Ten Microseconds Over LAN, for Free (extended)

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

***Abstract*—The status quo for timestamping in PCs is *ntpd*, which under general conditions is accurate to 1 ms at best. For precision applications this is inadequate, but it is a low cost solution which suits many generic applications. IEEE-1588 provides mechanisms for sub-microsecond accuracy, but to achieve this more hardware is needed. We have developed the TSCclock, which gives performance between these two, around 10 microseconds on LAN, sub millisecond beyond, but using commodity hardware. We benchmark the TSCclock to show its potential as an inexpensive yet accurate software clock, which could be used with IEEE-1588 for LANs, but has wider applicability as a replacement to *ntpd*.**

I. Introduction

Clock performance, like any system, is subject to tradeoffs of cost and ease of use. Consider the status quo for software clocks in PCs, the *ntpd* daemon [1], which is used to synchronize system software clocks via a time server hierarchy across networks. It is widely recognised that *ntpd* is inadequate for precision timekeeping, being limited to around 1 ms at best (see Figure 3 for an example) in typical configurations. However, for many applications this is sufficient, and it does enable a time server to be inexpensively accessed over networks ranging from LANs to the Internet.

Over LANs, IEEE-1588 implementations provide for far more precise synchronization, at the microsecond level to well below if hardware support is provided, such as when a hardware clock is embedded into a Network Interface Card (NIC). This large improvement in terms of clock performance brings with it two main constraints. First, the additional hardware comes at a financial cost, and makes legacy hardware incompatible with IEEE-1588. Second, since the IEEE-1588 protocol must be implementable in hardware, its design has to remain relatively simple. This fundamental constraint restricts the usage of the IEEE-1588 protocol to LAN islands.

There is a wide gap between the quality and constraints of these two timing solutions, *ntpd* and hardware based IEEE-1588[1]. This leaves open the prospect for a solution which lies in between that would be reliable, able to provide synchronization within or across LANs, whilst remaining inexpensive and compatible with legacy hardware. Over the last few years ([3], [4], [5], [6], [7]) we have developed a solution which takes this middle ground: exploiting the relatively high stability of commodity hardware to provide a robust clock with accuracy well above that of the *ntpd* solution, whilst retaining the low cost and ease of use of network based synchronization. The *TSCclock* has as its hardware basis the TSC register, available in common architectures, which counts CPU cycles. Its synchronization algorithms, based on a client-server paradigm, are effective in filtering out delay jitter from network elements and the host system, and use selective kernel modifications to dramatically improve timestamping performance at zero dollar cost.

To support the first release of the TSCclock over Linux and FreeBSD systems, in this paper we perform a careful benchmarking study of the TSCclock over LANs. Using weeks of live data, our results show performance over Ethernet LANs to be around 10 μs, a noise level essentially set by the jitter inherent in the multi-tasking operating system. Whilst not being able to compete with more hardware oriented solutions over LAN, this raising of the bar by an order of magnitude nonetheless opens up numerous possibilities for applications where cost is a factor. We illustrate the potential of the algorithm to go beyond the LAN setting with only a modest performance penalty.

This paper expands on earlier work presented at ISPCS 2007 in Vienna [8]. In particular sections IV-B, IV-C contain new results using new hardware and an improved methodology.

II. The TSCclock

The TSCclock [3], [4], [7] leverages the fairly high stability of commodity oscillator hardware, specifically the CPU oscillator, whose cycles are conveniently counted and readily accessible via the TSC register in common PC architectures. Contrary to the *ntpd* based software, the TSCclock does not actively vary its rate to track drift in a feedback loop [9]. Instead, it is built around obtaining stable long-term clock rate estimates using a feedforward approach. This enables it to provide two clocks, one for time differences, and one for absolute time. This approach circumvents the usual tradeoff problem, for example in PID controllers, whereby gain parameters which improve short term tracking do so at the detriment of rate stability over the time-scales at which tracking operates.

The synchronization algorithm operates in client-server mode. Each round-trip generates four timestamps, two in `timeval` format from the server (taken at times t_a and t_f in Figure 1), and two raw TSC timestamps taken at the host (taken at times t_b and t_e in Figure 1). These timestamp 4-tuples are used to provide an estimate $\hat{p}$ of the average CPU oscillator

[1]Software implementations exist [2], however, the very high accuracies commonly quoted for IEEE-1588 refer to hardware implementations.

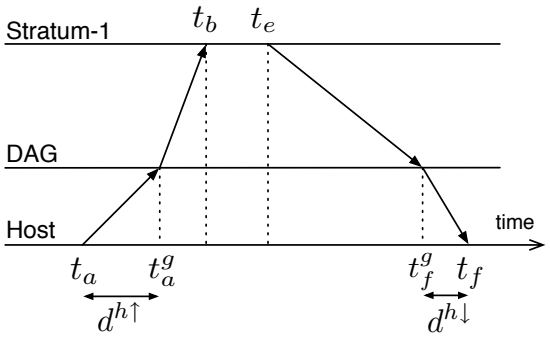

Fig. 1. Description of an NTP round-trip packet exchange showing the packet departure (t_a, t_e) and arrival (t_b, t_f) event times. The DAG card provides reference timestamps near the host, taken at times t_a^g and t_f^g, corresponding to t_a and t_f respectively. The *Host Round Trip Time* is $R^h = d^{h\uparrow} + d^{h\downarrow}$.

period p. An *uncorrected* clock which does not take clock drift into account can then be defined as $C_u(t) = \hat{p} \cdot TSC + K$, where K is an estimate of the initial offset which is **not** updated. Instead, an estimate $\hat{\theta}$ of the error in $C_u(t)$ is updated at each new incoming stamp. The two clocks are then:

$$\mathbf{C}_d(t): \quad \Delta(t) = C_u(t_2) - C_u(t_1) = \Delta(TSC) \cdot \hat{p}, \qquad (1)$$

$$\mathbf{C}_a(t): \quad C_a(t) = C_u(t) - \hat{\theta} = \hat{p} \cdot TSC + K - \hat{\theta}. \qquad (2)$$

The *difference clock* $C_d(t)$ does not incorporate the correction $\hat{\theta}$, so the constant K cancels exactly. Hence, $C_d(t)$ directly benefits from the underlying rate stability of the TSC over small to medium timescales, and is not perturbed by estimates of drift, which are irrelevant over those scales. For example, a rate stability of 1 part in 10^7 over a RTT of 1 s yields an error of only 0.1 μs.

Measuring absolute time requires that drift be tracked. Hence, $\hat{\theta}$ is incorporated into the definition of the *absolute clock* $C_a(t)$, which results in medium term variability since estimates must be based on a limited time window, and used even if not ideal (for example due to congestion). However, by not changing K, instead applying a correction only when reading, the absolute clock avoids varying its underlying rate to track drift, which improves stability and enables meaningful sanity checking.

The key problem in synchronising clocks over a network is the variable delays due to queueing in network elements, and interrupt, queueing and processing delays in the host and/or server hardware and operating systems. Whereas in systems using the *Precision Time Protocol* (PTP, ie. IEEE-1588), these can be reduced to near zero by the use of boundary clocks which are typically hardware based (see however [2]). The TSCclock is designed to be robust to large and highly variable delay jitter as it receives its timestamps via packets which travel across the network and back. Thus, apart from the feedforward design described above, the entire structure of the TSCclock synchronization algorithms is aimed at compensating for this jitter successfully, that is both accurately and robustly. If there were no jitter, then synchronization reduces to the calculation of propagation delays, which apart from path asymmetry issues (which are intrinsic and cannot be overcome by any algorithm), is trivial.

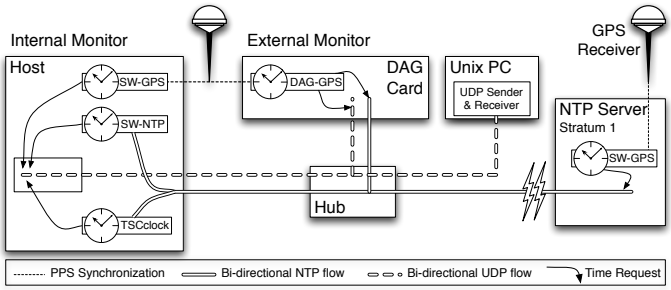

Fig. 2. Testbed: UDP packets are timestamped externally by DAG and internally by each of the *SW-GPS* or *SW-NTP* and TSCclock inside the host.

The TSCclock performs non-linear filtering based on round-trip times. It uses a simple but empirically very well justified model of round-trip times, namely a minimum constant plus positive random noise. This approach is extremely effective in identifying those packets which contain the best (smallest) network delays. The estimators of $\hat{p}$ and $\hat{\theta}$ above are based on these together with windowing for variance reduction. More precisely, the excess of RTT above an estimate $\hat{r}$ of the minimum RTT r is used as a basis of rejection of distorted timestamps when measuring $\hat{p}$, and as a weight when averaging estimates made over several packets in the case of $\hat{\theta}$. A more detailed account is given in [4], [7], including how to deal with changes in the minimum delay level, which can occur for example following changes to layer 2 or 3 routing. The resolution of the TSCclock is tied to that of the CPU, and is around 1 ns.

III. Testbed

Any software clock running on typical computer architecture and operating systems has to deal with system delay created by resource sharing and competition among running processes. As a general term, we refer to these delays as *system noise*. System noise affects the timestamping of any event of interest by delaying the reading of the clock. In other words, timestamps of events used to synchronize a software clock (arrival time of a reference time packet for example) or events used to assess clock performance, are all affected. Based on this simple observation, we designed a testbed to gain insights into system noise and so distinguish timestamping errors from actual clock errors.

Figure 2 shows our testbed, consisting of a PC host running three clocks: *SW-GPS* (*ntpd* synchronized to local GPS), *SW-NTP* (*ntpd* synchronized to stratum-1 NTP servers), TSCclock (synchronized to stratum-1 NTP servers), and a precision external reference, a 3.7GE DAG card [10] which timestamps packets in hardware without loss, even at high rate. The DAG card embeds its own hardware clock synchronized using a GPS Pulse-Per-Second (PPS) input with a nominal accuracy of ± 100 ns. Since the publication of [8] we have added a precision PPS source, a PRS10 Rubidium oscillator locked to a GPS PPS signal for long term stability. This PRS10 outputs an extremely stable PPS signal (5×10^{-6} PPM, [11]) which we use to feed both the *SW-GPS* clock and the DAG, improving

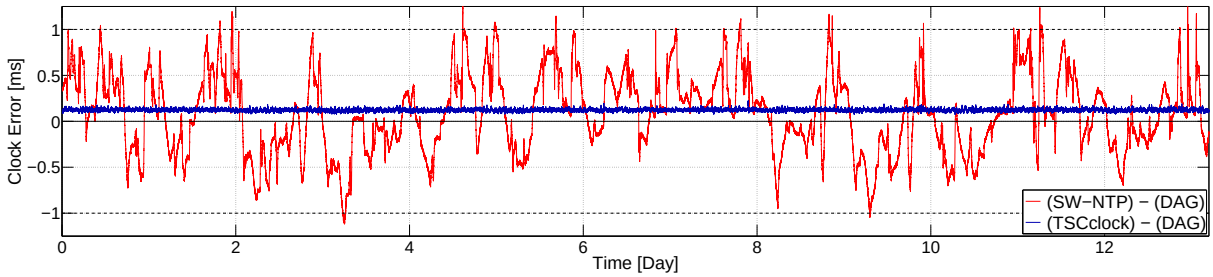

Fig. 3. Performance of *SW-NTP* a *ntpd* based SW clock (resp. TSCclock) using servers inside (resp. outside) the LAN.

the latter's timestamping error to $\pm 40\,\mathrm{ns}$ [12], comparable to IEEE-1588 specifications.

To assess the performance of each of the three clocks, UDP packets are exchanged between the Host and another Unix computer on the network. Packets are sent from the host monitor and for each one sent, the Unix PC replies with a similar UDP packet. This packet exchange constitutes the series of events used to trigger the reading of each of the clocks and translates into several series of timestamps which can be compared.

The comparison of the timestamp time-series made available by our testbed is twofold and we refer to it as the *Internal* and *External* comparisons. The Internal comparison relies on slight modifications to the Linux and FreeBSD kernels for the timestamping of each incoming or outgoing UDP packet. The main objective of these modifications is to be able to timestamp each packet event using two clocks in a "back to back" fashion. The packet timestamping is done using the packet capture library *libpcap*. The modifications applied to the Linux and FreeBSD kernel are different and are driven by the implementation of the kernel timestamping function made available to libpcap on these two platforms. On FreeBSD, the call to timestamping functions takes advantage of the Berkeley Packet Filter subsystem, but our modifications slightly improve the timestamping location for each of the clocks, moving them slightly closer to the network card driver code. On Linux kernels, timestamps are taken in the kernel, right after the driver code returns and are exported using RAW sockets opened by libpcap.

We implemented these new timestamping calls so that timestamps of the same event obtained from different clocks are taken one after the other, with no out-scheduling, and so share almost exactly the same system noise. As a result, the Internal comparison provides timestamps for which the clocks share the same timestamping error. When comparing corresponding timestamps from two different clocks the timestamping error cancels, revealing the relative performance of the two clocks under study. However, although free of timestamping error, this comparison does not provide an indication of absolute performance as none of the clocks used can be considered as references.

The use of the DAG card provides us with the External comparison that is absolute with regards to its embedded clock. However, this comparison is **not** free of timestamping noise since the packets do not reach the DAG at the same time as they pass the boundary between the NIC and the host operating system, which is the target event we aim to timestamp. These delay gaps between the target host event times and the corresponding ones at the DAG are shown as $d^{h\uparrow} = t_a^g - t_a$ and $d^{h\downarrow} = t_f - t_f^g$ in Figure 1. As extensively detailed in [6] the measurement of the sum of these, the "Host Round-Trip Time" $R^h = d^{h\uparrow} + d^{h\downarrow}$, provides information on these 'gaps' between the host and the DAG measurement, and helps us quantify host noise in general. There are two components to this gap/noise: a constant offset containing a fundamental ambiguity arising from asymmetry in the systems' handling of packet transmission and reception, and a variable component which in principle can be filtered out, but in practice can only be done imperfectly. More precisely, the asymmetry component is bounded by $2 \times \min(R^h)$ and implies the same bound on the clock error uncertainty, whereas R^h is the per-packet bidirectional noise that superimposes to the clock error measurement, producing a limit on the interpretation of the results. Note that similar issues, including asymmetry, hold within a software server.

Combining both Internal and External comparison we are able to give a lower bound on system noise at the host. We are also able to provide bounds on, as well as remove, the inherent ambiguity of the results due to the network asymmetry existing in the exchange of NTP packets. More details, a precise description of the methodology associated to the testbed, and the analysis of operating system noise and path asymmetry, can be found in [6].

IV. Experimental Results

In this section we present experimental results to illustrate how the TSCclock is able to fill the gap between the two timing solutions *SW-NTP* and IEEE-1588.

A. Performance in the NTP world

Figure 3 motivates our work and is a good starting point to illustrate the External comparison part of the methodology. It shows *ntpd* performance under ideal conditions: the *SW-NTP* clock running on the host being synchronized to a stratum-1 peer on the same LAN. Despite this, its absolute error (given by the External comparison using DAG packet timestamps) varies in a $1\,\mathrm{ms}$ band. In contrast, the TSCclock, synchronized to a stratum-1 NTP server *outside* the LAN, is over an order of magnitude smaller.

From the perspective of PTP, we are interested in seeing how well the TSCclock can perform in a LAN environment.

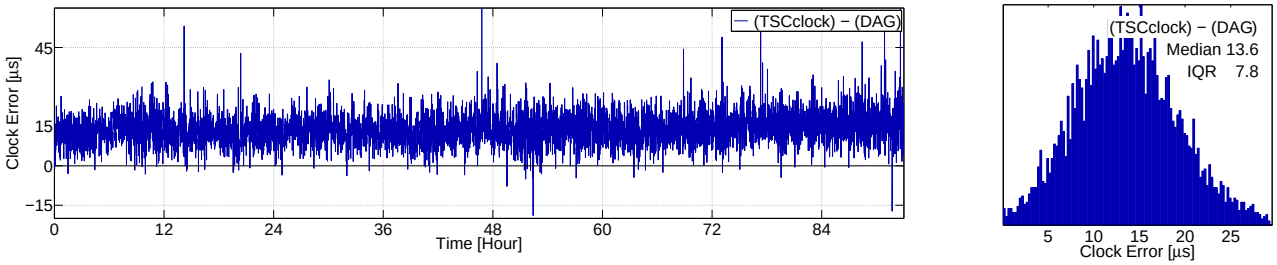

Fig. 4. TSCclock synchronized to stratum-1 server on LAN, polling period 256s, clock error using External DAG comparison. Left: time series over 92 hours. Right: normalised histogram (from 1st to 99th percentile), median = 13.6 μs, IQR = 7.8 μs.

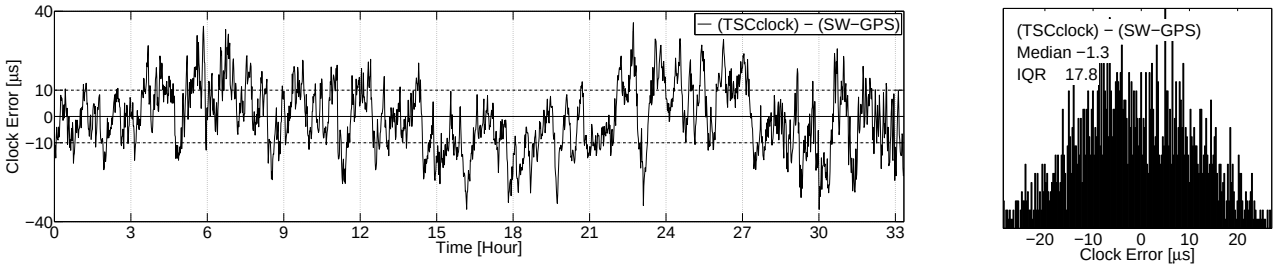

Fig. 5. TSCclock synchronized to stratum-1 server two hops away, polling period 16s, Internal in-host comparison against *SW-GPS*. Left: time series over 33 hours. Right: normalised histogram (from 1st to 99th percentile), median = -1.3 μs, IQR = 18 μs.

Figure 4 gives the absolute performance of the TSCclock over 92 hours, measured externally via DAG. In this experiment, the TSCclock is synchronized to a GPS synchronized stratum-1 server located on the same LAN. Again, the TSCclock relies on NTP packets to exchange time information with the server and to synchronize to it. The inter-quartile range (IQR) of this External comparison is only 7.8 μs, a value identical to the level of system noise measured on this particular host. The median error is only 13.6 μs. The performance of the TSCclock is then particularly good, especially with regards to the large polling period used in this experiment (256 seconds). We measured, however, an uncertainty of 104 μs on the median clock error value due to path asymmetry effects.

While this performance is still far from what is achievable using dedicated hardware, we believe that few software solutions reach this level of accuracy. Furthermore, the server used itself suffers from large system noise, as its timestamps are not created in the kernel and so suffer from operating system delays. We anticipate significant performance improvements if IEEE-1588 boundary clocks were used instead of stratum-1 NTP servers, and we explore this further below.

Next (Figure 5), we perform a direct comparison against a GPS synchronized software clock in the same host, using a modified kernel. We observe that the two clocks agree to less than 2 μs with a spread of 18 μs over the 33 hours. While this experiment is a strong challenge for the TSCclock (the quality of each clock synchronization source is radically different) the TSCclock performance is comparable to that of the GPS synchronized server.

This short section illustrates how the TSCclock outperforms *SW-NTP*, and acts as a proof of concept showing how the TSCclock can be easily integrated into the existing NTP world and take advantage of the existing stratum architecture for fast deployment. The comparison to a hardware-based solution such as *SW-GPS* highlights the impact of the synchronization source on TSCclock performance. The following section gives a taste of what can be achieved if one takes advantage of IEEE-1588 or similar synchronization sources.

B. Approaching the IEEE 1588 world – Typical server

Because typical NTP stratum-1 servers are computers built from commodity hardware, the system noise they undergo when processing NTP packet contributes to the degradation of the time information exchanged. While this 'server noise' may be neglected when synchronising to a server located several hops away, on a Gigabit Ethernet LAN, the server noise, minimum processing delay, and the RTT are all in the hundred(s) of microsecond range. Naturally, using a master clock as defined in IEEE-1588 should remove these server side errors and thereby improve TSCclock performance. However, we do not currently have such a master clock in our testbed.

A simple way to assess the impact of the NTP server is to use the DAG instead. In terms of the timestamps indicated in Fig. 1, this involves using t_a^g and t_f^g in place of t_b and t_e respectively. As mentioned before, the DAG card is a GPS synchronized hardware clock with similar characteristics to an IEEE-1588 master clock. Hence, this manipulation allows to replay TSCclock captured traces with the network and server defects removed. Such re-processed traces exhibit the TSCclock algorithm performance where the only inherent limitation is the system noise of the host it is running on.

To observe the impact of the above DAG 'pseudo-master', we first look at Internal and External comparisons of the TSCclock using the stratum-1 NTP server as normal. We then replace the server timestamps with the DAG ones as described above and compare.

Figures 6 and 7 show the TSCclock performance, synchronising to a GPS synchronized stratum-1 server located

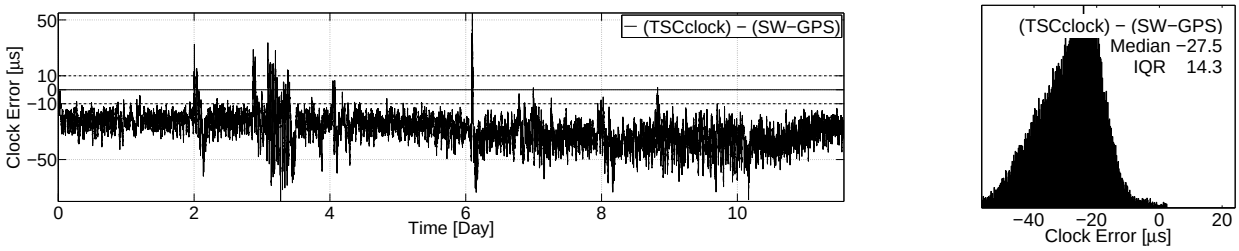

Fig. 6. TSCclock synchronized to stratum-1 server outside the LAN (2-hops away), polling period 16s, Internal in-host comparison against *SW-GPS*. Left: time series over 12 days. Right: normalised histogram (from 1st to 99th percentile), median = $-27.5\,\mu$s, IQR = $14.3\,\mu$s.

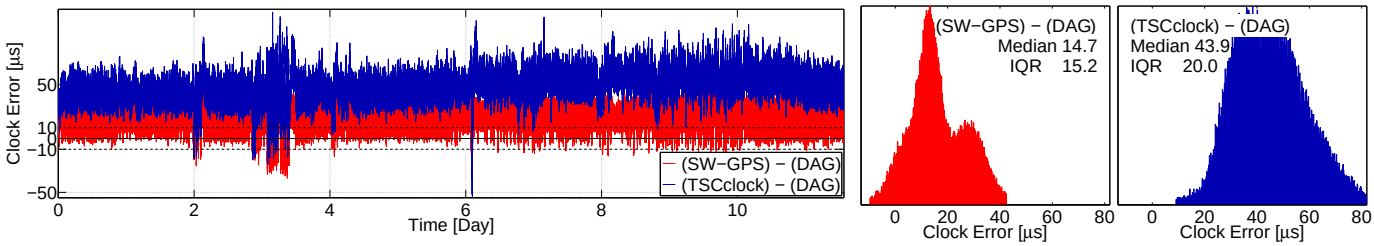

Fig. 7. TSCclock synchronized to stratum-1 server outside the LAN (2-hops away), polling period 16s and *SW-GPS* clock error using External DAG comparison. Left histogram: *SW-GPS*, median = $14.7\,\mu$s, IQR = $15.2\,\mu$s. Right histogram: TSCclock, median = $43.9\,\mu$s, IQR = $19.9\,\mu$s.

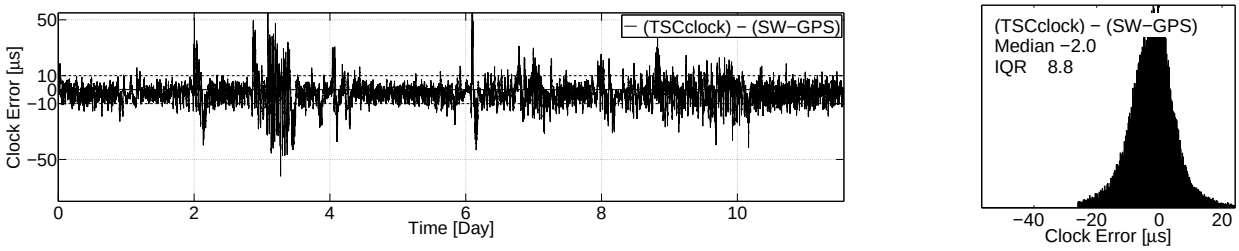

Fig. 8. TSCclock using DAG as a pseudo-master clock, polling period 16s, Internal in-host comparison against *SW-GPS*. Left: time series over 12 days. Right: normalised histogram (from 1st to 99th percentile), median = $-2.0\,\mu$s, IQR = $8.8\,\mu$s.

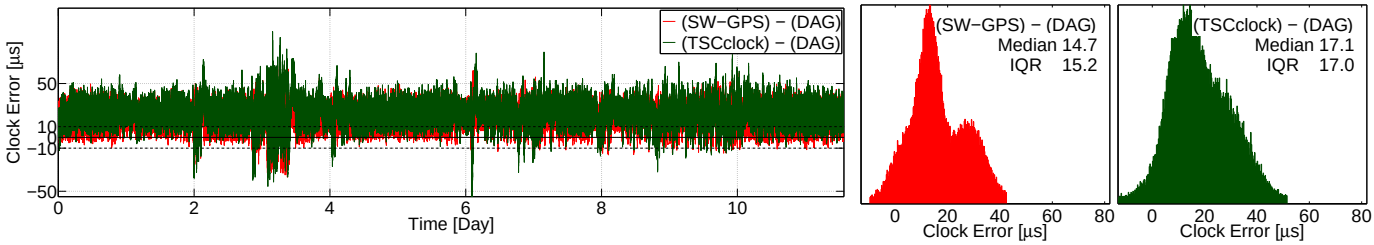

Fig. 9. TSCclock using DAG as a pseudo-master clock, polling period 16s, and *SW-GPS* clock error using External DAG comparison. Left histogram: *SW-GPS*, median = $14.7\,\mu$s, IQR = $15.2\,\mu$s. Right histogram: TSCclock, median = $17.1\,\mu$s, IQR = $17\,\mu$s.

2-hops away, against *SW-GPS*. Figure 6 shows the Internal comparison, namely TSCclock(t)-*SW-GPS*(t). The median difference between the TSCclock and *SW-GPS* is $-27\,\mu$s and the corresponding IQR is $14\,\mu$s. This is an honourable performance for the TSCclock as the server is **outside** the LAN. The External comparison of the TSCclock to DAG in Figure 7 is also good, with a median error of $44\,\mu$s and a spread of IQR=$20\,\mu$s. The unknown one-way delay incurred when the *SW-GPS* Processes the PPS signal obtained through the host serial port is the main contributor to its error. This delay cannot be measured but is reasonably estimated at around $10\,\mu$s.

A long term oscillation can be observed both in the Internal and External comparison for the TSCclock, but not *SW-GPS*. Because the TSCclock is designed to track and cancel the non-linear drift of the CPU oscillator, we suspect that this oscillation is due to the stratum-1 server itself (no sustained network congestion having been observed over this period). This hypothesis can be investigated by replacing the server timestamps with the DAG timestamps (figures 8 and 9). The results confirm our speculation, the large scale oscillation is absent when using the DAG card as a pseudo-master. The Internal comparison now shows that the TSCclock and *SW-GPS* only differ by $-2\,\mu$s with an IQR spread smaller than $9\,\mu$s, suggesting that the TSCclock essentially behaves as *SW-GPS* when using a master clock on the LAN. This conclusion

is consistent with the External comparison in Figure 9 showing extremely similar TSCclock and *SW-GPS* clock error characteristics when compared to DAG.

For the above experiment, the IQR of system noise has been measured at 11 μs, smaller than the IQR of the TSCclock (and *SW-GPS* also). This indicates that, despite the use of the pseudo-master clock (which improves the server timestamp quality but leaves the round-trip times unaffected) the server performance is still impacting slightly on the total error, and motivates using a server that is even better. This can be achieved by selecting a server on the LAN to reduce the RTT.

C. Approaching the IEEE 1588 world – Improved server

We now use a server on the LAN, and take advantage of our new Rubidium atomic clock. We use its PPS signal in three ways: to feed the *SW-GPS*, the DAG card, and to synchronize the *SW-GPS* of the NTP Stratum-1 server to which the TSCclock synchronizes.

Figures 10 and 11 show these new captured datasets. The Internal comparison (Figure 10) shows that the TSCclock and *SW-GPS* differ by a median value of $-14\,\mu$s with an IQR of $12\,\mu$s. This indicates a very clean dataset free of coarse stratum-1 server errors and significant network congestion.

The very low IQR values of the External comparison of the same experiment (Fig. 11), show that the use of the closer server and atomic clock improves the performance of both clocks. Note that the shape of the *SW-GPS* error distribution displays oscillatory behaviour (think sinusoid plus noise), indicating that the feedback based *ntpd* algorithm is not locking on. Such instability is of course highly undesirable, and is not present in the TSCclock. Note that the IQR of the bidirectional system noise on this host was measured at $62\,\mu$s, a large value, but despite it the variability of the TSCclock essentially matches that of *SW-GPS*. Regarding median performance, the *SW-GPS* is slightly behind the DAG reference clock by a median value of $-9\,\mu$s. The TSCclock is **apparently** more accurate being only $5\,\mu$s ahead of the DAG.

We now replace the NTP server timestamps of the trace by those from the DAG card. The Internal comparison using this pseudo-master is given in Figure 12. Interestingly, at $8.9\,\mu$s the IQR has not improved by any significant amount, indicating that server timestamps issues are in fact no longer the dominant source of error. The median results are far more interesting. The two clocks remain approximately the same distance apart, however the sign of the difference has changed! Since *SW-GPS* is unaffected by the use of the pseudo-master, this change is entirely due to the TSCclock which is now **apparently** behind the DAG reference.

The External comparison (Figure 13) confirms the Internal observations. The IQR values are marginally better but the medians have undergone a sizeable shift: the median TSCclock error is now $-21\,\mu$s compared to $5\,\mu$s previously. While we were expecting a large improvement we see instead an apparent performance degradation! In fact there is no contradiction in this. First, as we described above, the External comparison is subject to an uncertainty due to host noise, measured here at $32\,\mu$s, which is larger than the apparent shift. Moreover, the shift can be explained as a change in asymmetry. While the TSCclock performance of the original experiment is dependent on a total path asymmetry comprising host, server, and network components, the result obtained with the pseudo-master clock effectively 'cuts out' the network and server, leaving only the host component. As noted earlier the latter is determined by how the network adapter and corresponding driver implementation process outgoing and incoming packets, and is therefore highly hardware dependent, and cannot be measured here. However, as noted earlier it is bounded by $2 \times \min(R^h)$ which is measurable. As the NIC and its driver are responsible for the host asymmetry, it is important to choose them carefully in order to minimise R^h. Indeed, depending on the NIC, we have observed very different system noise characteristics, with a spread varying from IQR=22 μs for a RealTek 8139 Fast Ethernet card, to an IQR=85 μs for a Gigabit Ethernet Broadcom BCM5751 chipset (values observed on FreeBSD 6.1). Using the same two NICs, we have also observed the bound on the asymmetry varying from from $30\,\mu$s to $150\,\mu$s. The choice of NIC and driver is therefore essential to avoid losing the benefits of using a highly accurate master clock as server on a LAN.

Finally, it interesting to note that since the host and server hardware used in Figure 11 is similar, it is likely that the host asymmetry was originally partly compensated by the server asymmetry mirroring it, leading, ironically, to an increase in median error following the move to the more accurate pseudo-master server. We cannot however confirm this here.

D. Into the Internet

The previous section studied the parameters impacting the performance of the TSCclock on a LAN. In this section we provide some insight into the potential of the TSCclock as an Internet-wide synchronization solution.

Figure 14 looks into the respective absolute performance of the TSCclock and *SW-NTP* as a function of two parameters. First, the quantity of raw synchronization information as controlled by the polling period to the server, and second, the distance to the server measured in hops. As the hop count increases, the delay distribution moves to higher values, and the path asymmetry also (very likely) increases. For each experiment, the TSCclock and *SW-NTP* share the same stream of NTP packets. In each plot, the thick black lines show median errors, and the surrounding lines give $[2, 25, 75, 98]$ error percentiles over almost 2 days. *ServerLoc* is a stratum-1 server installed two hops away from the host PC running both clocks with a minimum Round-Trip Time (RTT) around $0.38\,\mathrm{ms}$. *ServerNear* is a stratum-1 server located 5 hops away with a minimum RTT measured at $0.89\,\mathrm{ms}$. Finally, *ServerFar* is a stratum-1 server located 3500km away, 10 hops away (as observed over stable routes) for which we measured a minimum RTT of around $37.7\,\mathrm{ms}$.

For each clock we observe the expected qualitative behaviour: as the polling period and the distance to the reference clock increase, the performance degrades. In this experiment the TSCclock performs clearly better than *SW-NTP*, in particular its variability (IQR) is much smaller. Also in each case,

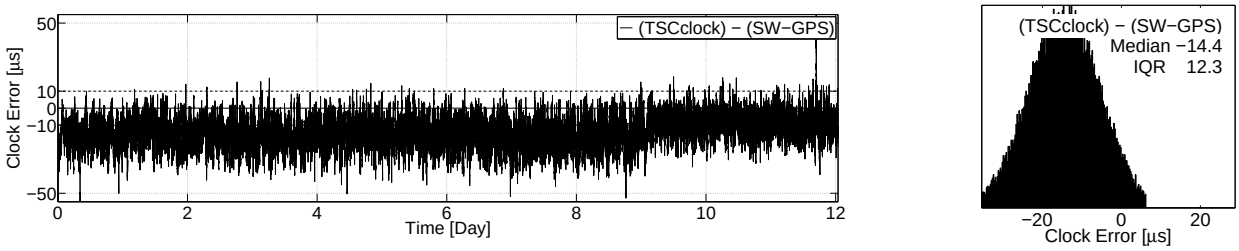

Fig. 10. TSCclock synchronized to stratum-1 server on the LAN, polling period 16s, Internal in-host comparison against *SW-GPS*. Left: time series over 12 days. Right: normalised histogram (from 1st to 99th percentile), median = $-14.3\,\mu$s, IQR = $12.3\,\mu$s.

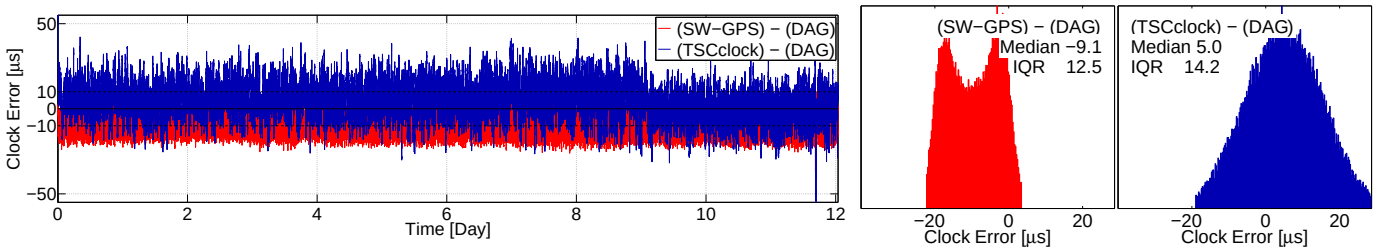

Fig. 11. TSCclock synchronized to stratum-1 server on LAN, polling period 16s and *SW-GPS* clock error using External DAG comparison. Left histogram: *SW-GPS*, median = $-9.1\,\mu$s, IQR = $12.5\,\mu$s. Right histogram: TSCclock, median = $5\,\mu$s, IQR = $14.3\,\mu$s.

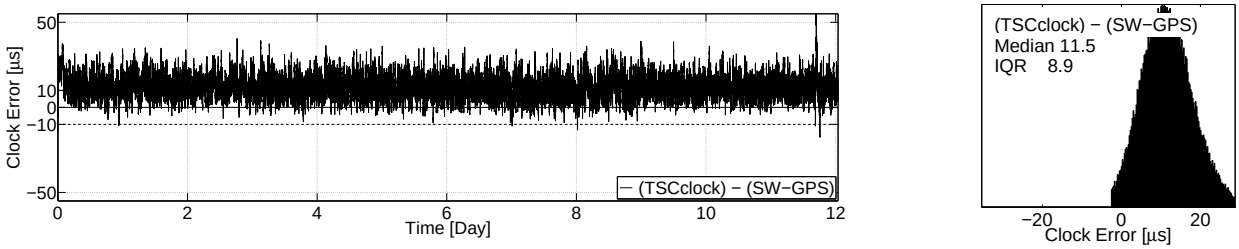

Fig. 12. TSCclock using DAG as a pseudo-master clock, polling period 16s, Internal in-host comparison against *SW-GPS*. Left: time series over 12 days. Right: normalised histogram (from 1st to 99th percentile), median = $-11.4\,\mu$s, IQR = $8.9\,\mu$s.

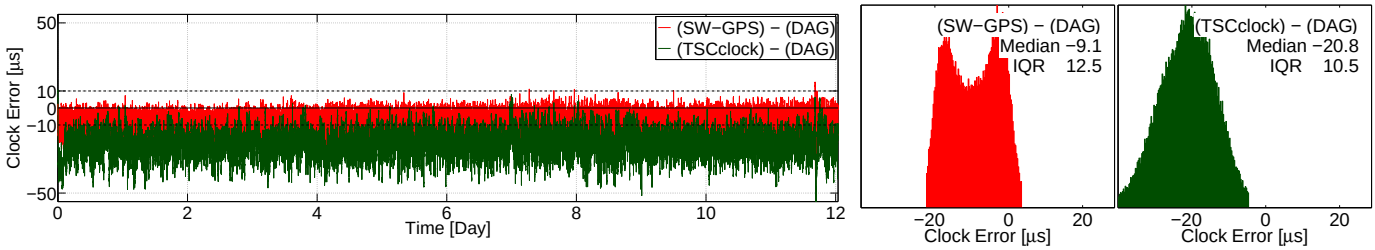

Fig. 13. TSCclock using DAG as a pseudo-master clock, polling period 16s, and *SW-GPS* clock error using External DAG comparison. Left histogram: *SW-GPS*, median = $-9.1\,\mu$s, IQR = $12.5\,\mu$s. Right histogram: TSCclock, median = $-20.8\,\mu$s, IQR = $10.5\,\mu$s.

the median value of the absolute performance of the TSCclock is closer to the reference given by the DAG card and much more stable as the polling period increases. This illustrates the potential of the algorithm to go well beyond the LAN setting with only a modest performance penalty.

V. Prior Work

The literature on practical software based systems capable of achieving clock synchronization over networks is relatively limited. The original NTP literature [13], [1], [9] contains much information but no formal analysis, and to the best of our knowledge no thorough benchmarking study of *ntpd* performance has been published using a detailed methodology such as we give here. A software implementation of PTP was described in [2] and made available by Correll et al. [14]. It uses a PI controller to adjust tick rate, a feedback mechanism, in contrast to our feedforward approach, and offers only a very preliminary evaluation of performance. Work addressing practical Internet measurement problems related to synchronization, for example [15], have tended to address the improvement of timestamping accuracy in post-processing, rather than tackling synchronization algorithms running live.

Recently there has been an increase in interest in rigorous analysis of network synchronization. Typically the approaches are reliant on strong assumptions, principally that drift is of an affine type (i.e., simple skew or pure frequency models).

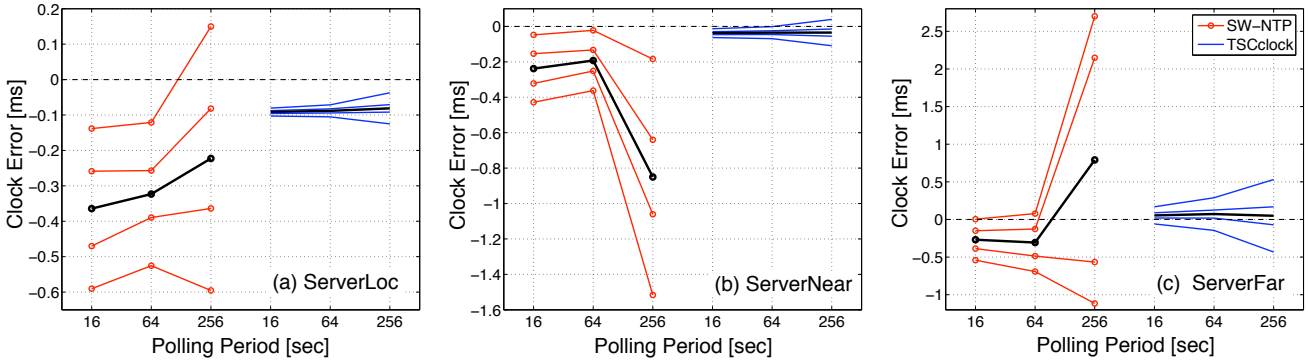

Fig. 14. TSCclock versus *ntpd* for three polling periods and three time servers, left to right: LAN; same campus; across the continent.

For example Berthaud [16] uses an affine drift model and the language of convex closures to combine local error bounds on clock error at a host into global bounds, whereas [17] tackles an optimization problem for determining one-way delays between nodes in a network assuming that clocks are perfect and run at the same rate, but have different constant offsets. There has been recent activity in synchronization over wireless networks, in particular adhoc IEEE 802.11 networks. For example in *RBS* a synchronization algorithm is described which exploits the broadcast nature of the medium to enable 'simultaneous' timestamping by multiple nodes of the same events. These timestamps can then be exchanged and compared by receivers to estimate relative node offsets. Elson et al. [18], in the context of a simple skew model (it is proposed that skew estimates be recalculated on a longer timescale), propose, but do not evaluate, a scheme for optimizing the transmission of synchronization messages over a network subject to an energy constraint.

VI. Conclusion

We compared the TSCclock performance against both the *SW-NTP* and *SW-GPS* clocks, the most widely deployed low cost alternatives at this time. The results presented help us gain an understanding of the performance of the TSCclock, and highlight the *ntpd* server system noise as a possible obstacle to further performance improvements. Specifically, we used the DAG card as a virtual IEEE-1588 master clock to measure the performance of the TSCclock under conditions where server noise and network asymmetry are greatly reduced, and observed (1) an accurate server (no long term drift, timestamping error below 40 ns) improves accuracy and robustness, (2) in a LAN environment, server error is not a significant cause of TSCclock error, (3) in a LAN environment the performance of the TSCclock is comparable to that of the *SW-GPS*, (4) in a LAN environment most of the TSCclock residual error is due to the asymmetry and noise characteristics of the network interface card and driver used.

Another interesting comparison would be to compare the performance of the TSCclock with the *PTPd* solution [2]. While having different objectives, we believe the TSCclock would be a robust alternative to *PTPd* in providing accurate software clock on LANs. In the future, we also will investigate the possibility of using the TSCclock as a boundary clock. The promising performance observed when comparing the TSCclock against *SW-GPS* calls for a modified version of the TSCclock capable of processing GPS input. Without being able to reach the level of accuracy offered by dedicated hardware, we believe this solution would offer an inexpensive alternative to network applications requiring accuracy in a 10 μs range, or even below.

References

[1] D. L. Mills, "Network time protocol (version 3) specification, implementation and analysis," IEFT, Network Working Group, RFC-1305 *www.ietf.org/rfc/rfc1305*, March 1992, 113 pages, papers in appendix.

[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] 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.

[4] 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.

[5] E. Corell, P. Saxholm, and D. Veitch, "A User Friendly TSC Clock," in *Passive and Active Measurement Conference (PAM2006)*, Adelaide Australia, March 30-31 2006, pp. 141–150.

[6] J. Ridoux and D. Veitch, "A Methodology for Clock Benchmarking," in *Tridentcom*. Orlando, FL, USA: IEEE Comp. Soc., May 21-23 2007.

[7] 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.

[8] J. Ridoux and D. Veitch, "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.

[9] D. L. Mills, "Improved algorithms for synchronizing computer network clocks," *IEEE/ACM Trans. Networking*, vol. 3, 1995.

[10] "Endace Measurement Systems," http://www.endace.com/.

[11] "Stanford Research Systems," http://www.thinksrs.com/.

[12] "Endace Measurement Systems," private communication, Dec 2007.

[13] D. Mills and P.-H. Kamp, "The Nanokernel," in *32nd Annual Precision Time and Time Interval (PTTI) Meeting*, Reston VA, November 2000, pp. 423–430.

[14] "The Precision Time protocol (PTP), ptpd," http://ptpd.sourceforge.net/.

[15] V. Paxson, "On calibrating measurements of packet transit times," in *Proc. ACM SIGMETRICS*. ACM Press, June 1998, pp. 11–21.

[16] J. Berthaud, "Time synchronisation over networks using convex closures," *IEEE/ACM Trans. on Networking*, vol. 8, no. 2, pp. 265–277, April 2000.

[17] O. Gurewitz, I. Cidon, and M. Sidi, "One-way delay estimation using network-wide measurements," *IEEE/ACM Trans. Networking*, pp. 2710–2724, June 2006.

[18] J. Elson, R. Karp, C. Papadimitriou, and S. Shenker, "Global synchronization in sensornets," in *6th International Symposium on Latin American Theoretical Informatics (LATIN)*, Buenos Aires, Argentina, April 5-8 2004, pp. 609–624.