ISPCS 2008 International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication
Ann Arbor, Michigan, September 22-26, 2008

The Cost of Variability

Julien Ridoux, Darryl Veitch
ARC Special Research Center for Ultra-Broadband Information Networks (CUBIN),
An affiliated program of National ICT Australia (NICTA)
EEE Department, The University of Melbourne, Australia
{jridoux, dveitch}@unimelb.edu.au

***Abstract*— We explore the robustness of synchronization performed in the presence of variable latencies using two software clocks: the TSCclock, designed to replace ntpd for Internet synchronisation, and *ptpd*, a software implementation of IEEE-1588. Using a precise comparison methodology the TSCclock is shown to be more accurate and far more robust. We discuss the reasons why and the implications for IEEE-1588 more generally.**

***Index Terms*— clock synchronization, TSCclock, ptpd, IEEE-1588, latency, robustness**

I. Motivation

It is well known that variability in latencies between system components separating clock masters from slaves impacts on synchronization accuracy. The IEEE-1588 Precision Time Protocol (PTP) was developed primarily for use in all-hardware solutions requiring high accuracy, and in such a context, latencies are close to constant. Even so, the introduction of *transparent clocks* was motivated by the need to control error accumulation, ultimately a latency effect, in network devices. Much greater latency variability occurs when commodity hardware, or worse, software, enters in.

There is now increasing interest [1], [2], [3], [4] In PTP-compatible software implementations using commodity hardware in Ethernet LANs, motivated either by cost issues, or through a desire to allow inexpensive clients to immediately take advantage of PTP devices already installed in a network. From the other direction, software based clock synchronization is the standard solution in computer networking in general, given the prevalence of the Network Time Protocol (NTP) daemon *ntpd* [5]. Here the focus is on low cost but low accuracy (1 to 100's of [ms]), based on slaves accessing masters over the Internet, but it also functions over Local Area Networks (LANs) where performance can be below 1[ms].

Over the last few years [6], [7], [8] we have developed the *TSCclock*, a servo design and software solution we hope will replace *ntpd*, whose performance has been detailed in [8], [9]. In [10] we demonstrated that it can achieve 10μs accuracy over LANs and compares favorably with GPS synchronised *ntpd*. Another available solution is *ptpd* [1], a software implementation of PTP described in [2], where errors below 10μs were also reported over a LAN.

In this paper we first compare the performance of the TSCclock and *ptpd*. Our second aim is to highlight through these examples the potentially destabilising effect of large latencies, the consequent need for robustness, and the danger in ignoring the 'servo side' of IEEE-1588 implementations.

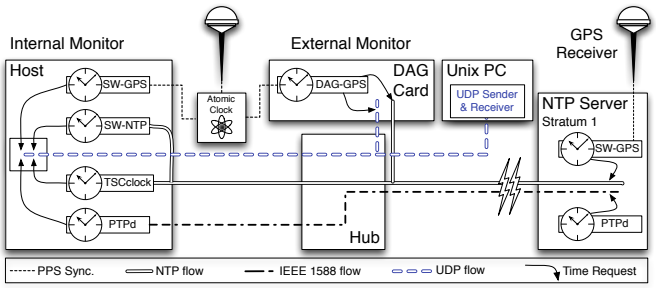

Fig. 1. Testbed. Timestamps, triggered by UDP packets, are taken internally by slaves in the Host, and externally in the External Monitor.

II. Experimental Setup

Figure 1 overviews testbed components. It shows a GPS synchronised PC acting as a time server (right), a host containing clocks to be evaluated/compared (left), an external monitor consisting of a GPS synchronised (with atomic clock corrected PPS) DAG capture card [11] using an Ethernet hub as a tap, and a UDP packet sender/receiver providing triggers for timestamping both in the host and external monitor. See [9] for a more detailed description.

We use a configuration with two hosts and one server PC:

Server: potoroo is both a GPS stratum-1 NTP server for the TSCclock, and runs the *ptpd* PTP master (which serves PTP requests but takes its time from the same NTP master).

Slaves: wallaby (FreeBSD 5.3) and **bettong** (Linux 2.6.20) run both the TSCclock and *ptpd* slaves (we access *ptpd*'s clock via the PC's system clock, which is slaved to *ptpd*).

Network: There are 2 switches between **potoroo** and the monitoring hub to which **wallaby** and **bettong** are attached.

Both the TSCclock [12] and *ptpd* software ([1]) are easy to install, run on both Linux and BSD Unix, and benefit from customised kernel timestamping. We use default parameters for each (no manual tuning), and set the server polling-period for each to 16[sec]. Note that IEEE-1588 does not specify the kind of servo controller used. It is common practice however to use a PI controller, and *ptpd* is no exception. In contrast, the TSCclock is *feedforward*, not feedback based.

III. Servo Design

We briefly outline here the key differences in approach between *ptpd* and the TSCclock, so that the experimental results may be better understood.

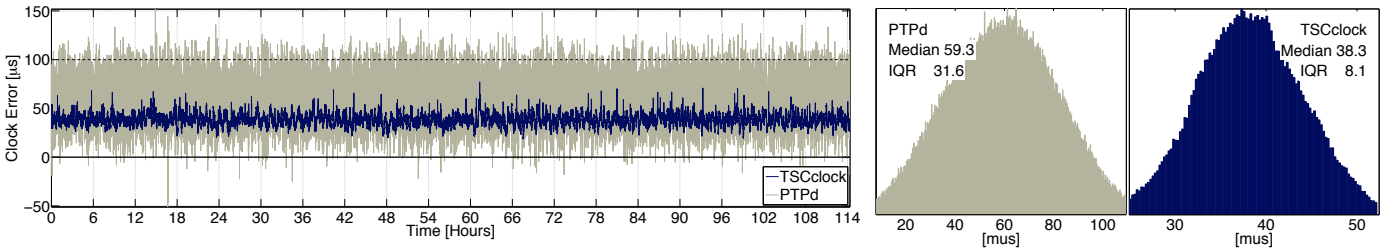

Fig. 2. Ideal environment: **bettong** by itself with minimal host and network load. Left: clock errors using external monitor, Right: histograms.

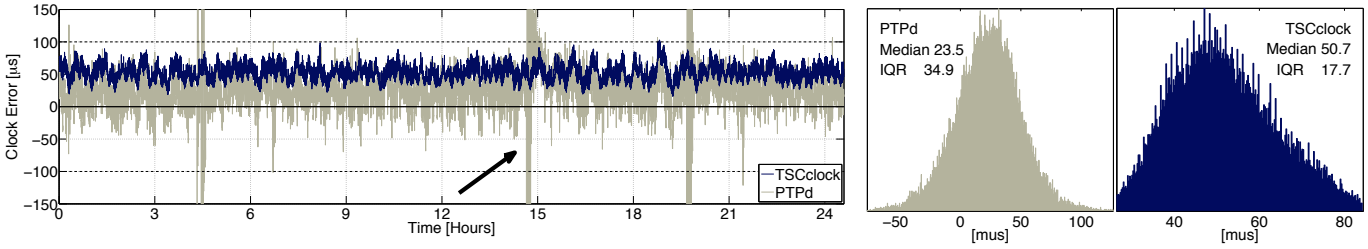

Fig. 3. Results for **wallaby** after it joins **bettong** on the hub. Spikes in host noise generate instability in *ptpd* (arrow marks event examined in Figure 4).

As described in [2], *ptpd* is based on a feedback control mechanism, more precisely a Proportional-Integral (PI) controller, which simultaneously tries to correct both clock rate and clock offset (or difference from true time). One disadvantage of this and other PID controllers is the fundamental tradeoff, inherent in their design, whereby parameter settings which improve short term tracking do so to the detriment of rate stability over the time-scales at which tracking operates, and in extreme cases threaten even the stability of the system as a whole. The need for global stability is paramount, which results in parameter values giving cautious initial convergence, and errors in clock rate (called 'noise in medium time scales' in [2]) which are due to the servo, not the underlying oscillator.

The TSCclock is feedforward based, meaning that it does not rely on a feedback loop 'locking onto' the input signal (indeed its operation is entirely asynchronous), but instead post-processes timestamps to estimate the current offset due to clock drift. It uses these estimates to remove offset error *only when* it delivers a timestamp to the user, the estimates are not used 'internally'. Underlying instability is therefore impossible. Furthermore, because of the nature of the nonlinear filtering used (see [7], [8] for details), timestamps fortunate enough to carry low noise can immediately be used to improve the estimate of offset error, which is of immediate benefit to subsequent timestamps. The result is almost immediate convergence both initially and throughout operation, and very fast recovery from periods of high noise.

IV. Results

The first set of results in Figure 2 show the error over time (evaluated using the external monitor) of each slave clock in **bettong** under ideal conditions: no other machine on the hub; negligible network traffic; very light host load. The time series show very consistent performance for each clock, however the TSCclock is considerably less variable, with an Inter-Quartile Range (IQR) of 8.1μs compared to 31.6μs for *ptpd*. To assess median performance we must compare it against the theoretical limit of $A/2$, where A is the path asymmetry between the slave and master. We measure the network component of A as $A_n = 28\mu$s, yielding a net median error of 24.3μs for the TSCclock and 45.3μs for *ptpd*. The asymmetry estimate can be futher improved by including the host component A_h (see [8]) but for space reasons we omit this and focus exclusively on variability below. The findings for *ptpd* are worse than those from [2], namely median errors under 10μs and an IQR of around 5-10μs. Details of how these were obtained were not provided. We use the evaluation methodology of [9] and apply it uniformly to each clock.

While keeping **bettong** running, we next add **wallaby** to the hub. Again the hosts are minimally loaded as is the network. The results in Figure 3 For **wallaby** are roughly similar to before although variability increases for each clock, since **wallaby** has a higher system noise. Note that it appears from Figure 3 that the TSCclock has higher median error, however since asymmetry effects have not been removed here, no such conclusion can be drawn. As mentioned above we focus on variability in this paper.

Before leaving this benign environment we examine two interesting points with the help of Figure 4. The left plot shows a zoom on the startup phase of both clocks for **bettong**. It

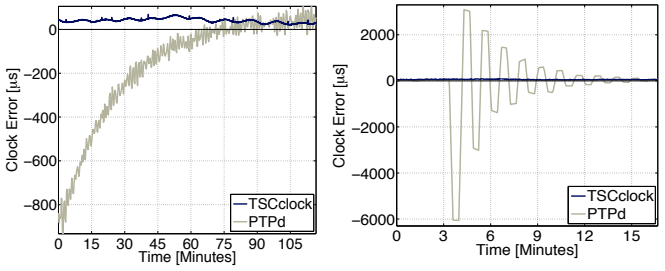

Fig. 4. Left: zoom on clock startup for **bettong** Right: zoom on reaction to host noise event for **wallaby** (marked by arrow in Figure 3).

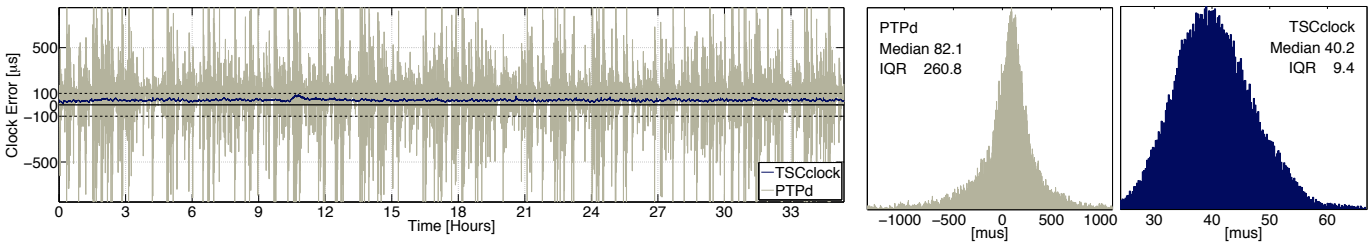

Fig. 5. Results for **bettong** with network congestion added (but still low host load). The TSCclock is basically unaffected, *ptpd* is strongly affected.

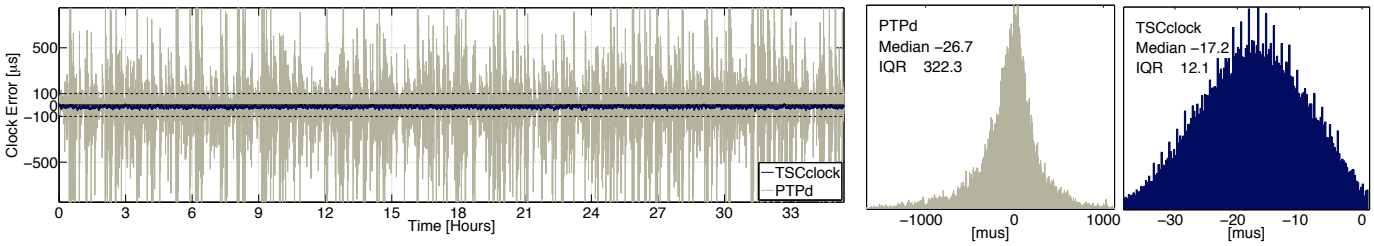

Fig. 6. Results for **wallaby** with both network congestion and high host load. Both clocks are affected, but TSCclock much less so.

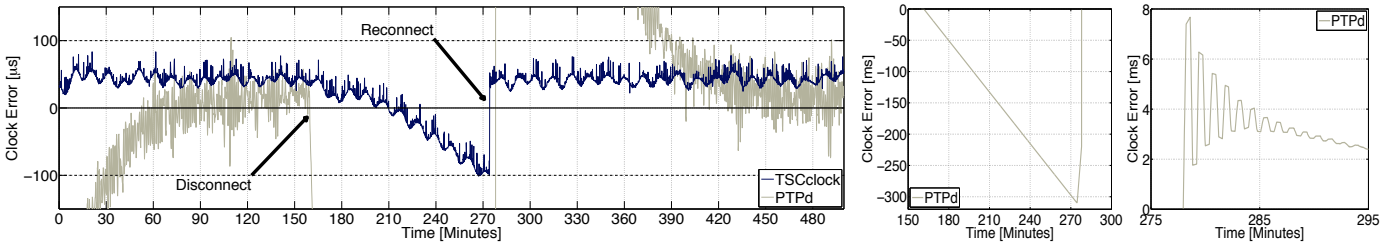

Fig. 7. Robustness test: disconnection from server. The TSCclock drifts gracefully, *ptpd* does not. Right: zooms on *ptpd* disconnection and reconnection.

takes an hour and a half for *ptpd* to converge whereas the TSCclock achieves nominal performance almost immediately. The right plot is a zoom-out (vertically) and in (horizontally) on the spike pointed to by the arrow in Figure 3. The triggering event for this spike is caused by operation system effects and is typical of FreeBSD – no such spikes were seen on Linux (it is likely due to implementation differences of the sockets used in the kernel timestamping). What is of interest here is how the clocks react to such an unusually 'large' system 'noise' input. In *ptpd* the event resulted in a very large jump in error and a slow, oscillating return to synchronisation. In contrast, the TSCclock is unaffected.

We next increase network congestion by causing traffic to be exchanged between **wallaby** and a third host placed on the monitoring hub. We repeatedly transfer a 750MB file simultaneously to and from **wallaby** via the UNIX *scp* command, separated by 10[sec] pauses. Each transfer is capped to 15000 kbit/s, corresponding to a total average load of around 30Mbps, and lasts around 7 minutes.

Figure 5 Shows the performance of the clocks on **bettong** under this scenario of high network congestion, but low host load. Comparing against Figure 2, We see that the TSCclock is barely affected, but *ptpd* suffers significantly, in particular in terms of its variability (IQR).

Figure 6 shows the performance on **wallaby** during the same experiment. Note that **wallaby** experiences not only significant network congestion but also significant system noise because it is the origin and destination of the traffic. We again find that the IQR for the TSCclock is barely affected when comparing either against Figure 5, or against the same machine under very low load (Figure 3, in fact it is lower than that 17.7μs found in that case!), whereas the IQR performance of *ptpd* degrades by a further 61.5μs from the already poor result of Figure 5. Thus the performance of the TSCclock under heavy load is far better than the performance found for *ptpd* even under the ideal conditions of Figure 2.

Note that under these high load conditions, the noise polluting the external comparison also increases significantly, and in fact dominates the actual clock error in the case of the TSCclock. This noise is in fact more severe on the incoming side. Thus, whereas in the other plots we showed errors as evaluated using the incoming direction, in Figure 6 we have instead given performance using the external comparison from the outgoing direction. Even the outgoing direction suffers from increased noise however in this more difficult environment, which means that actual clock performance is better than the results quoted above. Note that the change from incoming to outgoing brings with it a shift in median related both to asymmetry and the constant component of the external comparison noise, which we have not attempted to correct for here.

Finally, we examine the robustness with respect to a serious,

but quite common, network event: a loss of connection to the time server. We return to the light load scenario, and first allow each clock to converge. We then disconnect the monitoring hub from the network for about 2 hours, then reconnect it. Figure 7 shows the impact on **wallaby**. The TSCclock shows a gradual drift, the inevitable result of a lack of access to a master, and immediate recovery upon reconnection. In contrast the reaction of *ptpd* is extreme. As the middle plot in Figure 7 shows, following the disconnection at 150 minutes *ptpd* dives to reach an error of -300[ms] before reconnection, after which its error remained in the [ms] range (rightmost plot) for most of the hour required for convergence.

V. Related Work

We briefly describe here two other works, in addition to [1], [2], which have examined the performance of PTP under noisy environments.

In [3] the impact of jitter, quantization, and temperature on synchronization performance of a set of slaves receiving PTP Sync messages from a Master in a line topology, is examined. Each of the above factors are considered using simple models, such as a constant rate error resulting from a temperature change. Error formulae are derived, and a Matlab based simulation study is also provided. There is no examination of the role of filtering to reduce the errors found, and no use of real data or more complex non-linear drift and noise.

In [4] the authors evaluate the performance of a simple two-parameter servo design in an OMNET++ simulator. A scenario is considered with a single Master and Slave, with timestamp variability introduced through sharing a switching element with simulated cross-traffic. Clock error is given as a function of the servo parameters. Again neither filtering nor realistic non-linear drift in the Slave are studied.

VI. Conclusions

We have shown that the TSCclock is much more robust to deviations from an ideal, low noise environment, than the *ptpd* implementation of PTP. Indeed, the TSClock has been shown to behave stably under far noisier conditions with servers across the Internet [8], [10]. The broader point we wish to emphasize here however is: *if even a single component along the path from master to slave contains latency variability and/or extreme events, then servo performance can be at risk*. More specifically (recall IEEE-1588 does not specify servo):
(i) Expensive master and/or transparent clocks are pointless if a slave servo is not robust to its environment,
(ii) PI controllers may perform badly with variable latencies,
(iii) A servo of the TSCclock type (feedforward, not feedback, based) could offer greater stability, and performance, for both software *and* hardware PTP implementations.

In this paper we have exploited the robustness of the TSCclock servo as it stands. This is a client side solution which can interwork with the existing network of NTP servers. To complete the task of replacing the NTP based system however several more steps are required. First a server side solution is needed, which should interwork with NTP clients as well as communicating with TSCclock clients in a more generic way. One of the problems with the existing NTP edifice is that monolithic way in which the client servo, server servo, and timestamp exchange protocol are integrated. The TSCclock is in fact an implementation of a generic *Robust Absolute and Difference clock* (RADclock) which happens to use the TSC register as a counter and a NTP server as a remote reference. The underlying RADclock algorithms are modular in nature and we envision the server design to be along the same lines.

References

[1] "The Precision Time protocol (PTP), ptpd," http://ptpd.sourceforge.net/.
[2] K. Correll, N. Barendt, and M. Branicky, "Design Considerations for Software Only Implementations of the IEEE 1588 Precision Time Protocol," in *ISPCS*. Zurich, Switzerland: IEEE Computer Society, October 10-12 2005.
[3] N. Chongning, D. Obradovic, R. Scheiterer, G. Steindl, and F.-J. Goetz, "Synchronization Performance of the Precision Time Protocol," in *Proc. ISPCS 2007*, Vienna, Austria, Oct. 1-3 2007.
[4] G. Giorgi and C. Narduzzi, "Modeling and Simulation Analysis of PTP Clock Servo," in *Proc. ISPCS 2007*, Vienna, Austria, Oct. 1-3 2007.
[5] D. Mills, "The network computer as precision timekeeper," in *Proc. Precision Time and Time Interval (PTTI) Applications and Planning Meeting*, Reston VA, December 1996, pp. 96–108.
[6] A. Pásztor and D. Veitch, "PC based precision timing without GPS," in *Proc. ACM Sigmetrics Conf. Measurement and Modeling of Computer Systems*, Del Rey, California, 15-19 June 2002, pp. 1–10.
[7] D. Veitch, S. Babu, and A. Pásztor, "Robust Synchronization of Software Clocks Across the Internet," in *Proc. ACM SIGCOMM Internet Measurement Conf.*, Taormina, Italy, Oct 2004, pp. 219–232.
[8] D. Veitch, J. Ridoux, and S. Babu, "Robust Synchronization of Absolute and Difference Clocks over Networks," *Accepted for publication, IEEE/ACM Trans. on Networking, to appear June*, 2009.
[9] J. Ridoux and D. Veitch, "A Methodology for Clock Benchmarking," in *Tridentcom*. Orlando, FL, USA: IEEE Comp. Soc., May 21-23 2007.
[10] ——, "Ten Microseconds Over LAN, for Free," in *Int. IEEE Symp. Precision Clock Synchronization for Measurement, Control and Communication (ISPCS'07)*, Vienna, Austria, Oct.1-3 2007, pp. 105–109.
[11] "Endace Measurement Systems," http://www.endace.com/.
[12] J. Ridoux and D. Veitch, "TSCclock Webpage."