1998 IEEE INTERNATIONAL FREQUENCY CONTROL SYMPOSIUM

TIME SYNCHRONIZATION OVER THE INTERNET USING "AUTOLOCK"

Judah Levine
JILA and Time and Frequency Division
National Institute of Standards and Technology
and
University of Colorado
Boulder, Colorado 80303

Abstract

This paper describes the operation of an algorithm for synchronizing the time of computers using messages transmitted over packet-switched networks such as the Internet. The algorithm configures itself to realize any specified performance level at minimum cost (measured in computer cycles or network bandwidth). If the highest-possible accuracy is requested, the performance will be limited by the larger of the instability of the local clock oscillator or the noise in the measurement process between the client and the server. Lower accuracy can be realized at substantially lower cost. The algorithm uses a frequency-locked loop with unequal spacing between the calibration messages. It makes better use of scarce network bandwidth than previous methods. In addition, the algorithm is an improvement over the pure-FLL "Interlock" algorithm that I described previously because it is self-configuring. In addition to supporting an explicit trade-off between cost and accuracy, the algorithm provides better performance than previous methods because it is better able to adapt itself to fluctuations in the asymmetry of the network delay. This robustness can be realized without the preliminary tuning that was necessary to realize optimum performance using previous methods.

Introduction

This paper discusses a method for synchronizing the clock of a client computer using messages transmitted over the Internet from a remote server. The design principles would also be appropriate for other types of connections between the client and the server, provided only that the delay through the network connecting them is symmetrical on the average.

A number of other algorithms for synchronizing clocks in this environment have been published [1]-[5]. Each of these uses a different metric for evaluating the performance of the algorithm and for deciding what is meant by "optimum" operation. Some algorithms, for example, are designed to guarantee an upper bound on the absolute value of the time error of the client [3][4]. To realize this objective, they must make relatively frequent requests to servers that are close by (in a network delay sense), since the error due to an asymmetry in the network delay is bounded by one-half of the round-trip value. Supporting this density of servers may be quite expensive in practice, especially as the Internet becomes more crowded so that the average delay between any two points increases. Others algorithms place very strong emphasis on dealing with unreliable servers or with frequent large fluctuations in the network delay or in its asymmetry [2]. They realize these objectives by querying a number of servers on each calibration cycle, and by then choosing the "winner" based on various criteria.

All algorithms with these kinds of goals are relatively expensive from the point of view of the number of servers that they require to realize their design goals. The reason is that the *average* operating cost (measured in terms of the number of calibration requests that are generated by a client) is driven by the desire to limit the *maximum* time error or to detect server failures, which should be relatively rare events. While this machinery undoubtedly improves the worst-case performance of a system synchronized in these ways, it may have little impact on the RMS time accuracy of a typical client.

Our previous Internet-based algorithm [2] and the method that we describe here use the RMS accuracy of the clock in the client as the measure of performance, and they are designed to maximize the ratio of this RMS accuracy to the average cost (measured in terms of the load on the servers and the network). This optimization can be implemented either by finding the best performance that can be achieved at a fixed cost or by minimizing the cost for a given level of performance. As I will show below, the relationship between cost and performance is not linear – incremental improvements in

performance become increasingly expensive to achieve. Some levels of performance may not be achievable at any cost in the given environment because the frequency of the local clock is too unstable to support the amount of averaging that is required to cope with a noisy communications channel.

Synchronizing the local clock

All synchronization algorithms start from the same basic data: the measured time difference between the local machine and the distant server, and the network portion of the round-trip delay between the two systems. (Delays in the distant server are usually not a problem. Either they are small enough to be ignored or they are measured by the server and removed by the client.) These data are processed to develop a correction to the reading of the local clock. The usual approach is to use the measured time-difference after it has been corrected by subtracting one-half of the round trip delay. This model is based on the assumption that the transmission delay through the network is symmetrical, so that the one-way delay is one-half of the measured round-trip value. This corrected value may be used to discipline the local clock directly or it may be combined with similar data from other servers to detect outliers or to compute a weighted average time-difference which is then used to steer the local clock.

These steering corrections generally take two forms: time steps, which adjust the local clock by a fixed amount essentially instantaneously, and frequency steps, which adjust the effective frequency of the local clock oscillator and thereby slew the time relatively slowly. Frequency steps are usually realized by adjusting the size of the software "tick" – the amount added to the clock register in response to periodic interrupts from the physical clock oscillator. The frequency of the hardware oscillator itself is usually not accessible to software control.

Time steps, especially those that move the clock backwards in time, can cause unwanted side effects in other programs, and their use is usually limited to setting the clock during an initial start-up of the system. Frequency steps are the preferred means of adjustment, although they can have problems too. The minimum frequency adjustment that can be applied simply is to change the size of the tick by one least count. The resulting change in frequency is one least count per hardware interrupt, which is a fractional change of about 10^{-3} for most systems. This value is much too large. The actual frequency offset is likely to be one-tenth of this value or less, and the precision with which this offset can be determined is even greater.

Frequency steering generally requires a resolution of at least 5×10^{-8} if the granularity of steering corrections is to not degrade the frequency stability of the clock oscillator [1, fig. 1]. This level of frequency adjustment is usually implemented in two ways: either the kernel is modified to allow static frequency adjustments that are smaller than one least count per hardware interrupt to be specified, or the larger "standard" frequency offset is used periodically, with the ratio of the "on" interval to the "off" interval determined so that the average frequency offset corresponds to the estimate derived from the time-difference measurements. The first method has the advantage that the resulting time adjustment is smoother, but it requires access to the kernel source code. The second method is more general and can be applied to almost any system, but it results in a sawtooth-like variation in the time of the local clock with respect to the distant server. The amplitude of this sawtooth is a function of the system design, but it can be as large as a full tick (i.e., on the order of milliseconds) in some implementations. The first method is the best choice if it can be implemented in the client configuration, and it is the method that we use to keep our servers on time.

The maximum frequency offset that is required is usually on the order of s/day, which is a fractional frequency offset of about 5×10^{-5}. The dynamic range required in the frequency steering loop is thus about 1000:1. This requirement is usually not a problem, and almost any method that can provide the required resolution can also satisfy the dynamic-range requirement in steady-state operation. The maximum fractional frequency offset that is supported by the standard kernel of most systems is about 4×10^{-3}, so that it is not practical to remove time offsets of more than a second or so using this method because it would take too long. The clock is usually set initially using a single time-step for this reason.

Effects of measurement noise

Individual time-differences have a substantial uncertainty because of the noise in the measurement process itself. This noise arises from many sources – from fluctuations in the response time of the local operating system to interrupts, from jitter in the delay through the network or the interface hardware and from other hardware-related causes. Whatever the cause, this noise is not associated with the frequency of the clock oscillator. Using the time-differences themselves to set the local clock directly is therefore not a great idea -- no matter how the adjustment is performed. Doing this would convert the phase noise of the measurement process into frequency noise in the clock. The only reason for doing

this might be simplicity. A procedure that simply set the clock to the time received from the server would be simple to implement and would not have to run continuously as a background or "daemon" process. The price for this simplicity is that the performance of the local clock oscillator is degraded.

The contributions to the noise in the measurement process vary very rapidly with time, and consecutive measurements where the interval between them is large compared to the characteristic period of these fluctuations will be affected by noise that is almost completely uncorrelated both with the value on the previous measurement and with the underlying frequency of the clock oscillator. The spectrum of these variations is therefore approximately white, and averaging closely-spaced time-interval measurements results in an estimate that converges to the underlying mean value of the time difference, provided only that the measurements are made quickly enough so that the parameters of the oscillator have not changed.

It is usually easy to satisfy both of these time requirements. The characteristic period of the hardware is on the order of microseconds or less, so that even measurements made a few seconds apart are affected by very different noise environments. On the other hand, the time dispersion due to the fluctuations in the frequency of the oscillator is usually negligible on this time scale.

However, the averaging process cannot be continued indefinitely. It will only do the "right" thing as long as the spectrum of the fluctuations in the data can be characterized as predominantly white phase noise. The range of averaging times over which this is true is usually pretty small for typical computer clocks, and the non-stationary statistics of the delay in a typical Internet path further restricts the averaging times over which this is an appropriate strategy. The result is that algorithms that develop a correction based on the average measured time-difference are likely to be useful only in local-area networks and with a relatively short interval between calibration cycles (on the order of 1000 s or less).

It is not always easy to detect the effects of too much averaging, that is averaging for a time interval that exceeds the domain in which the noise is dominated by white phase noise. The average time difference still exists in this domain, but it is no longer a stationary quantity. The synchronization loop appears to be operating normally, but it is fact increasingly dominated by flicker (and, at sufficiently long averaging times by random-walk) processes [1, fig. 9].

Algorithms that stabilize the frequency of the local clock (as opposed to its time) have an easier job in principle. These algorithms can operate at much longer averaging times where white phase noise is no longer the main problem, and the performance is limited by the frequency stability of the local clock. It turns out that clock oscillators in many computer workstations are surprisingly good. Many of them have Allan deviations of less than 10^{-7} at averaging times of 10^4 s, so that they have a free-running time stability of better than 1 ms RMS [1, figs. 1 and 3]. One way of realizing this stability is to average the individual time-differences for a time that is well within the domain where white phase noise dominates the noise spectrum, and to then switch to averaging the frequency (that is, the first difference of these time differences) as long as the noise process can still be characterized by white frequency noise. For the oscillators typically found in computer hardware, this extends the averaging interval to about 15 000 s. The details of the design will depend on the stability of the local oscillator and on the noise in the network link to the server. However, a design configured for the highest-possible accuracy would involve averaging time difference measurements over a period of a few seconds combined with averaging the frequency for a period of a few hours. This strategy is *not* equivalent to using the same number of measurements that are equally spaced in time, even though the average load on the servers would be the same in both cases [2].

Frequency-locked loops are not better or more accurate than those based on phase-lock techniques -- they are advantageous because they are almost always cheaper to operate for essentially the same performance. This is true because the high level of measurement noise over the Internet means that the local clock is usually more stable at short times than the distant server seen through the noisy Internet. A frequency-locked loop can better exploit this stability by longer averaging, and therefore can be configured with longer intervals between the calibration cycles than the corresponding phase lock. Although neither design is *a priori* more sensitive to glitches, the longer interval between calibration cycles in a frequency loop and its correspondingly longer time constants, means that a glitch is likely to persist for a longer time in that design before it is detected and removed.

Details of the method

The algorithm design is based on the principle of separation of variance -- that is that it is possible to separate the contributions of the clock and the measurement process using statistical techniques. A second implicit assumption is that the variances of both the measurement process and the clock frequency can be modeled using stochastic parameters. These turn out to be good approximations to the performance of real hardware

over a wide range of operating parameters, although the second assumption tends to break down at averaging times approaching one day (see below). In order to achieve this separation, the program evaluates the following two statistics after each calibration cycle has been completed:

S-1. The standard deviation of a group of closely spaced time-difference measurements that are made quickly enough so that the parameters of the local clock have not changed significantly while they are being made. As we discussed above, this requirement is easily satisfied if the ensemble of measurements is completed within a few seconds.

S-2. The error in predicting the currently observed average time-difference using the previously measured value and the estimated frequency offset. This prediction is done using a simple linear relationship:

$$\hat{x}_i = x_{i-1} + \hat{y}_{i-1}\tau, \qquad (1)$$

where x_i is the time-difference measured at time t_i, y_i is the frequency difference between the local clock and the server estimated at the same time and τ is the time interval from t_{i-1} to t_i. We use equation (1) to predict the time difference; the error in this prediction is the difference between the measurement and the prediction. It is given by

$$\varepsilon_i = \left| x_i - \hat{x}_i \right|. \qquad (2)$$

Both statistics are maintained in two versions: the value determined in the current calibration cycle, and a sliding average over the most recent 12 hours or 3 calibration cycles, whichever is longer.

The first statistic is sensitive primarily to the phase noise in the measurement process. The program first compares the average value of this parameter with the desired level of performance that was specified by the operator when the software was installed. The number of measurements in each group is adjusted once per day until the two are roughly equal -- the size of the group is increased if the RMS of the mean is larger than the requested accuracy and decreased if the mean is much better than necessary. If we assume that the measurement noise is approximately white phase noise, the RMS value of the mean of a group of measurements varies as the square root of the size of the group. The specified performance level therefore has a quadratic effect on the cost of the algorithm. In the limit, it may not be possible to achieve the desired level of performance with a reasonably sized group (less than 25 or 50 members). The program defaults to a specified maximum group size in this case. Likewise, the program will never decrease the size of a group below a defined minimum size (usually 3) no matter how small the RMS becomes.

The RMS of the group of time differences sets an upper bound to the RMS performance of the synchronization loop at longer averaging times. No matter how stable its frequency is, on the average the local clock cannot be set more accurately than this RMS dispersion. This RMS value may be as small as 75 - 100 μs when the server and the client are on the same network, but is more typically on the order of milliseconds for a continental-length path.

If the standard deviation of the current group of time differences is much larger than the average, the most likely reason is that it is due to jitter in the symmetry of the network delay during the time that the group of measurements was being made. Although the measurements are made rapidly enough that the parameters of the local and remote clocks have not changed, the same cannot be said of the network (The magnitude of the delay is not a problem, since it is measured on each cycle -- only its asymmetry causes trouble.) The algorithm will drop a single member of the group as an outlier if doing so will reduce the standard deviation to a more reasonable value, and will try to repeat the entire group if dropping a single value will not fix the problem. The assumption that underlies this procedure is that asymmetries in the network delay are transient effects. Small asymmetries will be averaged by multiple measurements, while large ones will be short-lived and detected as outliers.

This method will fail if the path from the client to the server has a static asymmetry, which will bias the results but will not contribute to the variance of the group of measurements. The offset due to this asymmetry is bounded by one-half of the round-trip delay, so that *at the instant of synchronization*, the time difference between the client and the server is guaranteed to be less than the sum of the measured time difference and one-half of the round-trip delay. This may be good enough for some purposes, but, as we will show below, this guarantee becomes weaker with time because of stochastic frequency variations in the clock oscillator.

It is sometimes possible to detect a static asymmetry by requesting timing data from a second server, but the two data sets are likely to disagree consistently in this case, and it may not be possible to decide where the problem lies. In particular, if the local clock is synchronized initially to a server via a path that has a static asymmetry, then data from a second server received

via a more symmetric path may be rejected as coming from a machine that is thought to be broken.

Under normal, steady-state operating conditions, the magnitude of the second statistic is sensitive primarily to the frequency stability of the local clock, and its average value is proportional to the value of the Allan deviation for an averaging time equal to the time interval between calibration cycles. This magnitude is affected by two parameters -- the time interval between calibrations and the time constant of the frequency update loop [1, eq. 3].

The time constant of the frequency update loop represents the optimum averaging time for the frequency estimates. Averaging for a shorter time interval does not optimally attenuate the white frequency noise of the oscillator. Averaging for a longer time results in an average which exists in a formal sense but is no longer stationary because the procedure has entered the flicker and random-walk portions of the spectrum. This time constant is primarily a function of the design of the oscillator, and it needs to be determined only once when the algorithm is started for the first time. It is measured as part of the cold-start portion of the algorithm by evaluating the Allan deviation as a function of lag when the local clock is free running. This parameter is usually on the order of 10 000 s – 20 000 s, so that this initial evaluation process to determine it usually takes somewhat less than 1 day. (In order to have confidence in the Allan deviation estimates, it is usual to compute the deviation using a time series that is at least 3 times longer than the longest lag that is of interest.)

Once the algorithm is running in steady state, the average value of the second statistic and the desired performance level are used to adjust the interval between calibration cycles. If the average prediction error is much larger than the measurement noise (i. e., S-2 >> S-1) then the prediction error is due to stochastic frequency fluctuations in the local oscillator. If the prediction error is larger than the desired level of performance, then the time between calibration cycles is too long -- the frequency stability of the oscillator is not good enough to support free running for that time interval and more frequent calibrations are necessary. If the prediction error is not larger than the desired level of performance, then there is no need to have more frequent calibrations (even though the performance would be improved by doing so).

If the prediction error is much smaller than the measurement noise, then the calibration interval is unambiguously too small -- the frequency stability of the oscillator would support the same level of stability with less frequent calibrations and the extra ones are essentially being wasted. The interval between calibrations can be increased in this case with minimal impact on the performance of the loop. The only reason for not doing so would be a concern about outlier detection, and the user can specify an absolute maximum interval between calibrations if this is a concern.

The relationship between the performance of the loop and the interval between calibrations can be calculated if the noise type is known. This relationship tends to make predictions that are too optimistic when the averaging time exceeds a few hours – the performance tends to degrade faster than predicted as the time between calibration cycles is increased in that domain. There are two reasons for this. The first is that when the interval between calibration cycles becomes longer than the time constant of the frequency update loop, then that time constant is effectively forced to have a value that is larger than optimum. In other words, the distribution of the prediction errors calculated using eq. 2 above become increasingly dominated by flicker and random-walk processes and these errors are effectively fed back into the synchronization loop. The second reason is that at longer periods the frequency of the oscillator begins to be driven by non-stochastic effects such as fluctuations in the ambient temperature. These fluctuations are in addition to the increase in the stochastic level of the frequency variations. Temperature fluctuations in a typical office environment may pull the frequency of the oscillator by up to 1 ppm with a period ranging from several hours to 1 day. Those frequency variations are larger than what would be predicted based on a smooth extrapolation of the Allan deviation from its value at shorter periods.

Parameter update

If the statistical comparisons indicate that the current time differences are consistent with the past performance, then the current data are used to update the parameters of the procedure. These include the average values of both statistics and the average frequency of the oscillator. This average frequency is used to steer the local clock using one of the methods described above – either it is transmitted directly to the kernel or it is used to schedule small periodic time adjustments which realize the computed offset frequency on the average. The details of these adjustment procedures are more fully discussed in [1].

Error detection

A comparison of the values for each of the two statistics found on the current cycle with the corresponding average values can be used to detect many kinds of problems. We have already discussed this point

above with regard to the standard deviation of the group of time-difference measurements, which can be used to detect jitter in the symmetry of the network delay. Likewise, this standard deviation is a measure of how well closely spaced time-differences received from two different servers should be expected to agree.

If the tests based on the first statistic are okay, but the prediction error on this cycle is much larger than the value expected based on the average of this parameter, the software attempts to determine the cause of this apparent change in performance. (This test is significant only if the algorithm is operating in steady-state. If, based on the procedures outlined above, the interval between calibrations has just been increased, then an increase in the prediction error is to be expected, and a comparison with the average value is not appropriate until the average value comes to equilibrium at this increased time interval.) The software conducts a number of tests in the following order:

1. The simplest possibility is that the change is due to a temporary change in the symmetry of the network delay (because of congestion, for example). The program repeats the entire measurement protocol several times to see if the prediction error is improved on subsequent attempts. These repeated measurements are made to the same server. This procedure removes the discrepancy about 60% of the time after a single retry and about 80% of the time after a second one.

2. If the error persists, the program queries an alternate server and uses majority voting to decide if the problem is in the first server or in the local clock. (A network problem that is common to both servers will be considered -- incorrectly -- as a problem in the local clock). Note that the time of the local clock contributes to this comparison based on its expected stability as estimated by its Allan deviation. If the two servers agree (to within the phase noise of the measurement process as estimated by S-1), then the local clock has experienced a change in its characteristics. The response to this conclusion is outlined below. If the data from the second server indicate that the first server is broken (or its path delay is significantly asymmetric), then the two interchange roles, and the alternate server becomes the primary one for the next 24 hours. The program tries to switch back to the first server at that point, and does so if its data satisfy the tests described above.

3. If the problem cannot be resolved in step 2 (i.e., if the time-differences between both servers are not consistent with the expected prediction error and if the data from the two servers are not mutually consistent either) then either the problem cannot be assigned to one cause or our expectations based on the previous average performance are too optimistic. The operating parameters are not modified and the program waits a short time and tries again. This can happen if the network is very congested or if the path to one or both of the servers has a significant static asymmetry. The program will use a third server in an analogous manner if one is defined. If only two servers are specified (of if data from a third server fails to resolve the question), the program increases the average variances and goes into "holdover" mode. The holdover performance preserves the last estimate of the frequency offset of the local clock and continues to steer it in frequency based on that estimate. It makes no changes to its time. The clock accuracy will be limited by the flicker and random-walk frequency fluctuations of its oscillator. The time dispersion due to these fluctuations depends on the quality of the oscillator; typical values for the Allan deviation of the frequency are on the order of 10^{-6} at 1 day, which would result in a time dispersion of 50 - 100 ms. If the problem is due to a transient degradation of the network, then it will probably disappear on the next cycle; if it is due to a more permanent degradation of its characteristics then the synchronization procedure will slowly learn this by repeated increases in the average variances which form the basis for all of the tests that we have described.

The threshold for deciding that the current estimate of a statistic is "much larger" than the average value is set based on the usual compromise between the need to maintain adequate sensitivity to problems while not being overly sensitive to the statistical fluctuations in the amplitude of the noise. We have used a threshold of three times the standard deviation in all of our experiments. If the underlying statistics of the processes were Gaussian, a threshold of three times the standard deviation should cause the measurements to trigger these error tests about 1% of the time in normal operation. In fact, the data are not Gaussian, and the tests are triggered in about 10% of the calibration cycles. These "outliers" are often quite large so that the algorithm is not sensitive to the exact value used for the threshold.

Error modeling

The most common cause of an error in older systems was a time-step caused by a lost interrupt. These were easy to detect because the prediction error was very close to an integral number of systems ticks. This problem is usually not important in newer systems, where a large prediction error is usually due to a significant secular change in the frequency of the clock oscillator. As we pointed out above, the usual cause of this frequency drift is a change in the ambient temperature, although a change in line voltage may also contribute to a lesser extent. The

relationship between ambient temperature and oscillator frequency is complicated, because it depends both on the temperature and on its spatial gradient. Attempting to correct the oscillator frequency based on temperature measurements is too complex for most installations and is not a very profitable strategy in any case. The best strategy is to limit both the time constant for the frequency update loop and the maximum interval between calibration cycles so that these temperature-induced frequency variations are modeled as deterministic effects by the frequency estimator and removed. In other words, the parameters of the loop must be short enough so that these temperature-induced effects are treated as real frequency changes, rather than as stochastic effects that should be averaged without appreciably changing the loop parameters.

Synchronization experiment

In order to test these ideas, we conducted two synchronization experiments. The first was designed to synchronize the local clock as accurately as possible and to find out what accuracy could be achieved and how expensive it would be to realize it. The second experiment was designed to synchronize the same clock so that it was correct only to the nearest second using the same servers and network path as in the first experiment. The goal was to see how much this relaxation in accuracy would save in the operating costs.

In both experiments we synchronized the clock of a computer which was located in Boulder, Colorado to a server on the West Coast in Redmond, Washington and one on the East Coast in Gaithersburg, Maryland. The time of the local clock was also measured using diagnostics derived from other systems in Boulder, but these diagnostics were not used to discipline the clock. The server in Redmond was the primary one; the one on the East Coast was only used as part of the error evaluation procedure as outlined above.

The details of the paths between our client in Boulder and our two servers were not under our control. Although the path to Gaithersburg, Maryland had a delay that was always very nearly symmetrical, the path from Boulder to Redmond was often 42 ms longer than the return path (because it was routed through Atlanta, Georgia). When this was true, the two servers seemed to have a time offset of 21 ms – an offset that was large compared to the time dispersion due to all other causes for averaging times up to several hours. As we mentioned above, static asymmetries of this type sometimes cannot be unambiguously removed by any statistical procedure. Our solution was to add a third server located in Reston, Virginia, and to use its data in a simple extension of the majority-voting algorithm described above. The result was to exclude the server in Redmond whenever its delay became asymmetric and to make the server in Gaithersburg the primary one with the server in Reston as its backup when needed. Even when three servers are specified in the configuration file, only one is actually used most of the time – the others are used only to resolve ambiguities when a large prediction error indicates a possible problem or a change in the parameters of the system.

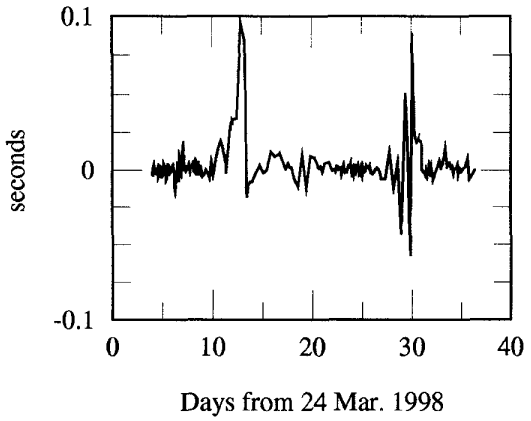

Fig. 1. The time difference in seconds between the local clock and UTC.

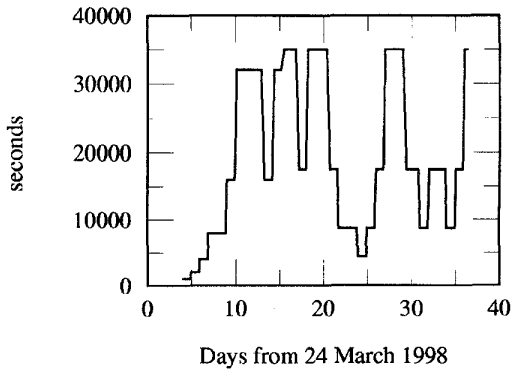

Fig. 2. The interval between calibration cycles as a function of epoch for the first experiment.

Fig. 1 shows the measured time difference between the local clock and UTC for the first experiment, and fig. 2 shows the interval between calibration cycles for the same period. Note that the interval between calibrations is gradually increased during the first week of operation without significantly affecting the accuracy of the clock.

This situation changes abruptly when the interval reaches 32 000 s – an interval comparable to the length of the working day -- because both the local temperature and the network delay have deterministic fluctuations near this period. Although it does not happen immediately, one of these fluctuations eventually exceeds the threshold and the algorithm is forced to decrease the time between calibration cycles to maintain the desired performance level The same story is repeated on a smaller scale several more times until about day 28 – a day which had both a large change in temperature followed by the failure of a network element when the same kind of problem is repeated. The RMS accuracy of the entire data set is about 30 ms, but most of this is due to the two large events – the RMS of the data excluding these events is about 10 ms.

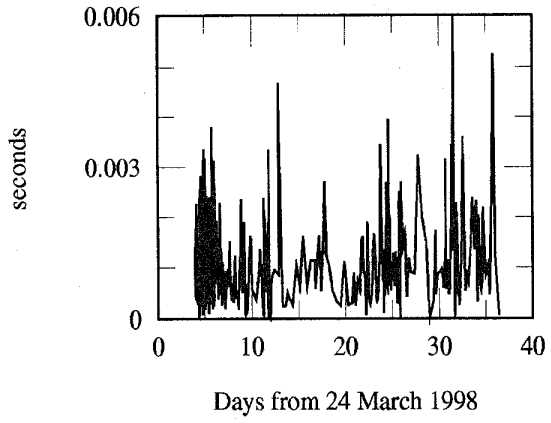

Fig. 3. The RMS spread of a group of five time difference measurements made over a very short time interval.

The large fluctuations shown in fig. 1 are primarily due to large and slowly varying network asymmetries or glitches in the local clock itself. As shown in fig.3, they are not due to more rapid jitter in the network asymmetry because they are not visible in the RMS spread of consecutive measurements (S-1). Based on fig. 3, the synchronization loop should have been able to synchronize the time of the clock with an uncertainty of 1-2 ms RMS; the actual loop achieved this on occasion, but the overall performance (even excluding the glitches) was not that good. The reason is that the interval between calibrations was allowed to grow to values larger than the optimum time constant for the frequency averaging loop. The best performance would have been realized if we had clamped the interval between calibrations to a value less than 12 000 s – the measured optimal time constant for the frequency loop. Note that the interval between calibrations was almost always larger than this value.

Fig. 4 shows the average frequency offset of the clock as estimated by the algorithm. Diurnal effects are visible on occasion, but they are smaller than the random-walk frequency noise, which dominates the noise spectrum at this period. These frequency fluctuations have an amplitude of about 3×10^{-6} peak to peak with an irregular period of about 1 week, and they can be used to suggest an initial configuration for a synchronization loop designed to keep the clock on the machine accurate only to the nearest second.

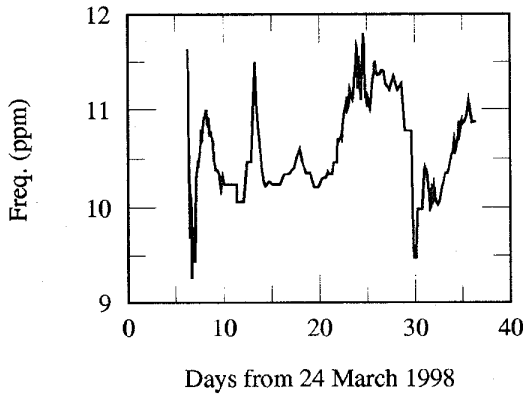

Fig. 4. The frequency offset of the local clock as estimated by the algorithm. The diurnal temperature-induced effects are masked by the random-walk frequency fluctuations.

If a static offset of about 10.5×10^{-6} were applied to the clock oscillator of this system and if the machine was then left free-running with no other updates, then the time-dispersion over the one-month period shown in fig. 4 would have been somewhat less than 1 s RMS. This result emphasizes the fact that keeping the clock correct to the nearest second requires very little external information – although the clock in the system we were using would gain almost 1 s/day if it were run with no correction at all, this frequency offset is very stable and can be predicted with considerable accuracy. In practice, the interval between calibration cycles would be set more by our concern that we detect a glitch relatively promptly than by the need to adjust the steering of the clock when it is operating normally.

We tested these ideas by configuring the algorithm to achieve an accuracy of 1 s RMS, using the same servers as in the previous experiment. We set the initial interval between calibration cycles to 32 000 s, and we specified that the interval should not exceed 200 000 s. The time constant for the frequency update loop was left at 12 000

s; since the minimum interval was larger than this value, the effective time constant was three sample times as discussed above.

It took just over 10 days for the interval between calibrations to reach the maximum specified value. The RMS time difference increases with the interval between calibrations, but has not yet reached the specified accuracy of 1 s RMS even with an interval between calibrations of 200 000 s (see Fig. 5).

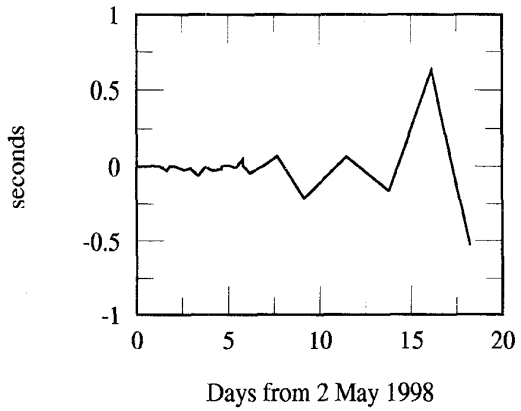

Fig. 5. The time difference in seconds between the local clock and UTC when the algorithm is configured to realize an accuracy of 1 s RMS.

The highest accuracy with this configuration would have been achieved with an interval between calibrations of about 4 000s, since that would have resulted in 3 calibration cycles in a time equal to the time constant of the frequency update loop. The performance shown in this figure is about 50 times less expensive in terms of network bandwidth and server load.

Conclusions

We have designed an algorithm for synchronizing the clock of a computer using messages transmitted over the Internet. The algorithm uses the network and message formats of the Network Time Protocol [3], and can communicate with conventional unmodified NTP servers.

The algorithm is based on our previous frequency locked designs, but this version automatically adapts to changes in the symmetry of the network delay. It also automatically adjusts its operating parameters so as to realize a specified level of performance. The adjustment algorithm uses the ratio of the RMS performance to the average cost as the metric for choosing the optimal operating conditions.

In addition to synchronizing the local clock, the algorithm has a number of other diagnostic modes – it can be used to evaluate the Allan deviation of the freely-running local clock or to estimate the jitter in the delay in a local network. These evaluations use the same machinery as the synchronization loop, except that the local clock is not steered in these modes.

We have conducted two experiments to illustrate the capabilities of the procedure. The first experiment was designed to synchronize a clock as accurately as possible; it achieved an accuracy of 30 ms RMS, and would be capable of achieving an accuracy of better than 10 ms RMS if its configuration were changed as described in the text. This performance was realized using servers that were more than 2000 km away, and neither the servers nor the paths to them were conditioned in any way for this work.

The second experiment was designed to synchronize a clock so that it was correct to the nearest second – a level which may be adequate for many users. The maximum interval between calibrations was set to 200 000s, and the performance of the loop exceeded the specified performance level even at the longest permitted interval between calibrations.

Acknowledgements

This work is supported in part by the National Science Foundation through grants NCR-9115055 and NCR-9416663 to the University of Colorado. We gratefully acknowledge this support.

References

1. Levine, Judah (1995), "An algorithm to synchronize the time of a computer to universal time," *IEEE/ACM Trans. Networking*, Vol. 3. pp. 42-50.
2. Levine, Judah (1998), "Time synchronization using the Internet," *IEEE Trans. Ultrasonics, Ferroelectrics and Freq. Cntrl.*, vol. 45, pp. 450-460.
3. Mills, David L. (1992), "Network time protocol (version 3); specification, implementation and analysis: *DARPA Network Working Group Rep. RFC-1305*, Newark, Delaware, Univ. Delaware.
4. Arvind, K., (1994), "Probabilistic clock synchronization in distributed systems," *IEEE Trans. Parallel Distrib. Syst.*, vol 5, pp. 474-487.
5. Cristian F. (1989), "A probabilistic approach to distributed clock synchronization," *Distrib. Comput.*, vol. 3., pp. 146-158.