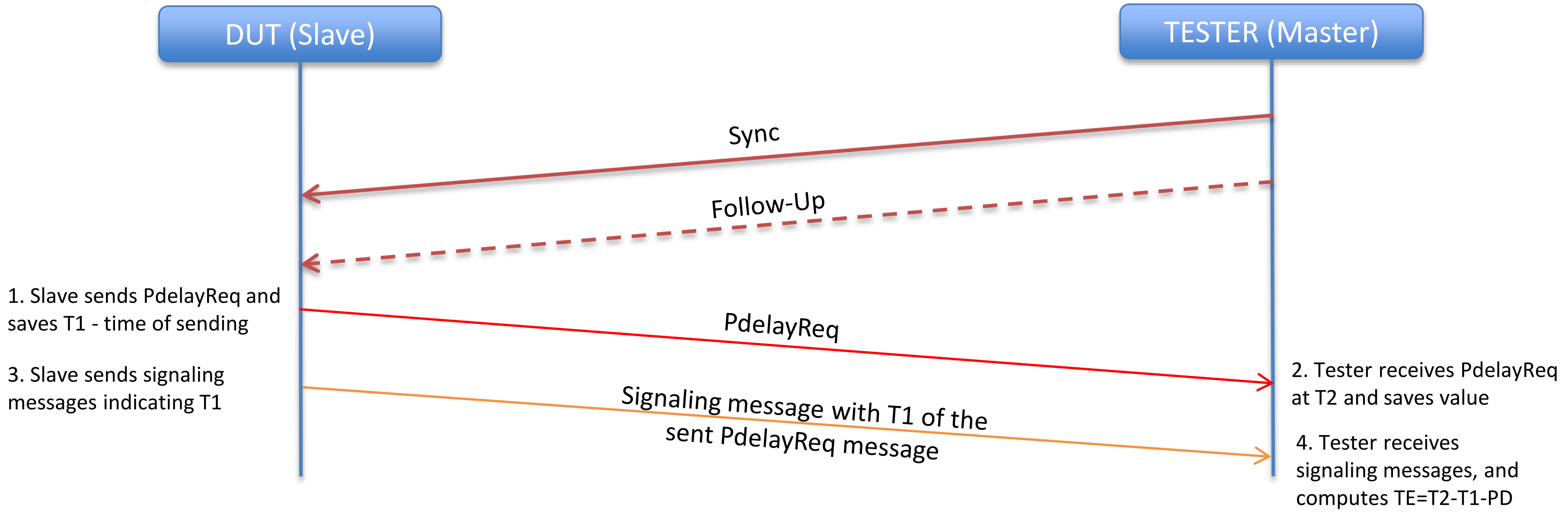
Shortcomings of Ingress method:

- **Sampling of time error is dependent on the timing of incoming Sync messages**
  It might not reflect the actual maximum time error between two incoming Sync messages
- **The Slave can report a value that is not the actual one**
  This could be due to a fixed phase offset, coding error, or due to intentionally returning values that make the device seem better (very hard to detect, example in backup slides)

=> the Ingress method is not adequate for certification testing



Egress method


DUT (Slave)
TESTER (Master)
Sync
Follow-Up
1. Slave sends PdelayReq and saves T1 - time of sending
PdelayReq
2. Tester receives PdelayReq at T2 and saves value
3. Slave sends signaling messages indicating T1
Signaling message with T1 of the sent PdelayReq message
4. Tester receives signaling messages, and computes TE=T2-T1-PD

Egress method - Slave reports it's recovered time at moment when an event message is sent
(Example: *"Slave: the last PdelayReq I sent was at global gPTP time X as per my recovered clock"*)

Advantages of Egress method:

- **Tester evaluates the error, it does not rely on the Slave to report it**
- Reporting of can be done in-band with 1588 proposed TLVs attached on signaling message
- Time of reporting is decoupled from the receive time of the incoming Sync messages

Shortcomings of Egress method:

- **Complicated to use in deployed networks – needs support from directly connected Master of DUT**

We need both Egress & Ingress

1588 defined both because they are both needed

- Egress provides the right amount of visibility to be used for certification testing purposes
- Ingress can be used for post-deployment testing, if the ingress measurement method has been verified during certification against the Egress method

If we try to use only one of the methods then either:

- We may lower interoperability (as some devices may not actually sync the clock correctly)
- We may make it hard/impossible to do post-deployment testing

We have received requirements for both lab testing / certification and post-deployment validation from different industry groups industry groups (automotive, ProAV and industrial)

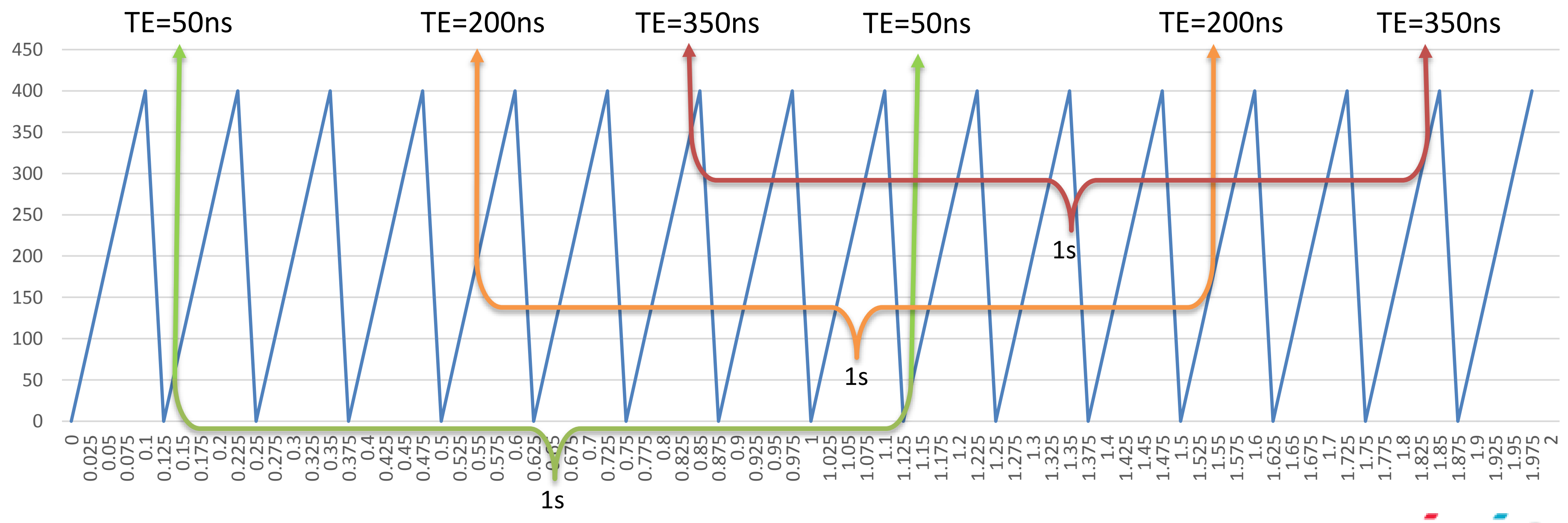
We believe that achieving both is important

Problems in using Egress method for gPTP

For gPTP common sense is that reporting would be based on the sent PdelayReq event messages. If PdelayReq messages are timed immediately or close to receiving a Sync message (when correction of the clock is made), they might not reveal the actual time error the device is experiencing.

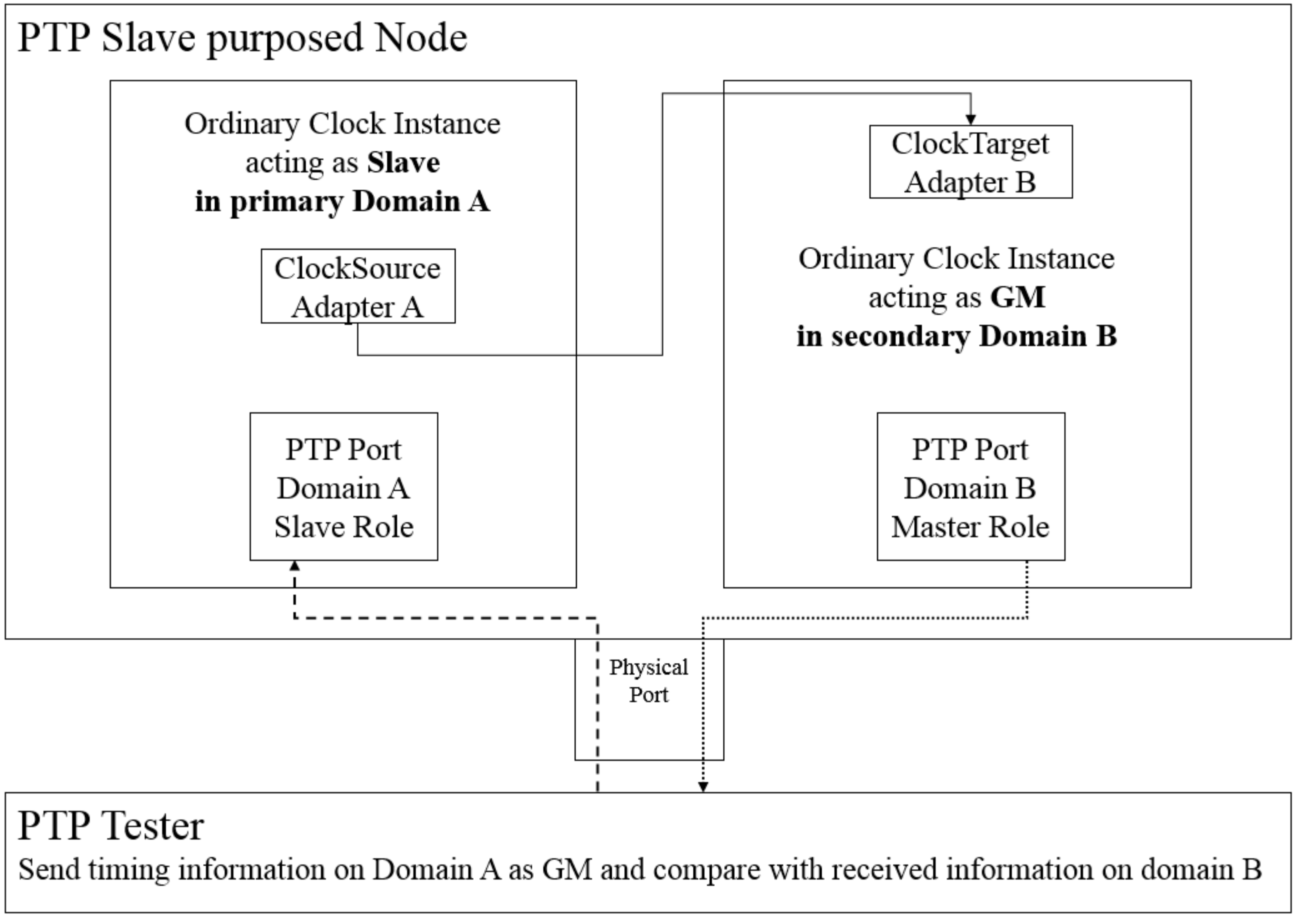
1. Consider a Slave clock drifting 400ns between each 2 Sync messages (125ms interval)
2. AND not properly implementing syntonization (algorithm or rate ratio calculus)
3. The time error function looks similar to a saw (resets to ~0 at each Sync received)
4. If event PdelayReq is sent close after Sync is received measured error is lower than one
5. Because PdelayReq interval is a multiple of the Sync interval, this would happen at every time

Not just theory, this was experienced with real devices in lab testing !



Proposed solution

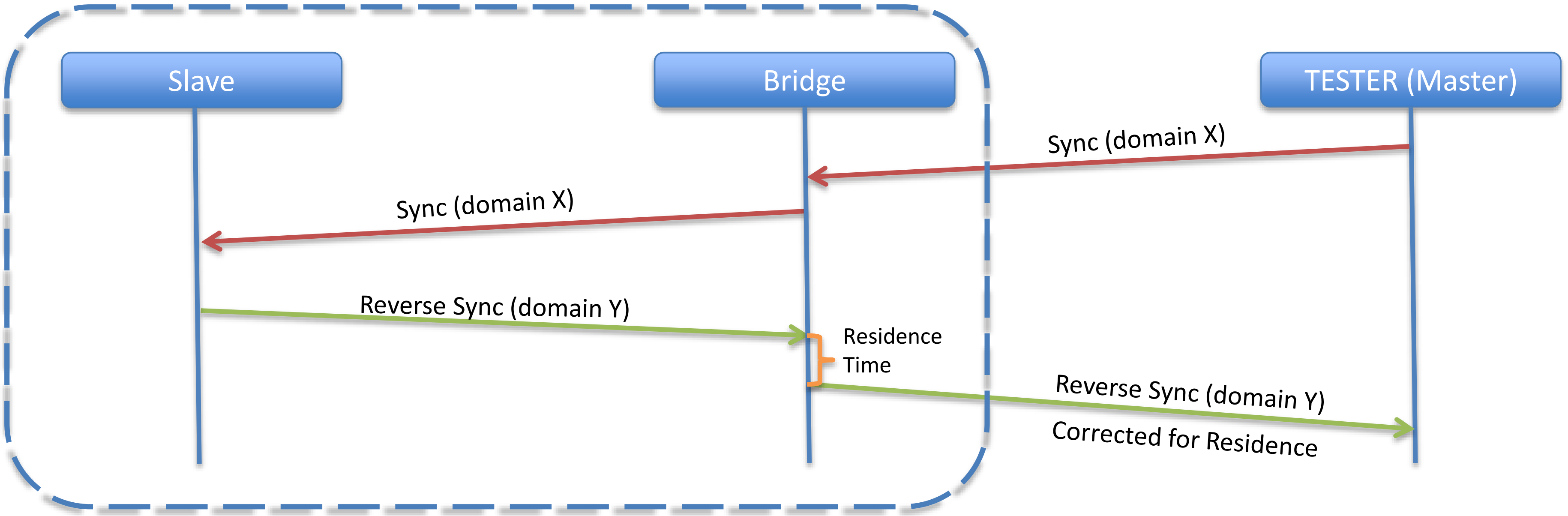
Use other DUT generated egress message that can be timed totally independent from the gPTP instance we are trying characterize => **use a second gPTP instance within the same PTP node that outputs the recovered clock of the first instance, with flexible message timing** (does not need the full instance, may skip BMCA and just send Sync/FollowUp)


PTP Slave purposed Node
Ordinary Clock Instance acting as **Slave** in primary Domain A
ClockSource Adapter A
PTP Port Domain A Slave Role
ClockTarget Adapter B
Ordinary Clock Instance acting as **GM** in secondary Domain B
PTP Port Domain B Master Role
Physical Port
PTP Tester

PTP Tester
Send timing information on Domain A as GM and compare with received information on domain B

Further advantages for using a Sync messages for Egress

- On integrated devices such as automotive ECUs that have an integrated Slave + Bridge, there is no observability of the Slave Pdelay mechanism (use-cases of this seen also in ProAV)
- Using Sync messages as Egress, the Bridge can correct for the "residence time" of the Reverse Sync the same way as they would normally do for the standard gPTP instance
  - Bridge would just forward the Reverse Sync on it's Master port where the Tester is
  - This is effectively running a subset of gPTP-rev, and using one domain for Synchronization, and a different domain to do the recovered clock measurement
  - Uses the same principle that will probably be used in gPTP-rev for cases in which the Relay would be the one doing the merging of timing information for resource constrained Salves
- If proper time is set in Reverse Sync preciseOriginTimestamp there is no need for additional egress TLV
- **Implementation is straight forward – can be done from existing "Send Sync" code**



DUT: Integrated ECU with Slave + Bridge

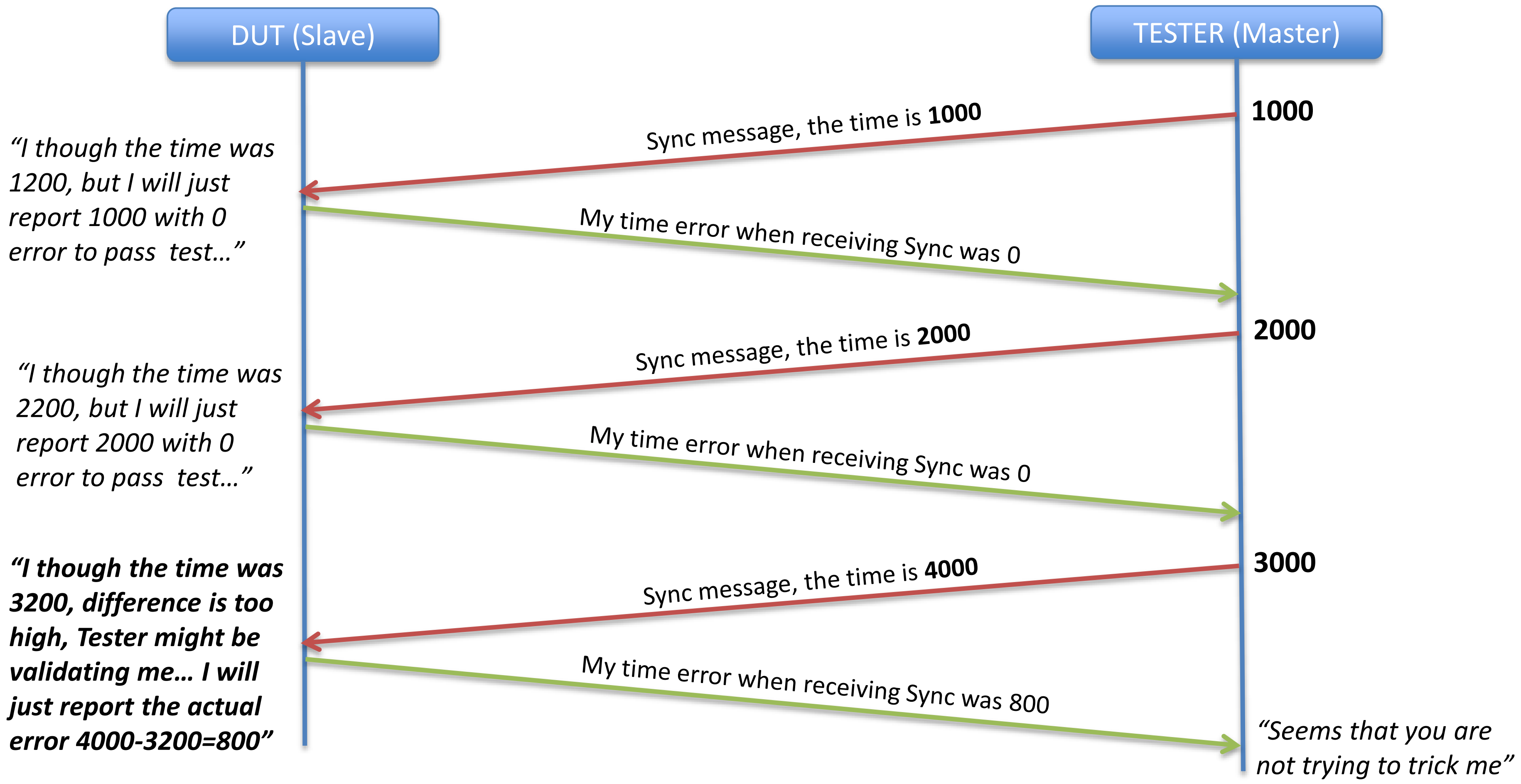
Add the following example Section 7.2

“A PTP node that has two Ordinary Clock gPTP Instances, in a design that has timing information coming from the ClockSource of one domain to the ClockTarget of another domain, the later will output the recovered clock quality. If the PTP Node is connected to a tester device that can compare timing information sent by itself on primary domain A to the timing information received back on secondary domain B, the same tester can sample the time error of the recovered clock of the Slave Instance in domain A. The GM Instance used for regenerating that recovered clock need not have a full Instance implementation – for example BMCA could be disabled and the role could be set static. If the GM Instance always outputs the PTP Clock of the domain A, even when not locked, holdover behavior could be tested. The timing and rate of Sync messages of the GM instance can be adjusted to improve the sampling as per application dependent needs. The GM Instance can be disabled when not used in a test bench environment.”

Detailed specification will be defined by interested industry alliances.

Backup slides

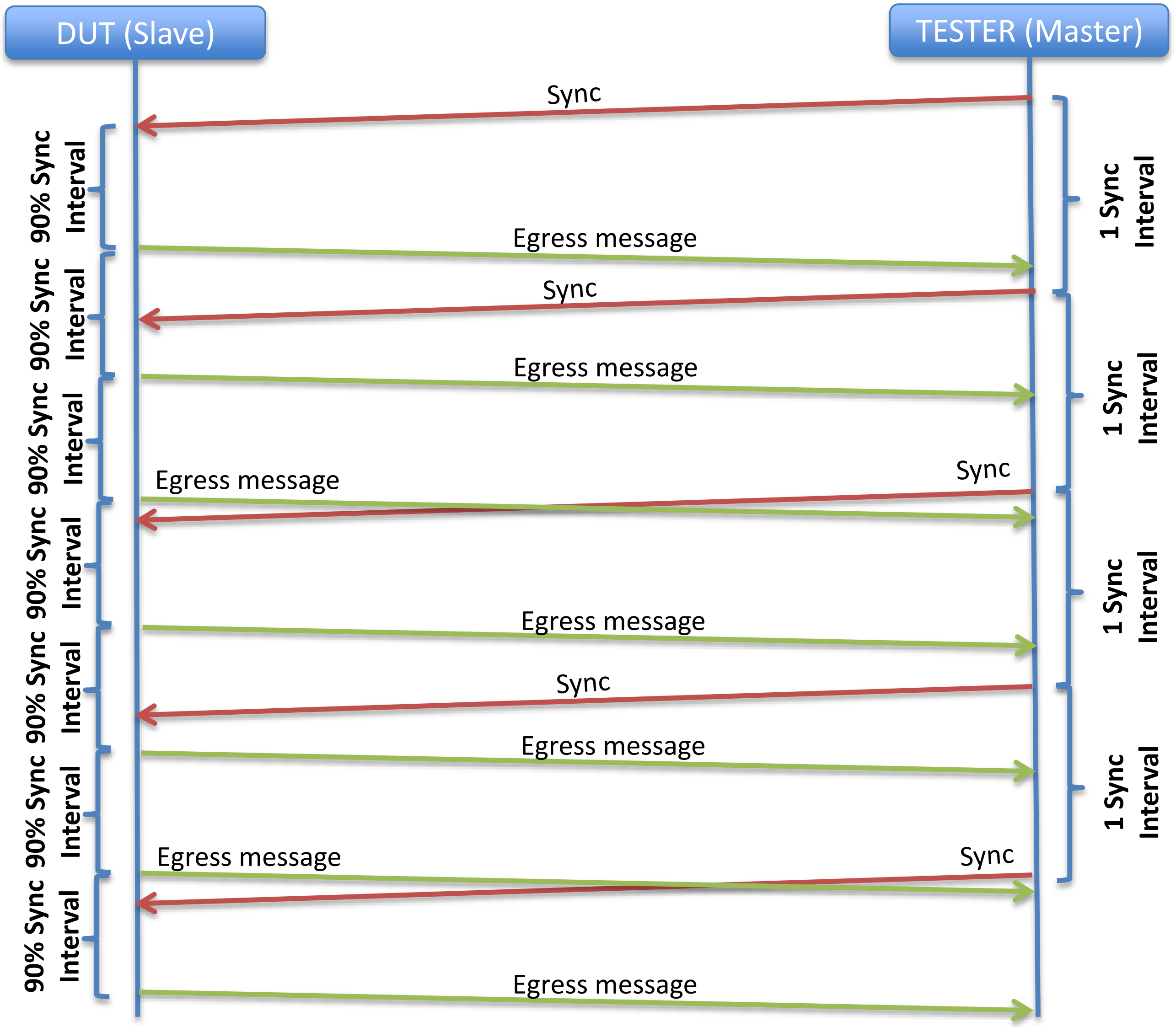
Example of Slave reporting an invalid time error with Ingress method


DUT (Slave)
TESTER (Master)
1000
Sync message, the time is **1000**
"I though the time was 1200, but I will just report 1000 with 0 error to pass test…"
My time error when receiving Sync was 0
2000
Sync message, the time is **2000**
"I though the time was 2200, but I will just report 2000 with 0 error to pass test…"
My time error when receiving Sync was 0
3000
Sync message, the time is **4000**
"I though the time was 3200, difference is too high, Tester might be validating me… I will just report the actual error 4000-3200=800"
My time error when receiving Sync was 800
"Seems that you are not trying to trick me"

An ill intended Slave can report a smaller than actual time error (potentially even 0 just by echoing back T1) when it detects it is off by not too much, and report a big time error when the Tester tries to check it reporting correctly. Example – slave is synchronizing with 200ns time error, higher than the 80ns required for gPTP certification (perhaps due to timestamping granularity). It can report to have an error of 80ns for usual cases, and detect when a tester is trying to validate it is reporting correctly report the actual one – **hard to detect if tricking tester!**

Adjust Egress message rate for better sampling

Need a method to vary the timing of the message that generates the measurement, without affecting when the PdelayReq message is sent (that would modify gPTP internal behavior)



ixia