

Design Considerations for Software Only Implementations of the IEEE 1588 Precision Time Protocol

Kendall Correll, Nick Barendt
VXI Technology, Inc.
Cleveland, Ohio, USA

Michael Branicky
EECS Dept., Case Western Reserve University
Cleveland, Ohio, USA

Abstract – This paper investigates adjusting computer clock frequency and time to provide a precise clock for test and measurement systems. In particular, it is concerned with the precision achievable using IEEE 1588 Precision Time Protocol systems without the support of specialized hardware. This paper outlines the design of a free IEEE 1588 implementation named PTPd. Particular attention is paid to the design of the clock servo—the system that steers the clock rate. This paper evaluates the implementation by the precision of the time coordination between networked test and measurement systems.

I. INTRODUCTION

The IEEE 1588 Precision Time Protocol (PTP) [1] provides a means by which networked computer systems can agree on a master clock reference time, and a means by which slave clocks can estimate their offset from master clock time. PTP implementations typically have a clock servo that uses a series of time offset estimates to coordinate the local slave clock with the reference master clock time, a process referred to as clock discipline.

This paper presents our software-only implementation of PTP. Precise time coordination with PTP relies on precise estimates of the send and receive times (time stamps) of messages exchanged between the master and slaves. High precision time stamps can be achieved with the support of specialized hardware interfaces in the physical layer of the network; however, many legacy systems lack such hardware interfaces. A PTP implementation that is not supported by specialized hardware is referred to as a software-only implementation. These implementations must time stamp in higher layers of the network, which introduces large degrees of non-determinism in the time stamp latencies, known as jitter. Achieving precise master-slave time coordination with jittery time stamps is the primary obstacle in the design of software-only PTP implementations.

This paper is organized as follows. Section II is a brief introduction to IEEE 1588 (PTP). Section III introduces PTPd, our open-source, software-only PTP implementation. Section IV provides an overview of clock servo design and the specifics of PTPd's clock servo. Section V presents test results of PTPd's performance in a target application. PTPd achieved precision on the order of microseconds. Section VI presents conclusions, comments on future work, and a link to PTPd's source code.

II. PTP IN BRIEF

A. Masters and Slaves

In PTP, master clocks provide the reference time for one or more slave clocks through the exchange of messages over a network. The protocol determines a unique master among a group of clocks using the Best Master Clock algorithm (BMC). The BMC selects the most stable and accurate clock.

B. Sync Messages

PTP masters send Sync messages. The master records the send time of Sync messages (t_1), and slaves record the receipt time (t_2). The difference between the send and receipt times of Sync messages is the master-to-slave delay (d_{m2s}):

$$d_{m2s} = t_1 - t_2. \quad (2.1)$$

Sync messages are sent once per Sync interval (T_{sync}) (typically 2 s). This makes the master-to-slave delay sampling period (T_{m2s}):

$$T_{m2s} = T_{\text{sync}} = 2 \text{ s}. \quad (2.2)$$

C. Delay Request Messages

PTP slaves send Delay Request messages. Slaves record the send time of Delay Request messages (t_3), and the master records the receipt time (t_4). The difference between the send and receipt times of Delay Request messages is the slave-to-master delay (d_{s2m}):

$$d_{s2m} = t_3 - t_4. \quad (2.3)$$

Delay Request messages are sent on intervals uniformly distributed between 2 and 30 Sync intervals. This makes the slave-to-master delay sampling period (T_{s2m}):

$$T_{s2m} = T_{\text{sync}} * U[2,30]. \quad (2.4)$$

D. One-Way Delay

PTP calculates an estimate of the message propagation delay. This calculation assumes symmetric propagation delays, so that an average of the master-to-slave and slave-to-master delays cancels the time offset between master and slave. This yields the message propagation delay, which the specification refers to as the one-way delay (d_{prop}):

$$d_{\text{prop}} = (d_{m2s} + d_{s2m})/2. \quad (2.5)$$

Assuming symmetric propagation delays is often, but not always, valid. Asymmetric propagation delays cannot be observed by the protocol. They will cause a constant bias in the one-way delay and, in turn, the overall time coordination. The bias will equal half of the magnitude of the delay asymmetry.

Assuming a constant delay asymmetry, an asymmetric delay bias can be eliminated by adding a latency correction to the master-to-slave or slave-to-master delay that cancels the asymmetry; however, assuming constant delay asymmetry also may be invalid.

E. Offset From Master

PTP estimates the time difference between master and slave clocks. This is the master-to-slave delay corrected for message propagation delay, and it is referred to as the offset from master (Δt):

$$\Delta t = d_{m2s} - d_{\text{prop}}. \quad (2.6)$$

III. PTPd IN BRIEF

A. Background

The Precision Time Protocol daemon (PTPd) is a software-only PTP implementation. It was developed by two

engineering students at Case Western Reserve University over a period of approximately six months as part of an undergraduate senior project.

B. Test and Measurement

PTPd is currently developed for Test and Measurement (T&M) systems. For T&M devices (e.g., volt meters and thermocouple instruments), PTP provides time and frequency coordination for the time-stamping of acquired data, and PTP provides a common time-base for time-triggered data acquisition.

The needs of T&M systems significantly influence the current design of PTPd's clock servo. Most notably, the servo is optimized for the stable network topology typical of test and measurement set-ups.

C. Hardware Constraints

PTPd is a software-only system. It lacks two notable systems found in hardware-supported implementations. First, PTPd uses software time stamps. It records message send and receive times in the software layers of the network stack rather than in the physical layer of the networking hardware (e.g., snooping the MII bus of an Ethernet PHY [2]). Second, PTPd uses a software clock. It adjusts the magnitude of the periodic increment of a time quantity stored in memory. However, PTPd was outfitted with a hardware clock for the tests included in this paper. This was done to allow the clock to be read with minimal jitter by isolating jitter in clock reads from jitter in clock coordination.

PTPd is intended for embedded computer platforms that have minimal computing resources. This includes platforms with sub-100MHz CPUs. The program's CPU utilization is below 1% on a 66 MHz m68k processor, as observed by standard resource utilization monitors like the UNIX `top` utility. Also, PTPd does not require a Floating Point Unit (FPU), or FPU emulation, because it uses only fixed point arithmetic. Efficiency and limitation to fixed-point arithmetic are significant considerations in the design of the clock servo.

D. Software Constraints

PTPd is currently ported to Linux. Most of the PTPd system, including the protocol stack and the clock servo, runs as a background user-space process. This allows PTPd to "play nicely" in typical multi-task computing environments. PTPd relies on simple kernel-space routines for its timely components: the frequency adjustable clock and the message time stamps.

PTPd interfaces with the kernel through standard Linux system calls. Receive time stamps are recorded in the Network Interface Card (NIC) driver, in or close to the receive interrupt handler. The receive time stamps are passed to user-space through an `ioctl()`. The receive time stamp mechanism is included in vanilla (unmodified) Linux version 2.4 and 2.6 kernels. A similar send time stamp mechanism is not included in vanilla Linux kernels, but kernel send time stamps can be added to Linux with only small modifications. The entire modification typically amounts to less than ten lines of code. PTPd can operate acceptably without kernel send time stamps, but it performs better with the lower jitter afforded by kernel send time stamps, especially under heavy CPU loads.

PTPd uses the Linux kernel's software clock along with the `adjtimex()` interface for clock tick-rate adjustment. Linux's clock is an implementation of the hybrid kernel Phase-Locked Loop/Frequency-Locked Loop (PLL/FLL) designed by David Mills for the Network Time Protocol (NTP) project [3]. The interface provides many types of clock adjustments, including a self-tuning PLL servo; however,

PTPd uses its own servo loop and relies on only `adjtimex()` frequency adjustment. This combination is effective because the user-space servo is efficient and is not sensitive to execution latency, and `adjtimex()` is accurate and responsive to rate adjustments.

Vanilla Linux is not a real time operating system (RTOS); therefore, it guarantees no bounds on interrupt servicing latencies. Both message receipts and clock ticks are interrupt driven events. Variations in interrupt latencies create jitter in the delay estimates that PTPd uses to coordinate clocks. Jitter presents the greatest challenge to precise time coordination, and it is the most significant consideration in the design of the clock servo.

IV. CLOCK SERVO

A. Overview

Figure 1 is a diagram of PTPd's clock servo. The diagram from left to right shows the data path from the protocol to the clock. The protocol regularly samples the master-to-slave delay (cf. Equation (2.2)), and it intermittently samples the slave-to-master delay (cf. Equation (2.4)). Correspondingly, the offset from master is updated regularly, and the one-way delay is updated intermittently. The figure shows the delay and Sync interval inputs, the offset and one-way delay calculations, the offset and one-way delay filters, and the PI controller that mediates the servo output. The output is a fractional tick-rate adjustment that disciplines the clock.

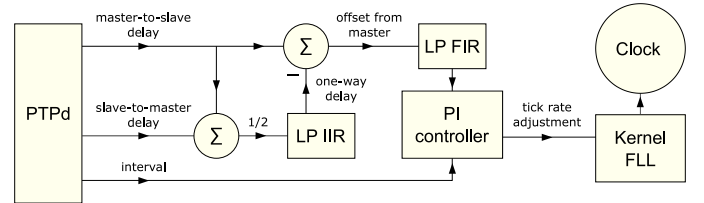


Fig. 1. Clock Servo Diagram

B. Design Parameters

Three characteristics were considered during the design PTPd's clock servo. First is the closed-loop response, including convergence and stability. The acceptable period of initial convergence is on the order of minutes, and the quantity tracked by the servo changes slowly. This allows convergence to be attained and maintained with conservative controller tuning, and conservative tuning largely eliminates stability concerns.

The second characteristic is time error. This represents the time-dependent applications that require two clocks to read the same time at any given point in time. An example of this requirement would be two systems that must take a measurement at precisely the same time. Another example would be two systems that must precisely measure the coincidence in time of two events. A useful metric of time coordination is the root-mean-square (RMS) time difference between clocks.

The third characteristic is rate error. This represents the time-dependent applications that require two clocks to progress at the same rate over a given period of time. An example of this requirement would be a system that measures the frequency content of a signal. It might seem that low rate error must follow implicitly from low time error, but this is not so. A servo design that minimizes time error may sacrifice rate error, and vice versa. This could occur with an aggressively tuned servo that tracks closely but with a lot of ringing, and the converse case could occur in a sluggishly

tuned controller that tracks with a significant offset but is noiseless and steady over short intervals.

A useful metric of rate error over a given period is the modified Allan variance [4, 5] versus summation time (herein referred to as variance versus time scale). The relative tick-rate between clocks typically exhibits three modes of variance: a minimum variance at some medium time scale (typically nanoseconds to many seconds) with increasing variance for small and large time scales. The increasing variance for small time scales represents jitter in the physical oscillator driving the clock. The increasing variance for large time scales represents wander between oscillators caused by changes in the tick-rate due to supply voltage or ambient temperature changes. Clock discipline typically aims to correct oscillator wander and cannot correct oscillator jitter. Ideally, clock discipline should not corrupt the naturally low oscillator variance on medium time scales.

C. Clock Servo Input

The following plots provide a rough picture of the input to PTPd's clock servo. Figure 2 plots PTPd's offset estimate versus master clock time over a roughly one-hour run, and Figure 3 plots the relative tick-rate estimate (the first derivative of the offset in master clock time) versus master clock time. The offset was sampled without PTPd performing any clock discipline. PTPd was a slave to a hardware-supported PTP implementation that achieves sub-microsecond precision. PTPd was running on a 66 MHz m68k embedded Linux platform, with kernel send and receive time stamps.

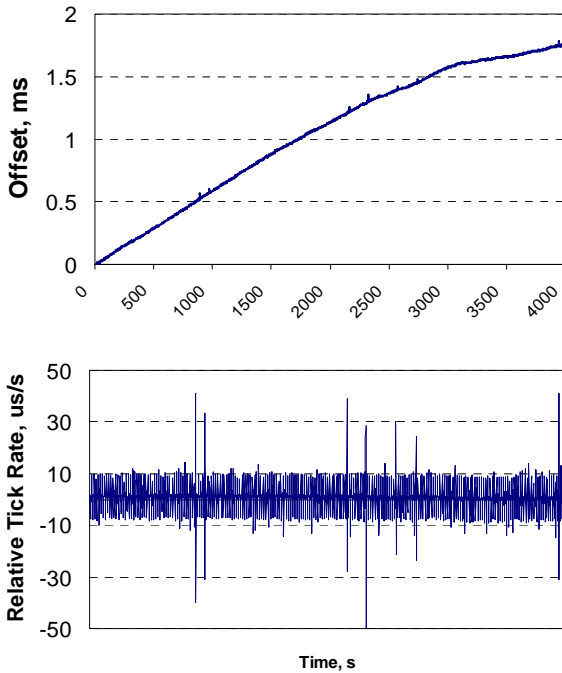


Fig. 2 (top) and Fig. 3 (bottom). Clock Servo Input

The offset signal in Figure 2 is typical of two undisciplined clocks. They drift away from each other in a nominally linear fashion due to inherent tick-rate differences, with some slight curvature due to variations in ambient conditions, including temperature [6].

Figure 3 reveals the microsecond-order noise in the offset signal that is obscured by the large magnitude of the signal in Figure 2. Figure 3 shows two modes of noise. One mode is persistent, high frequency, lower energy noise. Another mode is intermittent, higher energy impulse noise. The noise is not

an artifact introduced by the protocol or PTPd because the same modes of noise are exhibited in offsets sampled with an interrupt time stamped master-to-slave pulse-per-second (PPS), one of the simplest means of sampling clock offsets. The persistent noise is likely due to the nominal level of interrupt servicing latency jitter. The impulses may be due to interrupt latencies from extremely long periods of time when interrupts are disabled, or they could be due to periods of delayed execution due to bursty CPU or interrupt loads. Both of these sources of jitter are common in a non-RTOS.

Overall, the noise might appear small because it is orders of magnitude smaller than the long term time loss; however, the 10-30 $\mu\text{s/s}$ noise is orders of magnitude larger than the roughly 0.5 $\mu\text{s/s}$ tick-rate difference that the clock servo must extract from the offset signal to discipline the local clock.

D. PI Controller

The clock servo inputs the offset from master signal into a Proportional-Integral (PI) controller to produce a fractional tick-rate adjustment that coordinates the local clock with master clock time. The PI controller corrects both the time and rate of the local clock. The proportional term tracks and corrects the direct input, which is the time difference between two clocks. The integral term tracks and corrects steady-state error, which is the rate difference between two clocks.

The PI controller approach works well in terms of time error. The controller will drive the time error to zero in stable operation, and there are many analytical tools to optimize PI controller tracking.

The PI controller approach also works fairly well in terms of rate error. The controller tracks just as closely over short intervals as it does over long intervals. This characteristic is effective for correcting oscillator wander, which pushes the Allan variance to zero for long time scales. However, a problem with the PI controller approach arises on medium time scales. The PI controller attenuates noise in its input, but some noise will pass through to its output. This will increase the Allan variance for medium time scales. This problem is often the motivation for windowed and non-linear clock servos [7].

E. Filters

The clock servo uses filtering to mitigate the detrimental effect of input jitter on clock coordination. The filtering attenuates noise in the clock servo input to keep it out of the controller, which keeps jitter out of the clock.

What must be filtered out of the input signal is the persistent noise and the impulse noise described previously. The clock servo uses low-pass filters to attenuate input noise. Low-pass filters are reasonably effective in discriminating between noise and good input. This is because much of the energy in the input signal is close to zero frequency (within the pass-band of a low-pass filter), whereas much of the energy of the input noise is at higher frequencies (within the stop-band of a low-pass filter).

Low-pass filtering is a useful but problematic component of the clock servo. Typically, noise does have low frequency energy that can pass through low-pass filters (e.g. impulses, which have an even energy distribution throughout the frequency spectrum). Lowering the cutoff of the filter attenuates more noise, but lower cutoffs incur greater filtering delays. Delays make the controller less responsive to wander, which increases the tracking error.

Another problem is that low-pass filters can be biased by colored noise. This could be caused by asymmetric jitter, and would result in a constant offset in the clock coordination. Such biases are typically not a problem because, as previously

described, constant offsets can be zeroed by adding a latency correction to the master-to-slave or slave-to-master delay.

The clock servo filters both the offset from master and the one-way delay. The offset from master filtering is only a simple, two-sample average:

$$y[n] = x[n]/2 + x[n-1]/2. \quad (4.1)$$

This is a Finite Impulse Response (FIR) low-pass filter with a rather high cutoff near the Nyquist rate, but it has minimal delay. This filter effectively attenuates high frequency noise, which the controller does not attenuate as effectively. The one-sample delay incurred through the filter introduces negligible tracking error.

The one-way delay filtering is more involved than the offset filtering. The one-way delay filter is a variable cutoff/phase, first-order Infinite Impulse Response (IIR) filter:

$$s*y[n] - (s-1)*y[n-1] = x[n]/2 + x[n-1]/2. \quad (4.2)$$

Figure 4 shows the one-way delay filter's frequency response, plotted from zero to the Nyquist rate.

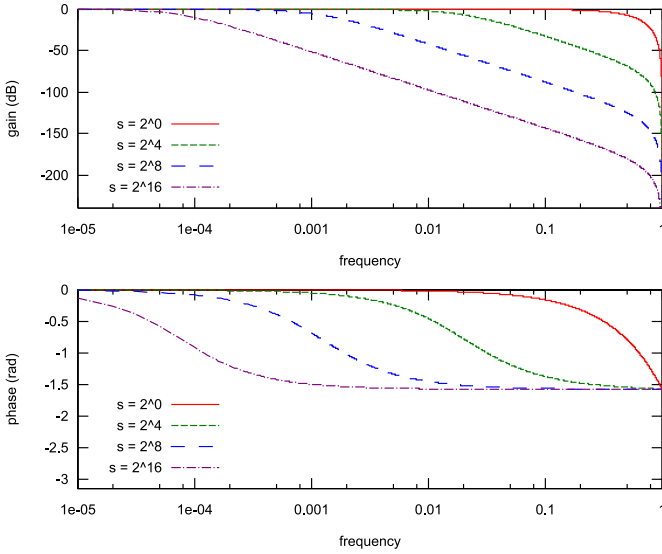


Fig. 4. Frequency Response of Equation (4.2).

For those that are more comfortable with statistical analysis than signal processing, the one-way delay filter can be viewed as a modified exponential smoothing calculation. The standard exponential smoothing form is modified for fixed point arithmetic, and a two-sample average is added to improve the response characteristics at high frequencies.

The 's' term in Equation (4.2) controls the cutoff and phase of the filter, and the term is herein referred to as the stiffness. With a stiffness of one, recursion is eliminated, leaving only a two-sample average (a low-pass FIR filter). Increasing the stiffness lowers the cutoff, but increases the delay.

The clock servo uses this variable cutoff/phase to allow the filter to overcome initial filtering delays at start-up. The servo begins with a stiffness of one, and then increments the stiffness each sample until reaching some maximum stiffness. As the stiffness is increased, the filter cutoff is lowered, and the one-way delay signal becomes smoother.

PTPd's clock servo filters the one-way delay separately from the offset from master. This is for two reasons. The first reason is that the one-way delay signal has a lower nominal sample rate than the offset signal (cf. Equation (2.1-2.6)). The

one-way delay signal is therefore interpolated in the combined offset from master signal. This interpolation lowers the frequency of the one-way delay noise, which pushes more noise into the pass-band of the low-pass filters. Filtering the one-way delay directly eliminates the interpolation seen by the filter.

The second reason why the one-way delay is filtered separately is due to the one-way delay signal's having different characteristics than the offset signal. The one-way delay signal reflects the message propagation delay, and its characteristics depend upon the network topology. In the case of the typical T&M set-up, the one way delay is nominally close to constant. A constant one-way delay signal can be filtered through a low-cutoff, high-phase, low-pass filter without increasing the tracking error of the clock servo. This is because there is no time delay of a constant signal through a real filter.

Some applications may not offer a stable network topology; therefore, the one-way delay signal would not be nominally a constant. The current filtering scheme in PTPd's clock servo may not be appropriate for such applications. However, the general approach of treating the one-way delay separately from the offset from master would remain a useful approach.

V. TESTS

A. Test Set-up

PTPd is currently being developed for the VXI Technology EX1048 precision thermocouple instrument [8]. The EX1048 is a 66MHz m68k embedded Linux platform. The following tests exhibit PTPd running as a slave connected over an Ethernet hub (except where noted) to a hardware-supported master clock. The EX1048 was coordinated with a hardware-supported master clock because it is expected that a T&M set-up coordinated with IEEE 1588 will include a hardware supported master clock. The master clock for the test is an Agilent LXI IEEE-1588 Demonstration Kit. It is a non-production device made available to the LAN Extensions for Instrumentation (LXI) Consortium for IEEE 1588 testing. Information on the LXI Consortium is available at [9].

The Linux kernel receive time stamps are used, and the kernel is modified to add kernel send time stamps. The Linux software clock is replaced by a frequency adjustable hardware clock implemented in an FPGA. The hardware clock is able to latch the time of received pulses with sub-microsecond precision, but the time is recorded with only microsecond quantization. This is sufficient to test PTPd's coordination, which is on the order of microseconds. Again, the hardware clock is used to allow the clock to be read with negligible jitter by isolating jitter in the clock from jitter in the observation.

The clock coordination is observed by the slave clock recording the time of pulses-per-second (PPS) generated by the master clock. This yields a 1Hz sampling of the slave clock's time with respect to the master clock.

B. Filtering

Figure 5 shows time offset between master and slave with various levels of filtering in the clock servo. The top run shows the results of sending unfiltered input to the PI controller. The jitter in the input makes it through to the clock and results in a poor time base. The low frequency undulations are likely due to the large impulses in the input, and the higher frequency noise on top of the undulations is likely due to the persistent noise in the input.

The middle run uses the fully configured clock servo with filters, but with a one-way delay filter stiffness of one

(equivalent to a two-sample average). The offset from master is also filtered by a two-sample simple average. The high frequency noise appears slightly smoother, and the undulations seem slightly smoother as well.

The bottom run has a one-way delay filter stiffness of 2^6 , and the coordination is significantly smoother. Most notably, the large undulations have been cut down to small intermittent excursions. These excursions are likely due to impulse noise in the input that is not fully attenuated in the clock servo.

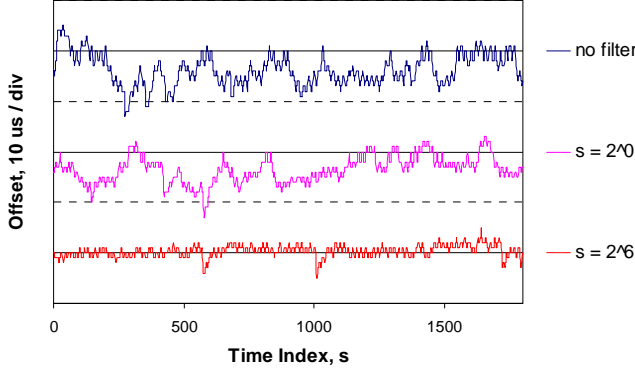


Fig. 5. Filtering Test

C. Convergence

Figure 6 shows the time offset between master and slave during the first ten minutes after PTPd starts-up and performs an initial clock reset. Figure 7 shows the next roughly hour-and-a-half after the initial convergence period.

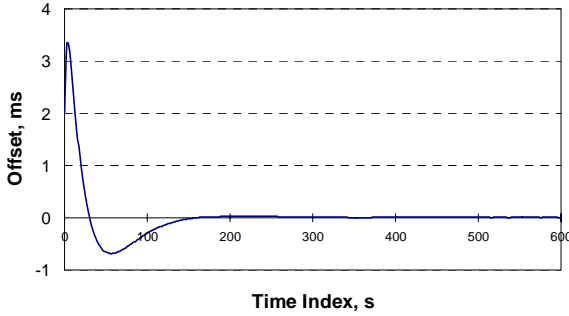


Fig. 6. Convergence Test, 0-10min

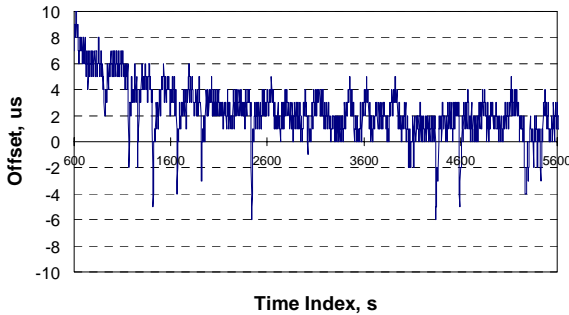


Fig. 7. Convergence Test, 10-90min

Figures 6-7 show that coordination is within $\sim 100 \mu s$ after roughly two minutes, and it is within $10 \mu s$ after roughly ten minutes. The response characteristics of the PI controller dominate the initial convergence because the one-way delay filter has low stiffness values during this period. The servo does not fully converge for about an hour. During this fine convergence period, the one-way delay filter stiffness is

increasing and the filtering delay of the one-way delay signal dominates the convergence.

D. Precision

Figures 8-9 shows two histograms of the time offset between master and slave after the clock servo is well converged. The histograms contain $1 \mu s$ bins with 50,000 offset samples at 1 Hz (almost fourteen hours). Figure 8 is from a test in which the slave was connected to the master through an Ethernet hub, and Figure 9 is from a test in which the slave was connected to the master through an Ethernet switch.

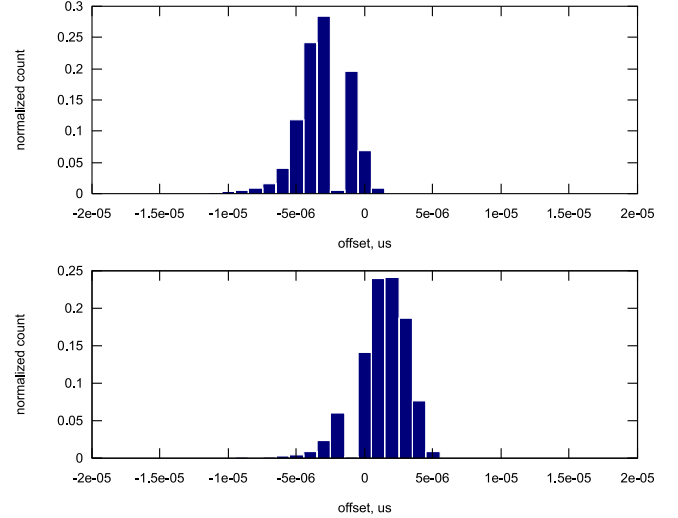


Fig. 8 (top) and Fig. 9 (bottom). Offset Histograms

The histograms show that the offset distributions for both runs are within $10 \mu s$. The offset distribution of the switch run is nearly as tight as the hub run. This indicates that jitter due to switch queuing is insignificant with respect to the slave's internal jitter. There is a bias in both of the distributions, but this is not a concern because the tight distribution indicates that the bias is stable; therefore, it can be eliminated with a latency correction as described previously.

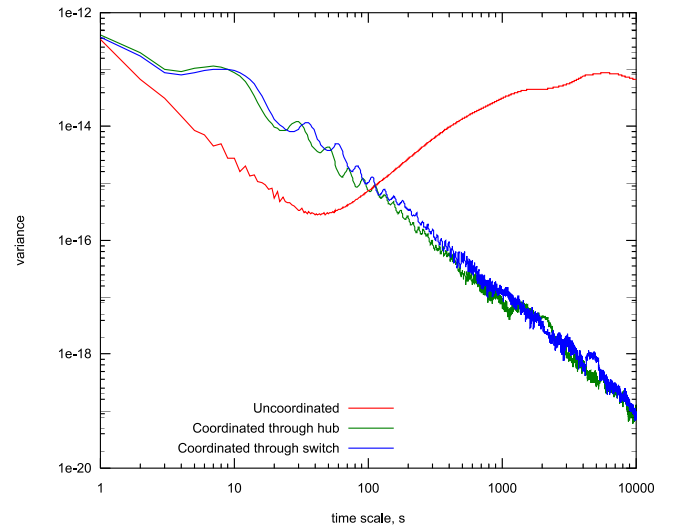


Fig. 10. Allan Variances

Figure 10 shows Allan variance plots of the same 50,000 sample runs versus the variance of an uncoordinated run. The

variances for the coordinated hub and switch runs are nearly on top of each other in the plot. The uncoordinated variance has the V-shape typical of uncoordinated clocks due to oscillator jitter on small time scales, a naturally low oscillator variance on medium time-scales, and oscillator wander on large time scales.

Figure 10 shows the advantage of using a PI controller. The variance of the coordinated clock goes to zero for large time scales. This indicates that the clock servo is properly correcting the wander between the oscillators, and it is a result of stable controller tracking.

Figure 10 also shows the troubles with PI controllers. The coordinated clock's variance on medium time scales (1-100 seconds shown) is larger than the uncoordinated variance. This is likely due to jitter in the offset estimate passing through the filters, into the PI controller, and in-turn into the clock. The jitter disrupts an oscillator that is naturally smooth on these time scales.

VI. CONCLUSIONS

A. Performance

PTPd coordinated the EX1048 with a hardware-supported master clock within 10 μ s. This precision comfortably exceeds the needs of the application in which sampling rates will not surpass 1 kHz.

PTPd can fill the needs of applications requiring sub-millisecond precision. PTPd exhibited coordination within ten microseconds on a platform with a slow (66 MHz), fairly busy CPU. It is reasonable to conjecture that PTPd could approach single-microsecond precision on a modern desktop platform with a more powerful (typically multi-gigahertz) CPU running under light CPU loads.

B. Future Work

PTPd is currently in the early stages of development. The clock servo is still quite simple and naive. PTPd's clock coordination precision could be increased with improvements to the clock servo design. Most notably, the noisy coordination on medium time-scales could be smoother. This could be addressed with the addition of a non-linear filtering element that could more effectively attenuate impulses in the clock servo input.

There are other improvements that also may be effective. The PI controller could benefit from improved tuning with the aid of formal analytical methods. The controller might also benefit from the use of gain scheduling. Finally, the one-way delay filter could be improved to accommodate unstable network topologies. The addition of some form of dynamic stiffness adjustment would keep the clock servo responsive to changes in the nominal one-way delay.

C. Open Source

PTPd is open source software. The source code is available under same BSD-style license as NTP. The project is hosted on SourceForge at ptpd.sourceforge.net.

VII. ACKNOWLEDGEMENTS

Michael Branicky was supported by the NSF (grant CCR-0309910). VXI Technology, Inc. has contributed to the open source community by supporting Kendall Correll and Nick Barendt during PTPd's development. Chad Greenebaum contributed to PTPd's initial development at Case Western Reserve University. Matt McConnell, a Digital Designer at VXI Technology, Inc., implemented the hardware clock used for PTPd's testing.

REFERENCES

1. IEEE Std 1588-2002. (for more information see <http://ieee1588.nist.gov/>)
2. Weibel, H., Béchaz, D., "IEEE 1588 Implementation and Performance of Time Stamping Techniques." *Proceedings of the 2004 Conference on IEEE 1588*, 2004.
3. Mills, D.L., and P.-H. Kamp. "The Nanokernel." *Proceedings of the Precision Time and Time Interval (PTTI) Applications and Planning Meeting*, 2000.
4. Allan, D.W., and J.A. Barnes. "A Modified 'Allan Variance' with Increased Oscillator Characterization Ability." *Proceedings of the 35th Annual Frequency Control Symposium*, 470-475, 1981.
5. Allan, D.W., N. Ashby, and C.C. Hodge. "The Science of Timekeeping." Hewlett Packard Application Note 1289, 1997.
6. Allan, D.W., et. al. "Precision Oscillators: Dependence of Frequency on Temperature, Humidity and Pressure." *Proceedings of the 1992 IEEE Frequency Control Symposium*, Report of Working Group 3 of the IEEE SCC27 Committee, 1992.
7. Veitch, D., Babu, S., Pásztor, A., "Robust Synchronization of Software Clocks Across the Internet." *Internet Measurement Conference*, 2004.
8. VXI Technology, Inc., <http://www.vxitech.com/>
9. LXI Consortium, <http://www.lxistandard.org/>