Initial experiences with a clock synchronisation test bed

P Ashton and J Mackenzie,
Department of Computer Science
University of Canterbury
paul@cosc.canterbury.ac.nz
jon@cosc.canterbury.ac.nz

TR-COSC 06/98, August 1998

Abstract

In a distributed system, no hardware facilities exist to synchronise the clocks of the computers within the system. Algorithms designed to synchronise clocks in such systems have been the subject of much research. The lack of a global clock has meant that the accuracy of these algorithms has generally been assessed using indirect measures of clock offsets. We have developed a clock synchronisation test bed that allows for direct measurement of clock offsets. The test bed is now operational, and we have used it to produce initial results for the widely-used `xntp` package, and for some off-line synchronisation algorithms.

1 Introduction

Clock synchronisation in distributed computer systems is a widely studied problem. The primary aim of a clock synchronisation algorithm is to ensure that the maximum difference (or offset) between any two clocks amongst a group of clocks being synchronised is small. Different applications have different requirements as to the degree to which clocks must be synchronised. For some, such as systems that compare timestamps as part of an authentication process, clock offsets of up to a few seconds may be acceptable. For others, such as accurate timing of packet delays, the maximum acceptable clock offset may be in the order of tens of microseconds or less.

Many clock synchronisation algorithms have been developed over the years. Invariably these algorithms have been assessed using indirect measures of clock offset, such as offsets **estimated** by the synchronisation algorithm, or offsets computed by simulation or analytical models. As part of work on developing high-accuracy clock synchronisation algorithms we have developed a test bed that allows for direct measurement of clock offsets. The test bed was developed:

1. to provide a highly accurate measurement tool for use in assessing the effectiveness of clock synchronisation algorithms, and
2. to enable us to determine whether clock offsets estimated using traditional indirect measurement techniques differ markedly from directly measured clock offsets. Knowing this will allow us to assess the validity of using indirect clock offset measures to determine the accuracy of clock synchronisation algorithms.

The overall design of our test bed is detailed in an earlier paper [MA98]. The primary purpose of this paper is to report on the successful implementation of the test bed hardware and software, and to show that the test bed meets its design goals by presenting initial results produced in some preliminary experiments.

In the body of the paper, we start by providing an overview of the hardware and low level software components of the test bed. We then present in two sections experimental results for evaluations of `xntp` and (very briefly) of off-line clock synchronisation algorithms. Included in each section is a description of the software needed to interface the test bed with the algorithms under test. Finally, the test bed is compared with other systems, conclusions are presented and future work discussed.

2 Test bed hardware and low-level software

In most of the literature on clock synchronisation, the clock offsets reported for an algorithm are generated by an analytic or simulation model (see, for example, [Arv94, DHSS85, CFN91, PB95]) or estimated by the algorithm under test (see, for example, [AP92, GZ89, Mil94]). This is understandable given that measurement requires some global (reference) clock, the presence of which would remove the need for software synchronisation algorithms.

To facilitate measurement of clock offsets, it is possible to connect an external source of reference time to each computer under test. The problem then becomes how to record a local timestamp and a reference timestamp at the same instant, or at least with some known fixed delay between them that can be compensated for. A common form of external reference time is a device (such as a GPS receiver) that provides interrupts regularly, often at the start of every second according to some global timescale. The time that elapses between the external device causing the interrupt and the current local time being read varies depending on how long it takes for the interrupt to be delivered, and how many cache misses, TLB misses and similar events occur in the handler for the interrupt before the local timestamp is recorded. If, for example, this delay can vary by up to five microseconds, then an error of up to ten microseconds is introduced into the computation of a clock offset (two clocks are involved).

We wanted to construct a test bed that could measure clock offsets with accuracy close to the clock resolution. Our design goals were:

1. to deliver to all systems under test a reference clock whose values are synchronised by hardware to achieve a maximum offset of well under one clock tick.

2. to provide a mechanism to be able to associate a timestamp read from a computer's local clock (whose timekeeping is under the control of a synchronisation algorithm under test) with a reference timestamp with a maximum delay between reading the two timestamps of well under one clock tick.

Meeting our goals requires hardware support. The hardware components are of the test bed are introduced below—see [MA98] for a more detailed description. We decided to develop a "clock card" that contains counters that form the basis of the local and reference clocks (each computer is connected to the test bed via a clock card plugged into its parallel port). When a local time value is required, the card latches simultaneously the values of the local and reference counters, and returns both values. Because the values are latched simultaneously in hardware, the time that elapses between recording a local timestamp and a reference timestamp is negligible.

Each of the two counters on a clock card must be driven by an oscillator. A single 5 MHz reference oscillator is connected to the reference counters on all clock cards in the test bed via shielded coaxial cables. Time domain reflectometry was used to ensure that differences in cable propagation delays were insignificant. An additional circuit is provided to reset all clock card counters upon the push of a button (the reference clock signal is suppressed for a time after a reset to ensure all counters start at the same time). Because all reference counters are driven by the same signal received from the reference oscillator via cables of the same delay and start counting from the same value they remain tightly synchronised.

Timekeeping in most operating systems is based on a quartz oscillator supplied as one of the computer's built-in devices. We could have tried to have the local counter on each clock card driven by the built-in oscillator of the computer it is connected to, but doing so would not have been straightforward, and would have resulted in a clock card much harder to use on a wide variety of platforms. Instead, we have included a 5MHz quartz crystal oscillator on each clock card, and this oscillator drives the local counter on the clock card. The oscillators used on the clock cards are typical of those incorporated into computer systems.

A problem in the original design (as detailed in [MA98]) was uncovered during initial testing. The counters used have a maximum ripple time (time between an increment occurring and the counter being stable) of 4μs. With a 5 MHz oscillator, 20 pulses occur in 4μs. This caused problems when trying to read the counter into a holding register—the value was not stable and incorrect values were sometimes read. To solve this problem two further counters were introduced giving a total of four: a master local counter, a slave local counter, a master reference counter and a slave reference counter. During normal operation the master counters are connected to the oscillators. When a read begins, clock signals are routed to the slave counters instead. After a 5μS delay the master counter values are latched into holding registers on the card, then the clock signals are switched back to the master counters. The master values are then read from the registers, and then the slave counters are latched to and read from the registers. Low-level software adds each master value to the previous reading of the matching slave counter so that a pair of timestamps (local, reference) is returned to higher software layers.

Functions to initialise the clock card and to return a timestamp pair, consisting of local and reference timestamps, have been written for use inside the Solaris 2.6 kernel. A number of short delays are needed during the read process to give the clock card time to respond. The current function takes approximately 60μs to read a timestamp pair, with the clock signals switched to the slave counters (which is the point at which the timestamps are determined) right at the start of the reading process.

A clock synchronisation algorithm needs to be modified before it can be evaluated using the test bed. This involves getting the algorithm to synchronise a local clock that is derived from the oscillator on the clock card, rather than the normal behaviour which is to synchronise the time of day maintained by the operating system. Descriptions of the necessary modifications are included in the following two sections.

3 Experiments with `xntp`

`xntp` is a widely used clock synchronisation package. It is one member of a large set of programs that we call real-time clock synchronisation programs. These programs attempt to maintain synchronisation between the time of day clocks on some collection of computers on an on-going basis. An alternative approach, off-line synchronisation, can be used where times present in event records logged on several different hosts must be synchronised after event recording has finished, as a first step in data analysis. Off-line algorithms are the subject of the next section. We begin by describing how `xntp` was interfaced to the test bed, then go on to present results of preliminary experiments on `xntp`.

3.1 Interfacing `xntp` to the test bed

`xntp` is a large, complex package with many configuration options. Its traditional method for reading and changing the time of day on a Sun running the Solaris operating system is to use the Solaris calls `gettimeofday`, `settimeofday` and `adjtime`.

Within the kernel, Solaris maintains the current time of day as a pair of values stored in a `timeval` structure: the number of seconds and microseconds since January 1st, 1970. We will call this the *tick resolution time*. Upon each clock interrupt a value equal to the length of the clock tick is added to the tick resolution time (a clock tick length of 10ms is common). When `gettimeofday` is called it reads the number of microseconds since the last clock interrupt (maintained in a device register by most built-in clocks) and returns in a `timeval` structure the sum of this value and the tick resolution time. `settimeofday` sets the tick resolution time to the contents of the `timeval` structure passed as a parameter. `adjtime` adjusts the tick resolution time by the delta passed as a parameter. The change does not occur immediately. Instead, the amount added to the tick resolution time upon each clock interrupt is increased or decreased slightly until the complete adjustment has been made. The change occurs at a rate of $\pm 500\mu$s per second to avoid large steps in the time of day.

We decided that the most realistic way of interfacing `xntpd` to the clock card was to add three system calls to Solaris 2.6: `ccgettimeofday`, `ccsettimeofday` and `ccadjtime`. These system calls operate in the same way as `gettimeofday`, `settimeofday` and `adjtime` except that the `cc` calls use the oscillator on the clock card for time keeping rather than the built-in clock. An advantage of this approach is that many other clock synchronisation programs interface to the clock using the same set of system calls as `xntp`. It will be easy to interface such programs to the test bed.

There are some internal and external differences in the way the `cc` family of system calls operates, some of which are caused by the fact that the current version of the clock card does not have the ability to produce regular interrupts. The main differences are:

- Because there is no regular interrupt to cause the tick resolution time to be updated, a record is kept of the most recent local timestamp read from the clock card. Whenever `ccgettimeofday` or `ccadjtime` is called a local timestamp is read from the clock card, and the difference between the time read and the saved value is added to the tick resolution time. The resulting value is the one returned by `ccgettimeofday`.
- The local counter on the clock card wraps around about every 14.3 minutes. If calls to `ccgettimeofday` and `ccadjtime` are infrequent then the clock card local counter might wrap around twice or more between updates of the tick resolution time, causing the clock to lose some multiple of 14.3 minutes. To prevent this the standard system clock code was changed to (effectively) call `ccgettimeofday` every 10 minutes.
- `ccgettimeofday` and `ccadjtime` both return the newly-calculated time of day, along with the 32-bit local and reference timestamps read from the clock card as part of calculating the current time of day. Because the local and reference counters are latched simultaneously, and because the value read from the local counter is used in updating the time of day, the reference timestamp returned gives the value of the reference clock at the instant the local clock was showing the time of day returned. These extra values are returned for data recording purposes.
- The standard approach to applying adjustments cannot be used because there is no regular clock tick interrupt. Instead, when the current tick resolution time is updated and an adjustment is in progress the amount added is scaled by a factor of up to $\pm 500\mu$s per second. This actually results in a slightly more accurate adjustment function.

Very minor changes were made to `xntp` (version 3-5.91) itself. Programs were changed to use the `cc` family of system calls, and code was added to log the current time of day and the reference and local timestamps returned by calls to `ccgettimeofday` and `ccadjtime`. Also, code was added to report clock card time stamps as part of the information `xntp` prints when it reports one of its estimates of clock offset.

3.2 Analysis of `xntp`

`xntp` uses NTP (the network time protocol) to achieve clock synchronisation. Our departmental NTP setup is a simple one. One machine, `kaka`, is a stratum 3 NTP server that synchronises to two stratum 2 servers at the University of Waikato, which is also in New Zealand. The stratum 2 servers synchronise to a collection of stratum 1 servers (each of which has a direct source of universal time), all of which are outside New Zealand. All departmental computers other than `kaka` synchronise to `kaka`.

The aim of the first `xntp` experiment was to investigate how closely `xntp` was able to synchronise the clocks of two departmental SPARCstation 4 computers (`rua` and `s455`) running as NTP stratum 4 servers synchronised to `kaka`. `rua` and `s455` were connected to the test-bed. On each computer the modified `ntpdate` was executed to provide initial synchronisation to `kaka`. Then the modified `xntpd` was run on each machine for a period of nearly 23 hours. The 5 MHz oscillator on a third clock card was used as the reference oscillator in this experiment.

`xntpd` calls `ccadjtime` once a second. For every call to `ccadjtime` we recorded: the time of day, the reference time and the adjustment requested. The first step in analysis is to correct each reference time for wraparound (wraparound is detected when a reference time stamp is less than the previous one recorded; no wraparounds are missed because a reference time is recorded every second).

Further processing is needed before clock offsets can be determined. Consider rua_X, a record logged on `rua`. If contains the time of day according to the local clock ($rua_X(tod)$), the reference time at which rua's clock showed that time of day ($rua_X(ref)$) and the adjustment requested by the call to `ccadjtime` ($rua_X(adj)$) To, determine the time of day on `s455` at reference time $rua_X(ref)$ it is necessary to locate successive records $before$ and $after$ logged on `s455` such that:

$$s455_{before}(ref) < rua_X(ref) < s455_{after}(ref)$$

In other words, we find the `s455` records that bracket rua_X. From the three reference timestamps we can determine the percentage P of the interval ($s455_{before}(ref)$, $s455_{after}(ref)$) that had elapsed at the instant rua_X occurred. The time that elapsed according to `s455`'s local clock between the two `s455` records is:

$$elapsed = s455_{after}(tod) - s455_{before}(tod) - s455_{before}(adj)$$

We can then calculate $s455_X(tod)$ (the time of day on `s455` at reference time $rua_X(ref)$) by adding to $s455_{before}(tod)$ the sum of $P \times elapsed$ and however much of $s455_{before}(adj)$ would have been applied at $rua_X(ref)$. We then subtract $s455_X(tod)$ from $rua_X(tod)$ to get the offset between the two time of day clocks at reference time $rua_X(ref)$.

This calculation assumes that the rate of the oscillator on the clock card connected to `s455` does not change significantly between $s455_{before}(ref)$ and $s455_{after}(ref)$. It is well known that the rate of a quartz crystal is very close to being constant [EK73]. `xntp` reports an estimate of the current rate of the local clock every 64 seconds. Given two successive rate estimates we can see how much the rate has changed over a 64 second period. The largest rate change (of the 6580 recorded in our data) was 1.611μs per second (in only 5 cases was the magnitude over 0.6μs per second). In the calculation described above, we are interpolating between readings taken 1 second apart. If the 1.611 rate change is spread evenly over the 64 seconds, then the change over a 1 second period would be about 25ns per second. Even if a rate change of this magnitude occurred as a step in the rate at the start of the 1 second period interpolated over, we are looking at a maximum interpolation error of the order of 25ns. Also, because the reference clock used in this experiment is an oscillator of the same type as used on the clock card an error of up to the same magnitude (that is, 25ns) is introduced into the calculation of P. However, these errors are worst case are are still small compared to the clock tick length of 200ns.

Our first analysis of the data collected in the experiment involved computing a clock offset using the method described above for every (time of day, reference time, delta) triple recorded on each computer. This resulted in 165032 clock offset measurements over the 23 hour period. Plotting these offsets as a function of reference time revealed a large spike—for a period of 33.8 seconds the clock offsets were over 240ms. Investigation of NTP logs showed that the spike was caused by `kaka` switching its primary synchronisation source from one of the stratum 2 servers in Waikato to the other. After switching servers, the offset calculated by `kaka` between its own clock and that of the new server was so large that `kaka` simply advanced its own clock by 242ms as a single step. Once the `xntp` daemons on `rua` and `s455` detected this large step they in turn stepped their clocks (by 243 and 241ms respectively). The step occurred earlier on `s455` than on `rua`, hence the half a minute of high offset. It is not clear why `kaka` found it necessary to make such a large step; this is something we plan to investigate further.

Figure 1 shows 164964 clock offsets plotted against reference time for the 23 hour period monitored (the 66 offsets in the 33 second spike are not shown, because if they are included the other offsets, being so much smaller, show up as a horizontal line along the X axis). From the figure we can see that (with the exception of the spike period) `xntp` maintains a good level of synchronisation between the two clocks; the clocks are within 2ms of each other for the entire period, and for much of the time (86% of the offsets) the clocks are within 500μs of each other. The discontinuity at the 22 hour mark is the point at which both local clocks were stepped.

One of the issues we want to investigate with the clock card is the accuracy of traditional estimates of clock offsets such as those produced by the synchronisation algorithms themselves. In the case of `xntp`, the goal is to synchronise all clocks to UTC (universal time). `xntp` records every 64 seconds an estimate of the offset between the local clock and UTC. Figure 2 shows the local clock/UTC offsets reported by `xntp` as a function of reference time. The range of these offsets is two orders of magnitude wider than the range in offsets between the two clocks as measured by the testbed. The fact that the offset between the local clocks of `rua` and `s455` is quite small is evident from the `xtnp` data shown in Figure 2 because the two sets of points follow each other very closely.

By subtracting two UTC/local clock offsets (one recorded on each machine) recorded at the same time we can arrive at the `xntp` estimate of the clock offset

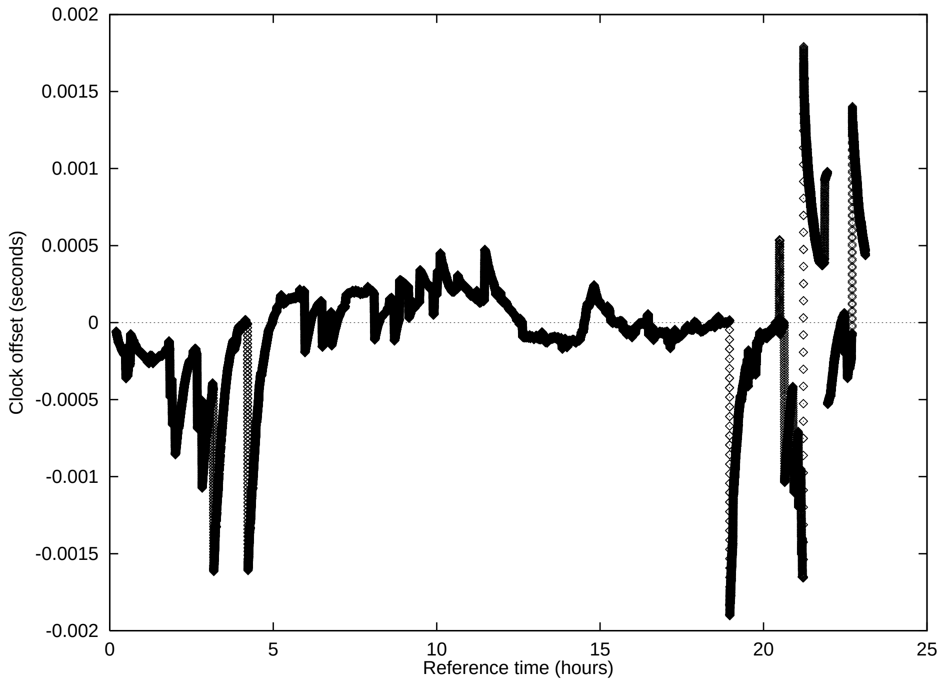

Figure 1: Clock offsets between `rua` and `s455` measured by the test bed

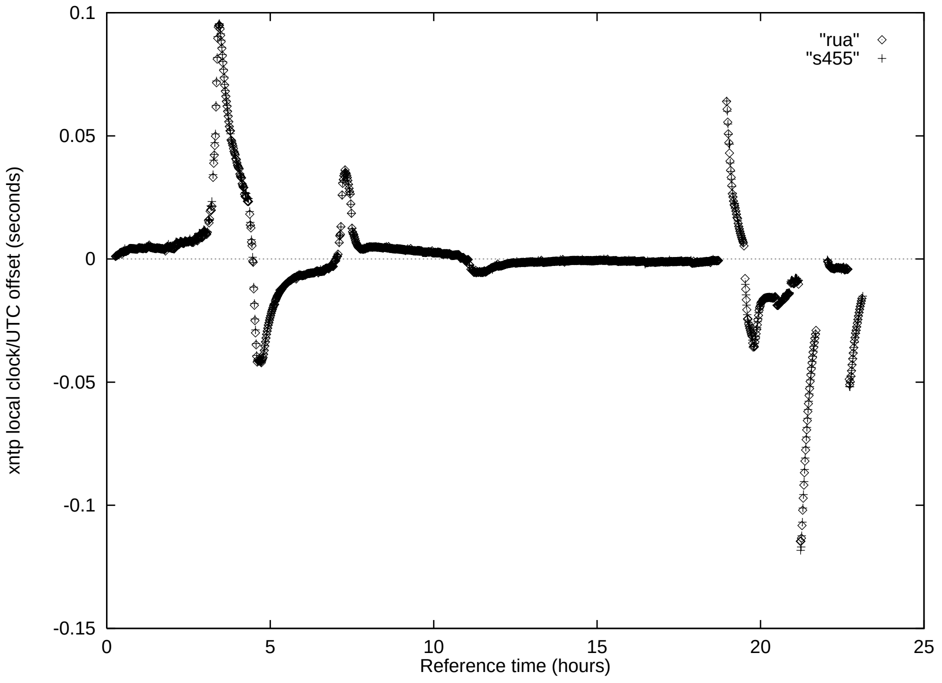

Figure 2: Offsets between the local clock and UTC reported by `xntp`

between `rua` and `s455` at that time. Clock offset estimates are reported by `xntp` at 64 second intervals—again we have to use an interpolation type of technique to arrive at two offsets as at some common reference time. For the `s455` log file an `xntp` offset was associated with each `ccadjtime` record A by taking the most recent offset reported by `xntp`, and adding to it all adjustments (between 1 and 64) made between when the offset was recorded and A. Then for each clock offset reported by `xntp` on `rua`, reference times were used to match it with the immediately preceding `ccadjtime` record on `s455`. The offset recorded by `xntp` on `rua`, and the offset estimate associated with the `ccadjtime` record on `s455` are then subtracted to determine the offset between the clocks on `rua` and `s455` according to `xntp`. We can then subtract the `xntp` determined offset from the offset measured by the test bed at the reference time in the `ccadjtime` record from `s455` to get the amount by which the offset estimated by `xntp` differs from that calculated by the test bed.

The errors introduced into the determination of the clock offset according to `xntp` are in the order of tens of microseconds, because (amongst other things) the `xntp` offsets determined for `s455` are the result of extrapolating over periods of up to a minute from the point where an offset reported by `xntp`. Nevertheless, looking at the differences between clock offsets measured by the test bed and calculated by `xntp` will give a good indication of the accuracy of `xntp` clock offsets (are they within hundreds of microseconds, or milliseconds, or tens of milliseconds, or even worse?). Figure 3 shows the difference between the test bed and `xntp` clock offsets plotted as a function of reference time. Outlying differences of -61, 43 and 104ms do not appear in the figure.

It is clear that most `xntp` offset estimates are accurate to a sub-millisecond level; the mean magnitude of the difference between the `xtnp` offset and the test bed offset is 525μs, with 91% of differences less than 1ms in magnitude. Differences of the order of a few milliseconds are not uncommon, with an occasional difference in the tens of milliseconds range. A second `xntp` experiment was done where `s455` synchronised to `rua`. This allowed the peer offsets between `s455` and `rua` reported by `xntp` to be matched very accurately with an offset measured by the test bed, as each peer offset record contains a time of day and a reference timestamp. This experiment was run over two days, and in that time only 4 of the 2090 differences between the `xntp` peer offset and the clock card offset were larger than 1ms in magnitude (1.9, 1.9, 2.0 and 2.7ms). The average difference in magnitude was 64μs.

4 Experiments with off-line algorithms

Space precludes more than a brief description of an initial experiment in which the test bed was used to determine the accuracy of some off-line clock synchronisation algorithms developed in earlier work [Ash95]. The Solaris ethernet device driver has been instrumented to record packet send and receive events, and to include in each packet local and reference timestamps returned by the clock card. A logging program writes event records to disk for later analysis.

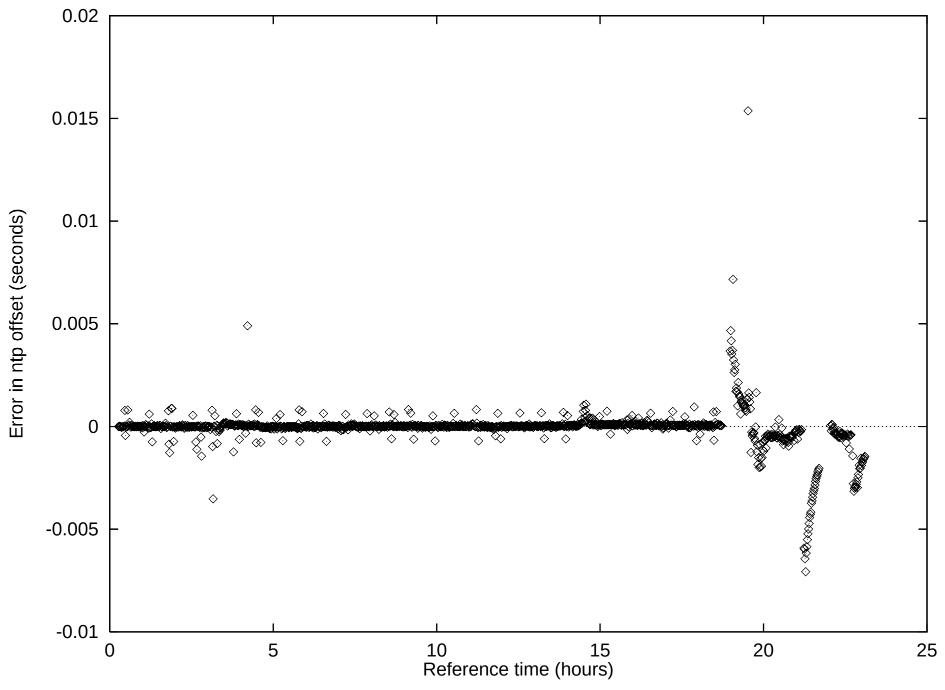

Figure 3: Difference between clock offsets measured by the test bed and offsets determined by `xntp`

In a small initial experiment the ranking of four algorithms was consistent with that given by indirect measures of algorithm accuracy used in earlier work. A very interesting quantitative result was that for the best algorithm used, the maximum clock offset in a 90 second experiment was 10μs.

5 Related work

As discussed above in section 2, nearly all papers quote clock offsets that are either estimates produced by a model or are estimates printed by the running algorithm. `xntp` can be configured to measure an offset between the local clock and an external reference clock. The accuracy of this method suffers from variable delays between taking readings of the local and reference clocks.

Troxel has developed (independently) a TURBOChannel card (for use with DECStation systems) that has some similarities to our clock card [Tro94]. The GDT-TIME board includes a 10MHz oven-controlled oscillator and a 64-bit counter driven by the oscillator. It also includes two external inputs, each associated with a 32-bit capture register. On a rising edge of an external input the least significant 32 bits of the counter are copied to the capture register associated with the external input port.

This hardware can be used to generate (local, reference) timestamp pairs in the following way. The local timestamp is a value recorded from the 64-bit counter on the GDT-TIME board. To make it possible to compute a reference timestamp, an external source of time must be connected to one of the external input ports. Troxel suggests that the 1 PPS (one pulse per second) output of a GPS receiver be connected to an external input port. Once a second a pulse is sent from the GPS and a value recorded in the capture register. Software can be written to monitor the relevant capture register and record values captured for each GPS pulse. For a given local timestamp, the fractional part of a reference timestamp in seconds can be determined by linear interpolation using the local timestamp itself and the PPS timestamps that bracket the timestamp. It is straightforward to determine the integral (seconds) component of each reference timestamp when analysing data from multiple computers.

Each approach has its advantages. Troxel's board does not rely on physical connection of all boards to a master oscillator, which means it can be used where the computers whose clocks are being synchronised are distributed over a wide area. Such experiments can still be done using our clock cards, however, even though the computers under test will generally be in the same room. For example, a computer connected to the test bed can be networked to the other computers in the test bed via a SLIP connection to a distant host, and wide area internet links from the distant host back to the local subnet.

Our clock card will give more accurate reference timestamps because of the offset between the GPS PPS signal and universal time. Troxel quotes an accuracy of 50ns 95% of the time with selective availability disabled and 100ns 95% of the time with selective availability enabled. When comparing two timestamps recorded on different machines, if both GPS receivers are within their stated accuracies then the error in reference times could be up to 100ns (200 with selective availability disabled) and some of the time the error between the GPS receiver and universal time will be greater than this.

Another issue is that while the counter and capture registers on the GDT-TIME board may well be more convenient to read than the clock card registers, our clock card can be interfaced to any system with a PC-type parallel port, so is much more portable. Finally, the local oscillator on each clock card is typical of those currently used in workstations, whereas the GDT-TIME board uses an oven-controlled oscillator. This means that algorithms evaluated using our test bed will be operating under more realistic conditions in terms of the properties of the local clocks being synchronised.

To the best of our knowledge, Troxel's board has not been used in the sort of studies that we are undertaking.

6 Conclusions

In an earlier paper [MA98], we detailed the hardware and software designs for a clock synchronisation test bed. We encountered some subsequent problems while implementing the test bed (such as the problem that required adding slave counters to the clock card), but the test bed is now operational. The test bed is inexpensive (components for each clock card cost less than US$100), portable (it can be interfaced to any system with a PC-compatible parallel port) and very accurate.

The main goal of this paper is to report successful use of the test bed in initial experiments on the accuracy of both real-time and off-line clock synchronisation algorithms. These very preliminary experiments produced some interesting results. For `xntp`, outside the period in which the clocks were stepped, the test bed showed that `xntp` maintained the two clocks within 2ms of each other; whereas even ignoring five large outliers `xntp` estimates indicate a maximum offset between the two clocks of over 8ms. For the off-line experiment, results increase confidence that off-line clock synchronisation algorithms can deliver very accurate synchronisation results.

Now that the test bed is operational, and has been successfully interfaced to real-time and off-line clock synchronisation algorithms, we are in a position to use it in much more thorough investigations of the accuracy of clock synchronisation algorithms.

References

[AP92] Paul Ashton and John Penny. Experiments with an algorithm for high-resolution clock synchronisation. In *Proceedings of the 15th Australian Computer Science Conference*, pages 41–55, Hobart, January 1992.

[Arv94] K. Arvind. Probabilistic clock synchronization in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 5(5):474–487, May 1994.

[Ash95] Paul Ashton. Algorithms for off-line clock synchronisation. Technical Report TR-COSC12/95, University of Canterbury, Department of Computer Science, December 1995.

[CFN91] D. Couvet, G. Florin, and S. Natkin. A statistical clock synchronisation algorithm for anisotropic networks. In *Proceedings of the Tenth Symposium on Reliable Distributed Systems*, pages 42–51, Pisa, Italy, October 1991.

[DHSS85] D. Dolev, J. Y. Halpern, B. Simons, and R. Strong. Dynamic fault-tolerant clock synchronization. *Journal of the ACM*, 42(1):143–185, January 1985.

[EK73] C. E. Ellingson and R. J. Kulpinski. Dissemination of system time. *IEEE Transactions on Communications*, COM-21(5), May 1973.

[GZ89] R. Gusella and S. Zatti. The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3 BSD. *IEEE Transactions on Software Engineering*, 15(7):847–853, July 1989.

[MA98] Jonathan Mackenzie and Paul Ashton. Direct measurement of the accuracy of clock synchronisation algorithms. In *Computer Science '98—Proceedings of the 21st Australasian Computer Science Conference*, pages 563–574, Perth, February 1998. *Acceptance rate 55%.*

[Mil94] David L. Mills. Precision synchronization of computer network clocks. *ACM Computer Communications Review*, 24(2):28–43, April 1994.

[PB95] M. J. Pfluegl and D. M. Blough. A new and improved algorithm for fault-tolerant clock synchronization. *Journal of Parallel and Distributed Computing*, 27:1–14, 1995.

[Tro94] G. Troxel. *Time Surveying: Clock Synchronization over Packet Networks.* PhD thesis, Massachusetts Institute of Technology, 1994.