Time and Time Interval Measurement with Application to Computer and Network Performance Evaluation

Technical Memorandum

Sponsored by: ARPA/ARMY contract DABT63-95-C-0046, NSF grant NCR-93-01002, NSWC/NCEE contract A30327-93, and HP Laboratories (Bristol, UK)

David L. Mills
Electrical Engineering Department
University of Delaware
31 January 1996

1. Introduction

In most experiments in which time is involved, it is necessary to develop estimates of time, frequency and measurement errors from a series of time measurements between the clocks of a number of computers and ancillary devices interconnected by some kind of computer network. However, time is not a physical quantity, such as mass, nor can it be measured relative to an absolute frame of reference, such as velocity. The only way to measure time in our universe is to compare the reading of one clock, which runs according to its own timescale, with another clock, which runs according to a given timescale, at some given instant or epoch. The errors arise from the precision of time comparisons and the accuracy of frequency estimates between the timescales involved.

The usual data collected during a performance run of some experiment might include time offsets, time delays, frequency offsets and various error statistics. While time offsets between two clocks can be measured directly, frequency offsets can be estimated only from two or more time offsets made over some time interval in the experiment. In practice, a sequence of time comparisons can be performed over the lifetime of the experiment and the instantaneous frequency estimated either in real time with a recurrence relation, or retrospectively with a polynomial fit to the data.

Estimating time and frequency errors in real time has been studied by a distinct subspecies of physicists who have made a career of the technology involved. Various means including autoregressive models, Kalman filters and simple weighted-average algorithms are used extensively by national standards laboratories to model cesium-clock ensembles. These techniques have been adapted to computer network and transmission engineering problems as well [MIL95a]. This memorandum explores issues in performing experiments of this type and summarizes various techniques found useful in practice.

2. Determination of Time and Time Uncertainty

The time of a particular clock at some time t is ordinarily represented

$$T(t) = T(t0) + Rt(t - t0) + \frac{D}{2}t^2(t - t0) + x(t) \ ,$$

where T is the time measured at epoch $t0$, R is the rate or frequency and D is the drift (ageing) over the interval since this epoch. The random variable x represents the measurement error (jitter). This formula is equally valid whether the measurements are absolute relative to a given standard timescale or relative to some other clock in the network. In this case, $T(t)$ includes the effects of the (constant) propagation delay and $x(t)$ represents the uncertainty due to jitter in the network and measurement process.

The frequency spectrum of $x(t)$ over the interval $t - t0$ can be quite useful in exposing anomalies due to measurement errors and unexpected synchronization effects. The usual application, however, is in the construction of delay histograms, where the statistic of interest is the probability of exceeding a given delay value. Usually, the region of interest is near the maximum of this characteristic, which is also usually the region of lowest probability. Accordingly, the most useful display is a log-log plot of $p(x)$ versus x, where $p(x)$ is the probability that the measured delay exceeds the x value. The interesting portion of the display lies in the bottom right quadrant, which shows the characteristic for high delay, low probability events.

Frequency offsets between two clocks cannot ordinarily be measured directly; rather, they must be estimated from two or more time comparisons. The error in frequency comparisons has different names corresponding to the underlying physical process that causes them and will be discussed below. In order to understand the nature of these errors, it is helpful to consider the general case of comparing a given specimen oscillator with another oscillator treated as a standard. In the usual case, the standard oscillator has much better stability characteristics than the specimen oscillator, but this may not always be the case.

The most reliable frequency comparisons are done using a series of time offset measurements taken over some period long enough to capture the effects of those factors known to affect the oscillator frequency, in particular, physical vibration, crystal ageing, fluctuations in power supply voltage and fluctuations in temperature. Typical periods range from seconds to months, depending on the intrinsic oscillator stability and precision of measurement.

As an example, consider the measurement of the lunar month as a multiple of the solar day as observed on Earth. Even though ancient astronomers had no accurate clocks, they still could count the number of days (nights) and new moons over some sufficiently long interval, like 26 years, and refine the ratio to a high degree using only long division. However, this assumes the stability of the heavenly oscillators is sufficiently good that errors due these causes can be neglected. In point of fact, if the astronomer is interested in just these causes, then observations may have to be continued over may years and even centuries. This of course is how our Gregorian Calendar was devised.

If only the average frequency error is of interest over some interval, the best technique is the simplest. Simply divide the difference in time offsets measured between the specimen oscillator and the standard oscillator at the beginning and end of the interval by the length of the interval. It can be shown that the mean error in this technique is no larger than any other technique involving more than these two measurements. If a more detailed frequency analysis is necessary, such as to determine the maximum error over some integration interval, the number of comparisons must of course be increased.

In many cases the statistic of interest is the inherent stability of the specimen oscillator, which can be determined by a sequence of frequency measurements over some interval. Since the frequency measurements themselves are determined from a sequence of time comparisons, a simpler method might be to determine the stability directly from the latter, which is usually the case in measurements of this type. There are two ways in which these data can be assessed: Fourier spectrum analysis and Allan variance analysis. The two are interrelated and stability characteristics can be determined from either one. The choice is largely made on the basis of the experimenter's socio-mathematical culture.

A frequency (power) spectrum can be obtained directly from the frequency measurements described above or indirectly from the time comparisons using the Fourier transform. Ordinarily, one would

expect the variance in the frequency measurements to be relatively large at small intervals between measurements and decrease as the length of these intervals increase, as long as the errors inherent in the time comparison process itself are uniformly distributed. Therefore, one would expect the frequency spectrum to display a similar falling characteristic with increasing Fourier frequency. For the smaller intervals between comparisons, the errors are typically called jitter and the underlying physical effect is due to oscillator phase noise and network delay variations.

On the other hand, as the intervals between comparisons become relatively large, like in the order of days or weeks, the errors become dominated by other factors, such as temperature, ageing and other random effects. Therefore, one would expect the frequency spectrum to reach a minimum and then inflect upwards. In the transition region the falling and rising portions of the characteristic, the errors are typically called wander and the underlying physical effect is due to oscillator frequency noise (flicker frequency noise). In the region where the characteristic is rising, the errors are typically called drift and the underlying physical effects are due to crystal ageing and related phenomena (random-walk frequency noise).

An alternative to frequency analysis is Allan variance analysis, in which a series of time comparisons is processed to produce a graph of variance versus interval similar to the frequency spectrum, but differing in the slope characteristic. As in the frequency spectrum case, the graph displays a broad minimum between two maxima, one corresponding to the jitter regime, then ranging through the wander regime and finally to the drift regime.

As pointed out in [MIL95a], ordinary room-temperature quartz crystals display a broad minimum in the Allan variance characteristic in the range of 20 minutes to several hours, depending on the quality of the oscillator and its temperature and voltage regulation. One conclusion to draw from this observation is that it doesn't make much sense to attempt to reduce time errors and improve frequency estimates using measurement intervals much longer than this. Another conclusion to draw is that the measurement intervals should be long enough that the expected jitter shown in the Allan variance characteristic is comparable to the accuracy required.

3. Time and Time Interval Measurement Technique

It is likely that most applications in which computer network time measurements are involved will operate at speeds where real-time data analysis is difficult or impossible. In these applications, timestamp data may be collected in real time and refined in only rudimentary ways before recording on some medium for later analysis. A typical application may involve a transmission system between two geographically separated locations, with a recording device at each location. In some cases it may be possible to synchronize the recording clocks at each location, such as using the Global Positioning System (GPS). so that the intrinsic frequency errors can be suppressed. In other cases, this may not be possible, so that the frequency errors must be compensated as part of the data analysis procedure.

The general model of a test and measurement interface consists of an oscillator and buffered counter, which can be read on demand by the resident application program. In some designs, a counter overflow or compare with a programmable register can signal an interrupt to the application program. In order to monitor a telecommunications system, a set of data receivers and transmitters is provided, along with buffers, FIFOs and combinational logic as required to capture frames, cells or packets as they transit the system. The intent of this design is to capture a precision timestamp relative to the local clock when an event such as a cell arrival with predetermined header is found.

Timestamps captured in this way can be preprocessed in the instrument itself, then either saved in a local file for later retrieval or, when possible, transmitted to another machine via a local net for real-time data reduction.

In order to measure statistics such as frame or cell delay of a ATM switch, for example, a number of test instruments can be deployed in a distributed configuration and connected to other machines for data collection and processing. In such cases it may be necessary to synchronize the oscillators and counters in the instruments, which may be widely separated. Depending on the requirements, the instruments can be synchronized in real time, which requires them to exchange messages in a protocol such as the Network Time Protocol (NTP) or equivalent, or the timestamps can be corrected during post processing of the data.

3.1. Time and Frequency Synchronization

There are several methods by which time and frequency synchronization can be accomplished, some using external references (e.g., timing and navigation receivers using any of a number of frequency/time dissemination radio, satellite or modem services). Others might use inband signalling, such as common network clocks, piggyback messages inserted in the data stream, or periodic calibration facilities, such as a portable precision clock. The specific means used will depend on the accuracy required, the characteristics of the instrument clock oscillators, and the control of environmental factors such as oscillator temperature.

In general, precision synchronization of instrument clocks requires periodic comparisons between them or, if absolute synchronization to a global clock such as UTC is required, periodic comparison between them and a source of national time standard, such as a radio or satellite signal. This requires some means to capture timestamps upon departure and arrival of the comparison messages, which could be exchanged using the same local net used for data collection, or using the telecommunications system itself, as described below.

In any case, consideration needs to be given to synchronization in both time and frequency. In general, time synchronization requires a two-way channel, so that propagation delays can be measured. However, frequency synchronization can be achieved with only a one-way channel. For example, if the instrument clock of an upstream device is chosen as the master and all downstream instruments can receive a common frame or cell flagged by the upstream device, the downstream devices can discipline their clock frequencies to the upstream instrument. However, this method does not provide for time synchronization; that requires a two-way transmission media or, at least, a backchannel using the same local net as used to collect the data timestamps. In principle, this scheme can result in rather good measurement accuracies, since once the various oscillator frequencies have been stabilized, the baseline time delays can be refined over relatively long integration times to improve accuracy.

If accurate time synchronization is required in all instruments of the distributed network, it is necessary that precision timestamps be collected upon departure and arrival of the time-synchronization messages. If these messages are embedded specifically in a (bidirectional) data stream, accurate timestamping facilities may already be available, since this is required for the data collection function itself. However, if these messages are embedded in the data collection stream between the instruments and the data collection machine, existing off-shelf interfaces and/or the software drivers may not have sufficient precision. In such cases, it may be necessary to modify the

drivers. This comment applies specifically to the ubiquitous PC, which is discussed later in the memorandum.

3.2. Measurements Using Aggregation and Sampling Techniques

While the above methods assume some kind of inband or outband signalling between the test instruments, a particularly interesting scheme could use asynchronous statistical sampling of the data stream in order to synchronize the test instrument clocks in time and frequency. One way to develop such a scheme is described in this section.

Most telecommunications devices being monitored, such as a ATM switch or SDH trunk, include traffic transiting the device in both directions, so that the chance of finding frames or cells in the aggregate data stream going either way across the device is statistically significant. For many applications, this assumption can be justified. For instance, a typical ATM switch is used to switch between two or more SDH trunks, with some input and some output ports of the switch dedicated to each of the trunks.

In this scenario with a statistically unbiased sample of cells collected at the input and output trunk interfaces, some will be in transit one way across the link and some will be in transit the other way. If an arbitrary input cell could be marked in some fashion, such as a message digest, and the same cell identified in the output population by means of this digest, it would be possible to develop a delay distribution for the switch in steady state. Delay samples from this distribution can be processed by the NTP algorithms, for example, to develop an estimate of the baseline switch delay. Using these data, and assuming the baseline delay for one direction of propagation is the same as the other, the timestamps collected at two instrument monitoring points on the switch could be used to synchronize the clocks in time, as well as frequency.

If the two directions of propagation across the switch are statistically different, the baseline delays can be calculated for each direction separately. If it is not possible a-priori to determine the direction of propagation, it may be possible to apply mathematical theories designed to extract individual distributions from a mixture of distributions. While the existence of these theories is known, it is not known whether they can be directly applied to the problem or whether additional development will be required.

From the probability density functions constructed from the bidirectional statistics above, it should be possible to construct probability density functions for the roundtrip delays and clock offsets relative to a distributed set of test instruments. In general, these will take the from of convolutions of the individual density functions. Statistical quantities such as means and variances can be readily estimated from these convolutions. Besides forming the primary data used to drive the various algorithms of NTP, for example, they form the basis for estimating the maximum error and expected error of the paradigm. These data can be used either in real time to adjust the local clocks in the instruments themselves, or the data can be passed to a measurement host and used later to resolve local clock offsets when the actual test data are processed.

3.3. Measurement Experiment Examples

Past measurement experience on ARPAnet and NSFnet has demonstrated considerable value in some unusual measurement technique. The experiments typically involve one-way and roundtrip delay measurements using NTP and designated time servers independently synchronized to a source of standard time, such as a radio or satellite receiver. The experiments have been of two types, one

designed to measure the roundtrip delay versus message length characteristic, the other designed to calibrate the transmission path routing delays. Both depend on accumulating a relatively large number of measurement samples exchanged on the order of one message per second and with data collected over many minutes or hours.

In one experiment, roundtrip delay data was recorded for various message lengths randomly distributed from the minimum size to the maximum transmission unit on the ARPAnet. The results form a scatter diagram plotting delay versus message length. Under moderately heavy load, the points tend to cluster along a straight line sloping upward to the right. A regression line is plotted and the residuals determined. The y intercept of this line is the propagation delay, while the slope of this line is the incremental delay per octet, the reciprocal of which is proportional to the link speed. The residuals with respect to the regression line are the result of other traffic sharing the link and thus represent a crude indicator of traffic intensity. In this particular experiment, a peculiar effect was noticed, in that the line had an interesting inflection in slope at about the ARPAnet maximum packet size, 1008 bits. This was analyzed and was found to reveal an interesting consequence of the ARPAnet fragmentation/reassembly scheme. The conclusion to be drawn from the experiment is that scatter diagrams such as this may be valuable in diagnosing unexpected effects of a similar nature.

In another experiment, delay and offset samples were collected using NTP and a very busy ARPAnet path across the country. Ordinarily, the scatter diagram plotting (signed) offset versus delay would appear as a wedge-shaped locus of points, where the apex of the wedge represents the baseline propagation delay and clock offset. However, in this experiment, there were four wedges apparent by eye in the plot, one at the apex, another about 540 ms behind it on the centerline and two at about 270 ms behind it on both the top and bottom of the centerline. The ultimate explanation of this phenomenon turned out to be a previously unsuspected and intermittent route flapping between the landline ARPAnet path and a backup satellite path. It is unlikely the flapping would have been discovered by any other conventional experiment.

Another experiment was designed to calibrate the time taken in the kernel to process a gettimeofday() system call. It was designed to validate the performance of kernel timekeeping changes to support a multiple-CPU system where each CPU had an individual clock oscillator and counter. The program looped on the system call and reported the system time at each call to an in-memory array, while carefully avoiding page faults and swaps. The results, as displayed in histogram form, revealed intricate details on kernel latency, CPU scheduling, timer interrupts, context switches, CPU and lookaside caches, and various housekeeping functions. Perhaps the most alarming result, so far unexplained, of this particular experiment was an extremely long tail up to almost a second in the delay histogram.

These techniques are obviously non-exhaustive and only scratch the surface of the possibilities. They are included here to suggest that a great deal of information on system performance and operational anomalies can be extracted from time-series data with suitable experiment design and data analysis procedures.

4. Instrumentation for Time and Time Interval Measurement

With the exception of general purpose laboratory test instruments such as oscilloscopes, frequency counters and signal generators, most test instruments and measurement equipment are purpose designed to serve some relatively narrow purpose. Frequently, these instruments are used in a

complex test arrangement involving direct or indirect attachment to another device, such as an embedded computer, transmission link or packet switch. This may involve the collection of performance data requiring accurate timestamps relative either to a synthetic clock maintained as part of the measurement apparatus or an absolute national standard. This section explores some of the issues involved in synchronizing the various clocks in the measurements and maintaining the expected accuracy of the timestamps.

Consider a generic measurement subsystem, for example, one designed to capture some feature or another of a transmission system and record the occurrence and time for later analysis. An example might be a system which triggers on the passage of an ATM cell header and determines its flight time from one ATM switch to another. A general purpose logic analyzer could be used for this purpose, or a special purpose instrument designed specifically to capture data from a transmission system such as ATM, SONET or FDDI could be used. What provisions could be made in the instrument manufacture that would result in better quality measurements? What provisions could be built into the switch itself that would enhance its measurability? What algorithms would be appropriate to refine the data and present an accurate picture of the experiment results?

It is worth noting at this point that almost every device involved in tests and measurements of this type involve a computer, overt or embedded, and a surprisingly large number of asynchronous clocks. Some of these clocks may be inherent in the measurement system and used to derive data used in performance analysis. Others may be intrinsic to the system under test, such as codec clocks, while still others may be implicit in the data processing and reducing equipment, including workstations used for experiment control and data recording. Each of these clocks can affect the experiment quality in sometimes devious ways.

For example, consider some hardware provisions that might be included in a generic ATM switch and which might facilitate instrument synchronization for performance measurements. A typical system may have a separate line card for each transmission link. The line card can contain an oscillator to derive the transmit link clock and a phase-lock loop to derive the receive link clock. In some configurations, the transmit clock of one node is slaved to the receive clock in that node, so that the link timing is determined from a single frequency source in an adjacent node. In any case, it may be useful for an external device to access a signal derived directly from the link clock(s) in use, so provisions should be made in the line card to connect to these source(s). Sometimes, this amounts simply to the provision of a test point on the card suitably buffered to prevent accidental shorts from affecting the transmitted or received signals.

The following subsections discuss issues involved in providing high resolution timekeeping in workstations and PCs, with special emphasis on hardware and software adjuncts to improve stock system performance. A related issue is how special purpose bus peripherals can be used to synchronize the kernel time-of-day facility to the same order of magnitude. Ordinarily, this is not required if all timestamps are captured relative to the peripheral local oscillator and counter; however, if the instrument clocks are to be synchronized in real time, it would be convenient of the kernel system clock were synchronized to that order of magnitude. Kernel system software is available for Unix workstations which can discipline the kernel system clock with respect to an external interface oscillator in this fashion. It would be of interest to explore how and in what form this software could be used in telecommunications monitoring and measurement instruments.

4.1. Synchronizing the Instrument Clocks

In many measurement and calibration experiments, commonly available workstations will be used to generate scripts, record and archive the data and in general control the experiment. While it may be the case that the actual data produced by the experiments are synchronized to a common experiment clock as described above, it will generally be the case that the management workstations need to be synchronized to standard time in order to provide a network-wide common timescale, even if the quality of this timescale is rather poor in comparison with the actual experiment clock.

The technology of computer network time synchronization has developed to the point that means for synchronizing workstations to the order of microseconds are commonly available for most workstations manufactured today; however, special considerations may affect some architectures, including the ubiquitous PC. Most PCs based on the ISA or EISA bus have very poor system clock resolution compared to Sun and HP workstations, for example (DEC is a special case, since recent DEC kernels do not yet support a high resolution system clock). If system clock precisions comparable to these workstations are to be maintained using a PC, special provisions need to be made.

The following subsections discuss issues involved in providing high resolution timekeeping in workstations and PCs, with special emphasis on hardware and software adjuncts to improve stock system performance. A related issue is how special purpose bus peripherals can be used to synchronize the kernel time-of-day facility to the same order of magnitude. Ordinarily, this is not required if all timestamps are captured relative to the peripheral local oscillator and counter; however, if the instrument clocks are to be synchronized in real time, it would be convenient of the kernel system clock were synchronized to that order of magnitude. Kernel system software is available for Unix workstations which can discipline the kernel system clock with respect to an external interface oscillator in this fashion [MIL96a]. It would be of interest to explore how and in what form this software could be used in telecommunications monitoring and measurement instruments.

4.2. Auxiliary Clock Architecture

If the PC or workstation motherboard does not support a high resolution system clock, it may be necessary to provide some sort of bus peripheral with this capability. A suitable board would include a stable oscillator, together with counter and buffer registers providing a multiple-word value which can be read as an atomic operation. There are numerous enhancements that can be included in the board design, including provision of a built-in radio clock and various Auxiliary synchronizing circuitry; however, the basic oscillator-counter-buffer functionality may be all that is required. TrueTime and Bancomm both manufacture boards of this type.

The architecture of a typical bus peripheral suited for high resolution time capture could be modelled on a particular interface for the Sun SBus made by Don Nelson at University of Delaware. It consists of a temperature-compensated crystal oscillator, a pair of Xilinx programmable gate arrays and associated buffers. One of the gate arrays is used for the bus interface buffering and handshaking functions, while the other, which can be programmed from the host machine, is used for the clock functions.

The clock functions include a 5-MHz oscillator and divider, which can operate with division ratios of 4, 5 or 6. The particular ratio is controlled by an auxiliary counter, which is also driven by the 5-MHz oscillator. Ordinarily, the divider operates with a ratio of 5, which yields a nominal master

clock frequency of 1 MHz. However, when the Auxiliary counter overflows, the ratio can be temporarily changed to either 4 or 6, in order to gain or lose an increment of 200 nanoseconds, respectively. On overflow, the auxiliary counter is initialized from a register that can be loaded from the host machine. Thus, the master clock frequency can be adjusted over a small range by the host machine in order to discipline its frequency to an external source.

The master clock oscillator drives a 64-bit counter in Unix timeval format; that is, a 32-bit value for microseconds and a 32-bit value for seconds. It is important in this design that these values be read as an atomic operation in a memory-mapped fashion in both user space and kernel space. Since many machines have bus widths smaller than 64 bits, it is necessary to provide some means to perform the atomic read, as well as insure monotonicity and consistency, since more than one virtual process may attempt to read the values at the same time. Furthermore, the values are produced using a simple ripple counter, in which carries can take some time to complete, possibly longer than a bus cycle. Finally, we would like to minimize the number of gates involved, in particular, to avoid the need for complicated FIFO buffers and latches.

An appropriate design which solves the above problems involves both a modest amount of hardware and a software counter-read procedure which assures monotonicity and avoids inconsistencies. The hardware consists of a 32-bit latch which is loaded by a software command from the microseconds value. The seconds value is read directly from the counter, so may on occasion deliver inconsistent data due to incomplete carry propagation, for example. The software routine that reads the 64-bit counter first reads the seconds value, which also latches the microseconds value, then reads the latch, then reads the seconds value again. If the first and second reading of the seconds value are different, the procedure is repeated.

The above software counter-read procedure is classic, having been in use for many years in similar applications. It remains to show that this procedure delivers monotonic, consistent time when multiple processes attempt to read the counter near the same time and where one proves (e.g., the kernel) may preempt another. In such cases, the intended semantics is that the time returned to a process may lie anywhere in the interval between the first and second readings of the seconds value. This applies whether or not the process was preempted between these readings.

In the design described here, the most recent reading of the seconds value causes the microseconds value to be latched. It could happen that a user process reads the seconds value and is then preempted as the result of a kernel interrupt, for example. If the kernel process also reads the seconds value, the microseconds value will be updated. However, this is completely consistent with the above semantics. As a practical matter, if the preemption results in a tick of the seconds value, the user process will restart the procedure anyway. Since the apparent microseconds value is guaranteed monotonic and must be consistent with the most recent seconds value, the values returned are always monotonic and consistent

Experiments designed to evaluate the performance of computer clocks suggest strongly that the configuration of the oscillator and clock divider circuitry be readable using designated management protocols, such as SNMP. This facilitates automatic measurements in those configurations involving different processors with different clock speeds and divider configurations. For example, in both Alpha and SGI processor architectures, the CPU clock frequency can be determined directly, while the Alpha includes a CPU clock frequency divider that can be read by a CPU instruction. This allows performance data to be collected and later analyzed specifically for each CPU clock frequency

without specific configuration management. This both simplifies data analysis and minimizes cockpit errors.

4.3. Software Interface

There are two issues with this approach that need to be explored: how the operating system and application program can access the auxiliary system clock and how the clock can be synchronized with the clocks of other machines in the network and with standard time.

It goes without emphasis that an auxiliary clock feature should be useful and effective without requiring PC operating system modifications. In fact, it may be possible by hooking interrupts and providing suitable board circuitry to synchronize the PC system clock to the auxiliary clock, so that all system and application programs that call the system clock routine will read the auxiliary clock time; however, that may not be necessary for all or even most experiments. Obviously, it is a simple matter to link a suitable replacement for the system library clock routine in a generic application program. Assuming the auxiliary clocks in all PCs are synchronized to standard time by some means, the application programs used in the experiment will all read standard time to some degree of acceptable error.

This still leaves the question of system programs and other application programs which are not easily changed to use the auxiliary clock. There are a couple of strategies to deal with these situations. The most useful may be simply to keep a journal file in every PC during the experiment. At suitable intervals, perhaps once per minute, a background TSR program appends both the system clock and auxiliary clock values to a file. Should it be necessary to correlate the system clock and auxiliary clock at some future time, a regression line can be fitted to the journal data and corrections interpolated for the data.

Another way to approach these issues is with available implementations of NTP for Windows 3.1, Windows 95 and Windows NT. These shareware programs discipline the operating system clock, so that no further corrections are necessary. However, this approach assumes the availability of a NTP subnet serving the PC; in particular, a workstation running the NTP Unix daemon and either synchronized to a local radio clock or externally via the network. It should be emphasized that this approach does not in itself provide a high resolution clock, since the typical PC motherboard provides a clock resolution of only some milliseconds.

4.4. Hardware Interface

For the best accuracy likely to be useful in experiments, it is necessary to use an auxiliary clock as described above. For those applications not needing the high resolution clock, the ordinary system clock can be used with no changes in software. In this case the system clock is assumed disciplined to national time by some means, as described above. However, this still leaves open the issue of synchronizing the auxiliary clocks themselves, assuming there is more than one and each is to be disciplined to a common source, national standard or otherwise.

There are several ways to synchronize the oscillators in a network of PCs using a suitable auxiliary clock board. Examples found in the products currently manufactured include pulse-per-second (PPS) signals, Inter-Range Instrumentation Group (IRIG) signals, standard time signals and indirectly using NTP or other computer network time synchronization protocol. It is in principle also possible to synchronized the clocks sharing a common bus using, for example, GPIB bus signals or the FDDI token, but these techniques are beyond the scope of this memorandum.

PPS signals are produced by a number of laboratory instruments, including GPS receivers, cesium clocks and even some auxiliary clock boards. An auxiliary clock oscillator can be disciplined to the PPS signal using a divider, phase detector and phase-lock loop, which is included in the board circuitry. In this scheme some means must be provided to determine the seconds numbering, since the PPS signal itself does not provide this function. It is possible to use NTP for this function, as described in [MIL93].

The same thing can be done using IRIG signals, which have the advantage that they can be transmitted over ordinary telephone wires and AC-coupled circuits. An interesting consequence of this scheme is that an ordinary audio port can be used to demodulate and decode the IRIG signal and provide a synchronization signal without the need for and auxiliary clock board. This scheme has been demonstrated successfully using the Sun audio port and a special Sun audio driver included in the NTP software distribution, but has not been ported to other architectures.

While the IRIG signals provide the time-of-day and day-of-year function, they do not provide the year and, in many schemes, do not provide advance warning of a leap second. Nevertheless, there may be some utility, for instance, in a scheme to synchronize the clocks in a distributed network of PC monitors for an ATM switch, for example, using a voice channel supported by the switch, along with a set of inexpensive sound boards for the PCs.

Much the same utility with the IRIG signals can be achieved using off-air standard time signals, such as disseminated by some standard frequency and time stations (in the US and Canada), which provide a voiceband timecode transmission which can be read by a computer equipped with a suitable interface. The same comments as above for the IRIG signals applies in this case, at least with the US transmissions, although no audio driver is known to exist for this signal. The Canadian signal can be demodulated by an ordinary wire-line modem and read by a computer equipped with a standard serial port. A driver compatible with this scheme is included in the NTP software distribution.

There has been considerable experience using high stability oscillator sources, such as standard frequency oscillators, ovenized quartz oscillators and radio clocks which provide only a PPS signal and not a computer readable time-of-day or day-of-year indication. In many applications, such signals are available at various nodes in the network, for example as derived from a common system clock. However, some means must be available to number the seconds with respect to a common timescale. In practice, NTP has been used to do this with a high degree of reliability. The technique is described in [MIL94b].

5. Computer Workstations as Measurement Platforms

One of the most useful time and frequency measurement devices is the ordinary computer workstation. Most modern operation systems, including those based on Unix or DOS, provide extensive file and program management features necessary to run typical measurement programs. There are several commercial products specially designed for data reduction and analysis, including the S system from AT&T and the MATLAB system from MathSoft, as well as a number of general purpose spreadsheet programs, including Excel, Lotus 1-2-3, and mathematical modelling programs such as Mathematica and MathCAD. The availability of these programs that run on the same platform used for the actual real-time data collection function, is a powerful inducement to their use.

A major drawback to the use of general-purpose workstations for precision data collection is usually the facilities available to actually capture the data in real time. For instance, a typical experiment may involve a transmission test set which detects frame preambles or cell headers on a transmission line. Such a device may buffer some or all of the cell contents and provide these data on a serial port. However, since the usual serial port has no provision for precision time capture, the device may signal the cell time of arrival by generating a pulse on a signal connector. While the workstation can easily capture the serial data stream, most general-purpose workstations provide no means to capture the timing signals.

Most Unix kernel programmers are far more concerned about the performance of the workstation with respect to average kernel overhead than to the hardware and software latencies of the individual requests. For instance, the average hardware and software latency for a serial port interrupt is about 8 ms for a Sun IPC kernel, but the maximum is several times this. Without artful use of hard and soft interrupts, the Sun kernel would ordinarily be considered a poor real-time measurement platform.

At issue with these kernels is a generic capability to create a kernel interrupt upon receipt of a synchronization signal, such as the PPS signal from some timing receivers and laboratory standards. There are two issues inherent in this paradigm, the capability to recognize and respond to such signals, perhaps in the form of a modem control-leads interrupt, and the capability to discipline the kernel system clock using these signals.

Kernel code to implement both capabilities described above has already been demonstrated using Unix kernels for a number of workstations, including those made by Digital, Sun and HP. An agenda to be pursued involves a generic, standard interface recognizable by all of these platforms in the form of a POSIX-compliant interface, most conveniently in the form of standard ioctl() system calls. This would allow portable application programs in conjunction with suitable hardware synchronization sources, to synchronize the system clock to accuracies in the order of a few microseconds with current high performance workstations.

There are several ways in which the performance of a general-purpose workstation can be substantially improved with very little additional cost in hardware or software. This can be done by minor changes in the serial port driver code common to many Unix kernels, including those for HP, DEC and Sun, and very likely kernels for most other Unix machines. The following section contains some suggestions on how this may be done.

5.1. Hardware Timers and Counters

Most modern workstations are equipped with a hardware time-of-day clock/calendar and a high resolution counter used for kernel "tick" interrupts. The counter interrupts the kernel at rates from 10 ms for most workstations to about 1 ms for the fastest workstations. A counter interrupt causes a number of microseconds or nanoseconds equal to the tick to be added to the kernel time of day. In some cases where the number of counter increments per second does not evenly divide the second in microseconds or nanoseconds, a fudge value is added to the time of day each second to make up the difference.

For those architectures used by Sun and HP, for example, the high resolution counter can be read by the kernel and used to interpolate between counter interrupts. This feature can be used to provide a precision time-of-day function with resolution limited only by the signal source driving the counter. Curiously, some producers have elected not to provide the interpolation function even when

the hardware is available, so the resolution of the time-of-day function is limited to the tick interval selected. In general, this makes these workstations unsuitable for precision time measurements.

Fortunately, some producers, including Digital and HP, recognize the need for a precision time interval function, if only to measure the performance of software modules. The Process Cycle Counter (PCC) of the DEC Alpha architecture is an example. This counter runs at the CPU clock rate, or some submultiple of it, and can be read by a special machine instruction. Since current Alpha clock frequencies are in the 100-300 MHz range, the ultimate time measurement precision available with this counter is truly awesome. When used as the basis for external time events, however, the counter reading must be converted to standard kernel time, which is usually in seconds and either microseconds or nanoseconds. In the case of the Alpha, this means probing the CPU to find out what its intrinsic clock frequency is, then converting the PCC to standard time on every reference. While the number of CPU cycles to do the division is not normally a burden, the need to do the conversion is awkward.

In architectures such as the Alpha, which includes a high speed counter in the CPU itself, it may not be hard to provide an extremely fine time-of-day resolution; however, in architectures without these features, this function must be provided using an external counter which must be read via an external bus. In order to avoid stressing this bus, the counter is typically operated at rates well below the CPU clock rate, typically some submultiple of the bus clock rate. However, it may well happen that the oscillators used to derive the time-of-day clock, tick interval counter, CPU clock and external bus clock are each asynchronous relative to the other. In addition, in the case of symmetrical multiprocessor systems with internal CPU counters such as the DEC Alpha, each CPU may have its own clock oscillator as well. The issues may have less to do with synchronizing the system clock as with conducting a clock orchestra.

Fortunately, the techniques for disciplining possibly more than one oscillator used to derive the system time-of-day clock are well known. The logical frequency of each oscillator is controlled over a small range by periodic additions or subtractions of a computed value which determines its frequency. The logical oscillator is then encapsulated in a software feedback loop implemented as a phase-lock loop, frequency-lock loop or hybrid phase/frequency loop as described elsewhere ([MIL95a]. The result is a uniform nominal system timescale such that a programs running on each CPU sees a nominal timescale within a microsecond or two relative to the other CPUs and always monotonically increasing.

In summary, it should be the expectation of every modern hardware architecture and kernel implementation that the precision of the time-of-day function should be at least to the degree of the native time-of-day format or the microsecond, whichever is larger.

5.2. Controlling Timed Events

While with modern hardware architectures and kernel implementations it is possible to achieve timing accuracies relative to an external event to within a few microseconds, it is much a different matter when the kernel is asked to generate an event at a designated time. The reason for this is that external events can be initiated only at the instances of tick interrupt, which is typically at intervals in the 1-10 ms range; in other words, far coarser than the precision of the clock itself. For most purposes, this is not a problem, since the system is seldom asked to generate some output at a precisely determined time and, when it is, there is generally some sort of dedicated peripheral to do this, such as a counter subsystem board peripheral, for instance.

A precisely timed external event could be implemented in some kernels quite easily using the tick counter. Ordinarily, this counter is programmed to interrupt at fixed intervals; however, in most designs using off-the-shelf counter/timer chips, the intervals can be programmed to produce a precisely timed interval equal to some number of clock cycles. It is a relatively simple matter to program each interval as determined in a queue of requests by the operating system. While the design is conceptually simple, modifications of typical Unix kernels and specification of a suitable application programming interface would probably not rank high on the typical producer's agenda.

In summary, we conclude that a precision-time event generation feature will probably not soon become a ubiquitous operating system feature. Where such features are required, they can be implemented as an external bus peripheral.

5.3. Reducing Serial Port Latency

Perhaps the most obvious way to capture the time of an external event is using a serial port, which has become a ubiquitous feature in modern workstations. At its simplest, the signal to be timed is connected to the serial port data leads, perhaps using a level converter/pulse shaper which produces a valid character facsimile for each signal occurrence. The signal causes a serial port interrupt, which unblocks the process or causes an asynchronous signal. The program then reads the system clock to establish the time of signal occurrence.

While this method is simple and straightforward, other than the design of the level converter/pulse shaper, it is accompanied by several sources of error which can add up to an unacceptable burden. The serial port hardware itself contributes a jitter equal to about half the character time, while additional jitter is caused by the serial port driver and the time to buffer the character and wake up the process. In the case of typical architectures, the various processing latencies that contribute to the jitter budget can easily exceed 20 ms. While some contributions to the jitter budget are hard to reduce, others can be fixed easily without significant impact on overall system performance are simplicity. Foremost among these are due to the hardware driver used with some serial port chips.

System programmers are well aware that the cycles to process a serial port interrupt can be a significant burden and limitation in line speeds supported by the system. In order to reduce the overhead of these interrupts, designers typically batch character arrivals when possible to reduce the number of interrupts. Thus, an interrupt is caused when either an internal buffer fills up or an interval equal to some number of timer ticks has expired after the first arrival of a character batch. However, the design of the typical Unix application interface provides a mechanism to read both formatted (canonical) and unformatted (raw) character streams. In the latter case, provisions are made to specify both the buffer size and timeout; but, the timeout can be specified only to a relatively coarse granularity. In the canonical case, there is no aggregation delay in the software interface; however, in order to reduce overhead, a delay is included in the hardware driver anyway. It is this delay that can ruin the ability of the kernel to accurately record the time of external events.

By sniffing though the source code of several Unix kernels, it has been possible to find and eliminate the serial port driver delays. Obviously, elimination of these delays is not justified in all cases; therefore, it may be necessary to provide a way of selectively enabling and disabling these delays on an optional basis, perhaps by a special system call known to the time synchronization daemon.

5.4.Reducing Interrupt Latencies

The design of the Unix kernel provides two levels of interrupt, one responsive to hardware interrupts and operating at an elevated hardware priority, and the other responsive to software-simulated interrupts and operated at a priority depending on the processes involved. A hardware signal causes an interrupt to a running process, regardless of the priority of that processs, an entry to be made on the software interrupt queue, and then a return to the interrupted process. If the priority of the software interrupt is greater than the running process, a context switch occurs to run the higher priority process immediately. This design minimizes the overhead for context switching, while preserving the multiple-priority model of process scheduling.

The problem with this approach for real-time systems is the overhead of the various prioritizing functions. Reference [MIL94b] discusses the various overheads incurred in the Sun kernel along with various techniques to avoid them, or at least reduce their impacts. One way to do this is through the use of Unix signals, with which a software interrupt can be simulated to a particular process without affecting the priority ranking with respect to other processes. In this way a time-critical event can be recorded in the state space of a process at the time of occurrence, while the processing which uses this information can be deferred to a more convenient time.

The use of Unix signals avoids system latencies due to process scheduling; however, it does not avoid latencies in the chain of programs between the hardware and the process-scheduling interface. In the case of System V Streams drivers, for example, the additional jitter due this cause can reach outrageous degrees - up to 1.5 ms for this alone. One way to minimize jitter contribution could be to "pop off" unnecessary streams modules, such as those which handle canonical formatting, for example. Unfortunately, this does not help much in typical (Sun) kernels, since the largest contribution to jitter is due to the character buffering and general data structure management operations, which are apparently unavoidable in this particular Streams implementation.

The most effective means to reduce jitter accumulation due to driver processing is to timestamp the external event at the hardware interrupt level, which ordinarily is only a few microseconds after occurrence in modern RISC machines. This of course requires intervention in the serial port device driver. One way to accomplish this is to add code to the driver which reads the system clock (a low overhead operation which reduces to a single instruction in 64-bit machines), then saves the timestamp in a data structure associated with the particular serial port. Upon the occurrence of a designated character, which can be designated by the programmer, the saved timestamp can be inserted in the data stream along with the serial data in the buffer which eventually is read by the application program. This scheme has been implemented for the Sun kernel in the form of the "ppsclock" module included in the NTP distribution.

The ppsclock module is of course specialized to the Sun kernel, indeed, to only the console port driver and a particular kernel version. Such would seem to be a worthwhile generic feature in other kernels as well. Supporting evidence for this agenda can be found in [MIL96b].

5.5. Precision Time Inputs

The precision obtainable using a serial port depends on the port data rate. An external signal that generates a designated character and timestamp as described above will result in a fixed delay of one character time or ten bit times, assuming an appropriate level converter/pulse generator as previously described. Most serial port chips sample the receive data signal eight or more times per bit, which introduces an additional peak-to-peak jitter equal to the sample interval. By operating

the serial port at the highest speeds, this jitter can be reduced to the order of a few tens of microseconds for most machines, but this may not always be possible, especially in common cases where the serial port is used to connect an external radio clock, for example. In these cases, an accurate timestamp is necessary regardless of port speed.

We first observe the common cases involving radio clock or similar device do not involve use of the modem control leads commonly supported by serial port chips. In such cases the control leads can be used to convey precision timing information. In most kernels available for inspection, the chips can cause an interrupt as the result of a control lead transition, if enabled. The interrupt can then be used to record a timestamp for later insertion in the buffer or for other purposes (see below). For this purpose, the best signal to use is probably the data carrier detector (DCD) signal, which ordinarily causes an interrupt at each state transition.

Since many signal sources which might generate a DCD signal use signal levels (TTL) which are incompatible with ordinary serial port signals, a level converter device may be required. In addition, if the signal has a very short duty cycle, it may be necessary to regenerate and stretch the pulse using a pulse generator, as described above. While devices to do these things are conceptually very simple, they are not commonly available and may have to be provided as an option or constructed locally. It may help to simplify the design to note that the power to operate the circuitry, typically one or two chips, can be provided by the other control leads supported by the serial port chip.

5.6. Latency Measurement and Calibration

In all the schemes presented so far, there has been an implicit assumption that it is possible to reduce incidental systematic delays and jitter to the point where they do not materially affect the calibration of the data eventfully delivered to the application buffer. Certainly, those contributions to the fixed delay budget can be predicted and correction factors included for them in the implementation. There are some contributions that may not be predictable, for instance, those due to the slewing-rate limiting characteristics of some level converter chips. In some cases, it may be necessary to calibrate out these sources of error. This can be done by generating an external signal as the result of a received DCD signal, for example. The external signal can be another modem control lead, for example, which can be detected by a conventional laboratory counter/timer. It would even be possible to use a third modem control lead along with a special calibration program which could automatically calibrate the system.

6. Summary and Conclusions

This memorandum has discussed issues in precision time measurement and synchronization. The target applications have been measurements likely to be involved in computer networks and transmission systems. A number of techniques have been described, some of which involve tinkering with existing workstation kernels. Some of the improvements suggested, such as a generic hardware and software interface for a 1-PPS signal, involve minor kernel changes that could be made without compromising other functions of hardware and operating system. Others, such as a programmed event function, require a shift in design philosophy unlikely to prevail widely in general purpose workstations.

7. References and Bibliography

Note: The following publications are available from the web page http://www.eecis.udel.edu/~mills.

[MIL89] Mills, D.L. Measured performance of the Network Time Protocol in the Internet system. Network Working Group Report RFC-1128. University of Delaware, October 1989.

[MIL91a] Mills, D.L. On the chronology and metrology of computer network timescales and their application to the Network Time Protocol. *ACM Computer Communications Review 21, 5* (October 1991), 8-17.

[MIL91b] Mills, D.L. Internet time synchronization: the Network Time Protocol. *IEEE Trans. Communications COM-39, 10* (October 1991), 1482-1493.

[MIL92a] Mills, D.L. Modelling and analysis of computer network clocks. Electrical Engineering Department Report 92-5-2, University of Delaware, May 1992, 29 pp.

[MIL92b] Mills, D.L. Network Time Protocol (Version 3) specification, implementation and analysis. Network Working Group Report RFC-1305, University of Delaware, March 1992, 113 pp.

[MIL94a] Mills, D.L. Unix kernel modifications for precision time synchronization. Electrical Engineering Department Report 94-10-1, University of Delaware, October 1994, 24 pp.

[MIL94b] Mills, D.L. Precision synchronization of computer network clocks. *ACM Computer Communication Review 24, 2* (April 1994). 28-43.

[MIL95a] Mills, D.L. Improved algorithms for synchronizing computer network clocks. *IEEE/ACM Trans. Networking* (June 1995), 245-254.

[MIL95b] Mills, D.L. Simple Network Time Protocol (SNTP). Network Working Group Report RFC-1769, University of Delaware, March 1995, 14 pp.

[MIL96a] Mills, D.L. A Kernel Model for Precision Timekeeping. Electrical Engineering Technical Memorandum, January 1996, 28 pp.

[MIL96b] Mills, D.L. A Kernel Programming Interface for Precision Time Signals. Electrical Engineering Technical Memorandum, January 1996, 3 pp.