Arguments and experimental data for allowing Slave remembering of GM frequency (i.e. simple holdover algorithms)

Version 0.01, 2007-11-01

Alan K. Bartky alan@bartky.net

Bartky Networks www.bartky.net

Notice of copyright release

- **Notice:**
  - This document has been prepared to assist the work of the IEEE P1722 and IEEE 802 Working Groups. It is offered as a basis for discussion and is not binding on the contributing individual(s) or organization(s). The material in this document is subject to change in form and content after further study. The contributor(s) reserve(s) the right to add, amend or withdraw material contained herein.
- **Copyright Release to IEEE:**
  - The contributor grants a free, irrevocable license to the IEEE to incorporate material contained in this contribution, and any modifications thereof, in the creation of an IEEE Standards publication; to copyright in the IEEE's name any IEEE Standards publication even though it may include portions of this contribution; and at the IEEE's sole discretion to permit others to reproduce in whole or in part the resulting IEEE Standards publication. The contributor also acknowledges and accepts that this contribution may be made public by the IEEE P1722 Working Group or the IEEE 802 Working Group.

Revision History

Rev	Date	Comments
0.01	2007-06-24	First version for discussion purposes

Issue summary

- There is a proposal to say that when an 802.1AS device transitions from a slave to a grandmaster that it must not remember the frequency derived from the previous grandmaster and instead when it becomes master, it should send sync/follow-up messages based on its nominal frequency.
- I took an action to do some experimentation, collect data and come up with a presentation.
- Based on data I've collected to date, I still believe it should be allowed for a slave to remember the frequency. There are benefits of maintaining frequency and that possible out of PPM range and other errors can be detected and corrected for.

Why allow frequency maintenance?

- Allows for smoother transitions of both time and frequency on transition from GM to the next in the 802.1AS network.
  - Frequency maintenance also makes for better time maintenance.
- Allows for cheaper devices when running as GM to automatically learn and create a more accurate time and frequency reference by allowing them to automatically inherit (i.e. calibrate to) the accuracy of the better and/or preferred master..
- Allows for 802.1AS devices to specify a preferred GM they can detect by its ID to remember the clock, and then better mimic that device if the preferred GM fails.

Why allow frequency maintenance?

- Other applications for 802.1AS to be used by besides AVB, AVBTP may have tighter restrictions and/or run better with less time and frequency changes. Possible examples:
  - High speed uncompressed video
  - Time Division Multiplexing Circuit emulation over Ethernet
  - Synchronized system time of day systems
    - i.e. could use 802.1AS as a poor mans NTP.
- If we put a little more smarts for devices that say they can become GM, then we may be able to simplify listeners as there should be less error recovery if 802.1AS devices are allowed to make a “reasonable” effort to maintain time and/or frequency
  - (similar philosophical argument was made for 1394 to AVBTP gateways is we want to make end stations as simple as possible and put adaptation complexity in devices that need to do adaptation).
- May make porting of applications that assume clock maintenance by lower layers (such as 1394/61883) less prone to errors and more likely to work.

HW Test environment

- 2 HW timestamp capable boards interconnected back to back using Gigabit Ethernet.
- Nominal Reference Frequency 133 MHz
- 100 PPM class oscillator
  - Measured error (24 hour versus stratum 2 NTP server):
    - System A: 12 PPM
    - System B: 84 PPM
  - Calculated PTP drift (delta PPM once stable): ~72 PPM
- Software Tunable HW counter for generating timestamps.
  - For this test, base period set to 40 nanoseconds to mimic 25 MHz frequency.
  - Capable of instantaneous small or large tuning up or down when running PTP code in slave mode.

SW Test environment

- Embedded Linux, 2.6.20 based kernel.
- PTP daemon with multiple command line options
- Key Test run options:
  - 40 ns measurement resolution
  - Sync interval 2^{-7} (128 sync/follow-up message pairs per second)
  - 802.1AS encapsulation
  - System B: Preferred Master (includes setting of priority to make sure it is preferred).
  - System A: Set to run as slave or master, defers to B.

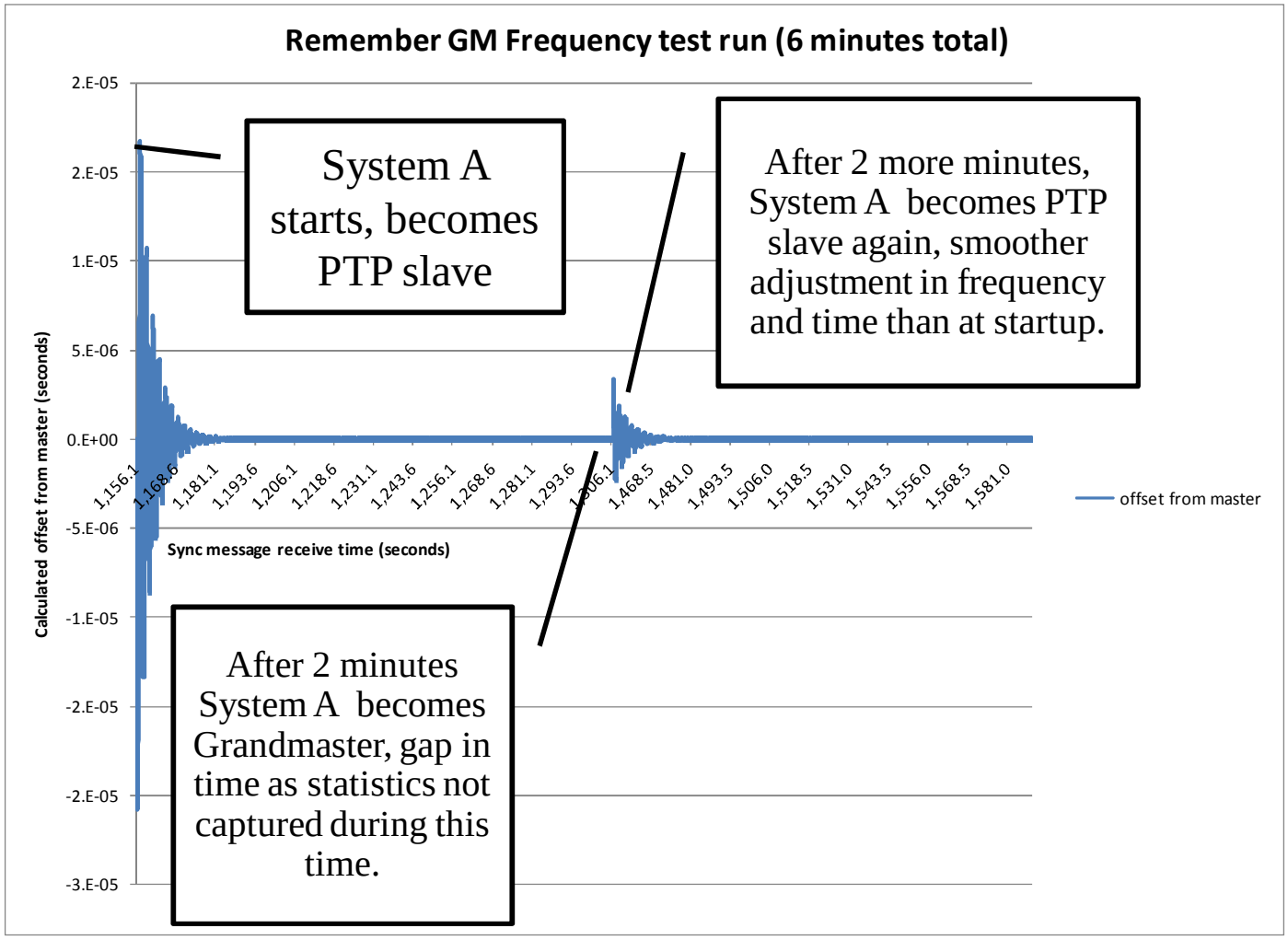
Test Description

- Goal to simulate short failure (2 minute) then recovery of a preferred grandmaster.
- Test description:
  - Startup System B , enable PTP.
    - System B fixed by priority as preferred GM.
  - Startup System A, enable PTP.
    - Sees system B as preferred GM, and goes into slave mode. Start statistics capture.
    - Note: for this test, statistics are only captured at system A when it is running as slave. This is why show a “jump in time”.
  - 2 minutes later: Disable PTP on system B.
    - System A then becomes Grandmaster and based on test run either remembers System B’s frequency or immediately returns to its nominal frequency.
  - 2 minutes later: Enable PTP on system B.
    - System A then becomes slave and then also has to adjust both time and frequency back to the value calculated from the preferred Grandmaster.

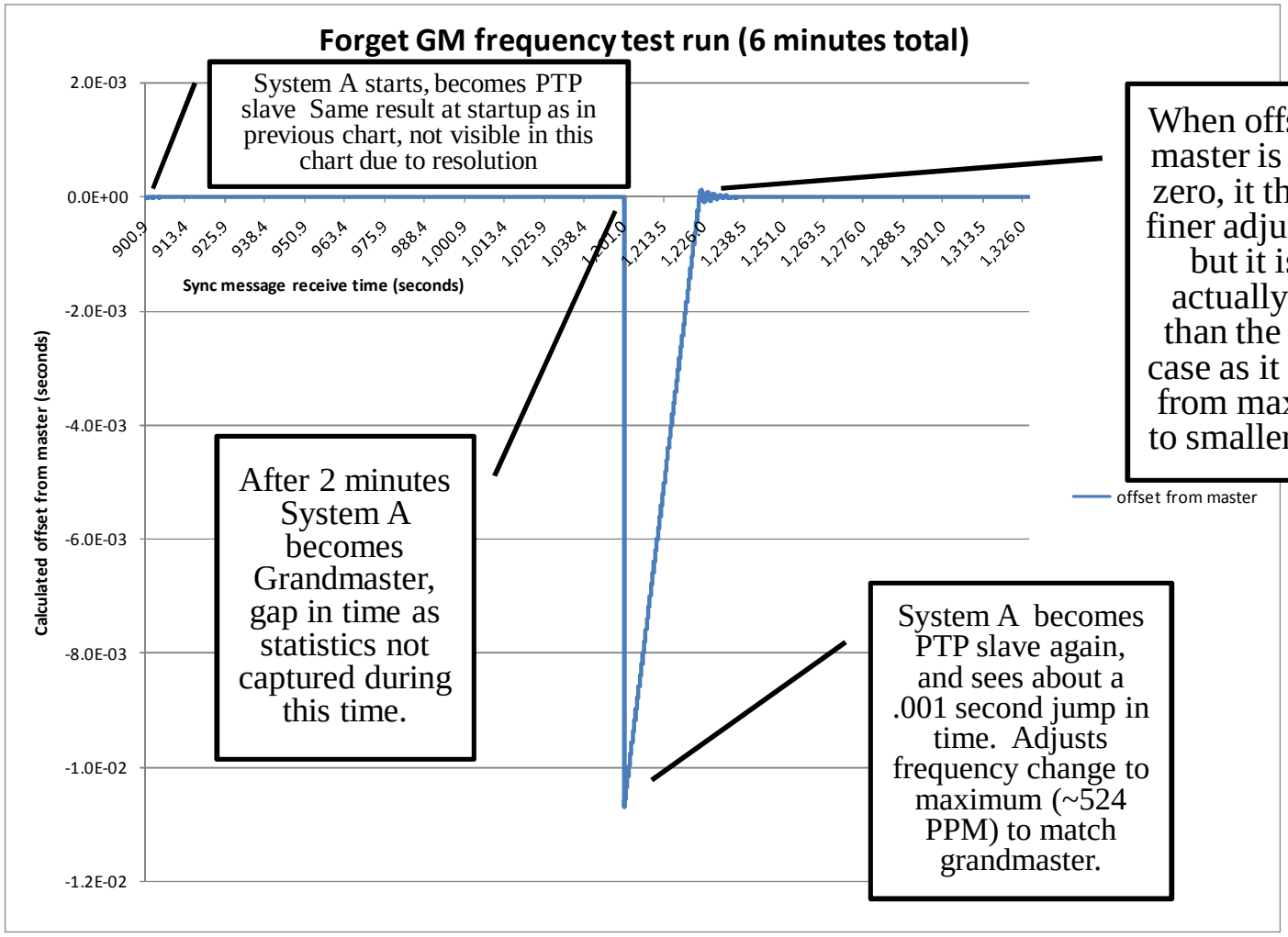
SW algorithm note

- Note on SW algorithm to adjust local frequency.
- The algorithm used to adjust time and currently has the following characteristics.
  - When the offset from master is calculated at the slave:
    - If the delta in time is more than one second, it changes the time, but does change the frequency.
    - If the delta is less than a second, it changes the frequency to adjust to the master based on a filter where the maximum frequency adjustment is 524.288 PPM
- This is done so that the slave device avoids changing time up or down unless it is greater than or equal to a second.
  - This assumes that the 802.1AS clock will be used for both frequency and time usage by the applications on the board (i.e. not just used for AVB/AVBTP, but for other Linux apps as well).

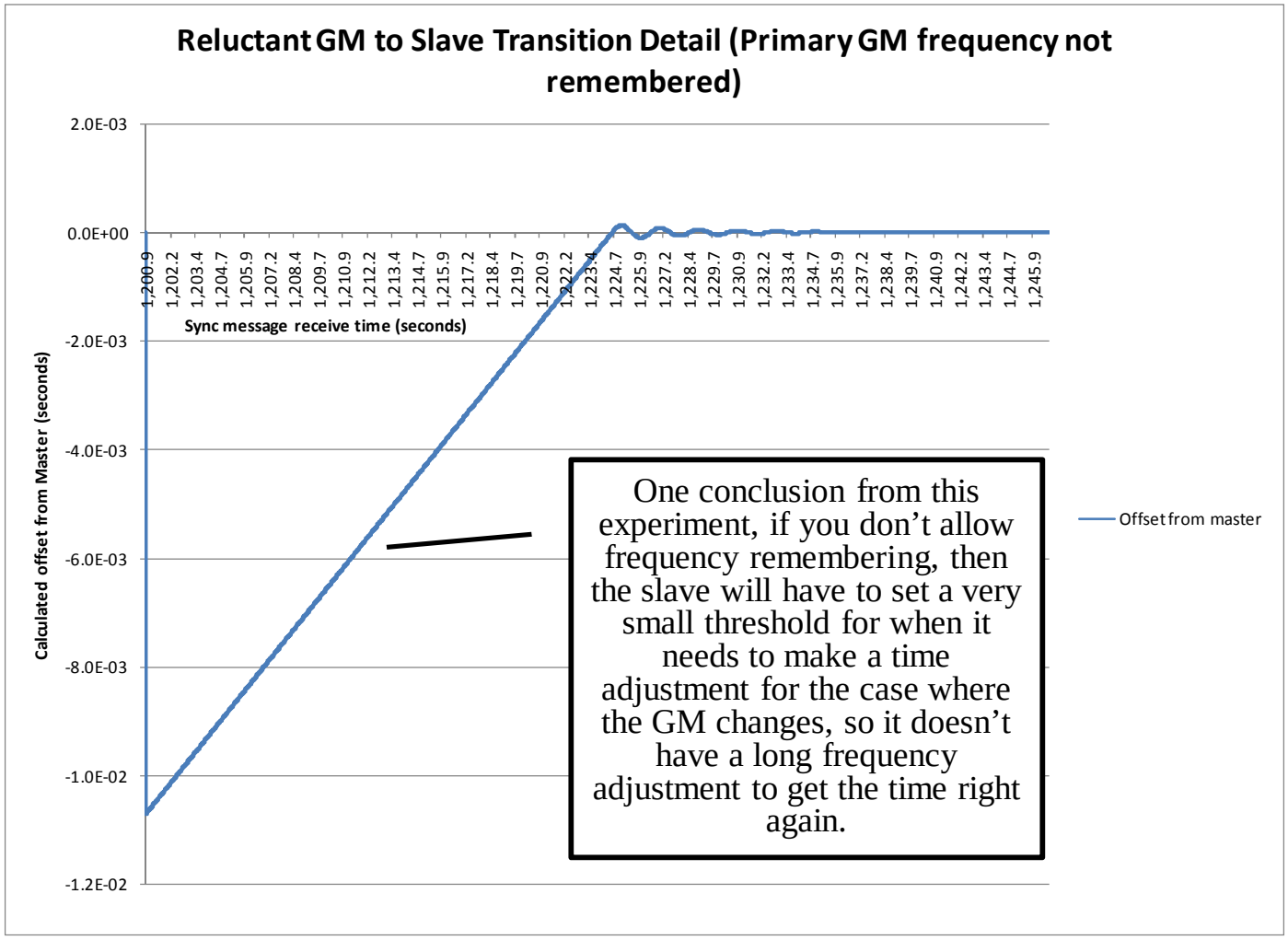
Remember GM frequency chart



Forget GM frequency chart



More detailed GM to Slave transition, forget GM frequency case



Potential errors and how to remedy them

- Error case:
  - 2 devices, 100 PPM class clocks. GM is at or near limit of accuracy, slave is cold and becomes GM, warms up, then becomes out of spec (>100 PPM error). Problem can get worse over time.
- Possible solutions:
  - Do not remember frequency if clock is in same class, only remember a more accurate clock based on announce message and delta accuracy is significant.
  - Do not allow unless devices run at 50 PPM or better accuracy.
  - If calculated drift is measured and out of spec, return to local frequency.
    - In my implementation, once the slave is stable relative to the GM, my implementation can actually measure a problem thus indicating either the local device or the GM is out of spec.

Backup

Sample PTP daemon command line options

```
-d                  display stats
-D                  display stats in .csv format
-z                  set debug level (0:none, 1:basic, 2:verbose)
-x                  do not reset the clock if off by more than one second
-t                  do not adjust the system clock
-a NUMBER,NUMBER    specify clock servo P and I attenuations
-w NUMBER           specify one way delay filter stiffness
-P                  specify hardware clock period in nanoseconds
-A                  specify base value for clock frequency adjustment
-R                  remember adjust value for slave to master transition
-8                  run in IEEE 802.1AS PTP Layer 2 mode instead of IP/UDP
-l NUMBER,NUMBER    specify inbound, outbound latency in nsec
-y NUMBER           specify sync interval in 2^NUMBER sec
-g                  run as slave only
-p                  make this a preferred clock
```