Timing in Telecommunications Networks

Judah Levine

Time and Frequency Division and JILA
National Institute of Standards and Technology and University of Colorado
Mail Stop 847
325 Broadway
Boulder, Colorado 80305, USA
e-mail: jlevine@boulder.nist.gov

1. 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.

The work was motivated by two observations: (1) The load on the network time servers operated by the National Institute of Standards and Technology (NIST) has been increasing at an average rate of 5% per month for several years with no sign that this rate of increase is slowing and (2) there are many indications that a significant fraction (perhaps even a large majority) of the users of NIST digital time and frequency services do not need the millisecond-level accuracy that these services can support. I therefore set out to design an algorithm that would support an *explicit* tradeoff between the synchronization accuracy that could be achieved using the procedure and the cost of realizing it, where the cost is proportional to the average number of calibration requests generated by each client system. The relationship between accuracy and cost is approximately quadratic, so that a modest relaxation in the accuracy requirement (to 100 ms RMS, for example) would be accompanied by a decrease of more than two orders of magnitude in the average cost of realizing it.

A number of other algorithms for synchronizing clocks in this environment have been published [1]-[6]. 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 [4][5]. 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 [1][3]. They realize these objectives by querying a number of servers on each calibration cycle, and by then choosing the "winner" based on various criteria.

We argue that 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 [3] and the method that we describe here use the RMS time offset of the clock in the client as the measure of performance, and they are designed to maximize the ratio of this RMS performance 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 we 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.

2. 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.

3. Effects of measurement noise

Individual time-difference measurements are likely to 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 contributions to the noise in the measurement process vary very rapidly with time, and consecutive measurements where the interval between them is long compared to characteristic period of the variations 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.

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 a 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). This argument is particularly relevant to algorithms which use a pure phase-lock loop to discipline the local clock [4], and it explains why algorithms based on frequency loops tend to achieve comparable synchronization at much lower cost [2][3].

Algorithms that operate by stabilizing 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. 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 [3]. The time between calibration cycles is often called the *poll interval*, and the argument presented here suggests that the average poll interval does not completely specify the performance capability of the procedure – that grouping the calibration requests makes a better use of the available resources than simply uniformly spreading them out.

4. 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. In order to achieve this separation, the program evaluates the following two statistics after each calibration cycle has been completed:

S-1. Compute 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:

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

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 the prediction on this cycle is the difference between the predicted and measured values:

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

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. Assuming 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 (in time) more accurately than this RMS value, and the subsequent algorithm can do no better than "remember" this adjustment without further degradation. 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. This limit is independent of the details of the synchronization procedure.

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 (i.e., one that is consistent with the average over the previous observations), 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.

5. Discussion and conclusion

The algorithm we have described is most useful for client systems that have only modest accuracy requirements – on the order of 0.5 s or greater, since the interval between requests needed to realize this accuracy is much longer than the default used by many current implementations (on the order of minutes). This is an important result in increasing the ability of the primary NIST servers to handle user requests without a corresponding increase in the hardware needed to satisfy them.

6. References

1. Mills, D. L., "Improved algorithms for synchronizing computer network clocks," IEEE/ACM Trans. Networking, 3, Vol. 3, pp. 245-254, 1995.
2. Levine, Judah "An algorithm to synchronize the time of a computer to universal time," IEEE/ACM Trans. Networking, Vol. 3, pp. 42-50, 1995.
3. Levine, Judah, "Time synchronization using the Internet," IEEE Trans. Ultrasonics, Ferroelectrics and Freq. Control., vol. 45, pp. 450-460, 1998.
4. Mills, David L., "Network time protocol (version 3); specification, implementation and analysis,: DARPA Network Working Group Rep. RFC-1305, Newark, Delaware, Univ. Delaware, 1992.
5. Arvind, K., "Probabilistic clock synchronization in distributed systems," IEEE Trans. Parallel Distributed Syst., vol. 5, pp. 474-487, 1994.
6. Cristian, F., "A probabilistic approach to distributed clock synchronization," Distributed Computing, vol. 3, pp. 146-158, 1989.