IOP PUBLISHING METROLOGIA
Metrologia **48** (2011) S203–S212 doi:10.1088/0026-1394/48/4/S12

Timing in telecommunications networks

Judah Levine

Time and Frequency Division and JILA, National Institute of Standards and Technology and the University of Colorado, Boulder, CO 80305, USA

Received 5 March 2011, in final form 14 April 2011
Published 20 July 2011
Online at stacks.iop.org/Met/48/S203

Abstract
I describe the statistical considerations used to design systems whose clocks are compared by the use of dial-up telephone lines or the Internet to exchange timing information. The comparison is usually used to synchronize the time of a client system to the time of a server, which is, in turn, synchronized to the time scale of a national timing laboratory. The design includes a dynamic estimate of the system performance and a comparison between the performance and a parameter that specifies the required stability based on external considerations. The algorithm adjusts the polling interval and other parameters of the algorithm to realize the specified performance at minimal cost, where the cost is taken to be proportional to the inverse of the interval between message exchanges using either the Internet or dial-up telephone calls.

1. Introduction

The National Institute of Standards and Technology (NIST) currently operates 35 public network time servers that are located at 21 different sites in the United States [1]. (Several sites have more than one server.) The servers provide time signals to users in a number of different time formats. The ensemble of servers currently receives about 5×10^9 requests per day; most of these requests are for time in the NTP (Network Time Protocol) format [2]. The number of requests has been increasing by about 5% per month for several years. Therefore, we are continuing to study methods of increasing the number of users we can support with only a smaller increase in the hardware needed to support the service. The results of these studies are presented in this work.

In addition, NIST operates an ensemble of servers that provide time in the ACTS (Automated Computer Time Service) format [3] using dial-up telephone lines. These servers currently receive several thousand requests per day and are also used to synchronize the network time servers that are not located at the NIST Time and Frequency laboratory in Boulder, Colorado. Both the network-based and the telephone-based time services use standard publicly available communication services, and we do not consider the methods used to synchronize the networks themselves.

All of the servers are synchronized to UTC(NIST). The time servers form a clock ensemble that is linked to the NIST atomic clock ensemble in Boulder, Colorado, by the use of a hard-wired connection for the systems that are located at NIST and dial-up telephone lines, which implement the ACTS protocol, for the remote systems. The users of the service form a second ensemble, which is linked to the first one using public networks such as the Internet. In some cases, a user may implement a third ensemble whose members are linked together by a network that may be either a private network or the public Internet. This hierarchy of ensembles is formalized in the 'stratum number,' which specifies the number of systems between a given system and a national timing laboratory such as NIST. Using this terminology, all of the NIST systems operate at stratum 1.

I will discuss the details of these clock ensembles in this paper. Although these ensembles use many of the techniques that are familiar from the atomic clock ensembles that are used by timing laboratories and national metrology institutes to realize UTC, there are also important differences.

The most important difference between the ensemble that provides the network time service and the atomic clock ensemble that realizes UTC(NIST) is that the members of the network time service ensemble are linked together by a noisy network. The network has delay fluctuations that cannot be well characterized by the use of the usual statistical methods that we use for atomic clock ensembles. The measurement noise is significant relative to the clock noise, and is very definitely not well characterized by a Gaussian distribution. In addition, the clocks at the remote systems are relatively poor quality quartz-crystal oscillators and have stochastic frequency ageing and a sensitivity to ambient temperature fluctuations that is difficult to model. Finally, there is a cost (in network bandwidth, computer cycles and telephone line charges) in implementing the synchronization algorithms, so that the time

scale must be designed with a cost/benefit analysis in mind, especially from the point of view of the servers and the network link between the servers and the community of users. Although the size of a time message is relatively small (less than 50 octets in general) and therefore does not require a large bandwidth, there is a significant overhead in establishing and monitoring each connection, so that there is a strong incentive to minimize the number of time messages needed to implement the synchronization procedure.

The goal of both the primary ensemble algorithm (which is used to synchronize the servers to UTC(NIST)) and the secondary ensemble algorithm (which is used by the client systems to synchronize the internal system clock to the time of the servers) is to compensate for the instabilities in the network delay and for the frequency noise in the local oscillators of the systems, with the understanding that the stability of the network is usually the limiting factor in the performance of the overall synchronization process. Improving the clocks at the system nodes can improve the performance to some extent, but not nearly as much as one would like.

2. The measurement protocols

The links between a remote NIST server and the NIST atomic clock ensemble and between an end-user and one of the NIST time servers use the two-way transmission protocol. The details of how the protocol is realized vary somewhat from one message format to another, but all of the methods share a common model: the round-trip transmission delay between two systems is measured and the one-way delay is modelled as one-half of this value. I will discuss the details of the measurement format for the two important versions of this idea: the Network Time Protocol (NTP) and the ACTS format.

In the NTP format, a client requests the time from a server and the client corrects the time data that it receives for the transmission delay through the channel using its own state and information provided by the server. The exchange begins when the client sends a request to the server at time t_1 as measured by its internal clock. The one-way transmission delay is d, so that the time of the clock on the client has advanced to $t_1 + d$ when the message reaches the server. The time on the server clock at that instant is t_2, so that the time difference (client − server) is

$$\Delta t = (t_1 + d) - t_2. \tag{1}$$

The server responds at time t_3 on its clock and the response is received back at the client at time t_4, as measured by the clock on the client. The two-way transmission delay is

$$D = (t_4 - t_1) - (t_3 - t_2) \tag{2}$$

where the first term is the total time elapsed during the exchange of the messages (as measured by the client) and the second term is the latency between when the server received the request and when it responded, as measured by the clock on the server. (The reply from the server contains the information in the second term of equation (2).) In a well-designed system, the magnitude of the second term is negligibly small compared with the magnitude of the first one. For example, the magnitude of the first term is tens of milliseconds on a typical Internet path, while the magnitude of the second term is a few microseconds on all of the NIST servers.

The assumption of the two-way model is that the one-way delay is one-half of the round-trip value. That is, that $d = D/2$. In order to see the impact of asymmetry in the delay, we will define an asymmetry parameter k, and set

$$d = kD = k(t_4 - t_1) - k(t_3 - t_2), \tag{3}$$

so that $k = 0.5$ is perfect delay symmetry. Substituting equation (3) into equation (1), we get

$$\Delta t = (1 - k)(t_1 - t_2) + k(t_4 - t_3). \tag{4}$$

The limiting values $k = 0$ and $k = 1$ correspond to the situations where the request or reply delays, respectively, make a negligible contribution to D. The sensitivity of the measured time differences to small asymmetries about the value $k = 0.5$ is given by

$$\frac{\Delta t}{\Delta k} = t_4 - t_1 - t_3 + t_2 = D, \tag{5}$$

so that

$$\Delta t = D\Delta k = D(k - 0.5). \tag{6}$$

Thus, any asymmetry in the path delay introduces an equivalent timing error proportional to the magnitude of the asymmetry and the path delay itself. A well-designed network will therefore minimize D, the path delay from the client to the server, for as many clients as possible. (Realizing this requirement suggests geographical diversity in the location of the servers. Unfortunately, the network delay and the physical distance between the two systems are often only weakly related to each other.) Since the network delay is typically of the order of 0.1 s, an asymmetry of order 1% would result in a time error of about 0.001 s. This is an optimistically small asymmetry for the public Internet, but it can be realized on private networks and using conventional dial-up telephone lines. A more typical value for the asymmetry on the public Internet is a few per cent, so that the protocol can support timing accuracy of a few milliseconds for many wide-area network paths. Note that even *a perfect* clock on the server does not change this conclusion. The timing error in equation (6) arises from our ignorance of the *true* asymmetry, and our assumption that it is 0. I will discuss this point in more detail below.

The ACTS system uses the same protocol, except that the network delay is measured by the server rather than by the client. The message from the server to the client contains an on-time marker character, which the client echoes back to the server with as little delay as possible. The server uses the time that it received the echo character to estimate the round-trip delay using essentially the same method as described above, and advances the on-time marker on the next transmission by the measured delay. The protocol assumes that the system latency, $(t_3 - t_2)$ in equation (2), is small enough to be ignored in the computation, and this is easy to realize in practice because the system latency is of the order of a few microseconds, whereas the network delay is at least tens of milliseconds. It is not too difficult to realize a delay asymmetry of less

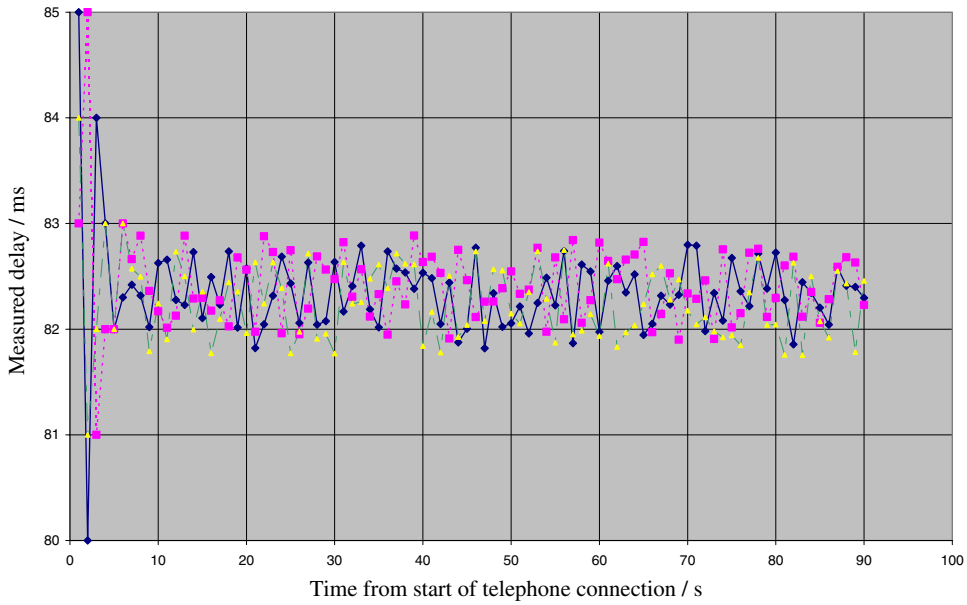

Figure 1. The delay measured by the ACTS server on several consecutive telephone connections spaced a few minutes apart. (Each connection is shown with a different colour.) The client echoes the on-time marker back to the server using a hardware loop-back cable that eliminates any latency delays in the client. The transients at the start of each call are caused by the equalizing circuits in the modems. (This figure is in colour only in the electronic version)

than 1% using traditional voice-grade telephone lines and standard communications modems, so that the ACTS protocol can realize a timing accuracy of about 0.001 s.

In both cases, the variance of the asymmetry is an important consideration in determining the precision of the process, and we will discuss this in more detail below.

3. The ACTS servers

Since the ACTS system is used as the timing reference for the network time servers that are not located at NIST in Boulder, Colorado, we begin by discussing the accuracy and precision of the ACTS protocol. The details of the ACTS time messages and the hardware that is used to support the ACTS system have been described in detail in previous publications [3, 4]. Here we focus on the statistics of the ACTS transmission system, since it sets the limit on how well the remote time servers will perform.

All of the ACTS computers are synchronized using a direct, hard-wired connection between the computer that supports the protocol and UTC(NIST) as realized by the NIST clock ensemble. The connection uses 1 Hz pulses, and it is a simple matter to synchronize the time of the computers that support the ACTS with a timing offset and jitter of $\leqslant 2\,\mu$s. Since the timing pulses are available every second, the software to synchronize the clock that is internal to the ACTS computer and that serves as the reference for the ACTS timing messages is simple and straightforward. The 1 Hz pulses generate system interrupts, and the seconds fraction of the internal clock is driven to 0 each time an interrupt is received. (The systems that are used as ACTS servers maintain the system time in two registers: a counter whose value represents the number of seconds that have elapsed since the epoch that was chosen by the designers as the origin of time and a second counter that show the fraction of a second in units of microseconds.) This algorithm can lock the system to the wrong second in principle, and this possibility is avoided by having the synchronization loop operate sufficiently often that there is no possibility of the frequency offset of the system clock producing a time offset of anything close to 0.5 s. Since the frequency offset of the internal oscillator is of the order of seconds per day, the polling interval, which is of the order of 1 min, is much faster than is needed to ensure that the synchronization process does not cause the clock to jump by an integer second.

Since the times of the clocks in the ACTS servers are synchronized with an accuracy of a few microseconds relative to UTC(NIST), the performance of the ACTS system as seen by a user will be dominated by the accuracy of the procedure that determines the one-way transmission delay and by the stability of the delay. Figure 1 shows the variation in the delay measured by the ACTS by the use of a simple loop-back configuration in which the client system echoes the on-time marker back to the server using a hard-wired loop-back cable. The various traces in the figure are the results of telephone connections made a few minutes apart. In each case, there is an initial transient in the delay as the equalizing circuits adjust to the line characteristics, but the delay then becomes stable with a variation of about 0.001 s peak-to-peak both within a single connection and between consecutive connections spaced a few minutes apart.

The delay reported in any message and used to advance the on-time marker character of that message was actually computed based on the exchange of the previous message. Therefore, the ACTS system will compute and implement an unbiased estimate of the true delay if the asymmetry of the delay is close to 0, if the delay is stationary (so that the time of

the measurement is not important on the average) and if it is well characterized by white phase noise (so that the average of the delay provides an unbiased estimate of the actual value). These were reasonable assumptions when the ACTS system was first installed in 1988, but they are less true now because more and more telephone circuits are implemented as logical circuits multiplexed by the use of analogue and, increasingly, digital methods.

The stability of the delay measured by the ACTS system is not the whole story, because the timing accuracy depends on the one-way delay, whereas the ACTS system (as with all two-way algorithms) uses one-half of the measured round-trip value as the one-way delay, which is used to advance the on-time marker in the next message. Since the ACTS system uses the delay measured on the previous transmission to adjust the advance for the next one, variations in the delay with periods close to 1 s are not handled correctly and are in fact amplified by the ACTS protocol.

The ACTS system can be affected by a second, more subtle problem when the telephone connection is implemented over packet-switched networks with appreciable switching delays. Although there is no difference in principle between measuring the round-trip delay at the server and measuring it at the client, the measurement at the server is less accurate when the asymmetry of the line delay cannot be well characterized by a Gaussian random variation. The ACTS server sees only the round-trip delay and adjusts the on-time marker of the next transmission to compensate for it by the use of the usual two-way assumption. However, the client sees both the delay measured by the server (which is part of the ACTS transmission) *and* the measured time difference between the time in the message and its local clock. Since the clock of the client system does not change its characteristics from one second to the next one, the client can detect a change in the symmetry of the path by measuring the variation in the measured time differences from second to second between the received messages and its clock. While it cannot measure an *absolute, static* asymmetry, it can easily detect changes in the asymmetry of the order of 1% from second to second. This information is not available to the server.

The NIST Internet time servers implement this strategy by measuring consecutive time differences between the ACTS time messages and the time of the system clock. These data are then combined with the delay values reported by the ACTS servers to estimate the statistics of the delay. This method is not any better than simply averaging the measured time differences when the asymmetry is always close to 0 and the delay is well characterized as a Gaussian random variable, but it can compensate for the flicker and random-walk character of the delay on newer telephone circuits. It is especially useful in detecting the linear trend in the asymmetry that is typical of the impact of the equalizing circuits in the telephone line and the modems. The result is that the more sophisticated algorithm can handle the newer circuits with almost no degradation in stability relative to the original system that was designed in the days of simpler analogue circuits.

No two-way protocol can correct for a static asymmetry or for an asymmetry in the delay that changes very slowly, since the slow variation is hard to distinguish from the random walk of the computer clocks. The modems make a significant contribution to the round-trip delay, and we calibrate all of the modems used in the NIST time service to measure the asymmetry in the modem delay. (The delay itself need not be calibrated, since it will be measured by the round-trip protocol.) For the particular modems that we use, the asymmetry in the delay is smallest when the modems operate at a transmission speed of 9600 baud, and all of our time severs use this speed. (This conclusion is based on tests of our particular brand of modems, and may not be generally true.)

In spite of all of these improvements, there are some telephone circuits that have delay variations that cannot be adequately modelled. The most serious problem is a telephone line that has a stable delay and asymmetry over the short term but a bi-modal variation in the asymmetry over longer periods. The bi-modal asymmetry produces steps in the measured time differences at the remote system with periods of a few hours. The time steps are of order 0.020 s, which is easily detectable because it is much larger than the time dispersion of the system clock over the averaging interval of about 1 h. There is no way to remove these time steps, since we do not know the 'true' time offset of the remote system. There is no basis for choosing the mean of the bi-modal time steps as the correct value or for using assuming that the minimum round-trip delay has a smaller asymmetry. (This latter assumption is sometimes used to characterize the delay on a wide-area network, and it can be useful on systems such as web servers, which often have a large difference between the inbound and outbound message sizes.)

4. Design of an ACTS-based time client

The NIST time servers that are not located at the NIST laboratory in Boulder, Colorado, are synchronized by the use of periodic telephone connections to the ACTS servers described in the previous section. They re-transmit the time to users on the public Internet. The synchronization process on these servers is designed to complement the statistical performance of the ACTS system that I described in the previous section. Since telephone calls are relatively expensive, the algorithm is designed to exploit the stability of the local system clock so as to maximize the time between calls. As long as the total length of the call is less than 30 s, the cost of a telephone call is not increased by processing consecutive time messages as I described in the previous section. Even in the best situation, the standard deviation of the measured time difference improves only as the square root of the number of measurements, so that there is not a great advantage to using more than about 10 measurements, which results in a telephone call about 25 s long.

The first step in the design of the system that will be used as a time server is to evaluate the stability of its free-running internal oscillator, and the typical results of this evaluation are shown in figure 2. We evaluate the oscillator using an external atomic clock as the reference and querying the operating system for the system time each time we receive a 1 Hz tick from the external clock. This procedure evaluates the oscillator

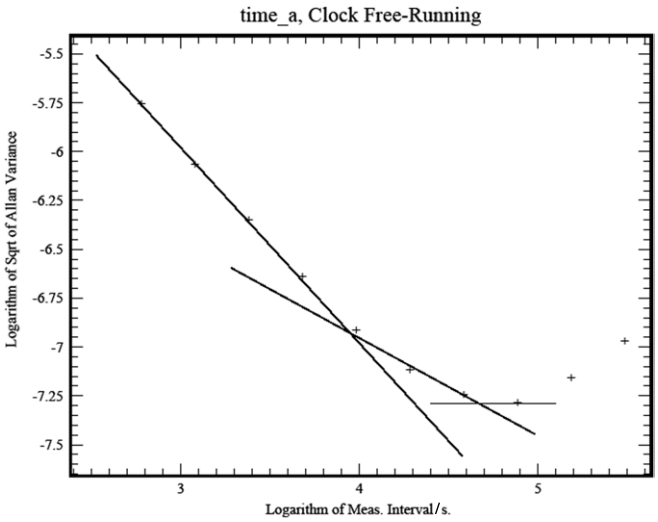

Figure 2. The Allan deviation (square root of the Allan variance) of the free-running clock on a typical NIST time server. The points show the measured values and the straight lines have slopes of −1, −0.5 and 0 to illustrate the domains of the various noise types. The data are obtained using the operating system to query the oscillator periodically in response to interrupts received from an external atomic clock. The data do not represent the performance of the oscillator itself, which is probably quieter, especially at shorter averaging time.

as it is seen from inside the operating system. We cannot access the actual hardware oscillator, whose performance is probably better than figure 2, especially at short averaging times, where we are limited by the system latency in processing our time requests. The performance of the physical oscillator is not very useful for this work anyway, since the software can see the clock only through the operating system request process. Because of this limitation, there is only little to be gained by replacing the internal physical crystal oscillator with one that is more stable. On the other hand, anything that reduces the jitter and latency in the operating system request process that is used to read the local clock is very helpful. The simplest way of realizing this improvement is to minimize the number of processes that are active at any time, with special emphasis on processes that expect near real-time responsiveness, such as a mouse-driven graphical-user interface. We have also modified the operating system kernel to improve the handling of requests for the system time.

The straight lines on the log–log plot have slopes of −1, −0.5 and 0 to illustrate the domains of applicability of the usual power law noise types. The time stability in the white phase noise domain is about 0.0006 s, calculated by $\tau \times \sigma_y(\tau)/\sqrt{3}$, where $\sigma_y(\tau)$ is the Allan deviation for an averaging time of τ seconds. This value is independent of averaging time in this noise domain, since the Allan deviation is proportional to the reciprocal of the averaging time. The free-running stability of the system clock in the white phase noise domain is about a factor of two better than the stability of the ACTS system, even with the averaging scheme described in the previous section. As we will discuss in the following section, a system that connects to this server through the Internet cannot expect a timing accuracy of better than a few milliseconds because of the jitter in the network delay. Therefore, there is no advantage to synchronizing the clock to much better than this value because the cost of doing so does not result in an improvement of the time service as seen by the users.

The previous considerations are valid only so long as the fluctuations in the local clock are dominated by white phase noise, and this becomes less true at averaging times greater than about 1 h, so that the server is configured to connect to the ACTS system about every 40 min. The ACTS time differences are used to adjust the local clock by means of the algorithm that has been described previously [5]. The only important difference in the current implementation is the handling of the ACTS messages described in the previous section. Many of the servers would not be able to function without this improvement.

5. Design of a network-based time client

5.1. Hardware and software support

In the following sections I will describe the design of a client system whose clock is synchronized to a national time scale such as UTC(NIST) by exchanging time messages over a public network such as the Internet. The messages could be exchanged with one of the time servers operated by NIST and described in the previous section, but this is not a requirement—any time server that conforms to the two-way delay measurement method and uses messages that conform to the network time protocol is acceptable. The algorithm will be based on the statistical principles I have discussed in the previous sections.

I assume that the algorithm will be implemented on one of the commonly available computer systems and that no modifications will be necessary to the usual hardware and operating system. The algorithm will require read/write access to the system time parameter; this access is typically available to privileged processes on all systems. Finally, I assume that the system is connected to the public Internet through a reasonably fast network connection. The details of the physical layer of the connection are not important, except that packet-based connections using dial-up telephone lines ('ppp' or 'slip' formats) are usually too slow and have delays that are too variable to be useful for timing applications.

5.2. Initial configuration

The synchronization software requires only two parameters: the names or network addresses of several servers that can be queried to provide the time messages and a parameter that specifies the desired RMS accuracy of the synchronization process, which we will designate as T_a. The value of this parameter will be determined based on the application that is using the system clock. This parameter will be used to configure the functioning of the synchronization process and will be used to estimate the minimum cost (in terms of computer cycles, network bandwidth, etc) that is required to realize the required accuracy, given the stability of the system clock and the network connection. Specifying a value much smaller than is necessary to support the application will

increase the cost of running the algorithm and will increase the load on the servers without improving the performance of the application. Choosing a value that is too large may degrade the accuracy of the system time to the point that it cannot support the application.

5.3. Message format

The client system will communicate with the time servers by the use of the usual two-way message exchange that I have described above. The message exchange format used by the Network Time Protocol is perfectly suited to this requirement, so that the client system we are describing will look like a standard NTP client to the servers that it queries. The job of our algorithm will begin once the time difference and network delay have been estimated from the data in the message exchange with the server.

5.4. Performance metric

In order to evaluate the performance of the system clock, we will use the Allan (two-sample) deviation as a function of the averaging time, τ, denoted by $\sigma_y(\tau)$. Although modified forms of this deviation are better able to characterize the spectrum of the system time noise process, this is not so important in this application, and the additional insight provided by these statistics is not worth the additional complexity needed to calculate them. In addition, the standard Allan deviation more accurately models the way the system clock is actually used and is therefore a better metric in this application. The synchronization process does not use the averaging of consecutive blocks that is the basis for the modified Allan deviation and the time deviation, TDEV, which is derived from it.

We will use $T_c = \tau \times \sigma_y(\tau)$ as a measure of the time dispersion of the system clock. (This estimate is too large by a factor of $1/\sqrt{3}$ for pure white phase noise, and the more exact statistic could be used in principle. Our experience is that the variance of the system time is rarely characterized by pure white phase noise, and that our statistic is usually a somewhat better description of the actual performance.) The algorithm computes a running estimate of the time dispersion parameter, T_c, using the time-difference data acquired on consecutive message exchanges. Each computation uses 24 h of data to estimate the Allan deviation and the time dispersion for averaging times out to a few hours. The goal of the algorithm, which we will describe in more detail in the following sections, is to choose the averaging time, τ, such that $T_c \leqslant T_a$. That is, the time dispersion of the system clock is not worse than the requirements of the application that depends on it. In most cases, the averaging time will be set to less than this value to allow for a safety margin and because the performance metric is a root-mean-square (rather than a peak) deviation.

5.5. System clock model

We model the system clock using the usual three-parameter model of a time difference, x, a frequency offset, y, and a frequency noise parameter, η. During the measurement interval, τ, the system time and frequency evolve as

$$x(t) = x(t-\tau) + y(t-\tau)\tau \tag{7}$$

$$y(t) = y(t-\tau) + \eta \tag{8}$$

and the message exchange between this system and a *single* remote server produces a time difference at time t given by $\Delta X(t)$. The phase noise of the system clock, which is typically of the order of a few microseconds, can be ignored relative to the noise of the measurement process and the time dispersion due to the frequency noise. These noise sources are generally at least 100× larger. The frequency noise parameter, η, includes the stochastic contribution of the frequency ageing, which is typically larger than the contribution due to deterministic frequency ageing. (The sensitivity of the computer clock oscillator to ambient temperature fluctuations and the sensitivity to variations in the power supply voltage are treated as stochastic parameters because we do not have any information to model them accurately. These are important sources of frequency fluctuations, and modelling them is a subject for future work.)

5.6. Measurement algorithm

The first step in the algorithm is to compare the measured time difference $\Delta X(t)$ with the prediction of the model equation (7). When the algorithm is operating in steady state, the state of the physical clock on the previous measurement cycle was adjusted to drive both $x(t-\tau)$ and $y(t-\tau)$ to 0. Therefore the expected value of the current measurement is $\Delta X(t) = 0$. We will take this requirement as satisfied if

$$\Delta X(t) \leqslant 3 \times \tau_c. \tag{9}$$

That is, the time offset on the current measurement cycle is consistent with the value derived from the statistical estimate of the time deviation of the system clock. The factor of 3 is chosen so that the test should be satisfied with a probability of about 80% if the measured time difference is consistent with our statistical estimate of the clock performance. Note that the noise processes are only approximately Gaussian, and that the probability of a large value of the time difference is greater than standard Gaussian statistics would predict.

If we pass this test, we assume that the time dispersion is mostly due to frequency noise, and we model the frequency noise as a Gaussian process at the averaging time that was used. We update the frequency estimate using an exponential filter whose time constant, T_y, is chosen to ensure that the assumption of Gaussian frequency noise is satisfied. (The initial value of this constant is determined by the free-running stability of the clock as shown in figure 2, and this parameter is updated by the periodic re-evaluations of the Allan deviation of the system clock as described above. Note that the data in figure 2 were measured using a special-purpose device that was connected to the computer bus and that has negligible measurement noise. The actual algorithm would use the time-difference measurements obtained through the actual network

connection and would almost certainly not be this quiet.) The updated frequency is given by

$$y(t) = \frac{y(t-\tau) + T_y \dfrac{\Delta X(t)}{\tau}}{1 + T_y} \tag{10}$$

and we apply this new frequency to the physical system clock. Depending on the details of the hardware and the operating system, this adjustment is made either by adjusting the software clock frequency in the system kernel or by making periodic very small adjustments to the system time if the kernel does not support clock frequency tuning. We also apply a time adjustment $-\Delta X(t)$ to the system clock to set its time to correspond to the time of the server. The details of this adjustment also depend on the details of the system kernel. This time adjustment is usually implemented by a temporary adjustment to the effective clock frequency so that there is never a step discontinuity in the system time. These adjustments are usually made to the time of the system clock as seen through the operating system—the physical quartz crystal oscillator itself generally cannot be adjusted. However, since the physical oscillator can be interrogated only in this way, the effect is to modify its effective physical characteristics. (In contrast, note that atomic clock ensembles *never* adjust the parameters of a physical clock—the estimates of the time and frequency offset are maintained as parameters that are distinct from the physically measured time of the clock.)

5.7. Error detection

If the measured time difference does not satisfy equation (9), then we have to proceed to various tests to determine the problem. The first possibility is that either the local clock or the remote clock (*as seen through the network*) has experienced a time step. We query a second server and repeat the tests above. The possibilities are the following:

1. The two servers agree and the local clock disagrees with both of them (within the expectation of the noise variance). We assume that the local clock has experienced a time step. This is actually quite a rare event and is usually an indication of a hardware problem. We adjust the time of the local clock by the measured time step and do not change any of the other parameters. The large time difference will propagate into the calculation of the Allan deviation. Since Allan deviation is an average over several cycles, it is not adjusted by the full magnitude of a single time step. However, the effect of the time step is to increase the tolerance to future fluctuations. If the time step is a one-time event, then the variance will gradually return to its previous value on subsequent computations. If the time step is really caused by a frequency step (which is generally more likely), the increase in the Allan deviation will allow the increased time dispersion to modify the frequency parameter on future cycles. If the time step is caused by relatively slow frequency wander, then the future calculations of the Allan deviation will adjust the time constant in equation (10), since the wander will affect the transition point where white frequency noise is no longer a good assumption for the variance.
2. The time difference of the second server satisfies equation (9). The algorithm accepts this time and continues. The algorithm assumes that the first server has an undetected internal time error since the program would not have used the data from a server that had declared itself to be unhealthy.
3. The two time differences do not agree with the expected value and they are not consistent with each other. This is most likely a network asymmetry problem, although it is possible that both servers have timing errors. The program will query a third server if it is available and repeat the tests. Otherwise, it makes no change to the state of the clock, since the cause of discrepancy is ambiguous. (When in doubt about what to do then do nothing.)

The error detection algorithm we have described has two important design features that are not present in many other network synchronization algorithms. The first is that the local clock is used to evaluate the accuracy of the measured time difference. The algorithm has information on the history of the statistics of the local clock, and that history can (and should) be used in the error detection process. The second important feature is that the algorithm does not contact a second time server unless there is some question about the accuracy of the message received from the first one. This is really a different aspect of the first feature but it has a very significant implication on the number of time requests that are needed on every calibration cycle.

For example, if the probability that the remote server has an *undetected* internal error is a few per cent, then the data from the remote system will be accepted if (*and only if*) the error in the remote data is such that it is consistent with the time of the local clock, which means that both the local and remote systems must be broken simultaneously in the same way and that the error is not detected by either system. There is no easy way of calculating the joint probability that both the local and remote systems have the same error, but, since the two systems have no common elements, an undetected error in the remote system is much more likely to trigger case 2 in the error algorithm, and the probability of a joint error is smaller than a few per cent, which was the probability of a single error in the remote system. Therefore, querying multiple servers on every cycle is a sub-optimal choice, since the extra queries multiply the cost in direct proportion to the number of servers queried, but the multiple queries provide little extra information most of the time. (At best, the standard deviation of the time difference improves as the square root of the number of servers queried, while the cost increases linearly with this number.)

5.8. Update of the polling interval

The polling interval (the interval between queries to the remote time servers) is set dynamically based on two considerations. (1) If the RMS time dispersion (as defined above) is very different from the performance specified then we will adjust the polling interval. If

$$T_c \ll T_a \tag{11}$$

then the RMS performance of the clock is much better than is necessary and we can increase the polling interval. Conversely, if the polling interval is already too long, then we must decrease it to compensate for the poor stability of the local clock (or of the remote server as seen through the network). This adjustment continues until either a minimum poll interval of the order of 10 s or a maximum interval of the order of a few hours is reached. Equation (11) may never be satisfied for any poll interval if the overall system is sufficiently unstable.

5.9. The cost–benefit model

Since each request for time from a remote system is a distinct event (as opposed to contacting the ACTS server using a dial-up telephone call, where the cost of establishing the connection is fixed and the incremental cost of receiving an additional time message is negligibly small), the cost of the algorithm is proportional to the reciprocal of the polling interval, or $1/\tau$. The performance of the algorithm is related to T_c, so that we can define the quality of the algorithm, Q, as a function of the interval between queries by the product

$$Q(\tau) = \frac{1}{\tau} T_c = \sigma_y(\tau) \sim \tau^m, \tag{12}$$

where m is the exponent that approximately characterizes the Allan deviation of the local clock for an averaging time of τ. We wish to minimize the value of Q by increasing the polling interval and/or by decreasing T_c. This function will be minimized by choosing the largest polling interval, τ, for which the exponent $m < 0$, which means choosing a polling interval so that the noise of the algorithm can be characterized as no more divergent than white frequency noise.

On the other hand, the performance of the algorithm is proportional to τ^{m+1}, where m is defined as above, so that, as the polling interval is increased, the performance of the algorithm will begin to degrade in the white frequency domain ($m = -0.5$) even as it becomes more efficient. Choosing the polling interval domain where $m = -1$ (white phase noise) is clearly a bad choice using either metric since the quality decreases and the performance does not improve. If the polling interval is made so short that the characteristics of the local clock do not change between queries, then the performance improves as the square root of the number of queries. In the limit of very rapid queries, the query interval is so rapid that the local clock is effectively not being used at all—the time tag is simply the average of the data received from the remote end.

If the polling interval is increased beyond the optimum point computed in the previous section, the cost–benefit calculation becomes less favourable. The cost continues to decrease as the polling interval gets longer, but the accuracy of the local clock is degrading faster than linearly with the polling interval. In spite of this fact, it may be desirable to operate in this regime if T_c, the RMS time dispersion of the synchronization process, is much less than T_a, the time stability required by the application. As an extreme example, the Allan deviation of the last point in figure 2 corresponds to an averaging time of about 3.6 days. The time dispersion for this averaging time is less than 0.1 s, which would be more than adequate for an application that needed a time accuracy of 0.5 s. Two somewhat different versions of this algorithm, called the 'adaptive frequency-lock loop', were described in [6, 7]. The first paper [6] also contains a test of the method using a real network path that is typical of many network connections and an externally defined performance metric of 1 s.

5.10. Synchronizing the clock by the use of data from the physical layer

If the synchronization algorithm we have described operates as a normal user process, then it must compete with other processes for computer cycles and for access to the hardware. If the system is moderately busy, then this competition may result in relatively large jitter in the stability and accuracy of the system time, since the jitter affects both the system time and channel delay estimates. It is tempting to reduce this jitter by moving at least part of the synchronization algorithm closer to the device that actually receives and transmits the time messages. The ultimate version of this idea would be to move the time-stamp processing of the synchronization algorithm into the device driver itself, where it would run at the interrupt priority of the system with all other processes locked out. This idea can improve the statistics of the synchronization procedure by a significant factor, but it may have a much smaller impact on the application that is using the time stamps. In an extreme case, it can make things worse.

We assume that the reason the client system is going to the trouble of synchronizing its clock is so that it can apply accurate time stamps to some sort of event, such as the receipt of some signal or the processing of some data. The interpretation of these events and the decision to apply a time stamp to them is generally an application-layer process, and this process must compete with other processes for system resources in general and reading the system time in particular. If the system is heavily loaded, there is likely to be a delay between the instant the event actually is received and the time when the application decides to apply a time stamp to it and between when the application requests the system time for the time stamp and when it is actually received back in response to the system query. Since there is always some delay between when the application process requests a time from the system and when the time is returned, the time stamp used by the application process has a bias that varies with the load on the system but always has the same sign. That is, the time-stamp applied to the event is always somewhat later than the time at which the event actually occurred.

The procedure used in the Network Time Protocol requests the time twice on each measurement cycle—once when the query to the remote server is sent (t_1 in equations (1) and (2)) and a second time when the response is received (t_4 in equation (2)). If the synchronization algorithm is running as an application level process then it is also subject to these same delays described in the preceding paragraph, and both of these requests will be delayed by the system latency. Since this latency is inside the two-way loop that measures the transmission delay, they are measured and corrected for (at least in first order) by the delay-measuring algorithm. On

the other hand, if the synchronization algorithm is running in the kernel or as part of the device driver, then it is much less sensitive to these delays and so they are not estimated at all. Thus, although the statistics of the synchronization process may look better in this case because the apparent two-way measurement delay is smaller and more stable, the actual end-user accuracy is worse, since the user process and the synchronization algorithm are really seeing the system clock through very different channels with different measurement delays. This discrepancy becomes larger as the system load increases, which is exactly the situation in which moving the synchronization algorithm into the driver will seem to have the greatest benefit.

The preceding argument assumes that all application-layer processes on the client system experience the same system latency in an RMS sense, and there are situations where this is not accurate. For example, it would not necessarily be the appropriate description for a special-purpose system where the application that used the time stamps was a kernel-layer process or where this application was realized in dedicated hardware. Moving the synchronization process for the system clock closer to the interface that receives the time messages would be appropriate in this case, since both the application and the synchronization procedure would be accessing the clock with very similar latencies. This dedicated-controller configuration was one of the motivations for the IEEE 1588 protocol [8].

6. Summary and conclusions

I have discussed the design considerations that are used to transmit and receive time using dial-up telephone lines and the Internet to exchange the information. The algorithms are similar to those used to characterize atomic clock ensembles, but there are a number of important differences. The most important difference is that the channel that supports the measurement process makes a significant contribution to the overall noise of the measurement process. This measurement noise is usually not well characterized as a Gaussian random variable, so that many of the usual statistical tools are not appropriate. In addition, I have considered the fact that exchanging messages and implementing the two-way protocol to estimate the network delay requires resources in terms of network bandwidth and computer cycles. The algorithms used in these applications should therefore be designed to take the efficiency of the synchronization process (measured as the number of queries needed to realize a given timing accuracy) into account, and I have provided a framework for implementing these considerations.

The synchronization process is driven by a real-time estimate of the Allan deviation of the time-difference data that is performed by the client system. The calculation is typically done about once per day, and the algorithm adapts the interval between computations to the observed variation in the results.

The algorithms I have described are used to synchronize the ensemble of time servers operated by NIST using data from the ACTS telephone time service. The method can provide a timing accuracy of $\leqslant 0.001$ s measured at the server in most cases. The accuracy of a system that receives the time over the Internet is poorer than this because the network jitter is larger and less well characterized.

The overall accuracy of the synchronization process is determined by three factors: the stability of the server time, the stability of the user clock, and the statistical performance of the channel connecting them. The user sees the server through the network, and so the performance of the server clock is not important once it is significantly more stable than the network. Improving the stability of the user clock can improve the efficiency of the algorithm by allowing a longer interval between calibration cycles and by allowing longer periods of operation in 'holdover' mode when the connection to the server clock is lost. This increased polling interval will translate into improved *accuracy* if (and only if) the average asymmetry of the network delay is 0 over sufficiently long averaging times. This need not be true in general, and we discussed telephone circuits where it is almost certainly not true for any averaging time.

Finally, we note that increasing the polling interval to improve the cost/benefit ratio of the algorithm or to decrease the cost of the algorithm by relaxing the required synchronization accuracy has the down-side that it will take longer for the algorithm to respond to and correct for non-statistical events such as large time steps. This is a characteristic of all methods that use root-mean-square statistical estimators. The maximum error may happen only rarely, but its magnitude can be arbitrarily large.

The algorithms we have discussed can be used to realize an unbroken chain of measurements between a user system and the atomic clock ensemble maintained by NIST to realize UTC(NIST). I have also provided methods of evaluating the uncertainty of each link in the chain. The time service software and hardware therefore support full traceability of the time of a user system to UTC(NIST). The major uncertainty in this measurement chain is the estimate of the asymmetry of the network link between the user system and the NIST server. Our experience is that the impact of the actual asymmetry is smaller than the upper bound I have presented in equation (6). On the other hand, asymmetries on the order of a few per cent are quite commonly observed, so that there is an incentive to minimize the round-trip path delay. The network time servers operated by NIST are distributed as widely as possible for this reason.

References

[1] The NIST web page contains a list of the locations of the current servers. In most cases, locations that have multiple physical servers combine the systems using a load balancer and present a single network address to the public network. See http://tf.nist.gov/tf-cgi/servers.cgi

[2] Mills D L 2011 Network Time protocol, version 3, RFC 1305. See tools.ietf.org/pdf/rfc1305
See also Mills D L 2011 *Computer Network Time Synchronization* (New York: CRC Press)

[3] Levine J, Weiss M, Davis D D, Allan D W and Sullivan D B 1989 The NIST automated computer time service *J. Res. Natl Inst. Stand.* **94** 311–21

[4] Levine J 2008 Improvements to the NIST Network Time Servers *Metrologia* **45** S12–22

Lombardi M L 2002 NIST Time and Frequency Services, NIST, Boulder, Colorado *Special Publication 432*

[5] Levine J 1995 An algorithm to synchronize the time of a computer to universal time *IEEE/ACM Trans. Netw.* **3** 42–50

[6] Levine J 1999 Time synchronization over the Internet using an adaptive frequency-lock loop *IEEE Trans. Ultrason., Ferroelectr. Freq. Control* **46** 888–96

[7] Levine J 1998 Time synchronziation over the Internet using AUTOLOCK *Proc. IEEE. Int. Frequency Control Symp. (Los Angeles, CA)* pp 241–9 IEEE Catalog number 98CH36165

[8] IEEE 2002 Standard for a precision clock synchronization protocol for networked measurements and control systems, New York, IEEE Std. 1588-2002. The original standard was revised and issued as 1588-2008, which is often called 1588v2. The two versions are motivated by the same considerations, but are not compatible. The original version was intended primarily for network connections with stable and symmetric delays—where all of the nodes are on the same local-area network segment, for example. The revision relaxes this requirement and provides support for wide-area networks.