Time Synchronization in a Local Area Network

Article *in* IEEE Control Systems Magazine · May 2004

DOI: 10.1109/MCS.2004.1275432 · Source: IEEE Xplore

Precise Time Synchronization across a Local Area Network

By Svein Johannessen

In this article we examine the problem of synchronizing the time-of-day clock in one node of an automation network system with a reference clock. The emphasis of this article is on switched, highly loaded networks, where unpredictable delays introduce excessive synchronization noise.

Time Stamping

Time stamping is the association of a data set with a time value. In this context, time may also include the date. Why is time stamp desirable? The closest example may be on your own PC. Whenever you create or save a document, the document is automatically assigned a date-and-time value. This value can be used to locate documents created on a certain date (last Monday), documents created within a certain time span (last half of 1998), and the order in which a set of documents was created (the e-mails in your inbox).

By looking at the examples above, it can be seen that the accuracy needed for the time stamping is about the same as we expect from a wristwatch. This comparison suggests an accuracy within a couple minutes, but as long as the clock does not stop, it does not really matter how precise it is.

Synchronizing Watches

Now that we know about PC time stamping, the next step is to connect the PC to a network, perhaps to the Internet, and start exchanging documents and e-mails. What happens if the clock in your PC, that is, the clock used for time stamping, is wrong by a significant amount?

If you have an e-mail correspondence with someone, their reply, which is time stamped at the other end, might appear to be written before the question, which is time stamped at your end. Alternately, if you collaborate on documents, identifying the latest version may be problematic.

The PC clocks are accurate enough when connected within a network, but a new requirement is for them to be *synchronized*, which means that they should show the same time at the same instant. We could go around to each PC, look at our wristwatch and set the PC clock to agree with it. However, this is a tedious job, and we desire a better solution.

One solution is to elect one PC to be the time reference, which means that every other PC could get the current time from it at least once a day, and set its own clock to agree with the reference time. This solution works satisfactorily on a local area network (LAN), but all PC clocks will lag the time reference by the time it takes a clock value to travel from the time reference to the synchronizing PC. This lag, which is generally less than one second, is good enough for office purposes.

Enter the Internet. Suddenly the synchronization problem escalates, since two collaborating PCs may be located in different time zones (remember to compensate for that) and a synchronization message may take a long time to travel from one PC to the other. Fortunately, the Internet Network Time Protocol has a solution to both problems. This protocol involves sending a time-stamped time request message to a time server. This time server adds an arrival time stamp and a retransmit time stamp before returning the request message to the requesting PC. The requesting PC time stamps the message when it returns and uses all of the time stamps to calculate the correct time. This protocol and its little sibling, the Simple Network Time Protocol, are able to synchronize computers across the Internet with a precision of only a few milliseconds.

Why is Clock Synchronization Important?

Consider two measurement nodes in a control system and assume that both nodes are connected to a central controller through a network. Both measurements are sampled at a precisely defined rate. Ideally, the central controller receives both measurements at the same time, but the vagaries of network communication often make that impractical. Basic automation controllers therefore time-stamp the measurements as they arrive, assuming that the difference between the actual sampling time and the arrival time stamp is negligible (for a treatise on the influence of network jitter on time series measurements, see [1]). Some applications, however, demand a more precise time stamp, which means that the measurements must be time-stamped at the measurement nodes.

Basic time-stamping is effective as long as the measurements at the two nodes are uncorrelated. If correlation between the two measurements is important, the clocks located at the two nodes must be synchronized in the sense that their difference must be below a given limit. This situation leads to the classical time-synchronization setup with one Time Server, which represents true time with respect to the network, and time clients, which strive to remain synchronized with the Time Server.

Time Stamping in Substation Automation

In the energy distribution world, a *substation* is an installation where energy is combined, split, or transformed. A substation-automation (SA) system refers to tasks that must be performed to control, monitor, and protect the primary equipment of the substation and its associated feeders. In addition, the SA system has administrative duties such as configuration, communication management, and software management.

Several tasks within SA need precise time synchronization, the most obvious being the phase difference measurement between two separate nets. In extreme cases, the measurement node may be kilometers apart, which means that the speed of light introduces a significant delay in the synchronization process; on a standard cable, the signal propagation speed is about 200 m/µs. Due to the extreme energy levels involved, SA synchronization requirements can be as challenging as ±1 µs. Therefore, we must compensate for the delay due to the speed of light whenever distances are greater than 200m!

The Network Time Protocol

The most prominent time-synchronization method is the Network Time Protocol (NTP) proposed by Mills and the Internet Engineering Task Force (IETF) group [1]. A subset of NTP, called the Simple Network Time Protocol (SNTP) [3], is protocol-compatible with NTP. The intended use of NTP is to synchronize computer clocks in the global Internet. NTP relies on sophisticated mechanisms to access national time, organize time-synchronization subnets, and adjust the local clock in each participating peer. SNTP, on the other hand, does not implement the full set of NTP algorithms, but rather focuses on simpler synchronization objectives, such as those in a local control system network.

These protocols were developed for the office automation field, where a precision of several milliseconds is sufficient. The one exception is in [4], which describes engineered refinements in the form of a modified driver and Unix kernel code for improving the time-stamping accuracy within SNTP time clients and servers. The adaptations of [3] achieve a timekeeping precision of a few hundred microseconds for an Ethernet network of workstations. Later in this article we discuss a technique for increasing the timekeeping precision to the order of one microsecond. Based on [5], we use the precision class terminology P1 (1 ms), P2 (0.1 ms), P3 (±25 µs), P4 (±4 µs), and P5 (±1 µs) to refer to timekeeping precision.

We restrict our analysis to a network based on *Switched Fast Ethernet*, since most Ethernet-based fieldbuses, such as PROFInet, Fieldbus Foundation HSE, Ethernet/IP, and Modbus/TCP use this standard as a basic packet-transport mechanism.

A New Time-Synchronization Standard

In 2002, IEEE ratified standard 1588 for time synchronization across Local Area Networks. IEEE 1588 is based on a protocol invented by Hewlett-Packard (now Agilent). In short, IEEE 1588 is based on one node transmitting a time-synchronization message, followed by another time-synchronization message containing the precise time of the previous time message. Although this protocol fulfills class P5 requirements on some types of local area networks, it is just as susceptible to random switch delays as the Network Time Protocol. The analysis below is thus valid for both IEEE 1588 and NTP.

Stating the General Synchronization Problem

Consider a time-server clock and a time-client clock. Let T denote the time measured at the time-server clock, and let t denote the time measured at the time-client clock. The general synchronizaton problem involves determining constants A and B such that $T = At + B$ within a specified accuracy and for a specified time interval.

Observations and Reformulations

The time-client clock is derived from a crystal oscillator with an absolute accuracy of better than 100 ppm. This accuracy includes initial accuracy as well as temperature and aging effects, implying that the constant A in the general synchronization problem is close to unity. The time-synchronization equation can therefore be rewritten as

(1) $T = (1 + a)t + B$,

where the *time-client clock frequency offset a* is the deviation between the time-client crystal oscillator output and the time-server reference oscillator, and B is the *absolute time offset*. For crystal accuracy, we have $|a| < 10^{-4}$. Since all microprocessors have a limited numerical precision, a numerically better version of (1) is

(2) $T - t = at + B$.

Our goal is to determine a and B such that the formula above is optimally satisfied for a given set of time measurements.

Mapping SNTP Time Stamps

The SNTP protocol depends on a time request sent from the time client to the time server and then returned from the time server to the time client. This message contains three time stamps: t_1, the time-client time the message was sent by the time client; T_2, the time-server time the message was received by the time server; and T_3, the time-server time the message was returned by the time server. An additional time stamp is t_4, the time-client time the message was received by the time client.

By incorporating the cable delay in the constant B, every SNTP measurement set would ideally satisfy the equation

(3) $T_2 - t_1 = at_1 + B$.

Observe that if an Ethernet switch is inserted between the time server and the time client or if a real-time operating system (RTOS) is implemented in either the server or client, then (3) should be modified to read

(4) $T_2 - t_1 = at_1 + B + \rho(t_1)$,

where $\rho(t)$ is a random delay function. The function $\rho(t)$ consists of a constant part, which is network dependent, and a delay distribution, which depends on the network load, the RTOS granularity, and the speed of the node CPU. Since $\rho(\tau)\geq 0$, its mean value over a set of measurements will not be 0, but rather a positive mean delay.

Restating the Synchronization Problem

A complete solution to the high-precision time-synchronization problem must reduce the randomness associated with the RTOS. This reduction can be accomplished by precise time-stamping at the hardware or driver level. In addition, it is important to verify that low-level time-stamp information is passed correctly to the time-synchronization (application) level. Finally, it is necessary to remove the influence of ρ(t) from the calculation of the constants *a* and *B*.

The Random Delay Problem

Size of the Problem

In [6], several sets of measurements were made of the end-to-end message propagation delay between two applications on a small network, with different loads on the nodes and the network. Two nodes were involved in the measurement, namely, a fast node based on the Motorola MPC860 and a faster node based on the Motorola MPC8255. Both nodes operated under the VxWorks RTOS. Running a large number of end-to-end measurements with no additional traffic on the switch resulted in a standard deviation of about 14 μs. These measurements involved the transmit and receive stack in both directions, and were probably dominated by the MPC860 node. Furthermore, running a large number of end-to-end measurements with a heavily loaded fast node, representing a busy time server, resulted in a standard deviation of about 18 μs when no long replies were scheduled. Using standard statistical techniques, we estimate that the standard deviation for the delays due to switch, drop link, and time server was about 11 μs.

Thus, even with powerful nodes, we are barely within class P3 at the standard deviation level. Observing that the worst-case is at least three times the standard deviation, we cannot hope for better than class P2 if no action is taken.

Handling the Problem

We consider three methods for dealing with the influence of the random delay function *ρ(t),* namely, statistical, filtering, and erasure methods. Statistical methods use a large number of measurements in the calculations, thereby reducing the influence of individual measurements. These methods require a large set of measurements, since the error of the calculated estimate is approximately equal to $\sigma / \sqrt{n}$, where σ is the mean measurement error and *n* is the number of measurements. Filtering methods provide a new estimate from the current measurement combined with a set of previous measurements and estimates. These methods work best when the errors are distributed symmetrically about zero. Erasure methods evaluate the probability that the new measurement is contaminated by a delay. If the probability is greater than a predefined value, the measurement is erased and not used. Such procedures are usually simple to implement, but may ignore valid measurements and thus yield erroneous estimates.

Low Level Packet Time Stamping

Traditional SNTP implementations are usually based on time stamping incoming and outgoing SNTP time packets. The accuracy achieved by adhering to such a time-stamping scheme suffices only for class P2 and below. To attain sufficient time-stamping accuracy for class P3 and above, a low-level time-stamping scheme must be implemented both in the time client and the time server. Such a low-level time-stamping scheme is considered an implementation issue and therefore is not in conflict with SNTP as such.

Basically, there are three different low-level time-stamping methods. First, we can implement hardware time stamping in, or close to, the Ethernet controller. Second, we can implement software time stamping in an interrupt service routine (ISR) outside the RTOS. Such an ISR should be connected to the Ethernet interrupt request signal and have a top hardware priority. The third possibility is to implement software time stamping in the Ethernet driver, which is controlled by the RTOS. This driver is connected to the Ethernet interrupt request signal with a normal hardware priority.

Measured Low-level Time-Stamping Accuracy

To evaluate the accuracy of the various options for low-level time stamping within the time client, a test was run with various loads on the time client. The measurement setup is illustrated in Figure 1. The precision attained when using a hardware time-stamping scheme is, of course, totally independent of the software load in the time-client node. Furthermore, the time differences between a hardware time stamp and the time stamp generated by an interrupt outside the RTOS under a high node-processor load were found to be virtually independent of the node-processor load. The standard deviation of the differences is 0.57 µs. Therefore, we can be 99.7% sure of a measurement accuracy of ±1.71 µs. Thus, by hardware time-stamping or by low-level software time-stamping outside the RTOS, most of the time-client inaccuracy in the error budget has been eliminated.

The time differences between a hardware time stamp and the time stamp generated by a standard interrupt under RTOS control for the given processor load depend heavily on the node-processor load. The standard deviation of the differences is 21.0 µs. Therefore, we can be 99.7% sure of a measurement accuracy of ±63 µs.

From these measurements we conclude that time-stamping with a sufficiently high priority interrupt (preferably non-maskable) easily fulfills the requirements for classes P1 to P3. This method may also be used for class P4 if a careful check of additional errors is maintained. In contrast, time-stamping using an interrupt under RTOS control fulfills the requirements only for class P1. This method can also be used for class P2 if a careful check of additional errors is maintained. Even with sophisticated filtering and statistical techniques, this time-stamping method is not suitable for classes P3 to P5.

Passing Time-Synchronization Information

When performing time synchronization over a communication network, the layered protocol concept is inadequate, since the protocol stack functions as a *processing pipeline*, where each layer does whatever it needs to do in whatever time it may take. Consequently, the hardware driver, which is the lowest layer, knows when a frame is sent or received but cannot use the information. On the other hand, the application layer needs the timing information but has no standard way to obtain it.

Time Stamping on Reception

Since the hardware driver layer does not know whether the application layer wants a time stamp or not when a frame is received, the safest solution is to generate a time stamp for each frame. The problem lies in connecting the time stamp to the frame. The possibilities for making this connection include storing the time stamp in the frame buffer, storing the time stamp in a separate structure, or passing the time stamp to the application using a *callback* function.

Storing the time stamp in the frame buffer is the simplest solution. The disadvantage of this approach is that there is no obvious place to store the time stamp. Storing the time stamp in the buffer outside the actual packet does not interfere with higher-level protocols, but since the data part of the packet is often copied between buffers as the packet winds its way up through the protocol layers, this information is lost when the first copy is made. Storing the time stamp in a separate structure is no better since there must be a connection between the time stamp and the contents of the packet. However, it is not possible to handle such connections within all known and unknown protocols.

Finally, there is the callback method, in which the (S)NTP protocol layer installs a function pointer in the Ethernet MAC layer. Whenever a network packet arrives, the MAC layer calls this function with two parameters, namely, a pointer to the packet buffer and the packet arrival time stamp. The called function first determines whether or not the packet in the frame buffer is an SNTP packet. If this is the case, the function copies the time stamp into its proper place in the packet. Since the callback method lets the application layer handle the problem, we conclude that a callback function is the best and most flexible solution.

Time Stamping at Frame Transmission

Time stamping at frame transmission is really two separate problems. The first problem is telling the hardware driver layer that a time stamp should be created when the frame is sent on the network, and the second problem is passing the time stamp back to the application. Various schemes can be invented to solve the first problem. The second problem is more difficult, and so far the only solution we have found is to use a callback function.

Estimating the Clock Frequency Offset

Referring to equation (2), we now estimate the time-client clock frequency offset *a* using one or more (t_1, T_2) pairs corrupted with an unknown random delay. The frequency offset is due to initial tolerance, aging, and temperature drift. The initial tolerance is constant, the frequency change due to aging is sufficiently slow that it can be neglected inside calibration periods, and temperature is usually stable after warm-up. As long as our measurement period is less than about one hour, these assumptions are usually fulfilled. One exception is a system in which the crystal temperature changes appreciably through the measurement period. In this case we measure and calculate the average value of *a* over the measurement period.

There are two distinct ways to estimate frequency offset, namely, off-line and on-line. The off-line method collects raw measurements for the entire measurement period before calculating estimates, whereas the on-line method recalculates the estimates after every measurement. Both methods have their advantages and disadvantages.

The off-line method always gives the best estimate for the given set of past measurements while the on-line method always gives the best estimate for each current measurement, although the estimates fluctuate from measurement to measurement. Both methods give the same estimate at the end of the measurement period. However, the on-line method tries to predict the final estimate earlier. If there is no extra time-server traffic, the on-line method gives an accurate estimate after 3 measurements, two of which are used to determine the offset and slope and the third to verify them. If there is significant server traffic, and therefore spurious delays, the on-line method gives fluctuating estimates until a sufficient number of measurements have been collected.

Statistical Methods

Statistics on a Set of Deviations

One possible algorithm for estimating the client frequency deviation is as follows. First, acquire a statistically large (more than 30) set of (t_1, T_2) pairs. Then, using two (t_1, T_2) pairs at a time, estimate the ratio between the reference clock and the client clock. Some of these estimates are incorrect due to switch delay caused by head-of-line blocking, but a large part of the estimates are correct. The average or median of the distribution of the frequency ratios can then be applied as a measure of the frequency ratio between the reference clock and the client-node clock.

This statistical algorithm was tested by collecting 1000 (t_1, T_2) pairs under no extra server load and under 50% additional time-server load conditions. In the case of no extra load on the server drop link, the distribution of the frequency deviation was characterized by an average of -0.223 ppm (ppm denotes parts per million, where 1 ppm accuracy equals a relative accuracy of 0.0001%), a standard deviation of 0.003 ppm, and a median of -0.223 ppm. Since the average and the median were equal within three significant figures, we used Gaussian statistics. The standard deviation implied with 99.7% certainty that the frequency deviation was between -0.214 ppm and -0.232 ppm. In the case of 50% extra load on the server drop link, the distribution of the frequency deviation was characterized by an average of -0.216 ppm, a standard deviation of 0.164 ppm, and a median of -0.211 ppm. Since the average and median were well within one standard deviation, the distribution was plausibly Gaussian. For this measurement set, the uncertainty in the frequency deviation was much larger, in particular, the calculated 99.7% confidence interval was between -0.71 ppm and +0.28 ppm.

Using Linear Regression

Linear regression calculates the best possible straight line through a set of given points. Best possible means that the sum of the squares of the differences between the line and the given points is minimal. In the case of 50% extra load on the server drop link, linear regression gives a frequency deviation estimate of –0.213 ppm, placing it midway between the average and median values for the set-of-deviations algorithm.

Using Sliding Linear Regression

Changing the linear regression method from off-line to on-line is easy. After every new measurement, the slope and intercept are calculated based on all measurement points up to and including the last. When completed, the newly calculated slope and intercept values are the best available estimates. In our test involving 50% extra load on the server drop link, this method gave estimates that fluctuated initially, but stabilized after about 100 measurements.

Filtering Methods

A Low-Pass Filter Method

Using a first-order recursive digital low-pass filter on the measured time differences gave a smoother curve than the raw data, but did not directly produce an estimate of slope or offset. However, such a method may be useful as a *prefilter*.

A Simple Two-Point Prefilter

A hybrid between a filter method and an erasure method provides a surprisingly effective tool for reducing the influence of random delays. Define $\delta = (T_2 - t_1)$, and let (t_n, δ_n) denote the nth value of $(t_1, T_2 - t_1)$. The (t_1, T_2) pairs are assumed to be measured at equal intervals or at least within 1% of each other. The last measurement is (t_{n+1}, δ_{n+1}). For the finite version of the algorithm, we create a prefiltered set of data $(t_n, \min(\delta_n, (\delta_{n-1} + \delta_{n+1})/2))$ from the set (t_n, δ_n). This data set is used in the calculations. A simpler, recursive version involves substituting the minimum of the two values δ_n and $(\delta_{n-1} + \delta_{n+1})/2$ into δ_n. Although the batch version is more reliable, the recursive version is stable and shows slightly better performance.

Other Prefilters

Calculating the mean value of two points preceding and two points succeeding the measurement point did not improve the performance of the prefilter. A dual version, which fared better, is similar to the simple two-point prefilter. In this algorithm we assume that the

(t_1, T_2) pairs are measured at equal intervals. The last measurement yields (t_{n+2}, δ_{n+2}). For the batch version of the algorithm, we create a prefiltered set of data (t_n, min(δ_n, $(\delta_{n-1} + \delta_{n+1})/2$, $(\delta_{n-2} + \delta_{n+2})/2$)) from the set ($t_n$, δ_n), and this data set is used in the calculations. For the recursive version, the minimum of the values δ_n, $(\delta_{n-1} + \delta_{n+1})/2$, and $(\delta_{n-2} + \delta_{n+2})/2$ is used to replace δ_n.

Non-symmetrical Prefilters

If the (t_1, T_2) pairs are not measured at equal intervals, we use the two-point prefilter with minor modifications and slightly higher complexity. Let $\alpha = (t_n - t_{n-1})/(t_{n+1} - t_{n-1})$. For the batch version of the algorithm, we create a prefiltered set of data (t_n, min(δ_n, $\alpha\delta_{n-1} + (1-\alpha)\delta_{n+1}$)) from the set ($t_n$, δ_n).

Erasure Methods

Contrary to expectations, no good erasure methods were found, since the remaining data sets were not equally spaced in time. Although linear regression methods were able to handle such data, the filtering methods ran into trouble.

Estimating the Absolute Time Offset

In the previous section we presented a method for making the time-client and time-server clocks track each other. In this section we estimate the absolute time offset *B* that can be added to T_1 so that the two clocks show the same value at the same point in time.

By calibrating the network setup beforehand, we know the nominal amount of time it takes a time packet to travel from the client to the server. By adding that amount to T_1, we know the nominal value of T_1 at the global time T_2. It is then a simple matter to adjust the offset value B. In the ideal case, the shape of the distribution of the differences between the calculated values as well as the small standard deviation shows with 99.7% certainty that using the estimated value for T_2 implies that the estimates are within the requirements for class P4. However, in the heavily loaded case, we must resort to other means. A basic illustration of the problem is given in Figure 2, which contains the measurement values after correcting for frequency offset.

Statistical Methods

Referring to formula (3), the linear regression formula gives values for both *a* and *B*. In our case, *a* represents the frequency difference between the reference-clock oscillator and the time-client-clock oscillator. In the same way, *B* represents the mean difference between the two clocks, as observed from the time client. Using *B* as a measure of the absolute time offset is not advisable, however, since the linear regression formula tries to distribute the measured values equally on both sides of the estimated values. Since all delays are positive, the intersection value will be the sum of the actual time offset and the average delay.

Statistical analysis of the distribution of the delays is given in Fact Sheet 2. The conclusion is that the distribution does not conform to any statistical formula, and the average delay is therefore unpredictable. Using the intersection formula **(** $B = \frac{t \quad (T-t) - \quad (T-t) \quad t^2}{(\quad t)^2 - N \quad t^2}$) to calculate the time offset thus gives an error equal to the average delay.

Filtering Methods

Figure 2 shows what happens when the estimated values provided by the above methods are subtracted from the measurements**.** It turns out that, except for a fixed offset, all of the methods yield the same result. We refer to these values as *CalcOffsets*. Since all delays are positive, we want to find the lower envelope of these values. Standard low-pass filters exhibit the same problems as the statistical intersection method in that they try to track the local *average* value. What we need is a simple, nonlinear filter that provides the minimum of a set of values.

A Sliding Minimum Filter

Figure 3 shows the result of applying an 8-point sliding minimum filter to the values in Figure 2. The algorithm for this filter is given by "substitute (t_n, min(δ_n, δ_{n-1}, ... δ_{n-7})) for (t_n, δ_n)". This filter finds the minimum over the last 8 CalcOffset values. Figure 3 shows the output τ_n of the filter. Most of the noise is gone, and we are left with the real offset plus some spikes. To eliminate these spikes we have two options, namely, using a wider interval for the minimum, or applying a low-pass filter to the sliding minimum values.

Figure 4 shows the result of using a 16-point sliding minimum filter instead of an 8-point. All of the spikes have been removed, and the estimate stays within 3.5 µs instead of the 70 µs spike variation in Figure 3. To be on the safe side, however, we use a low-pass filter to remove any remaining spikes. The average value of the CalcOffsets is -28.9 µs with a standard deviation of 1.2µs. Thus, we can be 99.7% sure that the offset is within ±3.6 µs of the correct value.

Erasure Methods

It is debatable whether the sliding minimum filter is a filtering method or an erasure method, although the filter works satisfactorily in either case.

Higher Precision with Special Time Servers

OnTime Networks, Inc., noted the problem of random delays through Ethernet switches and developed a series of industrial-grade Ethernet switches with built-in time servers [8]. Since the time server is inside the switch, the time server can precisely calculate when a packet is transmitted onto the destination drop link and thus time stamp the packet precisely, thereby eliminating randomness in both directions, from client to server and then back again. This technology may be the easiest way to attain class P1 precision without multiple measurements and associated statistics. However, this solution entails somewhat higher cost (advanced switches are far more expensive than simple switches).

Conclusion

A precision time-synchronization implementation should contain the following elements: a high-priority interrupt connected to the network controller interrupt to obtain precise time stamps; a callback mechanism for delivering time stamps to the time synchronization main application; a simple two-point prefilter, preferably a dual-point prefilter; a procedure for calculating the local clock frequency offset, either linear regression or mean slope; and an 8-point sliding minimum filter or, preferably, a 16-point sliding minimum filter, for calculating the absolute time offset. By careful attention to detail, such an implementation can fulfill the requirements for Class P4 (±4 µs).

The methods discussed in this article can help determine the frequency and time offset of a local time-of-day clock. The next challenge is to construct a local time-of-day clock and apply the synchronization information to it.

References

[1] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén, "How Does Control Timing Affect Performance?," *IEEE Contr. Sys. Mag.,* June 2003.

[2] D.L. Mills, "Internet time synchronization: the Network Time Protocol," *IEEE Trans. Comm.,* Vol. COM-39, pp. 1482-1493, 1992.

[3] D.L. Mills, "Simple Network Time Protocol," *RFC 2030* (IETF reference, online at various locations)

[4] D.L. Mills, "Precision synchronization of computer network clocks," *ACM Comp. Comm. Rev.,* Vol. 24, pp. 28-43, 1993.

[5] "IEC 61850-9-1 - Ed. 1.0: Communication networks and systems in substations - Part 9-1: Specific Communication Service Mapping (SCSM) - Sampled values over serial unidirectional multidrop point to point link," International Electrotechnical Commission.

[6] T. Skeie, S. Johannessen, and Ø. Holmeide, "The Road to an End-to-End Deterministic Ethernet," *Proc. 2002 IEEE Int. Workshop Factory Comm. Sys.*, pp. 3-9, 2002.

[7] T. Skeie, S. Johannessen and Ø. Holmeide, "Highly Accurate Time Synchronization over Switched Ethernet," *Proc. 8th IEEE Conf. Emerging Tech. Factory Automation*, October 2001.

[8] OnTime Networks, http://www.ontimenet.com/

[sidebar]

Delays and Probability Distributions

If a time server has an Ethernet connection that is busy a fraction β of the time, the probability that the drop link is busy is also β. Hence, the probability of the drop link being not busy is 1-β. Now, if the drop link is busy, a queue of packets may be waiting for the drop link to become free. Assuming that all packets arrive randomly and independently of each other, let ε denote the probability of the drop link being busy and no packets queued. Then ε^2 is the probability of the drop link being busy and exactly one packet in the queue, ε^3 for exactly two packets, and so forth. This probability chain bears a slight resemblance to the binomial distribution, although a Poisson distribution described the situation better.

Given the randomness of the packet arrival time, the delay due to a packet on the drop link has a rectangular distribution from 0 to the total packet duration. By including the specified interpacket gap in the total packet duration, the delay due to packets in the queue is the sum of these total packet durations. When all packets are of the same size, the delay distribution is somewhat like that in Figure 5.

If the measured delays follow a Poisson distribution, calculating the zero point is easy, since the mean and variance are dimensionless and equal. However, after extensive measurements and calculations, we concluded that the Poisson distribution was not a good approximation to the real distribution, and therefore could not be relied upon to accurately predict the zero point.

Biography

Svein Johannessen

Svein Johannessen is an independent consultant employed by ABB Corporate Research Norway. He received a Masters degree in mathematics from the University of Oslo in 1970. He was a research scientist at the Central Institute for Industrial Research in Norway, specializing in automation, microprocessor technology, and robotics. In 1985 he joined Tandberg Data Display, where he became Director for R&D. Johannessen has been an associate lecturer at the University of Oslo, Department of Physics, since 1982. At ABB Corporate Research he has worked on various projects in hardware design and communications technology.

Mailing address: c/o ABB Corporate Research Norway, Bergerveien 12, N-1351 Billingstad, Norway.

E-mail address: svein.johannessen@kode.no

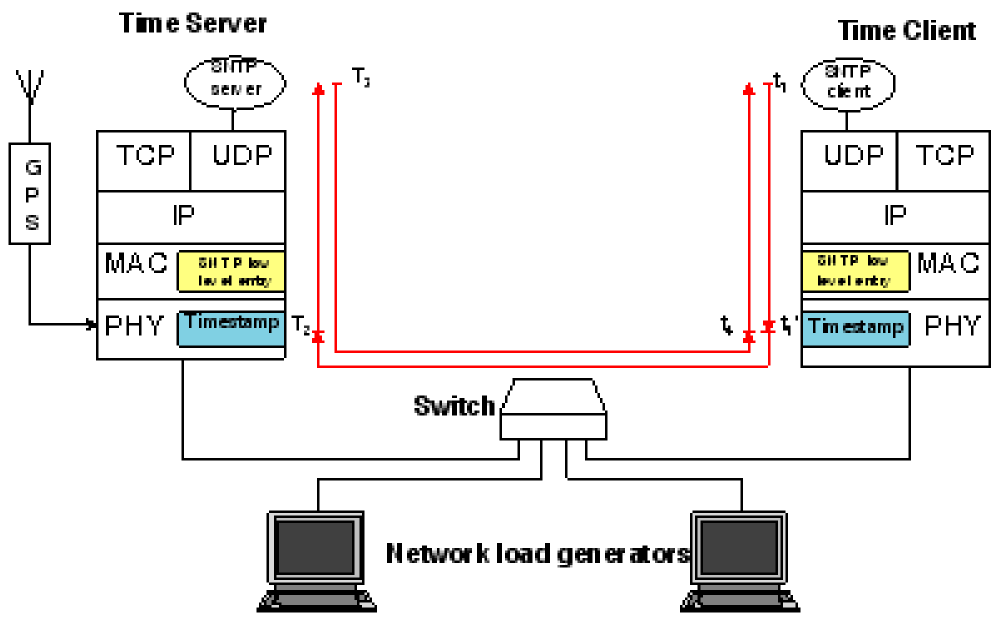

Figure 1. The SNTP Test Setup. The time server is from OnTime Networks AS, and the SNTP client is implemented on a medium performance platform based on the ARM7 processor.

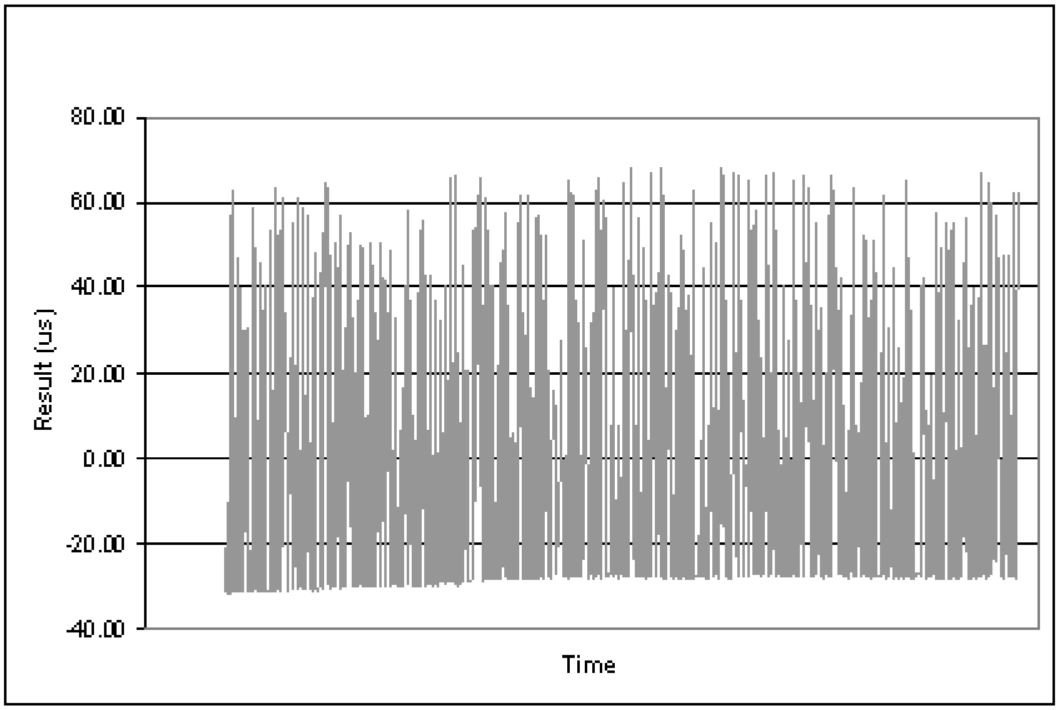

Figure 2. The Measured Time Differences Minus The Calculated Values. Knowing that all delays are positive, the human eye can easily estimate the correct offset. Mathematically, the situation is somewhat more complicated.

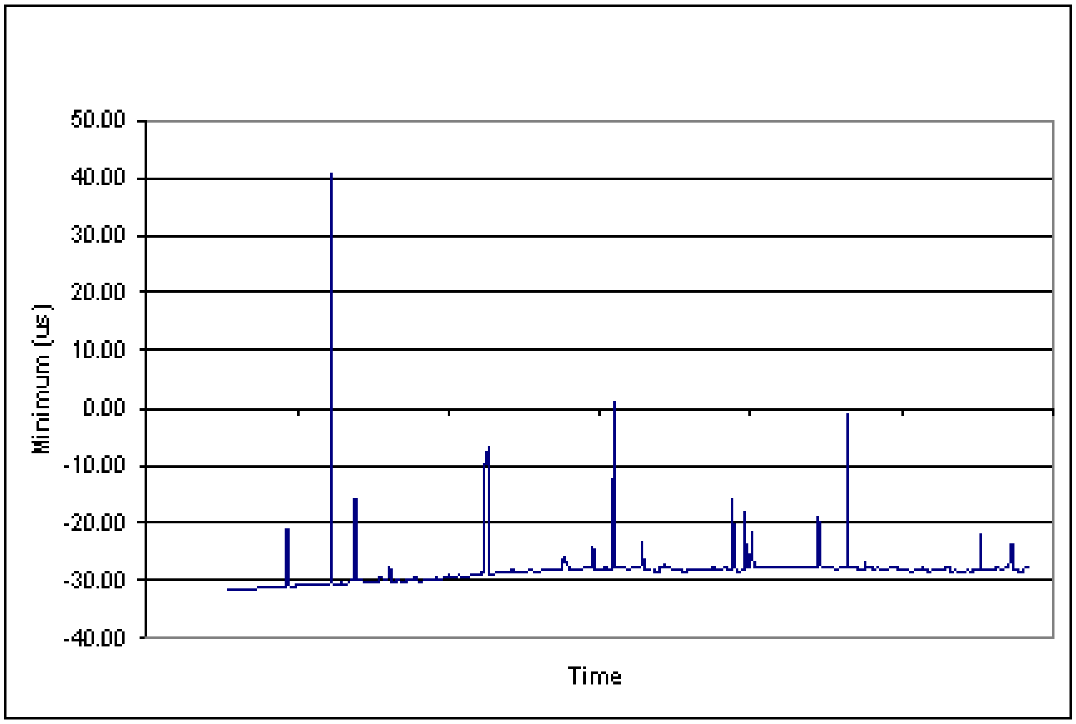

Figure 3. The Values in Figure 2 After Passing Through An 8-Point Sliding Minimum Filter. This procedure is the first step in determining the lower envelope of the values shown in Figure 2.

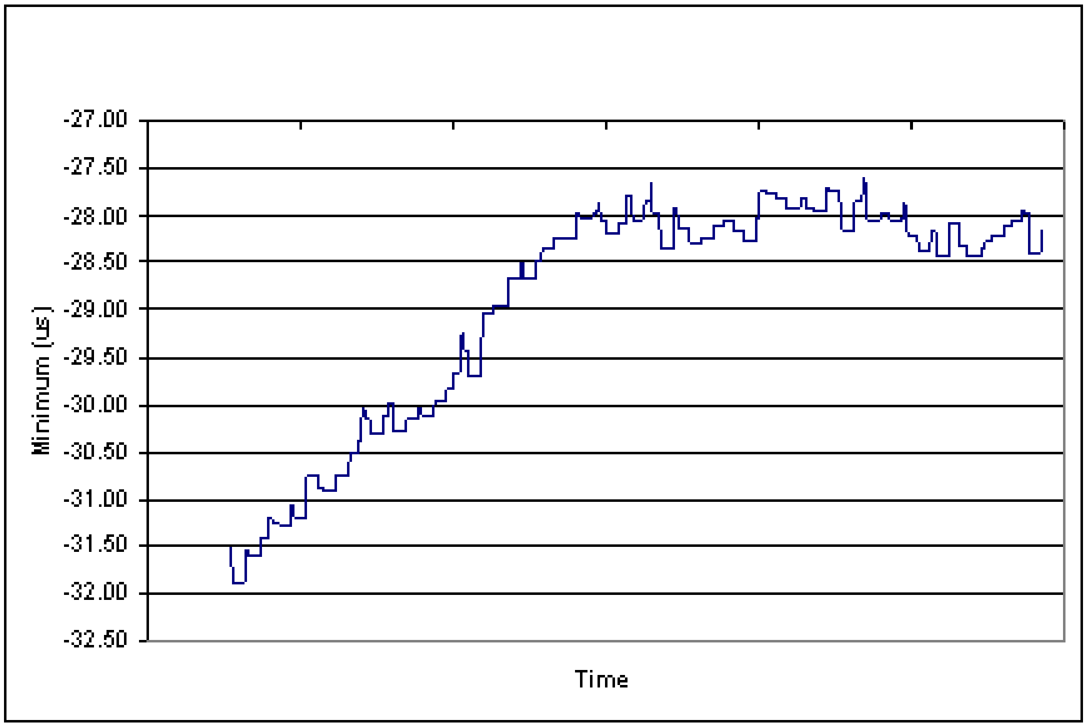

Figure 4. The Values in Figure 2 After Passing Through A 16-Point Sliding Minimum Filter.

This filter gives a much smoother result than the filter shown in Figure 3.

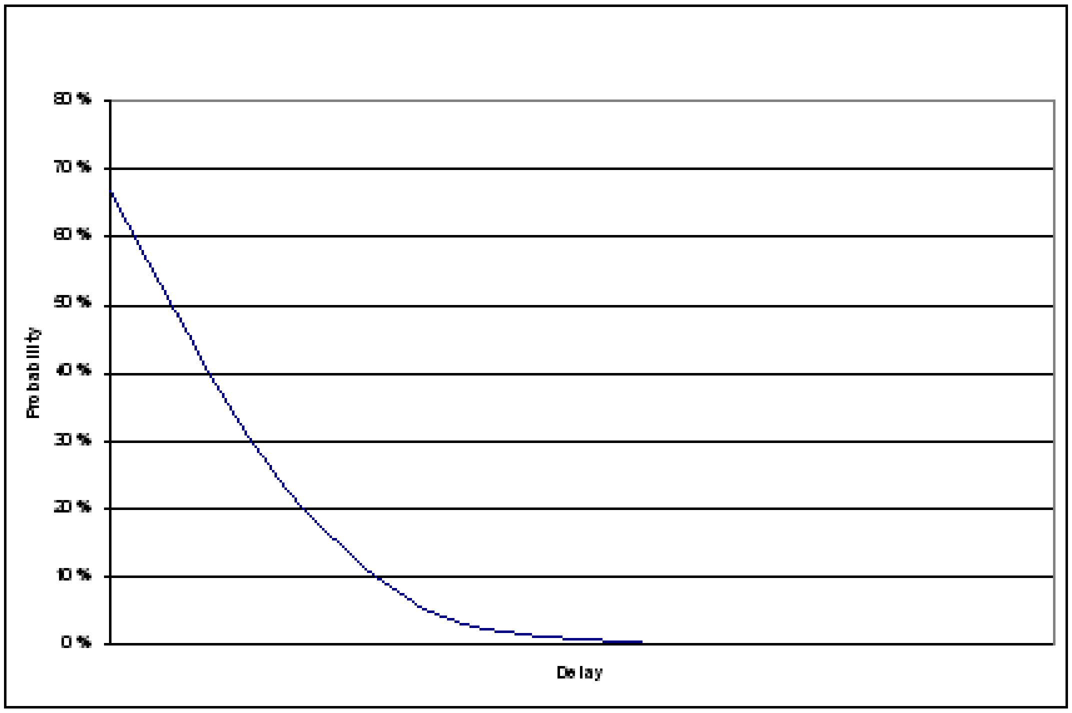

Figure 5. A Poisson Distribution. This distribution has the useful characteristic that the dimensionless mean and variance are equal.