
ENCYCLOPEDIA OF COMPUTER SCIENCE AND TECHNOLOGY

EXECUTIVE EDITORS

Allen Kent James G. Williams

UNIVERSITY OF PITTSBURGH
PITTSBURGH, PENNSYLVANIA

ADMINISTRATIVE EDITOR

Carolyn M. Hall

ARLINGTON, TEXAS

VOLUME 37
SUPPLEMENT 22



MARCEL DEKKER, INC.

NEW YORK • BASEL • HONG KONG

COPYRIGHT © 1997 BY MARCEL DEKKER, INC.
ALL RIGHTS RESERVED

Neither this book nor any part may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, microfilming, and recording, or by any information storage and retrieval system, without permission in writing from the publisher

MARCEL DEKKER, INC.
270 Madison Avenue, New York, New York 10016

LIBRARY OF CONGRESS CATALOG CARD NUMBER: 74-29436
ISBN: 0-8247-2290-6

Current Printing (last digit)
10 9 8 7 6 5 4 3 2 1

PRINTED IN UNITED STATES OF AMERICA

PRECISION SYNCHRONIZATION OF COMPUTER NETWORK CLOCKS

INTRODUCTION

In this article, we consider methods for controlling the clocks in computers so as to keep them synchronized to some external source of time information. We will discuss the statistical principles that govern the design of the control algorithms, and we will illustrate these principles using several examples.

There are two types of system for which time synchronization is important:

1. Processes which handle time-sensitive data where the time stamps must be traceable to a national timing center for legal or commercial reasons. Although it is possible for such a process to query an external time server whenever a time stamp is required, this is usually not an optimum strategy. More sophisticated algorithms can usually deliver higher-accuracy time tags at lower cost.
2. Processes which handle time-sensitive data and which are implemented in a networked environment using a number of independent machines. Keeping the times of all of the machines synchronized to each other (without having them necessarily synchronized to a national timing center) may be sufficient in this case in principle, but this simplification may be difficult to exploit if the machines are widely separated. Synchronizing the machines to a common external time reference may be easier to implement and may result in a synchronization algorithm that is more accurate, in general, and more robust in the event of a partial failure of the network.

It is difficult to specify a single accuracy requirement that will satisfy the wide range of client processes that are encountered in practice. As we will discuss, a number of problems make it difficult to synchronize the time of the clock in a general-purpose computer to much better than 1 millisecond (0.001 s) or to deliver time information to a client process while maintaining this accuracy (even if the client process is running on the same machine). Higher-accuracy time information can be obtained in a number of ways, but using the full accuracy of these time tags generally requires special-purpose equipment, and we will not discuss these special-purpose configurations in this article. We will confine our discussion to synchronization of general-purpose computers running essentially standard software.

STANDARDS OF TIME

The international standard of time is called Coordinated Universal Time (UTC); its definition is traceable to the Convention du Mètre, which was signed in Paris in 1875 (1). It is maintained by the Bureau International des Poids et Mesures (BIPM) in Sèvres, France using data from a worldwide ensemble of clocks whose frequencies are locked to a specified transition in cesium (2). (Data from hydrogen masers have recently been included in the computation as well.)

The rate of the UTC time scale is defined in terms of the quantum-mechanical properties of cesium and hydrogen, and its time was initially set to correspond as closely as possible to the older astronomical definition (3). The UTC time scale is adjusted (“coordinated”) by adding “leap” seconds as needed so as to keep it within ± 0.9 s of UT1, a time scale based on the rotation rate of the earth. At the present time, leap seconds are added about every 18 months—usually on the last day of June or December. Although it is possible, in principle, to drop a second instead of adding one (a “negative” leap second), this has never been done and is unlikely to be required for the foreseeable future because the rate of UTC is fast with respect to UT1 by about 0.03 ppm. This difference is larger than the typical variations in the length of the day as determined from the rotation of the earth.

The computation of UTC by the BIPM is done almost 2 months after the fact, so that real-time users must get time information from a national timing center. In the United States, time information is provided by the National Institute of Standards and Technology [UTC(NIST)] and by the U.S. Naval Observatory [UTC(USNO)]. The differences between UTC and the real-time scales UTC(NIST) and UTC(USNO) are published by the BIPM in its Circular T. These differences are typically on the order of tens of nanoseconds (i.e., 10^{-8} s), and we can take them to be zero for the purposes of the current discussion. Time information is also provided by timing centers in other countries; the offsets between these time signals and UTC are generally less than a few microseconds (i.e., 10^{-6} s) in almost all cases.

SYNCHRONIZATION SOURCES

In addition to the problems of time-scale definition (which are largely theoretical in the context of this discussion), most users must correct the time information that they receive for the delay due to the finite transit time of the information along the transmission channel between the timing center and their location. This delay is on the order of at least 3×10^{-6} s/km and can, therefore, be tens of milliseconds for a typical user who may be 1000 km or more from a timing center.

There are three general methods for dealing with the propagation delay:

1. One-way transmission methods with or without an ancillary model of the delay
2. Two-way methods which measure the round-trip delay and estimate the one-way delay as one-half of this measurement
3. Common-view methods which use simultaneous one-way transmissions to two (or more) sites that are approximately equally distant from the transmitter

Each of these methods has advantages and limitations, and all of them are widely used for distributing time and frequency information at the present time.

The one-way method is the easiest one to use. A timing source broadcasts messages periodically, and any number of users can receive the information. Each user may correct the received messages for the propagation delay using parameters derived from the distance back to the transmitter and possibly other parameters such as ambient temperature. The transmitter can supply information to any number of users, so this type of server does not saturate as the load increases. (As the operating costs are essentially independent of the load, the cost per user may actually decrease as the load increases.) The primary limitation of the method is that the delay is estimated from a model rather than actually measured—how well these estimates will work in any particular situation is difficult to judge a priori. In general, the method will be least useful if the delay varies appreciably with time, because such variation may be difficult to incorporate into a static model unless a significant effort to measure the properties of the path in real time is included in the measurement protocol.

There are two methods that are often used to estimate the properties of the transmission path. The simpler method is to combine end-point measurements of the relevant parameters with a model of the delay. The propagation delay through the atmosphere, for example, is primarily a function of the ambient pressure, temperature, and water-vapor density (4,5). The end-point measurements of these parameters can be combined with the physical distance and a theoretical expression for the refractivity to yield a real-time estimate of the propagation delay. The accuracy of this procedure is limited by atmospheric homogeneity: by how well the end-point measurements represent the average conditions along the path (6).

The second method uses simultaneous transmissions of the timing information using two different carrier frequencies. This is a useful technique if the path is dispersive: if the propagation delay depends on the carrier frequency and if the dispersion can be separated into the product of a function that depends only on the carrier frequency and a second function that depends only on atmospheric properties. The refractivity (the excess path length due to the presence of the atmosphere) can be estimated from the dispersion in this case, and this estimate can be combined with the physical length to compute the path delay (7). In addition to optical transmissions through the atmosphere, this method is also used by the Global Positioning Satellites (see below) to measure the propagation delay of the signals through the ionosphere (8).

Two-way methods directly measure the round-trip delay and estimate the one-way delay as one-half of this value (9). The method is more complicated than a one-way technique because a second, reverse channel is required and because the receiver must actively collaborate with the transmitter in the two-way data exchange. Instead of a client-server model, the relationship between the two stations is more like two peers.

The accuracy of the delay estimate is limited by the symmetry in the path delay (10). Any static inequality between the inbound and outbound delays can be measured and administratively removed in principle, so the real problem is with uncorrelated fluctuations in the delays in the two directions. Even when the path itself has a symmetric delay (as is generally true in the atmosphere), there may be a significant asymmetry in the inbound and outbound channels of the station hard-

ware. Many standard telephone modems exhibit this unfortunate characteristic, for example.

In addition to the complexity of the method, two-way systems have a finite capacity, so that the operating costs increase with increasing load. The load on the server increases linearly with the number of clients requesting service, and both ends must “know” about each other. The casual, one-sided association that is possible with broadcast services is not adequate.

Two-way methods may have an additional problem depending on how the reverse channel is implemented. There are three commonly used methods:

1. *Time-Division Multiplexing.* A single “half-duplex” channel is alternately used in opposite directions. The measurements in the two directions use mostly the same hardware, but they are not simultaneous and they may, therefore, be corrupted by temporal fluctuations in the channel delay. Any such fluctuations are potentially troublesome, but those whose periods are comparable to the “turnaround” time of the channel are particularly serious because they produce offsets that appear to be static; such asymmetries are not removed by averaging. The transmission of time over the Internet is a common example of this signaling protocol.
2. *Frequency-Division Multiplexing.* A single “full-duplex” channel is used in both directions simultaneously using signals at two different frequencies. The measurements in the two directions are now simultaneous, but any dispersion in the channel or any imbalance in the inbound and outbound hardware (which process different frequencies and are therefore likely to be different) produces a first-order error. Telephone modems and many satellite systems use this type of signaling protocol.
3. *Two One-Way Circuits.* The data in the two directions are carried on two independent, nominally identical one-way circuits. The limitation here is balance—how well the properties of the two one-way channels can be balanced and how well their fluctuations will track each other. Many long-distance telephone circuits are implemented in this way.

In summary, the conceptual simplicity of the two-way method often causes designers to underestimate the difficulty of realizing the potential power of the technique in a real-world environment.

In common-view methods, any number of receivers simultaneously listen to a single transmitter that is approximately equidistant from all of them (11). Unlike two-way methods which depend on the equality of the propagation delays in opposite directions along the same path, the common-view method depends on the equality of the propagation delays along different outbound paths. Two clients can synchronize to each other using the common-view method with an accuracy that is limited by the imbalance in the different paths back to the server and independent (to a good approximation) of the accuracy of the time messages themselves. In fact, the server need not even “know” that it is being used for synchronization—common-view systems have been built using the synchronization pulses of ordinary television signals or the zero-crossings of the AC power line. Signals from LORAN stations, which are widely used for navigation, have also been used for time synchronization in the common-view mode.

Although a common-view time server can operate in broadcast mode and need not know who is listening, the clients must know about each other—they must agree on a common observation schedule so that they all observe the signal simultaneously, and they may have to apply corrections to the data if the paths are not exactly equal. As with two-way methods, the station hardware may make an appreciable contribution to the overall delay. A constant difference in the delays through two receivers can be measured and removed, so that the limitation is ultimately from uncorrelated fluctuations in the properties of the two different paths or in the station hardware. Because the paths are *different*, and may be widely separated, uncorrelated fluctuations in the delays are often more of a problem here than in two-way methods, where the delays in the two directions along the same path (or two adjacent one-way paths) are more likely to be correlated.

None of the methods offers any independent check on the adequacy of the delay correction. Residual fluctuations in the corrected time signals provide some indication of the statistics of the estimate of the propagation delay (see below), but there is no good way of detecting a static offset in the delay estimate without using some independent check on the synchronization.

We now discuss several widely used distribution systems that are based on the general principles we have described.

Radio Clocks

Although many other more sophisticated distribution methods have been developed, radio-based transmissions are still widely used for distributing time to many clients who need only moderate accuracy. Most computer clocks fall into this category, so that radio transmissions play a central role in computer network synchronization.

The first transmissions used standard short-wave equipment. In the United States, the National Bureau of Standards began transmitting time and frequency information in 1923 from radio station WWV. This station currently transmits on standard frequencies of 2.5, 5, 10, 15, and 20 MHz. The transmitter is located in Fort Collins, Colorado. A second transmitter, named WWVH and located in Hawaii, was added in 1948 (12). Similar equipment is operated by other countries (13).

The accuracy of short-wave time-signal broadcasts is limited primarily by uncertainties in the propagation delay between the transmitter and the user. (Delays in the receiving equipment are also important, but these can be minimized by careful design and tend to be stable with time so that they can be removed by calibration.) The propagation delay depends on the straight-line distance between the transmitter and receiver but is also a function of the height of the ionosphere, the time of the day, the season, the sunspot cycle, and other factors (14). Propagation delays on the order of milliseconds are quite commonly found; as we will show below, it is the variability of this delay (and especially the variability at periods of a few hours and longer) that causes the most problems in computer synchronization. The variability in the delay at these periods can be significant, and so most computer networks no longer use short-wave broadcasts for synchronization.

The National Bureau of Standards added a third, low-frequency transmitter named WWVB in 1956 (12). Similar low-frequency services are also operated by

other countries (15). The purpose of this station is to take advantage of the more stable propagation delay that is usual at this lower frequency. The reason for this stability in the propagation delay is that waves at these low frequencies tend to propagate in a “waveguide” mode between the earth and the ionosphere. The main problem with this signal is that both the transmitting and receiving antennas are relatively inefficient because they are small compared to the very long wavelength (about 5 km) of the carrier that is used. Station WWVB transmits using a frequency of 60 kHz; similar frequencies are used by the low-frequency stations operated by other timing centers (15).

Although low-frequency transmissions are less affected by atmospheric effects than short-wave signals, there are residual effects that may be important. The most important fluctuation often has a period of 24 h; the path delay changes in one direction during sunset and the other way during sunrise. The average fluctuation over the full 24-h period is a small fraction of the peak-to-peak fluctuation, but, as we will discuss below, this is not of much help for time synchronization because the clocks in most computer are not stable enough to support this long an averaging time.

Signals from radio station WWVB are widely used to synchronize computer clocks, and machines synchronized in this way are often used as time servers to redistribute the time to a larger number of “stratum-2” clients.

The diurnal fluctuations in the propagation delay of the signal from WWVB are usually significantly less than 1 ms, but the interface between the receiver and the computer must be carefully designed so that fluctuations in the I/O path delay do not degrade the accuracy of the data. This is often not a trivial problem. Finally, the signal is often weak and hard to receive at some locations; an outside antenna is often required.

In summary, signals from WWVB are sufficiently accurate for most computer synchronization applications, provided that the signal is strong enough at the receiver location and the interface to the computer is designed with care.

Another approach to making the propagation delay more stable is to use much higher transmission frequencies. These signals tend to travel along “line-of-sight” paths so that the variations in the delay (due to ionospheric or tropospheric effects) tend to be smaller. These signals are usually transmitted from satellites because ground-base transmitters would have too limited a range to be useful. (This may change in the future as cellular telephone service becomes more widely available.)

There are two satellite systems that can be used to obtain time and frequency information: the GOES (Geostationary Operational Environmental Satellites) system, which is operated by the U.S. National Oceanic and Atmospheric Administration (NOAA) (16) and the Global Positioning Satellites (GPS) operated by the U.S. Department of Defense (17). The two systems differ in many technical details: The GOES satellites relay a time code that is generated in the control station at Wallops Island, Virginia, whereas the GPS satellites generate the time-code message onboard using information that is uploaded to the satellite periodically. There are normally two operational GOES vehicles in geostationary orbits located roughly above the Western Hemisphere. These satellites are called “GOES East” and “GOES West” and are located at 75° and 135° west longitude, respectively. A third satellite is located at 105° west longitude and is an orbiting spare. The GPS constellation provides worldwide coverage using 24 satellites in 3 orbital planes. The GPS satel-

lites are not in geostationary orbits, so the suite of satellites visible from any location varies with time.

In spite of these significant technical differences, they are conceptually similar systems from the point of view of the current discussion. Both transmit a complex message containing both time information and orbital parameters, and both require rather sophisticated receivers to parse the message and to correct the received time signal for the propagation delay through the atmosphere. The GPS system is more sophisticated than GOES, and the accuracy that can be achieved using it is considerably higher as a result. The corrected GPS time code is specified to be within 100 ns of UTC(USNO) at all times and is generally much closer to UTC(USNO) than this. The uncertainty in the fully corrected received signal may therefore be taken to be negligible in the context of the current discussion. This may not be true for the GOES system—station-keeping may occasionally cause temporary time offsets approaching 1 ms.

All of the broadcast services can be used in one-way mode with or without a model to correct for the propagation delay from the transmitter to the receiver. The delays from the satellite to the ground are significantly larger than 1 ms, so that correcting for the transit time is essential in these cases; satellite-based services transmit an extensive set of parameters to facilitate the computation of this correction for this reason.

Any broadcast service can also be used in the common-view mode to reduce the effect of the fluctuations in the satellite clock and in the common components of the path delay. The GPS signals are often used in this way, but this additional complication is usually not needed when synchronizing computer clocks because adequate accuracy can be achieved without it. The fluctuations in the unmodeled portion of the channel delay are negligible in our context under normal operating conditions.

Cable Systems

There are a number of time-distribution systems that are based on sending time information over communication networks. An example is the Automated Computer Time Service (ACTS) that is operated by the National Institute of Standards and Technology (NIST) in Boulder, Colorado (18). This service transmits time using standard modems and dial-up telephone circuits. The server transmits a time code every second using standard ASCII characters. The last character of each message is an on-time marker character (OTM); it is initially sent 45 ms early to compensate approximately for the typical transit time through modems and telephone circuits. If the user echoes the OTM back to NIST, the server measures the round-trip delay, estimates the one-way delay as one-half of this value, and adjusts the advance of subsequent OTM characters accordingly. This system is a standard two-way system; as we discussed earlier, its accuracy will be limited by any inequality in the inbound and outbound delays. Some modems have substantial asymmetries of this kind, and the accuracy of the ACTS system is limited to several milliseconds for this reason. The asymmetry depends on the details of the modem circuitry, and nominally identical and otherwise fully compatible modems from different vendors may have significantly different delays. Because the delay asymmetry tends to be dominated by the modem design, using the same hardware tends to result in a time offset that is

constant from one connection to the next. The asymmetries are often worse at speeds above 1200 bits/s, as these higher-speed connections often require dynamic equalizers to compensate for the characteristics of an individual telephone circuit. We have observed asymmetries of up to 20 ms using different 9600-bit/s modems and standard telephone circuits. The equalizers may change the group delay from second to second and degrade the stability of the OTM advance as well. This effect is observed as a large variation in the measured delay value from one message to the next. If this effect is large enough, the server hardware may conclude that the client is not working properly and may revert to the default advance of 45 ms. Other agencies, both in the United States and elsewhere, operate services that are quite similar to ACTS, and the accuracies are typically of the same order as we have discussed (19,20).

There are also a number of protocols that transmit time information over local and wide-area packet networks. The Internet versions are usually two-way methods and use message protocols based on udp/ip or tcp/ip. The most widely used is the Network Time Protocol (NTP), which is based on udp/ip (21,22). The accuracy of network-based time distribution is usually limited by the fact that both the total delay and its asymmetry are likely to vary much more rapidly than for any of the other methods we have discussed so far, and the variations are often difficult to characterize statistically. As we might expect, these problems tend to become more serious as the network load or the distance between the client and the server increase.

COMPUTER CLOCKS

Most computers keep time using a combination of hardware and software. The hardware generates periodic interrupts, and the system responds to these interrupts by incrementing a register or memory location. Some systems simply add one to the register at every interrupt so that the time is maintained in units of "ticks"; others increment the register by the nominal time interval between ticks. In either case, the time is kept with respect to some arbitrary origin, and the conversion to the more conventional civil date and time is performed by the software whenever a time tag is requested.

Some older machines used the zero-crossings of the input power line to generate the time interrupts, so that the interval between ticks was 1/60 s (or 1/50 s in many places outside of the United States) or some multiple of this interval. Most newer machines use an internal quartz-crystal oscillator to generate the interrupts. The size of each "tick" is on the order of milliseconds in most systems. This value is chosen as a compromise between time resolution, which would favor shorter periods, and minimizing the time spent on interrupt service, which would favor fewer, less frequent interrupts. Some hardware environments provide much finer time resolution using ancillary hardware to interpolate between ticks and some software environments make this enhanced resolution available to user processes. It is also possible for a client process to interpolate between ticks using calibrated software timing loops. The accuracy of this strategy depends on the system load, but it is generally quite difficult to reliably interpolate to better than about 5% of the tick period.

The raw accuracy and resolution of a computer clock set the lower bound on the uncertainty of the time tags that can be provided to a client process, but they are not the whole story. A client process normally requests a time tag in response to some event—the completion of an input/output request, for example. The delay between when the request for time is posted and when it is satisfied is a function of system load and operating system design, but it can be an appreciable fraction of a millisecond, even in lightly loaded systems. In some systems, requests of this type may be processed by a daemon process that is not started until the next clock tick, so that the average system latency may be 0.5 ticks, which is usually several milliseconds.

It is possible to minimize these latency issues by using a dedicated processor with a tightly controlled interrupt structure, but such configurations generally require special-purpose hardware and software. Thus, although it may be technically feasible to synchronize the time of a general-purpose computer to substantially better than 1×10^{-3} s, this improved synchronization may not be available to the average client process without special, possibly heroic, measures.

CHARACTERIZING CLOCKS

The accuracy of the time of a clock is usually limited by two classes of problems:

1. *Phase noise*, which is due to the fact that it takes a finite time to transmit, read, or process a time-tag. The servicing of a timer interrupt can be delayed because the processor is busy doing something else, for example, so that the actual advance of the internal clock is late with respect to the hardware tick that triggered it. In an extreme case, the interrupt might be lost altogether. The clock would appear slow to a client in this situation. Alternatively, the request for a time tag may spend some time on a queue before it can be satisfied; in an extreme situation, an additional clock interrupt may be processed while the request is pending. The time tag that is returned in this case specifies a time after the event which triggered the request. The oscillator hardware can also contribute to the phase noise budget, due to jitter in the zero-crossing detector, for example. This contribution is usually small compared to the software issues we have just described.
2. *Frequency noise*, which is caused by fluctuations in the frequency of the oscillator which triggers the time interrupts. The frequency fluctuations can be further separated into deterministic effects and stochastic effects. Examples of deterministic effects are frequency drift due to aging of the oscillator components or fluctuations driven by changes in the ambient temperature. Stochastic effects are of unknown origin and can only be characterized statistically.

Although we can identify the different underlying causes of phase noise and frequency noise, it is far more difficult to separate the two in a real device. This difficulty arises from the fact that the frequency of the clock oscillator is not directly observable—we see it only through the time data, which average the frequency fluctuations over the interval between readings and which are, of course, also con-

taminated with phase noise at each reading. It is important to try to separate these effects for a number of reasons—identifying the noise type might suggest how the system might be improved, for example, but the most important reason is that there are optimum statistical procedures for dealing with each type of noise. Therefore, knowing the cause helps define the solution (23,24).

Deterministic time or frequency offsets are relatively easy to detect and remove. The limitation in how well this can be done is usually set by the magnitude of the corresponding stochastic components, so that understanding them is really the central problem of time synchronization.

The underlying noise type can often be identified by examining how the power spectral density of the fluctuations varies with Fourier frequency. This description is particularly simple for most oscillators because the dependence on Fourier frequency can be approximated by a sum of five power-law processes (25):

$$S_y(f) = \sum_{\alpha=-2}^{\alpha=2} h_{\alpha} f^{\alpha}. \quad [1]$$

In this equation, $S_y(f)$ is the power spectral density at Fourier frequency f and the parameters h_{α} are the constant coefficients of its polynomial expansion. Each term in the expansion corresponds to a physical process:

- $\alpha = -2$ Random-walk frequency modulation
- $\alpha = -1$ Flicker frequency modulation
- $\alpha = 0$ White frequency modulation
- $\alpha = +1$ Flicker phase modulation
- $\alpha = +2$ White phase modulation

The adjectives, such as “flicker,” associated with each of the noise processes are formally defined by their association with a particular value of α , but they are also descriptive of the underlying physical mechanisms that dominate each type of noise process.

A “white” process implies that the corresponding parameter is a Gaussian random variable that can be completely characterized using a mean and a standard deviation. Consecutive observations are randomly distributed about the mean and are not correlated—knowing any number of previous values does not improve our ability to predict the next observation. Both flicker and random-walk processes have some correlation between consecutive observations. They differ in degree. As we will discuss below, the next observation in a random-walk process is randomly distributed about the *current value* (rather than about the mean as in a Gaussian process). A plot of consecutive values therefore looks more like noise superimposed on a slowly varying function rather than like plain white noise. From the mathematical point of view, this effect is due to the fact that the very lowest Fourier frequencies dominate the power spectrum. Although a number of models for flicker noise have been proposed, none of them provides a simple physical picture of the process—in some sense, it is simply intermediate between the random-walk and Gaussian processes and could be thought of as a mixture of the two.

The first step in computing the power spectral density is to measure the time difference x between the clock being tested and some calibration signal at equally spaced times t_0, t_1, \dots, t_N , where $t_j = t_0 + j\Delta t$. The difference at time t_j is $x(t_j)$ or just x_j . For the moment, we will assume that the calibration information is received

from a trusted source without degradation, so that our observations characterize the oscillator itself without the complicating effects of any noise in the transmission path.

Our goal is to parametrize the spectrum as a polynomial in Fourier frequency and, in particular, to see if it is possible to divide the Fourier frequency domain into regions where the power spectral density in each region can be approximated by an equation of the form of Eq. (1) with only a single polynomial term. It turns out that this can be done for most oscillators so that, to a good approximation, we can speak of the Fourier regime in which the system is dominated by white frequency modulation, and so forth. This characterization is particularly useful because there is a statistically optimum strategy for dealing with each type of noise process (26).

In order to decide which noise process is dominant in any Fourier frequency domain, we estimate the variation of the power spectral density there. More specifically, we estimate the slope of a log-log plot of the power spectral density as a function of Fourier frequency. Conventional fast-Fourier transform techniques can be used to compute these estimates, but they are not optimum for this purpose because they provide estimates that are linear in Fourier frequency, which means that they provide much more information than is really needed to estimate which value of α dominates the spectrum, especially at higher Fourier frequencies.

The Allan variance, usually denoted by $\sigma_y^2(\tau)$, is an alternative, time-domain measure of the noise of an oscillator. It is proportional to the time-averaged value of the square of the second differences of x_j as a function of averaging time τ , where τ is some multiple of the sample interval Δt . Specifically, if there are N time differences, the Allan variance for an averaging time of $\tau = k\Delta t$ is (25)

$$\sigma_y^2(k\Delta t) = \frac{1}{2(N-2k)(k\Delta t)^2} \sum_{i=1}^{i=N-2k} (x_{i+2k} - 2x_{i+k} + x_i)^2. \quad [2]$$

It is most useful for this discussion because its variation with sampling time τ can also be approximated by a polynomial expansion in τ :

$$\sigma_y^2(\tau) = \sum_{\mu=-1}^{\mu=2} a_\mu \tau^\mu. \quad [3]$$

If the expansion of the spectral density in Eq. (1) can be approximated by a single term, then the expansion of Eq. (3) becomes a single term as well. If $\alpha \leq 1$, the exponents of the single surviving terms in each expansion are related by $\alpha = -\mu - 1$ so that determining μ determines α and allows us to identify the underlying physical process that dominates the stochastic variation. [If $\alpha > 1$, the one-to-one relationship between α and μ can be preserved by computing the "modified" Allan variance as described in the literature (25,27).]

The relationship between α and μ assumes that the x_j are measurements of a clock whose time differences with respect to the calibration source are dominated by stochastic effects. A statistically significant peak in the Fourier spectrum will add complexity to the Allan variance for averaging times comparable to the inverse of its Fourier frequency. Neither the spectral density nor the Allan variance will obey the simple polynomial expansion we have assumed for it in the vicinity of the spectral peak, and the methods we will discuss are not really appropriate to deal with it. Statistically significant peaks are generally caused by some well-defined source (diurnal variations in ambient temperature, for example), so that dealing

with these effects means either attenuating the cause of the fluctuations or measuring them and removing their effects from the data.

The general principle, which underlies most of the following discussion, is that averaging a quantity will only be an optimum strategy when its underlying noise spectrum is white, and these sorts of analyses delineate the time-domain regime in which such averaging converges to a statistically robust estimate of the underlying physical parameter and is therefore appropriate. (Averaging will also attenuate bright lines in the noise spectrum and may remove them completely if the averaging period exactly matches the period of the perturbation. This is not the optimum way of doing this in general because it depends on an exact match between the averaging and noise periods and will fail if the phase of the noise is not stable. Standard regression and correlation techniques will usually do a better job.)

The coefficients in the polynomial expansion for the spectral density tend to vary very slowly in time, so that the evaluation procedure need not be repeated very often. The parameters vary somewhat among nominally identical machines, and we have found a much larger variation between machines made by different companies. Although all of these companies use clocks that are driven by quartz-crystal oscillators, the quality of the clocks varies widely. As we will show, an optimum statistical procedure can get the best the hardware can deliver, but there is really no way to make a good clock from low-quality hardware.

SYNCHRONIZATION STRATEGIES

We now turn to the principal topic of this article—how to combine calibration data from a time server with the current reading of the local clock so as to optimize the accuracy of the time on the local machine. The basis of our strategy is “separation of variance,” by which we mean extracting from the calibration signal information only in the Fourier frequency domains where its performance is better than that of the local clock. Our primary tool is the power spectral density as a function of Fourier frequency or, equivalently, the Allan variance as a function of averaging time.

The Statistical Characteristics of Clocks

Figure 1 shows the square root of the Allan variance (often called the Allan deviation) as a function of averaging time for two typical computer clocks. These plots are computed as discussed earlier: The time difference between the local clock and some external calibration signal is periodically measured, and the Allan deviation is the square root of the variance computed using Eq. (2). The points plotted using “+” are for a relatively good clock, whereas those plotted using “*” were obtained from a computer with a much poorer quality oscillator. In both cases, the data can be divided into three regions, each of which is approximately characterized by a single slope when the data are plotted on a log–log plot as a function of averaging time (as is done in Fig. 1). These slopes are indicated in Fig. 1 by straight lines drawn through the observations obtained from the better device, whose data follow the model quite closely.

The slope of the deviation at short times is -1 , so that the variance would

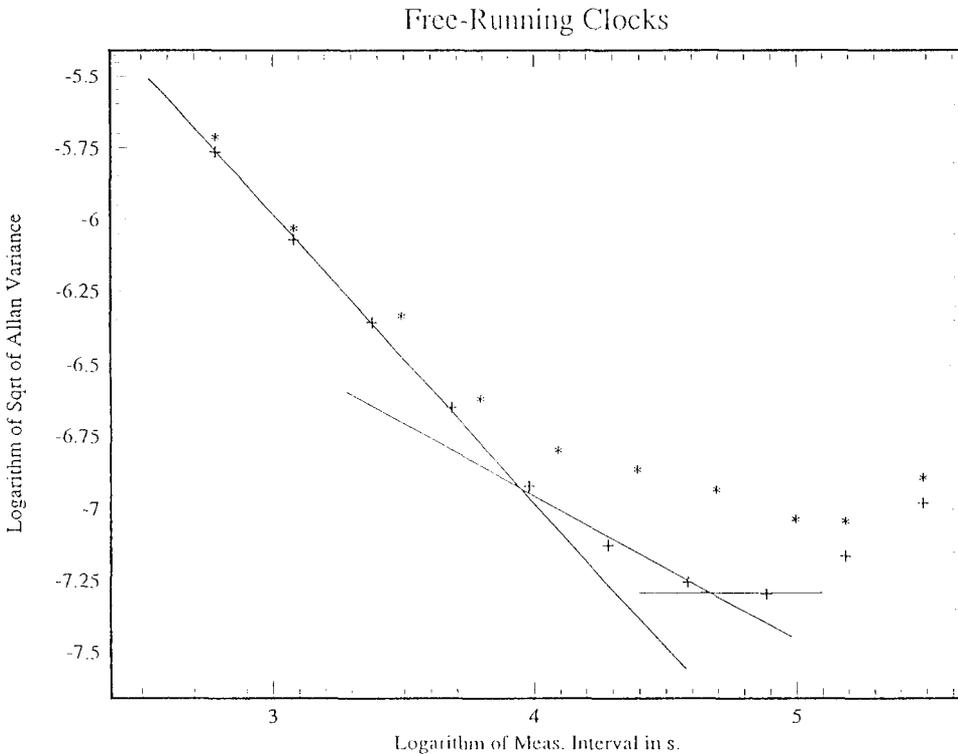


FIGURE 1 Allan deviations of free-running computer clocks. The points plotted using “+” are typical of better clocks, whereas those plotted using “*” are typical of machines with lower-quality oscillators. The reference lines drawn through the points have slopes of -1 , -0.5 , and 0 as discussed in the text. (Based on data from Ref. 28.)

have a slope of -2 . Using the results of the previous section, we see that the dominant noise process in this domain is white or flicker phase modulation. A computation of the modified Allan deviation shows that the process is, in fact, white phase modulation. Physically speaking, the noise in the data is due to the noise introduced by the measurement process itself rather than by fluctuations in the underlying frequency of the device. It would be a mistake to use these time difference data “as is” to adjust the clock because the underlying frequency of the clock is *more stable* than the observed fluctuations in the time data would indicate. Because the observed time differences show a nearly white spectrum, an underlying mean time difference exists and is well defined statistically. The average of consecutive data points converges to and provides an unbiased estimate of this mean, and the standard deviation of the estimate of the mean will improve as more data are included in the average. The most accurate way to read the time of this clock is to compute the average of many closely spaced observations.

It is important to remember that what we are discussing here is dealing with *stochastic* effects—by definition, the Allan variance is insensitive to a static frequency offset even though such an offset produces monotonically increasing time

differences and must be removed. The averaging process that we have described in the last paragraph must therefore be performed on time-difference data from which a mean slope has been subtracted. The physical significance of this mean slope is the static frequency offset of the clock; we determine this parameter as discussed below.

As the averaging time increase, we soon find that this strategy no longer works well. The slope of the Allan variance is now more like -1 , which means that it is white *frequency* noise that is now the dominant problem. Physically speaking, averaging the time-difference data has reduced the measurement (phase) noise to the point that the time-difference data are now dominated by the random fluctuations in the frequency of the oscillator itself. The time differences, which are the *integral* of the frequency over the averaging time, are no longer white, and a simple average of them over this longer averaging time is now the wrong thing to do—in other words, a mean time difference and a constant rate offset no longer suffice to characterize the observations.

The frequency, on the other hand, is now a white random variable, so that *its* mean is now well defined statistically. The average of consecutive frequency estimates would now converge to the underlying mean frequency without bias. The frequency estimate at time t_j (where $j > 0$) is f_j and is computed using

$$f_j = \frac{x_j - x_{j-1}}{\Delta t}. \quad [4]$$

The average after N estimates, denoted by F_N , is computed using the usual method:

$$F_N = \frac{1}{N} \sum_{j=1}^{j=N} f_j. \quad [5]$$

It is often useful to compute the estimates of the average frequency as the time differences are obtained, in which case the average can be rewritten in the recursive form

$$F_N = \frac{N-1}{N} F_{N-1} + \frac{1}{N} f_N \quad [6]$$

or

$$F_N = \frac{F_{N-1} + G f_N}{1 + G}, \quad [7]$$

where

$$G = \frac{1}{N-1}. \quad [8]$$

Note that the parameter G , the weight of the N th estimate in the average, tends to 0 monotonically as the number of observations increases. This happens because the N th estimate represents a smaller and smaller fraction of the total number of observations, all of which have equal weight in the white-noise regime.

As in the previous situation, the standard deviation of the mean of these estimates improves as more estimates are added. It is important to remember that

what we are estimating is the underlying mean frequency of the device so that these data must be used in a frequency-lock loop. The phase noise is no longer a white random variable at these averaging times, so that a phase-lock loop can no longer be used in this time domain—the averaging implied by the integrator in such a loop no longer converges to an underlying, well-defined mean value.

As we discussed earlier, the average offset frequency of the clock must be removed from observations in the white phase noise domain if the average is to converge to a statistically robust mean value. The average frequency, however, can only be determined using observations in the white frequency noise domain, so that the frequency of the device must usually be determined *first* before the phase data can be properly averaged. This results in a startup procedure that is perhaps counterintuitive—an uncharacterized clock must be allowed to free-run for several thousand seconds so that its frequency can be determined in a robust manner (28). During this evaluation process, the clock will drift off-time, of course, so that the algorithm appears to be doing nothing. This free-running evaluation is usually not needed on subsequent cold starts, as the previously estimated frequency is usually adequately accurate for the averaging procedure to converge.

As before, the strategy of averaging the frequency estimates begins to fail at longer times—the frequency average no longer predicts the time performance of the clock. Physically speaking, consecutive frequency estimates of the oscillator can no longer be characterized as having an underlying mean value. We could think of this as frequency “drift” or “aging,” but these terms imply a deterministic, predictable effect, whereas what we observe might be better characterized as a slow “wander” of the frequency with no discernible mean value.

The wander is often approximated as a “random walk.” At each instant, the frequency evolves by adding a small increment to its current value, where the *sign* of the increment is *random*. The best guess for where the frequency will be in the next instant is, therefore, the *current* value with *no* averaging of older data because, by assumption, the possible suite of next values is symmetrically distributed around the current value. The mean value of such a process is zero because negative and positive steps have equal probability, but consecutive values are clearly correlated in a way that is not present for a normal Gaussian process.

As in the previous case, the transition between the different domains of averaging time is gradual rather than sharply defined, and the previously optimum strategy becomes increasingly less optimum as the averaging time increases. As the averaging time increases, the frequency estimator must cope with this transition by starting to “forget” older estimates, until, in the limit, all older estimates are forgotten and only the most recent measurement is used.

One way of achieving this transition is to change the definition of G in Eq. (8) so that it remains finite as N increases. It is relatively easy to show that Eq. (7) implies that approximately $1/G$ values make a significant contribution to the current estimate of the average; keeping G finite, therefore, weights more recent data more heavily than older values. Using this picture, the value of G is determined so that the time interval needed to acquire $1/G$ observations is roughly the point at which the dominant noise process begins to deviate from white frequency noise. In other words, the averaging time $\Delta t/G$ corresponds approximately to the “break” in the slope of the plot of the Allan variance as a function of averaging time (28).

Things can get even worse at very long averaging times, as can be seen at the right side of Fig. 1. The slope is approaching $+1$ ($+2$ for the variance), so that the frequency no longer evolves symmetrically around the current value, and the current state begins to lose its ability to predict the future. Physically speaking, there have been so many random frequency steps during these long averaging times that the frequency at the end of the interval bears essentially no deterministic relationship to the value at the start. Although this sounds like our description of a Gaussian process, in that the previous values do not have any power to predict the future, the similarity is misleading. A Gaussian process is bounded on the average to variations about the mean that are only of the order of the standard deviation, whereas this process has no such constraint and has much wider tails.

The slope of the Allan variance specifies the best strategy at any averaging time, but it is its *magnitude* that specifies how well the strategy will work in any given device. The variance of the mean of a white process improves as the number of observations contributing to the average increases, but this improvement is, of course, with respect to the variance of a single observation. The cost of the synchronization process clearly also increases with the number of observations, so that any real implementation must balance the level of accuracy that can be achieved with a given clock against the cost of achieving it. As we discussed earlier it is often difficult to utilize the full potential of very accurate time tags because of latency and other system delays, so that there is no point in pushing synchronization to accuracies that will not be useful in a general-purpose environment.

In lower-accuracy applications, these considerations can be run in reverse—they can be used to determine the minimum cost for achieving a specified level of accuracy. Specifically, the product of the Allan deviation and the averaging time is a rough estimate of how well a synchronization algorithm can work using calibrations separated by a time interval equal to that averaging time. Physically speaking, the Allan deviation is a measure of how well a previous frequency estimate (computed using a given averaging interval) will predict the performance an equal interval into the future. The product of this quantity and the time interval is a measure of how far the time of the clock will drift during the interval due to the imperfect nature of the prediction. A more rigorous derivation shows that this product must be scaled by a number that is a function of the underlying noise type. This number is typically of order 1 and so does not change the general conclusion.

The Allan deviations of the two clocks whose data are shown in Fig. 1 are about the same at the shortest averaging times because the measurements are limited by white phase noise there. As we have discussed, this noise arises primarily from the *measurement process* and not from the statistics of the clock oscillator itself: The main contributions to the variance come from latencies in reading the time and in responding to interrupts. Therefore, the deviations at shortest periods tell us more about software latency (and possibly clock resolution) than about hardware quality.

The quality of the oscillator becomes important at longer averaging times, and it is here that the difference between the two systems becomes significant. The poorer oscillator leaves the white phase noise regime earlier, and its levels of flicker and random-walk frequency modulations are higher. There is a suggestion of a slight peak in the observations near an abscissa of 4.7, which is probably due to fluctuations in ambient temperature (or possibly power-line voltage) at approximately diurnal periods.

The Statistical Characteristics of Calibration Data

The second half of the technique of separation of variance is to characterize the calibration channel using the same method that we used for the clock oscillators. The ideal calibration channel would be one whose transmission delay was both known and stable to a fraction of a millisecond, so that its fluctuations could be ignored in the overall error budget of the system. The solution to the synchronization problem is relatively straightforward in this situation. The accuracy that can be realized is determined solely by the white phase noise of the oscillator—that is, by the unavoidable fluctuations in the delays associated with the measurement process itself. Because the calibration channel is essentially noise-free, we would focus on the properties of the oscillator alone and design an algorithm using the principles outlined in the previous section.

The same kind of results can be achieved even if the channel noise is not negligibly small, provided only that it can be characterized as white phase noise at all averaging times. This is the “best” kind of noise to have because an underlying mean time difference exists and can be estimated arbitrarily well by averaging increasing amounts of data. The effect of increasing the white phase noise in the channel is, therefore, predominantly a matter of cost rather than accuracy. It simply takes more data from a noisier channel to achieve a given level of performance. The calibration rate must be set fast enough in order that the required averaging can be realized in a time that is short enough that the oscillator noise is also dominated by white phase noise. This region can be rather small for low-quality oscillators, which tend to have rather high levels of random-walk frequency noise.

If the channel noise is high enough, it may dominate the noise budget for almost all averaging times. The optimum strategy is clear in this case—the local oscillator should be allowed to free-run until the averaging of the channel data causes its variance to fall below the free-running variance of the oscillator itself, and only then should a correction be applied. The resulting short-term performance may be quite poor in this case, but that is the best that can be accomplished using the data at hand—a faster correction time will degrade the clock stability by introducing channel noise into its free-running performance. Although this situation may seem pathological, it is, in fact, common—a good quartz-crystal oscillator is likely to be substantially quieter than most digital channels for averaging times on the order of seconds or less, and letting the oscillator free-run at short times is almost always the best strategy for this reason.

The situation is more complicated when the channel noise is not white phase noise at all averaging times. Many channels exhibit this rather unfortunate behavior. The principle is the same in this case: The channel data should be used only in those regions of the Fourier frequency domain where its noise is less than the free-running noise of the oscillator itself. The details will depend on the specific situation, but there are some characteristics that tend to be common to many situations. The local clock is usually better than the channel at short times—on the order of a few thousand seconds or less, and making a loop that tries to apply corrections much faster than this is quite likely to add phase noise. Conversely, the channel is almost always better than the local clock at very long averaging times (assuming that the reference at the other end is that good, too) because its variations are bounded and the Allan variance of its fluctuations must eventually decrease at sufficiently long averaging times. The intermediate period—on the order of a few hours—is usually

the most difficult region because the variance does not separate cleanly. Both the clock and the channel tend to have appreciable noise in this region of Fourier frequency, and there is no obviously optimum solution. Multiple clocks and channels may help here by providing a way of independently estimating the performance of each device, and this leads naturally to a discussion of ensembles.

ENSEMBLES OF CLOCKS

In the simplest arrangement, a clock to be synchronized receives timing information from a single trusted source via a (possibly) noisy channel. The difference between the time received and UTC is a random variable with a complex statistical distribution, but no long-term bias.

This configuration places a heavy load on the calibration source, especially if the measurement procedure involves some kind of active collaboration between the client and the server to estimate the propagation delay. In addition, this topology has a single point of failure, and the lack of redundancy makes it difficult to evaluate the accuracy of the received data.

One way of adding redundancy is to connect each client to several trusted sources—through independent channels, if possible. Two sources provide minimal redundancy and three sources provide the ability for error detection using majority-voting logic. These multiple connections are expensive in terms of the number of servers that are required to serve a given number of clients.

Some of the same redundancy and error detection can be realized by using data from other client machines (rather than from other servers), provided that these data can be combined in a statistically robust manner. The logical extension of this idea is an *ensemble*. Each member of the ensemble queries the others, requesting a time-difference datum. These time differences are corrected for any known systematic effects (such as a known rate offset, for example), and the resulting data are combined into a weighted average to yield an estimate of the time difference between the machine and the ensemble.

This averaging procedure is a statistically valid process if the data being averaged are randomly distributed about an underlying mean, and this will be true only if any systematic effects have been removed first. This is an imperfect procedure, of course, because the systematic offsets must be estimated from the data themselves in the presence of the noise processes we have described.

Each member of the ensemble is initially given equal weight in the computation of the average, but these weights are usually adjusted dynamically based on the subsequent measurements. One way of doing this is to set the weight of a given contributor equal to the reciprocal of the prediction variance of its data (29). In other words, a machine whose data exhibit a smoothly varying time difference with respect to the ensemble average is given high weight, as its data can be characterized by a simple rate offset with respect to the ensemble. We recognize this criterion as related to the Allan deviation of the time of the machine with respect to the ensemble average, so that the weights can also be characterized as proportional to the reciprocal of the corresponding Allan variances. The observation that the time difference of a particular clock with respect to the ensemble is smoothly varying is, of course, a statement about its level of phase noise in short term and frequency

noise at longer term, so that a clock that is given higher weight in the ensemble is “better” in the ways that we have discussed earlier. The sum of the weights of all of the clocks in an ensemble must be normalized to unity, of course, so that it is the relative Allan variances of the members that are important.

One of the primary advantages of an ensemble is that there is no single point of failure. Another is that it is usually easier to detect the failure of a single machine, as the ensemble average is likely to provide a more robust baseline for this test than the data from any single member machine. These advantages do not come for free. The software is considerably more complex, and casual associations are not feasible—each member must know all of the others and must maintain state information for each. Each estimate of the ensemble time requires that each of the N members request $N-1$ time differences, so that the network traffic to support the ensemble grows as N^2 . (The network traffic could be reduced by having these computations done in a single “master” machine, but this reintroduces a point of congestion and a single point of failure.)

We introduced the concept of an ensemble as a way of providing additional redundancy and error detection without the need for additional trusted time-servers. It is also possible to operate an ensemble without *any* primary time-servers. Each member of the ensemble is steered to the ensemble time, which is computed from the weighted average of the times of each of the members. If there is no external input, the actual ensemble average is arbitrary; this procedure will only guarantee that the times of all of the members agree—not necessarily that the ensemble time scale will have any relationship to UTC either in time or in rate.

This procedure will converge to a meaningful quantity if the times of the members are uniformly distributed about the ensemble average time with the deviations being characterized by white phase noise. Any systematic rate or time offsets must be removed before the average is computed; these systematic effects are usually estimated from the time differences themselves, and these estimates have inevitable uncertainties due to the noise inherent in the estimation process.

The time of each member clock in an ensemble is likely to be dominated by nonwhite processes at longer averaging times, and the ensemble average is likely to have the same statistical behavior. Thus, the frequency of the ensemble is likely to exhibit a random walk with respect to UTC at long averaging times, but this random walk will only characterize the behavior of the ensemble as a whole—the ensemble algorithm will keep the members synchronized to the average and, thus, to each other. This strategy might therefore be useful in a distributed environment where keeping the times of different machines synchronized to each other was more important than keeping them synchronized to UTC.

If the ensemble has only a few members and if one of them has significantly more weight than all of the others, then malfunctions in this clock (and especially changes in its frequency) may be very hard to detect. The ensemble tends to “follow” this clock because of its high weight, and the prediction error of the clock tends to be too small as a result. Therefore, any change in the parameters of the high-weight clock look like a change (with opposite sign) of the parameters of the other members. One solution is to limit the maximum weight that any member can have in the ensemble average; another is to compute the weight of a high-weight clock explicitly including the correlation between it and the ensemble average (30).

It is also important to keep in mind that each clock “sees” each of the other

members through a different, usually noisy channel. This noise must be taken into account using the procedures that we have outlined above. If the channel noise is asymmetrical or time varying and if each member computes its own ensemble average (to avoid a single point of failure), then the different computations may not see the same channel noise and may not agree as a result. The “ensemble average” is then not a well-defined quantity, but it may still be more stable than a simple client-server estimator using the same paths because the averaging process will attenuate the random component of the fluctuations.

RESET PROCEDURES

All of the methods we have been describing are based on the premise that it is possible to separate a data stream into two components—a deterministic portion derived from estimates of time and rate offsets, and a residual stochastic fluctuation than can only be described statistically. Many clocks and channels also exhibit occasional “glitches”—large deviations from the average statistical behavior. Such data could be treated as low-probability conforming events, but it is usually better to treat them as nonconforming glitches and to develop a special algorithm for handling them. The motivation for doing this is not philosophical but practical—the prediction error is improved on the average by this procedure.

The simplest glitch is a time step—a lost clock-interrupt, for example. It takes a minimum of three machines to detect the step and unambiguously decide which machine is at fault; therefore, glitch detectors are generally feasible only when some sort of ensemble software is being used or when one of the machines is a trusted server that is “never wrong.”

The case for a time step is quite strong if a time-difference datum differs from what we would have expected based on our prediction algorithm by an integral number of ticks. This identification also depends on the assumption that the difference is large compared to our estimate of the variance of the data, so that we can be reasonably confident that what we are seeing is really a glitch. The glitch must, therefore, be large both with respect to the phase noise of the clock and with respect to the frequency noise multiplied by the averaging time. Because the latter quantity increases with averaging time, time steps are easier to detect if the averaging time is kept short.

Frequency steps—discrete changes in frequency that are considerably larger than would be predicted from the Allan deviation—are much more of a problem. Because we observe the time of the clock (rather than its frequency), it can take some time before the frequency step produces a noticeable time offset. Frequency steps are obviously easier to detect if we are willing to average for a longer time, provided that they remain larger than the Allan deviation at the longer averaging time.

The situation is even more complicated if a frequency step and a time step happen at about the same time—the two effects will be eventually sorted out with additional data, but the resulting delay means that the system is not operating properly for some time.

One possible strategy is to treat all glitches as time steps initially. The best strategy for a time step is to adjust the time of the local clock to remove it. (This

adjustment can be done slowly or quickly, depending on the needs of the client processes—whether smoothness or accuracy are more important considerations.) If the glitch recurs on the next cycle, then a frequency step is probably the cause, and some adjustment to the prediction frequency is necessary.

Any approach to dealing with resets must be somewhat heuristic, because, by definition, they are events that cannot be characterized by some statistical function or distribution. A common strategy is to set the trigger for invoking the reset logic at three or four times the standard deviation of the measurements. With this kind of definition, the number of resets is independent of the quality of the clock—poorer clocks have larger standard deviations, so it takes a bigger glitch to trigger a reset. With these parameters, a typical clock will trigger the reset logic about once every 500 readings in normal operation (28).

The reset logic also has an impact on the interval between calibration cycles. As we discussed above, time steps become easier to detect as the time between calibrations is made shorter. Depending on the needs of the client processes, this consideration may drive the interval between calibrations to a very short value—much shorter than would be appropriate based on the statistical considerations we have discussed. Synchronizing a clock which has many time steps may require two types of calibration cycles—a relatively frequent set to test for time steps and a less frequent subset that is used to evaluate the statistical parameters of the oscillator.

FOREGROUND AND BACKGROUND PROCEDURES

The simplest synchronization procedure is for the primary foreground process that needs a time stamp to simply request one from the calibration channel and to use it either directly in the time-sensitive algorithm or indirectly to set the local clock to UTC. There are a number of widely available protocols that can be used in this way, including all of the broadcast services and the telephone time services such as ACTS and its relatives which were described earlier. There are also a number of internet protocols that are intended for this type of service. Some examples are the daytime (31) and time (32) protocols. These programs are often invoked automatically during the bootstrap process.

The accuracy of a time stamp received in this way is limited by the phase noise of the channel, and a clock set using this time stamp immediately begins to drift off the correct time at a rate given by its frequency offset. This rate offset is typically on the order of seconds per day, which is equivalent to milliseconds per minute. Unless the time stamp needs only to be accurate to the nearest second, the local clock will drift off rather quickly, and frequent calibrations will be necessary.

There are significant advantages to running a time-synchronization process continuously in the background as a “daemon” process. Because the process runs continuously, it can correct for the static rate offset of the local clock by making small time corrections which are invisible to the primary foreground activity. These corrections do not require additional calibrations and are made using the average frequency offset determined from previous measurements. In addition, such a process would schedule periodic calibration cycles to update its frequency estimate.

The accuracy of this procedure would be limited not by the static frequency

offset of the clock but by its (usually much smaller) Allan deviation evaluated at the interval between calibrations, because this value determines the fluctuations in the underlying frequency and, therefore, how well the static-frequency correction algorithm will work. Physically speaking, the gain in efficiency arises from the fact that although the rate offset of the local clock may be large, it is usually quite stable and predictable—or at least its stochastic component is much smaller than its static value. Once the steady-state operation has been reached, the calibration information is used primarily to determine the fluctuations in the rate offset and not the offset itself.

The Network Time Protocol that was mentioned earlier is widely used as a daemon process. The daemon actually consists of a number of logically independent procedures: a message format for interchanging time and clock-status information using udp/ip; a clock adjustment algorithm that applies periodic corrections to the local clock based on the information received and a daemon process that selects the servers, schedules the calibrations, and provides housekeeping functions. The algorithm can operate using a number of different calibration sources, including locally-connected radio clocks and network-based sources of time. In addition to synchronizing the local clock, the algorithm can act as a server, transmitting time to other client machines on the network.

The second algorithm is “Lockclock.” The published version (28) is designed to work using calibration data from the NIST ACTS telephone system, but it could be readily adapted to any other calibration source.

The principle difference between the two is that NTP is fundamentally a phase-lock loop with a variable compliance and Lockclock is a frequency-lock loop with an adjustable averaging time [the “G” parameter of Eq. (7)]. Using the considerations discussed above, it is clear that Lockclock can provide comparable performance to NTP with less frequent calibration data, because NTP must operate in the white phase noise regime of averaging times, whereas Lockclock operates in the white frequency noise portion of the spectrum. The NTP algorithm, on the other hand, is better able to cope with noisy channels or poor clocks—especially when the noise is not the relatively well-behaved white phase noise that dominates the ACTS data. As should be clear by now, there is no uniformly optimum choice between the two methods.

The methods we have discussed can also be used to evaluate any synchronization procedure. The ultimate limit of any procedure is the white phase noise associated with reading the clock itself—this noise is a characteristic of the clock hardware and the latencies in the host software and cannot be changed by the synchronization. Therefore, an optimum synchronization procedure is one in which the residual time differences between the clock and a trusted source transmitted via a noise-free channel are characterized by white phase noise at *all* averaging times. As above, this test can be implemented by computing the spectral density of the residuals using conventional Fourier methods, but it is usually easier to implement it by computing the Allan variance as a function of averaging time for the reasons we discussed earlier.

Figure 2 shows the Lockclock algorithm evaluated in this way. The time of the “better” clock shown in Fig. 1 is synchronized using it, and the time of the synchronized clock is compared to UTC using an independent, quiet channel. Comparing

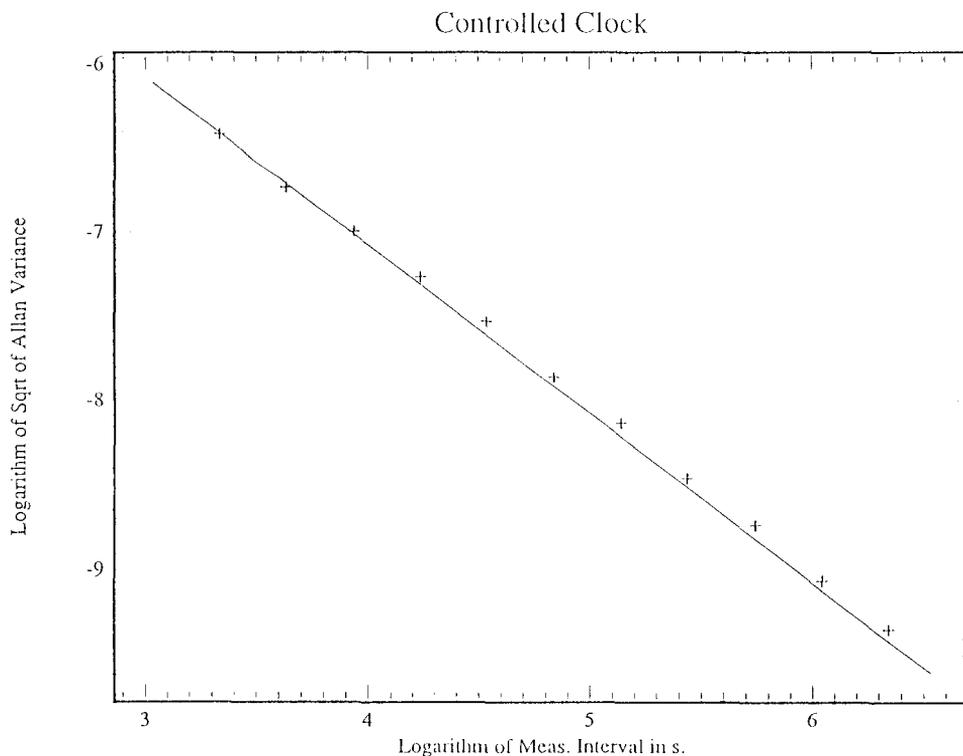


FIGURE 2 Allan deviation of a computer clock synchronized to ACTS using the Lockclock algorithm as described in the text. The reference line drawn through the points has a slope of -1 , which is characteristic of white phase noise. See Fig. 2 of Ref. 28.

the Allan deviation in Fig. 2 with the values in Fig. 1 illustrates the point and shows what a synchronization algorithm can and cannot do—there is essentially no improvement at the shortest periods because the algorithm cannot deal with the underlying white phase noise of the clock measurements. This noise might be minimized by averaging closely spaced time readings. Because the synchronization accuracy is already on the order of 1 ms with the simpler version of the algorithm, the improvement that this averaging could realize would be unlikely to be reflected in any increase in the accuracy of the time tags that could be delivered to client processes because of the inherent latency in reading the clock by a client process. Note the improvement over the unsynchronized clock at longer times and that the slope of the Allan variance continues to show that white phase noise dominates at all averaging times. The algorithm is, therefore, doing as well as can be done with the hardware and software at hand.

It is important to understand that the Lockclock algorithm could realize the same synchronization accuracy with the poorer clock shown in Fig. 1, because the limit to the synchronization accuracy is set by the level of white phase noise at the shortest times, and both clocks are essentially equivalent there. The difference

would be in the *cost*—the breakpoint in the Allan deviation plot for the poorer clock is at a smaller averaging time so that $\Delta t/G$ must be smaller as well. Therefore, the clock would need more frequent calibrations to achieve the same level of performance.

FUTURE DIRECTIONS

The separation of variance principles we have discussed are optimum in the sense that they describe a framework that can be used to design an algorithm to synchronize a clock as well as can be done with the hardware and information at hand or, alternatively, to find the minimum cost for achieving any lower level of synchronization. The method cannot be of much help in those Fourier domains where the variance does not separate cleanly—where it is not clear how much of the variance in a given region of Fourier frequency belongs to the local clock (and should therefore be removed by a time adjustment) and how much to the channel (and should therefore be averaged, otherwise attenuated or ignored). This problem tends to be most serious at intermediate periods on the order of several hours to 1 day, where almost all of the commonly used distribution channels have fluctuations in the propagation delay that are not white and often not even statistically stationary. Finding ways of characterizing these delays is an active research area at present.

If robust statistical measures can be found to characterize both the clock and the channel for all averaging times, then it may be possible to combine these measures into a single process which estimates both the channel noise and the clock performance simultaneously. The Kalman filter algorithm (33) provides a mathematical framework for doing this; its success depends on how well the statistical formulations can characterize the real-life performance of the hardware. As an example, Jones and Tryon (34) used the Kalman filter method to estimate the time of cesium clocks.

A second area of interest is the design of protocols that combine the simplicity and transmission efficiency of the one-way broadcast methods of time distribution with the increased accuracy than can only be achieved with two-way methods that measure the propagation delay directly. One way of accomplishing this might be to have a time-server that broadcasts time messages and is also willing to enter into two-way peer-type relationships for calibrating the delays to a number of individual clients. The broadcast and two-way modes could be supported simultaneously in the same hardware. Clients who did not need the increased accuracy that can be achieved by measuring the delay or who are unable to invest in the overhead needed to determine it could simply use the broadcast message “as-is,” whereas those who need a measured-delay mode would use the broadcast message to discover the name and address of the server and to make a preliminary estimate of their time offset.

Experience with the ACTS system suggests that only a small fraction of the clients use the two-way estimate of the propagation delay of the telephone system and that a simpler one-way system would adequately meet their needs. The continued popularity of the NIST radio broadcasts, which are simple one-way distribution systems, also confirms this idea. On the other hand, there is an increasing number of users who need more accuracy than can be provided by a simple one-way system, and finding ways of serving these customers economically is an important challenge

for the future (35). Satellite-based time services may provide part of the answer, but these are still quite costly to implement if many machines must be synchronized or if the antennas cannot be installed economically.

ACKNOWLEDGMENT

This work was supported in part by National Science Foundation grant NCR-9115055 through the University of Colorado.

REFERENCES

1. *The BIPM and the Convention du Mètre*, BIPM, Sèvres, France, 1987.
2. T. J. Quinn, *Proc. IEEE*, 79, 894-905 (1991).
3. D. D. McCarthy, *Proc. IEEE*, 79, 915-920 (1991).
4. H. Barrell and J. E. Sears, *Philos. Trans. Roy. Soc. London*, A238, 1-57 (1939).
5. B. Edlén, *Metrologia*, 2, 71-80 (1966).
6. K. E. Erickson, *J. Opt. Soc. Am.*, 63, 846-858 (1973).
7. G. D. Thayer, ESSA Tech. Rept. IER 56-ITSA 53, Environmental Science Services Administration, U.S. Dept. of Commerce, Boulder, CO (1967).
8. A. Leick, *GPS Satellite Surveying*, John Wiley and Sons, New York, 1990, pp. 219-226.
9. D. Hanson, *Proc. of the 43rd Symp. on Freq. Control*, IEEE, Piscataway, NJ, 1989, pp. 174-178.
10. J. Jespersen, *Proc. of the 43rd Symp. on Freq. Control*, IEEE, Piscataway, NJ, 1989, pp. 186-192.
11. W. Lewandowski and C. Thomas, *Proc. IEEE*, 79, 991-1000 (1991).
12. R. E. Beehler and M. A. Lombardi, NIST Special Pub. 432, NIST, U.S. Dept. of Commerce, Boulder, CO (1991), pp. 1-4.
13. G. Kamas and M. A. Lombardi, NIST Special Pub. 559, NIST, U.S. Dept. of Commerce, Boulder, CO (1990), pp. 64-65.
14. G. Kamas and M. A. Lombardi, NIST Special Pub. 559, NIST, U.S. Dept. of Commerce, Boulder, CO (1990), pp. 85-88.
15. G. Kamas and M. A. Lombardi, NIST Special Pub. 559, NIST, U.S. Dept. of Commerce, Boulder, CO (1990), pp. 105-106.
16. R. E. Beehler and M. A. Lombardi, NIST Special Pub. 432, NIST, U.S. Dept. of Commerce, Boulder, CO (1991), pp. 12-15.
17. A. J. Van Dierendonck and W. C. Melton, *Navigation*, 25, 127-145 (1984).
18. J. Levine, M. Weiss, D. D. Davis, D. W. Allan, and D. B. Sullivan, *J. Res. Natl. Inst. Stand. Tech.*, 94, 311-321 (1989).
19. F. Cordara, V. Pettiti, R. Quasso, and E. Rubiola, *Proc. 24th Annual Precise Time and Time Interval Applications and Planning Meeting*, NASA, Greenbelt, MD, 1993, pp. 243-254.
20. P. L. Petrakis, *Proc. 24th Annual Precise Time and Time Interval Applications and Planning Meeting*, NASA, Greenbelt, MD, 1993, pp. 255-266.
21. D. L. Mills, "DARPA Network Working Group Request for Comments," RFC-1305 (1992). This document contains references to earlier versions of NTP.
22. D. L. Mills, *IEEE Trans. Commun.*, 39, 1482-1493 (1991).
23. J. A. Barnes and S. Jarvis, Jr., NBS Tech Note 604, National Bureau of Standards, U.S. Dept. of Commerce, Boulder, CO (1971).

24. J. A. Barnes, *Proc. 15th Annual Precise Time and Time Interval Applications and Planning Meeting*, Naval Research Laboratory, Washington, DC, 1983, pp. 551–582.
25. S. R. Stein, “Frequency and Time—Their Measurement and Characterization,” in *Precision Frequency Control, Vol. 2*, E. A. Gerber and A. Ballato (eds.), Academic Press, New York, 1985, Chap. 12.
26. D. W. Allan, J. E. Gray, and H. E. Machlan, “The National Bureau of Standards Atomic Time Scale: Generation, Stability, Accuracy and Accessibility,” in *Time and Frequency, Theory and Fundamentals*, NBS Monograph 140, National Bureau of Standards, U.S. Dept. of Commerce, Boulder, CO, 1974, pp. 225–227.
27. D. W. Allan, *IEEE Trans. Ultrason., Ferroelectrics Frequency Control*, UFFC-34, 647–654 (1987).
28. J. Levine, *IEEE/ACM Trans. Networking*, 3, 42–50 (1995).
29. D. W. Allan, J. E. Gray, and H. E. Machlan, “The National Bureau of Standards Atomic Time Scale: Generation, Stability, Accuracy and Accessibility,” in *Time and Frequency, Theory and Fundamentals*, NBS Monograph 140, National Bureau of Standards, U.S. Dept. of Commerce, Boulder, CO, 1974, pp. 216–218.
30. P. Tavella, J. Azoubib, and C. Thomas, *Proc. 5th European Frequency and Time Forum*, 1991, pp. 435–441.
31. J. Postel, “DARPA Network Working Group Request for Comments,” RFC-867 (1983).
32. J. Postel and K. Harrenstien, “DARPA Network Working Group Request for Comments,” in RFC-868 (1983).
33. S. R. Stein, *Proc. 43rd Annual Symposium on Frequency Control*, IEEE, Piscataway, NJ, 1989, pp. 232–236.
34. R. H. Jones and P. V. Tryon, *J. Res. Natl. Bur. Stand.*, 83, 17–24 (1983).
35. J. Levine, *Proc. 25th Annual Precise Time and Time Interval Applications and Planning Meeting*, NASA, Greenbelt, MD, 1994, pp. 573–580.

BIBLIOGRAPHY

- Comer, D. E., *Internetworking with TCP/IP, Vol. I*, Prentice-Hall, Englewood Cliffs, NJ, 1991, pp. 293–302.
- Gelb, A. (ed.), *Applied Optimal Estimation*, The MIT Press, Cambridge, MA, 1992, pp. 102–155.
- Hackman, C. and Sullivan, D. B., “Resource Letter TFM-1: Time and Frequency Measurement,” *Am. J. Phys.*, 63, 306–317 (1995).
- Jespersion, J. and Fitz-Randolph, J., *From Sundials to Atomic Clocks*, Dover Publications, New York, 1982.
- Kochan, S. G., and Wood, P. H., *Unix Networking*, Hayden Books, Indianapolis, IN, 1989, pp. 49–93.
- Sullivan, D. B., Allan, D. W., Howe, D. A., and Walls, F. L., “Characterization of Clocks and Oscillators,” NIST Tech. Note 1337, NIST, U.S. Dept. of Commerce, Boulder, CO (1990), pp. TN14–TN127 and TN162–TN187.

JUDAH LEVINE