A Methodology for Clock Benchmarking

Julien Ridoux
ARC Special Research Center
for Ultra-Broadband Information Networks (CUBIN)
An affiliated program of National ICT Australia
EEE Department, The University of Melbourne
Australia
Email: j.ridoux@ee.unimelb.edu.au

Darryl Veitch
ARC Special Research Center
for Ultra-Broadband Information Networks (CUBIN)
An affiliated program of National ICT Australia
EEE Department, The University of Melbourne
Australia
Email: d.veitch@ee.unimelb.edu.au

***Abstract*— Accurate timestamping is a basic need in traffic monitoring as well as distributed computing in the broad sense, and is destined to become increasingly important as network latency becomes a hard barrier to improved performance across networks. Software clocks need to be improved to meet this challenge, however evaluating their performance is non trivial, as they are imbedded inside computing systems. We present a methodology for clock validation which allows many of the difficult problems to be resolved. Our method involves a combination of external and internal validation strategies, and makes use of GPS synchronized DAG cards and system clocks. We illustrate in detail how it may be applied using real data collected from 3 clocks implemented in UNIX PCs.**

I. Introduction

As has been foreseen for some time, the Internet is approaching a new step in its evolution. Giving users the ability to access high throughput networks via technologies such as Fiber-To-The-Home will soon change our needs and usage patterns, as will high bandwidth mobility. Applications will be more distributed and smarter, demands on providers to report quality higher, and the expectations of users, even higher. More and more, high performance will depend on tight integration over distributed servers and multiple simultaneous connections, and the careful management of latency between them to provide users with immediate response. Underlying this is the fact that the 'tightness' of distributed applications is fundamentally limited by the quality of clock synchronization. In this context, the design of inexpensive but accurate new timekeeping system is a crucial issue.

The development of more accurate clocks that perform well and reliably in practice cannot be successful without an ability to test their performance under operational conditions. However, there is a fundamental problem in that clocks are embedded inside computer systems. There is no perfect clock inside the system against which a given clock can be directly compared. In addition, clock comparisons suffer from system noise such as complex communication delays, interrupt latencies and resource race conditions. Furthermore, the act of reading a clock when desired, that is *timestamping*, is itself problematic and makes validation problematic.

One must admit that clock validation is a niche subject, as well as being somewhat dry. However, the fact is that it is important, difficult, and neglected. Previous studies exist ([1]), but we know of no published work describing how it can be done accurately for software clocks on modern systems, although an early form of part of what we present here appeared in [2], in the context of evaluating a new clock. In this paper we provide a methodology capable of circumventing, to a large extent, the inherent difficulties in this problem, enabling the accuracy of software clocks to be established up to 10's of microseconds. To do so, we make use of a testbed including GPS synchronized reference clocks, a DAG high performance measurement card, and modified BSD kernels.

Our methodology relies on performing two kinds of validation, 'external' and 'internal', which have complementary advantages, in order to avoid the inherent disadvantages of each. Our presentation is generic in that we clearly define and describe the underlying fundamental issues, and can be used in principle with any clock, but practical in that we use a real testbed, and actual implementations of three significantly different clocks. Using weeks of real data we illustrate how the methodology works in detail, using it to perform 'detective work' to discover true clock performance, which would otherwise be hidden by system effects.

Section II provides the background concepts of clock and timestamping error, and system noise. The clock validation methodology is then described in Section III. Section IV defines the clocks we use to illustrate the possibilities of the testbed and methodology, themselves given in Section V. We conclude in Section VI.

II. Clocks, Errors, and System Noise

A. Clock and Timestamping Errors

We denote true time, measured in seconds from some origin, by t. Any real world clock $C(t)$ inevitably suffers from errors. The curves in Figure 1 show what 4 different clocks read, as a function of true time, for example at time $t = t_k$. The diagonal line corresponds to a perfect clock.

The error or *offset* of $C(t)$ at true time t will be denoted by

$$\text{Offset:} \quad \theta(t) = C(t) - t. \tag{1}$$

Common terms used to characterise this error are *skew*, *drift* and *stability*. Skew is used to refer to a clock which runs at the wrong rate. For example it corresponds to the parameter

γ when $\theta(t) = C + \gamma t$, which may be a valid model for real clocks over small time scales. The upper curve in Figure 1, $C_s(t)$, shows such a clock. More generally one can define $\gamma(t) = d\theta(t)/dt$. The clock $C_o(t)$ in the figure has zero skew, but constant offset. Drift refers to the wandering of a clock (or equivalently its rate), in particular over longer timescales where offset is non-linear and cannot be described by a simple skew, such as $C_d(t)$ in Figure 1. Drift is strongly influenced by temperature in real systems. The stability of a clock is a measure of the variability of drift, and is usually measured by means of the *Allan Variance*, a kind of time-scale dependent variance measure (see [2] for details). For our purposes here we use 'stability' simply to refer to the non-linear part of the clock offset.

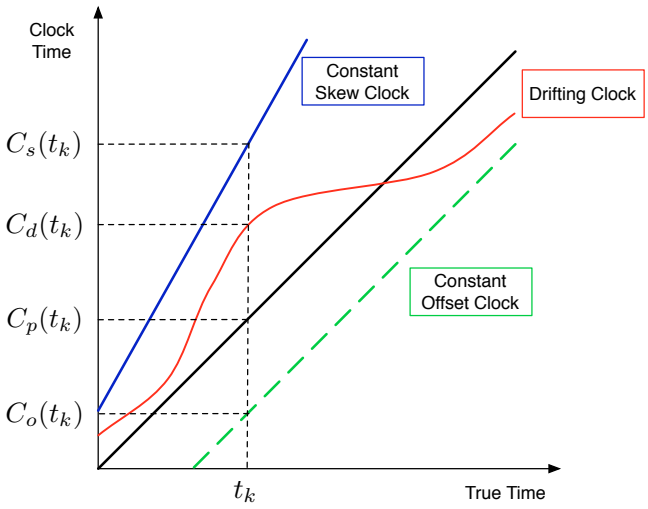

Fig. 1. Clock readings as a function of true time for 4 clocks. The realistic case is the $C_d(t)$ with non-linear drift.

Fundamentally, the role of clocks is to record the times at which events of interest occur. In real systems, however, clocks cannot be read precisely when desired. If t_k is the time[1] at which a *target event* occurs, and t'_k the time at which a clock $C(t)$ is read, then we define

$$\text{Timestamping error:} \quad \xi(t_k) = t'_k - t_k \tag{2}$$

as the *timestamping error* relative to the target event.

When attempting to measure the time of an event occuring at time t_k, using a clock $C(t)$, the timestamping and clock offset errors naturally combine, to produce the *total error*

$$\begin{aligned} \text{Total error: } E(t_k) &= C(t'_k) - t_k \\ &= \theta(t'_k) + t'_k - t_k \\ &= \theta(t'_k) + \xi(t_k). \end{aligned} \tag{3}$$

The total error and its components are illustrated in Figure 2. When comparing a clock against another, rather than against true time, the offsets and timestamping errors of both clocks

[1]True times will always be denoted by lower case t.

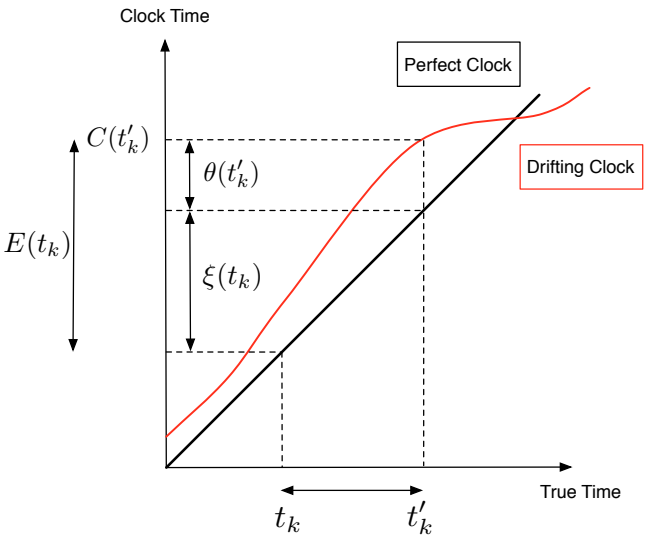

Fig. 2. Total error, and its components of clock offset and timestamping error, for a target event at true time t_k.

combine (constructively or destructively), to produce a *total relative error*

$$\begin{aligned} \text{Total relative error:} \quad & E_{C_1,C_2}(t_k) = \\ C_1(t'_k) - C_2(t''_k) &= E_1(t_k) + t_k - (E_2(t_k) + t_k) \\ &= \theta_1(t'_k) - \theta_2(t''_k) + \xi_1(t_k) - \xi_2(t_k). \end{aligned} \tag{4}$$

B. Clock Systems and System Noise

Clock systems consist of a hardware timing source such as a crystal oscillator, a system to convert this to a readable clock $C(t)$, and (almost always) an external reference time source $R(t)$ which is deemed to be more accurate than $C(t)$, together with algorithms to synchronize $C(t)$ to $R(t)$.

The system underlying $C(t)$ contributes both to offset and timestamping errors. For example, software clocks on PCs are ultimately software processes competing for resources. This competition translates to delays which clearly create timestamping errors whenever the clock is read. In addition however, since the synchronisation algorithms are based on timestamp processing, these delays also strongly influence the offset error of the final synchronized $C(t)$ itself.

This paper deals with software clocks defined in commodity computer hardware. The question we address is how such clocks can be evaluated, both in an absolute sense, and against each other. System effects play a very important role here. We therefore describe some of these in more detail. It is worth noting that, in many cases, the reference clock is itself a software clock built on commodity hardware, and suffers from similar system issues. However this is not our main focus. Such effects are subsumed within the final offset errors $\theta(t)$ of the clocks. Instead we focus on how $\theta(t)$ may be measured and interpreted through the 'haze' of system effects in the host.

In PCs there are many examples of delays preventing events being processed in real time. These include system

interrupts, the system scheduler, hardware/driver events and packet queueing and processing. Collectively, we use *system noise* to refer to any system induced delay which prevents part of the clock hardware or software from executing when it would like. Timestamping error, as one example, is therefore a direct result of system noise.

Scheduling delays in particular are strongly influenced depending on whether a process runs in user or kernel context. A convenient way to gain access to the lower noise of the kernel is to target events corresponding to packet events. We modified Linux and FreeBSD kernels so that packet timestamps from each clock used (described in more detail below) could be taken in the kernel and made available to user level. We do not present any measurement taken on Linux hosts in this paper to provide consistent comparisons. Nevertheless, all results obtained from FreeBSD kernels are similar to the ones obtained using Linux hosts.

The definition of 'system' above includes components of the reference clock itself. The important distinction for reference clocks is between hardware sources which are locally connected to the computer, or software sources accessible over a network. As we use each of these in this work, we provide some background here.

Atomic clocks, GPS or CDMA receivers are examples of local sources, which are used to build what is considered to be accurate clocks. In such cases, a Pulse-Per-Second (PPS) signal is made available via a serial port, along with a time packet to signal which second is which on an accepted timescale (like UTC). Here, a timestamp of $R(t)$ is effectively made when a pulse is processed.

Clocks synchronized over a network rely on packets travelling between the host and a time server containing timestamps. In the client-server mode of operation, this involves timestamping a packet as it leaves for the server, at the server, and as it returns. The network delays are part of an 'external system noise' and have a large impact. The Network Time Protocol (NTP [3], [4]) and the accompanying infrastructure of stratum-1 servers is the dominant solution today. Timestamps are carried in UDP packets with formatted payloads.

III. Testbed and Methodology

The heart of the testbed, shown in Figure 3, is a host computer where three different clocks are implemented. Two of these (SW-NTP, TSCclock) are synchronized to a stratum-1 NTP server, and the third (SW-GPS) to a locally attached GPS. In addition, the Host exchanges UDP packets with a Unix PC in each direction, in order to define target events in the host to be timestamped. Finally, a high precision GPS synchronized DAG3.6 series measurement card monitors, via a passive hub, the UDP packets, as well as the NTP packets exchanged between the Host and the NTP server. Note that, although three clocks are supported, only one of SW-GPS and SW-NTP can be used at any one time.

The central difficulty in evaluating a software clock is that we do not have access to a perfect clock inside the system against which to compare. The consequences of this are that

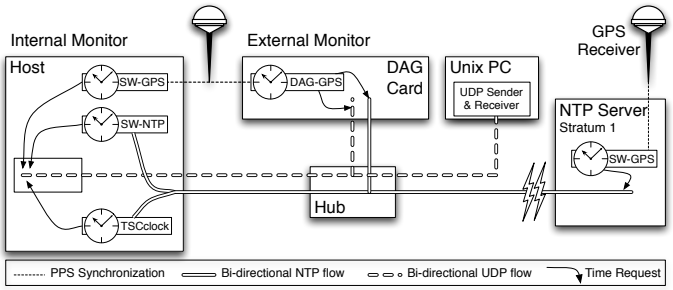

Fig. 3. The testbed. UDP packets are timestamped by clocks inside the Host.

(i) we must compare against clocks which are themselves imperfect, resulting in ambiguity in the estimated offsets, and (ii) timestamping errors, as clock readings are not contiguous. The second difficulty is greatly complicated by the fact that timestamping errors can be highly variable due to system noise. Our aim is to show how clocks may be evaluated despite these intrinsic difficulties.

Our approach is to combine measurements made from two different validation strategies with complementary strengths. The *external validation* makes use of the DAG monitor in order to provide a timestamp with low offset error, but high timestamping error because the monitor is external to the host. The *internal validation* makes use of a second clock inside the host, which greatly reduces relative timestamping error, but which has higher offset. By using these together, we can in many respects analyse the clock performance as if we had a validation clock with low total error.

Before describing the validation methodology in detail, we first define the events we wish to timestamp. These are in relation to an exchange of UDP packets, as shown in the (true) time line of Figure 4. The target events are the instants at which the last bit of the packet leaves or enters the network interface card (NIC) in the host. More precisely, t_a (resp. t_f) denotes the instant at which the last bit of the outgoing (resp. incoming) UDP packet leaves (resp. arrives to) the NIC.

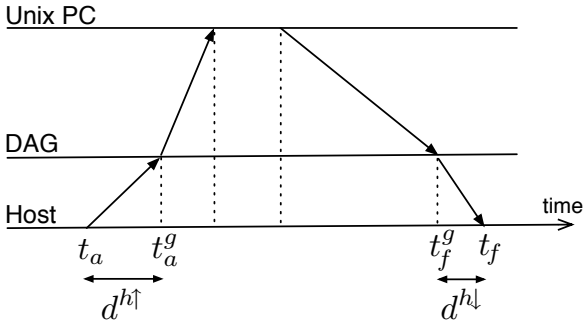

Fig. 4. A (true) time line showing the key target event times inside the host and at the DAG card. The components $d^{h\uparrow}, d^{h\downarrow} \geq 0$, of the minimum Host round-trip time $r^h = d^{h\uparrow} + d^{h\downarrow}$ are shown.

A. External Validation

One approach to validation is to try to compare against a reference clock, external to the host, which has very low offset error. A DAG card [5], an industry standard high performance passive packet timestamping card, is a good choice for this role. Its on-board clock can synchronize to a GPS pulse-per-second to around 200 [ns], and its hardware architecture minimises its internal timestamping noise. As a result, the offset error of a DAG timestamp $D(t')$ is considerable smaller than any other error we can quantify, and for convenience we assume it is zero below.

Figure 4 shows the DAG event times t_a^g (resp. t_f^g) corresponding to the last bit of the outgoing (resp. incoming) UDP packet passing by the DAG [2]. We can now define the external validation errors in terms of measured timestamps as

$$\text{Outgoing external error:} \qquad X_o = C(t_a') - D(t_a^{g\prime}) \quad (5)$$

$$\text{Incoming external error:} \qquad X_i = C(t_f') - D(t_f^{g\prime}). \quad (6)$$

Although the DAG offset is very small, there is a significant relative timestamping error in these external comparisons due to system noise on the host side, and NIC packet processing on the DAG side. This can be seen explicitly by setting the offset error for the DAG to be zero in Equation 5, yielding

$$\begin{aligned} E_{C,D}(t_k) &= \theta_C(t_k') + \xi_C(t_k) - \xi_D(t_k) \\ &\approx \theta_C(t_k) + \xi_C(t_k) - \xi_D(t_k) \end{aligned}$$

since the change in clock error over short periods is typically negligible.

The timestamping errors can be reduced by reducing system noise. To do so we modified the FreeBSD 6.1 and Linux 2.6.18 kernels to allow kernel level timestamping for each of the three clocks. In each system, packet timestamps made using the existing system clock are made available to the libpcap library at a generic place in the kernel. In the case of FreeBSD for example, this is achieved by means of the BPF subsystem common to all network drivers. Our modifications allow all three clocks to make timestamps in these standard locations in the kernel. In addition, for one of the clocks (the TSCclock) we provide an alternative modification (marked as *improved* in Figure 5) to allow timestamps closer to the driver and hence to the target events.

The external comparison using a quality external time reference allows the absolute performance of host clocks to be determined, but only up to an error, due to system and NIC noise, which can be large depending on system and network load. In Section V-A we explain that there are in fact two distinct impacts of this noise, one due to noise variability, and the other to a constant term arising from the fact that $t_a < t_a^g$ and $t_f > t_f^g$.

[2]Because the DAG card is designed to timestamp the first bit of each packet, we apply a post-processing correction to obtain a timestamp of the last bit, based on the packet size and network speed.

B. Internal Validation

A second approach to validation is to compare against another clock inside the host. Such a clock may often have an offset error greater or much greater than a quality external reference such as DAG, but its availability inside the host is a great advantage for reducing the impact of timestamping errors. If the clocks can be timestamped very close together, then $t_k' = t_k''$ in Equation 5, and the timestamping errors, although non-zero, cancel yielding

$$E_{C,D}(t_k) \approx \theta_{C_1}(t_k) - \theta_{C_2}(t_k),$$

where as before we have assumed that $\theta(t_k') \approx \theta(t_k)$.

Our kernel modifications also support a *Fair Compare* mode which allows back-to-back timestamping calls to be executed when triggered by a target event such as a packet arrival. In kernel context, this virtually eliminates the possibility that the timestamping processes will be interrupted, dramatically reducing system noise between two clocks' readings, and resulting in a tightly correlated timestamping errors.

We define the internal validation errors in terms of measured timestamps as

$$\text{Outgoing internal error:} \qquad I_o = C_1(t_a') - C_2(t_a') \quad (7)$$

$$\text{Incoming internal error:} \qquad I_i = C_1(t_f') - C_2(t_f'). \quad (8)$$

The internal comparison implemented with the Fair Compare strategy allows two host clocks to be cleanly compared: even if timestamping errors are large and highly variable, they will always cancel almost perfectly. The disadvantage is that this comparison is relative only. It provides insight into both the relative offset and stability of the two clocks, but can not tell us their absolute performance with respect to true time.

IV. Introducing the Clocks

This paper is about clock validation methodology, and not the characteristics or performance of any clock in particular. However, a knowledge of the basics of the clocks implemented in the testbed is necessary for an understanding of the results to follow.

Two of the clocks in the testbed, SW-NTP and SW-GPS, although differing significantly, are based on the standard system software clock supported by the PC's operating system. The system clock $S(t)$ is based around the periodic interrupt cycle, of period typically 1[ms], driving process scheduling. The interrupt cycle period is obtained by counting a number of periods of an oscillator of low frequency, located on the motherboard. The TSC (Time Stamp Counter) register records the number of CPU cycles starting at boot time. The system clock uses it to interpolate times between interrupts, but it is not the fundamental basis of the clock. The system clock derives a nominal rate at boot time, and then adjusts itself through three mechanisms: *reset*, *skew* and *phase*, informed by filtering timestamps through the `ntpd` daemon, with the aim of driving $\theta(t)$ to zero. Two important features of the `ntpd` generated system clock follow from this: it *actively adjusts clock rate*, and it provides an *absolute time clock only*. All timestamps are in `timeval` format (with 1μs resolution).

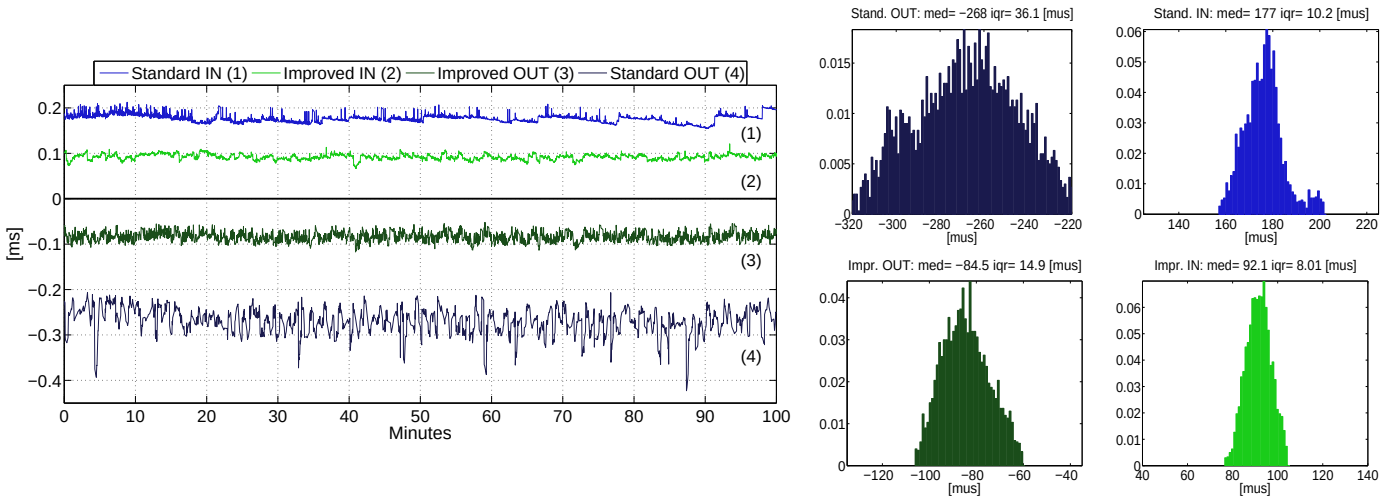

Fig. 5. *Bandicoot*, external error for both directions underlying the differences between standard and improved timestamping strategies.

1) The SW-NTP Clock: Present in most operating systems, the SW-NTP clock (with `ntpd`) works in client-server or broadcast mode, processing timestamps transported to and from a NTP server (or servers) in NTP packets (Fig. 3). Broadcast mode is typically used when the server is on the same LAN whereas the client-server mode is typically used in other cases. As the incumbent, the system clock benefits from existing kernel implementations and the standard libpcap interface, which timestamp packets in the kernel and make them available at user level.

2) The SW-GPS Clock: In the case of the SW-GPS, `ntpd` works in broadcast mode as above, but processes timing information arriving via serial port from a GPS receiver. Configuring the system to perform this reliably can be non-trivial. We achieved it using two drivers. The 'atomic driver' receives the pulse per second (PPS) output of the GPS (a PPS peer for `ntpd`), and the Palisade driver (candidate peer) receives a packet with timing information emitted after the pulse. We use a Trimble Acutime 2000 GPS receiver, whose pulse is synchronized to UTC to within 50 nanoseconds. However, the offset error is greater than this due to host system noise and processing within `ntpd`.

Since the SW-GPS clock uses exactly the same active control algorithms in `ntpd` as the SW-NTP clock does, it also provides an absolute time clock only. To calculate time differences, one must subtract two absolute times.

3) The TSCclock: The TSCclock [6], [2], [7] is based on the TSC oscillator only. Contrary to the SW clocks, it is built around obtaining stable long-term clock rate estimates. The TSCclock does not actively vary its rate to track drift and provides actually two clocks, one for time differences, and one for absolute time.

The synchronization algorithm operates in client-server mode. Each round-trip generates four timestamps, two in `timeval` format from the server, and two raw TSC timestamps taken at the host. These *stamps* (4-tuple of timestamps) are used to provide an estimate $\hat{p}$ of the average CPU oscillator period p. An *uncorrected* clock can then be defined as $C_u(t) = \hat{p} * TSC + K$, where K is an estimate of the 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:

$$\mathbf{C}_d(t): \quad \Delta(t) = C_u(t_2) - C_u(t_1) = \Delta(TSC) * \hat{p}, \qquad (9)$$

$$\mathbf{C}_a(t): \quad C_a(t) = C_u(t) - \hat{\theta} = \hat{p} * TSC + K - \hat{\theta}. \qquad (10)$$

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 [sec] yields an error of only 0.1μs.

Measuring absolute time requires that drift be tracked. Hence, $\hat{\theta}$ is incorporated into the definition of $C_a(t)$, which results in short 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 rate to track drift, which improves stability.

A more detailed account is given in [6], [2]. The resolution of the TSCclock is tied to that of the CPU, and is around 1[ns].

V. Comparing Clocks

In this section we show how the internal and external comparisons can be used together to learn about clock performance. We begin with an introduction to the external case, and show how it enables us to learn about the nature of system noise, which in turn gives insights into the clock. In Section V-B we show what the methodology is capable of through a series of examples using real data. Throughout we use the clocks described in the previous section, but with a strong emphasis on generic aspects.

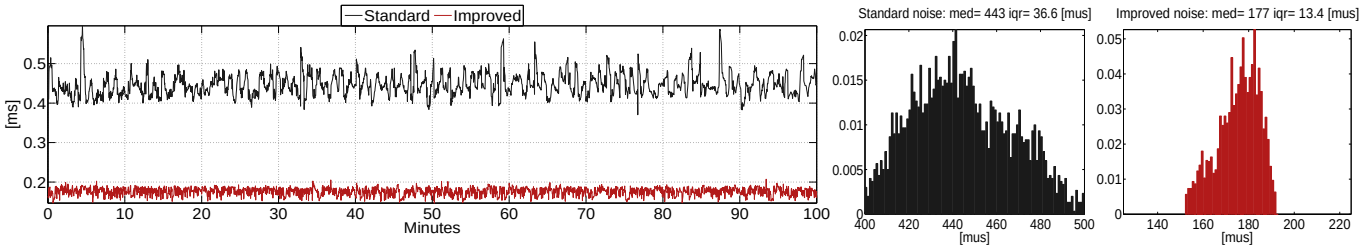

Fig. 6. *Bandicoot* noise measured with standard and improved timestamping strategies.

A. External Validation and System Noise

Figure 5 shows time series and associated histograms of an external comparison of the machine *bandicoot* over a 100 minute period, for both incoming and outgoing directions. *Bandicoot* is a 600Mhz Pentium processor host, running FreeBSD 6.1 and using a 3Com 10/100 Mbps network interface. The clock being validated is the TSCclock, for which we have both standard and improved kernel timestamping available. Both are shown for comparison.

Consider first the results using standard timestamping. The timeseries show an error of the order of 200μs in each direction. The immediate problem is that this number is only a bound on the clock offset. We do not know how much of it is due to the true offset, and how much to the timestamping error inherent in the external comparison. However, there is a clear difference in the time series in the two directions: the outgoing direction exhibits more variability. This is confirmed by the histograms: The Inter-Quantile Range (IQR) of the outgoing comparison (38 μs) is almost four times that of the incoming one (10 μs). However, each histogram is summarising the error of the *same* clock, seen by the same DAG reference! Extreme programming bugs aside, it is not possible for the offset of the clock to be consistently different for incoming and outgoing packets. We are led to conclude that the timestamping noise is in fact larger on the outgoing side, by an amount of the order of 30 μs. This is important as it indicates that it is better to work with data on the incoming side. However, it still does not help us to determine the actual magnitude of the timestamping errors for each direction and hence unambiguously recover the clock offset.

The above observations and conclusions hold for the improved timestamping also. The difference is that errors have decreased in both directions, particularly on the outgoing direction, compared to the standard case. Clearly the 'improved timestamping' has earned its name, but again, this does not give us a means to remove timestamping error. However, as we now show, using both directions simultaneously does give useful insight into the absolute value of timestamping error.

Using pairs of sent and received UDP packets, exchanged as indicated in Figure 4, the (minimum) *Host Round-Trip Time* (Host RTT) r^h is defined as the sum of the two minimum one-way delays between the Host and the DAG card:

$$r^h = d^{h\uparrow} + d^{h\downarrow} = (t_a^g - t_a) + (t_f - t_f^g).$$

In terms of available timestamps, the measured Host RTT R^h is expressed as:

$$R^h = (D(t_a^{g\prime}) - C_u(t_a')) + (C_u(t_f') - D(t_f^{g\prime})). \qquad (11)$$

Using the low total error of the DAG timestamps we set $D(t') = t$, and ignore noise on the NIC side, yielding

$$\begin{aligned} R^h &= t_a^g - C_u(t_a') + C_u(t_f') - t_f^g \\ &= t_a^g - (E(t_a) + t_a) + (E(t_f) + t_f) - t_f^g \\ &= (t_a^g - t_a) + (t_f - t_f^g) - E(t_a) + E(t_f) \\ &= r^h + \theta(t_f) - \theta(t_a) + \xi(t_f) - \xi(t_a). \end{aligned}$$

The RTT of a UDP packet pair is much smaller than two successive updates of any clock we study. Hence the clock offset remains unchanged during a packet pair exchange: $\theta(t_f) = \theta(t_a)$. As a result, the clock offsets of the timestamps in each direction cancel leading to

$$R^h = r^h + \xi(t_f) - \xi(t_a), \qquad (12)$$

that is the measured Host RTT is composed of the true Host RTT and the timestamping error on both directions.

There is an important asymmetry between the two directions. On the receiving side, the timestamping of an incoming packet can not be triggered before the packet actually arrives. This guarantees that $t_f \leq t_f'$, and so $\xi(t_f) \geq 0$. In the sending direction there is no such natural 'causality constraint', however it is important to ensure it via an appropriate implementation. Failure to do so could result in causality failing in the empirical timestamping sense, resulting in erroneous offset estimates. A causality respecting implementation guarantees that $t_a \geq t_a'$ and hence that $\xi(t_a) \leq 0$. Equation 12 then takes the form of a positive noise on top of a positive r^h, and r^h can be measured using minimum filtering.

Using the trace from *bandicoot* as before, Figure 6 shows the R^h calculated from Equation 11 for the same period of 100 minutes previously used. For both the standard and customised timestamping methods, the Host RTT is consistent with the picture given above, a constant minimum plus a positive noise. The figure then gives us both a measure of r^h, which is the smallest value the sum of the timestamping errors can take, and a measure of its variability via the IQR of the histograms. We have succeeding in isolating timestamping noise from offset, but we cannot do the converse, because the individual noises in each direction still (inevitably) elude us. Nonetheless, we have gained considerable insight into the size and nature of the system noise as it affects external validation.

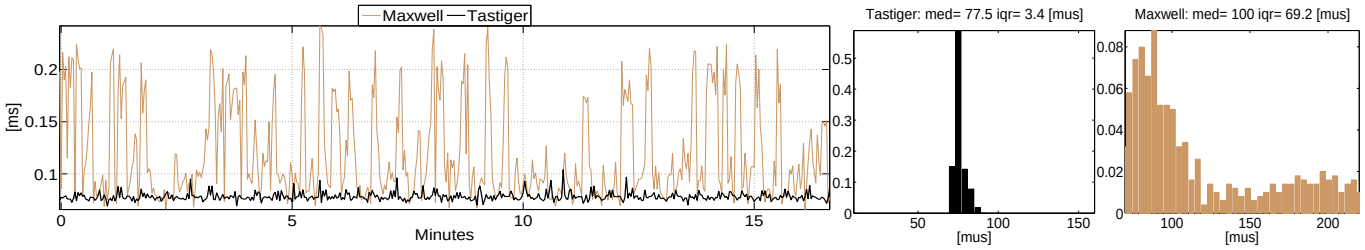

Fig. 7. Comparison of system noise between *maxwell* and *tastiger* measured with improved timestamping strategy.

We have seen that the variable part of the relative timestamping error superimposes onto the variability of $\theta(t)$ itself, resulting in a broadening of the observed histogram for the external error. In fact the relative timestamping error also has a constant term, which creates an impact of a different kind. Assuming $D(t') = t$, and assuming minimal timestamping (and NIC) noise for simplicity, the external comparisons in each direction are respectively $X_o = \theta(t_a) - d^{h\uparrow}$ and $X_i = \theta(t_f) + d^{h\downarrow}$. Now let c be a constant. We can write $X_o = (\theta(t_a)+c)-(d^{h\uparrow}+c)$ and $X_i = (\theta(t_f)+c)+(d^{h\downarrow}-c)$, which shows that the one-way NIC to DAG delays $d^{h\uparrow}$ and $d^{h\downarrow}$, as they cannot be independently measured, create an inherent ambiguity for the offset: *it can only be known up to a constant* c. It is easy to show that c is bounded by $c \in (-d^{h\uparrow}, d^{h\downarrow})$ which has a width $d^{h\downarrow}-(-d^{h\uparrow}) = r^h$. Although r^h is typically of the order of only a few tens of microseconds, the ambiguity places a limit on the ability of the external comparison to measure the true offset, in particular the average clock error. In practice, since $d^{h\uparrow}$, $d^{h\downarrow}$ are unknown but each bounded by r^h, $\theta(t)$ is only known up to $c \in (-r^h, r^h)$. In summary, there are two kinds of limitation in external comparisons: a variable component which broadens the measured histogram of θ, and a constant component which shifts the histogram by an amount which is unknown, but bounded by $2r^h$, which is measurable.

We conclude by showing how understanding the system noise can help configure the testbed itself. We use two recent PC systems called *maxwell* and *tastiger*. These machines have hardware of the same generation and have equivalent performance. They each run the same operating system (our modified FreeBSD 6.1). The only key difference existing between *maxwell* and *tastiger* concerns their network card and corresponding driver. *Maxwell* is equiped with a BroadCom 5157 Gigabit Ethernet network adapter embeded in its motherboard, whereas *tastiger* uses a 3Com 10/100 Mbps PCI network adapter.

We compare R^h for these two hosts using the improved timestamping implementation. We collected UDP packets timestamps during the same period on both hosts so that they share the same network conditions. Despite the many similarities in the machines, Figure 7 shows that the results are very different. *Tastiger* exhibits an extremely stable timestamping error with an IQR of R^h of only around 3 μs, whereas the histogram corresponding to *maxwell* is spread over 70 μs.

Tastiger and *maxwell* respectively exhibit the best and worst examples of system noise we have seen. This example highlights the need to carefully set up the testbed, in particular to select a good NIC/driver combination, in order to keep the timestamping error as low as possible.

B. The Methodology at Work

We begin with the comparison of the two clocks synchronized over a network, SW-NTP and the TSCclock. For this purpose *bandicoot* was configured to support each of these. The SW-NTP is synchronized to a stratum-1 server, located on the same LAN, that broadcasts NTP packets once every 16 seconds. The TSCclock is synchronized to a different stratum-1 server located 2 hops away, receiving time packets every 2 seconds.

Using the incoming direction, figure 8 shows the internal comparison I_i for the SW-NTP and TSCclock over a period of two weeks. The time series, corresponding to the difference between the offsets of the two clocks (recall Equation 8) shows large oscillations in a ± 1 millisecond band, as confirmed by the IQR of the corresponding histogram. Such a spread indicates that the offsets of the clocks differ significantly. We cannot tell however, if one or the other, or both, clocks are responsible.

To obtain independent information on each clock separately, we turn to the external comparison. Figure 9 shows the comparison with DAG for each clock. The time series clearly indicate that SW-NTP is overwhelmingly responsible for the magnitude of the relative offset error seen in the internal comparison. In fact the shape of the internal comparison clearly follows that of $X_i(t)$ for SW-NTP, with a similar oscillation within ± 1 millisecond band and an IQR of 545μs. On this scale the TSCclock remains close to the DAG clock, with an IQR for X_i of only 22μs. We measured the corresponding IQR of the bidirectional noise R^h to be in the order of 37μs. As this is small compared to the IQR for SW-NTP, it follows that the histogram of X_i for SW-NTP is an accurate view of its offset histogram. On the other hand, it is difficult to make detailed conclusions in the case of the TSCclock as the timestamping error may be of the same magnitude as the observed external error. During this capture r^h was around 10μs, corresponding to an ambiguity of $2r^h = 20\,\mu$s in the median values for both clocks.

We now compare the TSCclock and SW-GPS clocks, using another host, *potoroo*. *Potoroo* is similar to *bandicoot* in terms of hardware (same generation, comparable processor

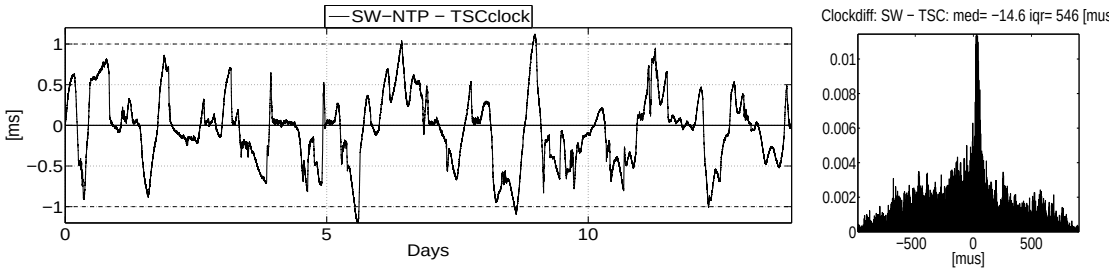

Fig. 8. Internal comparison between SW-NTP and TSCclock (captured on *bandicoot*).

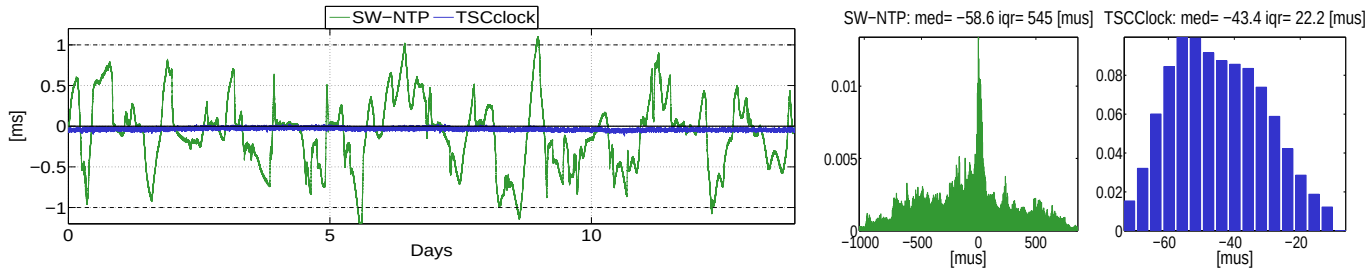

Fig. 9. External comparison between SW-NTP and TSCclock (captured on *bandicoot*).

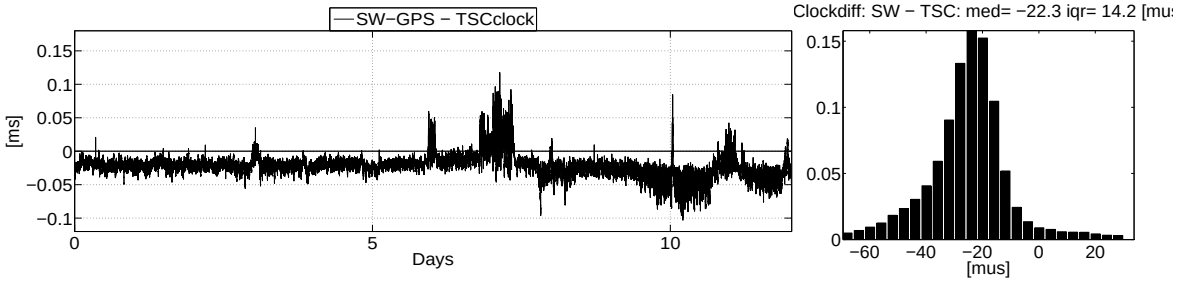

Fig. 10. Internal comparison between SW-GPS and TSCclock (captured on *potoroo*).

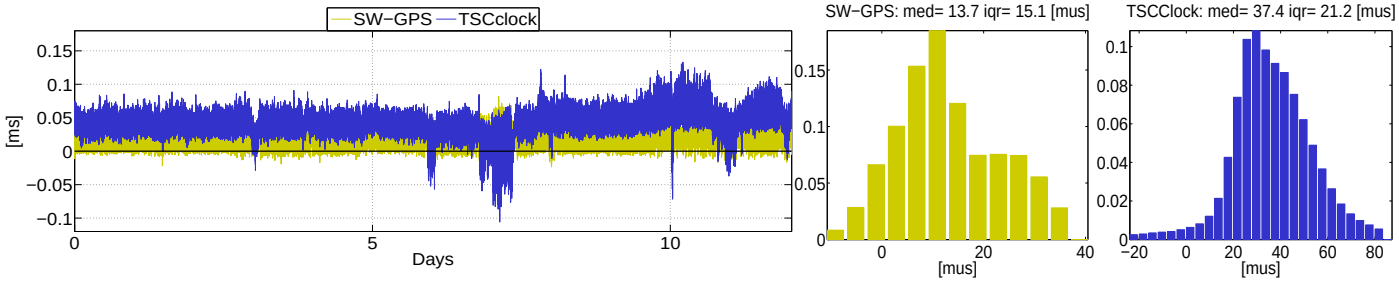

Fig. 11. External comparisons for SW-GPS and TSCclock (captured on *potoroo*).

speed and same network card) and operating system (modified FreeBSD 6.1). Results obtained for the TSCclock should therefore be consistent with those obtained with *bandicoot* as far as system noise is concerned. However, as *Potoroo* is configured to use the SW-GPS system clock (as described in Section IV), the clock comparisons should be very different.

The internal comparison, shown in Figure 10, shows the SW-GPS and TSCclock to be close to each other. The IQR of I_i is only 14μs, indicating that the two clocks offsets vary similarly most of the time. However, the time series show several periods where the clock behaviours diverge somewhat (for example during day 7). We are then interested in knowing if one of the two clocks is responsible for the increase in X_i or if both of them contribute to it.

The external comparisons are shown in Figure 11. Consistent with the internal conclusions, the histograms show that, in terms of clock stability, the two clocks behave similarly, with IQR for X_i values which are close to each other: 15μs for SW-GPS and 21μs for the TSCclock. In fact, since r^h was measured for this host as being around 10 μs, the external error

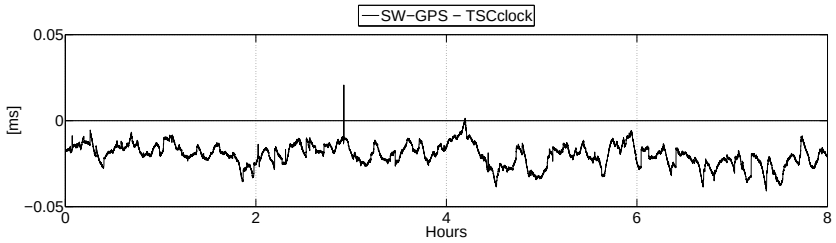

Fig. 12. Internal comparison between SW-NTP and TSCclock (zoom).

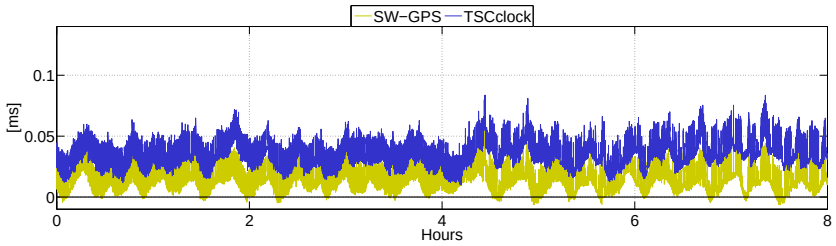

Fig. 13. External comparison between SW-GPS and TSCclock (zoom).

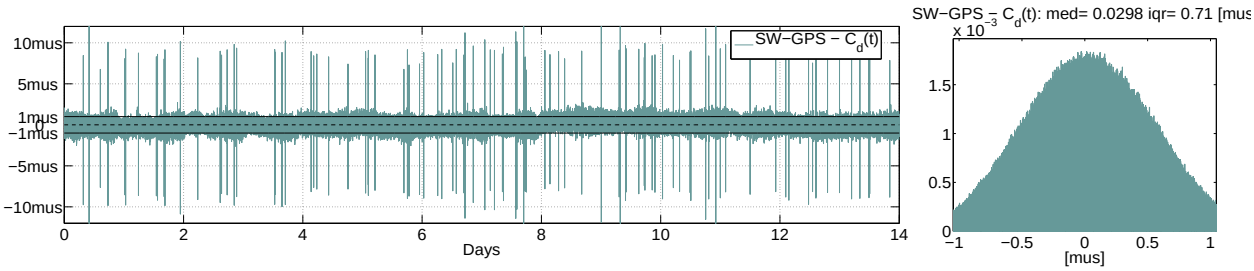

Fig. 14. Internal comparison between SW-GPS and the difference clock.

is consistent with the hypothesis that one of the clocks has an IQR close to zero (very high stability), to which timestamping noise has been added. However, it is impossible to verify this. The difference between the clocks is more pronounced in terms of the constant component of their offset. The TSCclock is slightly ahead compared to the SW-GPS (a median value of 37μs compared to 13μs). However, because the IQR of R^h is around 23μs that is, larger than the IQR of the external comparison for both clocks, we cannot conclude with certainty which one is worse.

The time series in Figure 11 also gives information about the different events occuring during the capture, as noted from the internal comparison above. Since they are present in the internal comparison we know they are not due to system noise, and since they match (mainly) the external comparison for the TSCclock but not for SW-GPS, we deduce that they are (mostly) due to the TSCclock. We investigated the log files of all machines involved to understand what the cause of such a variation in clock offset could be. These events correlate with periods during which the stratum-1 NTP server, to which the TSCclock synchronizes to, lost synchronization to its own GPS signal. We believe that this was due to a hiccup in connectivity arising from disruptive building work on the roof where the GPS antenna is located. Without a good source of synchronization, the TSCclock offset error naturally increased, explaining the additional drifts observed.

Because the SW-GPS and the TSCclock have offsets which follow each other so closely, it is worth taking a closer look to examine smaller scale behaviour. Figure 12 focuses on an 8 hour window showing the internal comparison I_i during the first day of the capture above. We observe oscillations in the time series with a period of about 20 minutes. Again, we want to know which clock is responsible for such behaviour. Figure 13 presents the external comparison over the same period. In this case we see that both clocks are responsible as both have the oscillations. Again, the phenomenon cannot be due to system noise as it is present in the internal comparison also. A comprehensive study of the testbed environment finally led us to discover that these oscillations can be matched with the pattern of temperature variations in the temperature controlled machine-room where *potoroo* is installed. These oscillations are thus due to temperature effects impairing the quality of both clocks.

Finally, we give an example using the difference clock $C_d(t)$. Using the incoming UDP packet events, we compute UDP packets inter-arrival times using both SW-GPS and

the difference clock. We compare inter-arrivals measurements for corresponding packets and present them in Figure 14. As confirmed by the histogram, the errors mainly fall in a narrow $\pm 1\mu$s band. Since $1\,\mu$s is the resolution of SW-GPS as it uses the `timeval` data structure holding a standard system timestamp, we can not interpret the errors in this band, beyond indicating the two clocks behave (almost) identically. However, as seen in the time series, a small number of spikes, of magnitude up to $10\,\mu$s, appear throughout the trace. By the way in which the difference clock is constructed (this was checked by looking in detail at internal algorithm data), we know that it cannot be responsible for these spikes. We are therefore able to diagnose their origin as the SW-GPS clock and/or system call. Note that in this case the external comparison would not have provided any additional insight, as the spike amplitudes are far smaller than the timestamping noise polluting the DAG timestamps.

VI. Conclusion

We presented a testbed and an associated methodology to evaluate software clocks running on computer systems. The use of the system was illustrated in detail using three clocks with very different characteristics, in three computers, two different reference clocks, and weeks of live data.

The system is capable of providing measurements, accurate to a few tens of microseconds, on clock offsets, clock stability, and also system noise polluting clock timestamps. The methodology is in fact a combination of two simpler methodologies. One is external, based on comparison against a highly precise clock located outside the system. We use a high precision GPS synchronized DAG card. The other is internal, based on comparison against a second clock inside the host. Modified kernels (which will be made generally available) were made to reduce system noise between clock timestamps to very low values, allowing in some cases meaningful comparisons to be made right down to the (system) clock resolution of 1μs. Through comparing and contrasting internal and external results, we showed how the disadvantages of each could be substantially alleviated, allowing the origin of anomalous measurements to be tracked down and reliably attributed to their true cause: system noise, the reference clock, or the clock under study. The system provides a well defined and comprehensive method to accurately benchmark software clocks under real conditions, and should be invaluable to the development of new clocks, which are needed for many distributed computing and networking applications.

References

[1] T. Wana, "Improving time measurement in the test traffic measurements project," Master's thesis, Fachhochschulstudiengägne Technik, RIPE-NCC, June 2005.

[2] D. Veitch, S. Babu, and A. Pásztor, "Robust Synchronization of Software Clocks Across the Internet," in *Proc. 2004 ACM SIGCOMM Internet Measurement Conference*, Taormina, Italy, 25–27 October 2004, pp. 219–232.

[3] D. Mills, "Internet time synchronization: the network time protocol," *IEEE Trans. Communications*, vol. 39, no. 10, pp. 1482–1493, October 1991, condensed from RFC-1129.

[4] ——, "Network time protocol (version 3) specification, implementation and analysis," IEFT, Network Working Group, RFC-1305, March 1992, 113 pages, papers in appendix.

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

[6] D. Veitch, J. Ridoux, and S. Babu, "Robust Synchronization of Absolute and Difference Clocks over Networks," *Submitted for publication*, 2007.

[7] A. Pásztor and D. Veitch, "PC based precision timing without GPS," in *Proceeding of ACM Sigmetrics 2002 Conference on the Measurement and Modeling of Computer Systems*, Del Rey, California, 15–19 June 2002, pp. 1–10.