Improved Algorithms for Synchronizing Computer Network Clocks

David L. Mills, *Member, IEEE*

***Abstract*— The Network Time Protocol (NTP) is widely deployed in the Internet to synchronize computer clocks to each other and to international standards via telephone modem, radio and satellite. The protocols and algorithms have evolved over more than a decade to produce the present NTP Version 3 specification and implementations. Most of the estimated deployment of 100 000 NTP servers and clients enjoy synchronization to within a few tens of milliseconds in the Internet of today.**

This paper describes specific improvements developed for NTP Version 3 which have resulted in increased accuracy, stability and reliability in both local-area and wide-area networks. These include engineered refinements of several algorithms used to measure time differences between a local clock and a number of peer clocks in the network, as well as to select the best subset from among an ensemble of peer clocks and combine their differences to produce a local clock accuracy better than any in the ensemble. This paper also describes engineered refinements of the algorithms used to adjust the time and frequency of the local clock, which functions as a disciplined oscillator. The refinements provide automatic adjustment of algorithm parameters in response to prevailing network conditions, in order to minimize network traffic between clients and busy servers while maintaining the best accuracy. Finally, this paper describes certain enhancements to the Unix operating system kernel software in order to realize submillisecond accuracies with fast workstations and networks.

I. Introduction

A *COMPUTER CLOCK* (or simply *clock*) is an ensemble of hardware and software components used to provide an accurate, stable and reliable time-of-day function for the computer operating system and its clients. In order that multiple distributed computers sharing a network can synchronize their operations with each other, a *synchronization protocol* is used to exchange time information and synchronize the clocks. In this paper the term *local clock* identifies the clock in a particular computer as distinguished from a *peer clock* in another computer with which it exchanges time information. If the clocks are to agree with Coordinated Universal Time (UTC) (sic), a radio clock (usually a special-purpose radio or satellite receiver) must be provided to synchronize one or more of them to UTC as disseminated by various means [14].

Computer clocks can be synchronized typically within a few tens of milliseconds in the global Internet of today [12]. However, as computers and networks become faster, there is every expectation that future applications will require accuracies better than a millisecond. This requires in essence a complete reexamination of all elements of the timekeeping apparatus described originally in [9], including the protocols which exchange timekeeping messages and the algorithms which process the data and discipline the local clock. This paper examines in detail the various design issues necessary to achieve this goal and, in particular, describes a suite of algorithms designed to exchange data with possibly many redundant peer clocks and to select an accurate, stable and reliable set of clocks from among them. Besides some new results, it contains some previous work published only in technical reports.

In this paper the Network Time Protocol (NTP) developed for the Internet is used as an example application of the new algorithms, but others, such as the Digital Time Synchronization Service (DTS) [2] could be used as well. After a review of terms and notation in Section II, Section III gives an overview of NTP. Section IV summarizes the clock filter, clustering and combining algorithms, which select the best measurement samples from among possibly several peers and combine them to produce the best available time.

The main results of this paper are in Sections V through VII. Section V describes the intersection algorithm, which is used to separate the *truechimers*, which represent correct clocks, from *falsetickers*, which may not. Section VI contains an analysis of the local clock model, including the effects of oscillator jitter and wander. Section VII details the local clock discipline, which is implemented as a hybrid phase/frequency-lock loop. These algorithms are primarily responsible for the increased accuracy and reliability of the current protocol compared to previous versions.

Section VIII contains a summary of related improvements and extensions of previous algorithms, including those utilizing special PPS and IRIG signals generated by some radio clocks. It also contains a description of certain modifications to four different Unix operating system kernels which provide extremely precise control of the clock time and frequency. Section IX discusses the present status of NTP in the Internet, Section X outlines future plans, and Section XI is a summary of this paper.

II. Terms and Notation

In this paper the terms *epoch, timescale, oscillator, tolerance, clock, and time* are used in a technical sense. Strictly speaking, the epoch of an event is an abstraction which

Manuscript received December 6, 1994; approved by IEEE/ACM Transactions on Networking Editor R. Guerin. This work was supported by the Advanced Research Projects Agency under NASA Ames Research Center Contract NAG 2-638, the National Science Foundation under Grant NCR-93-01002, and the U.S. Navy Surface Weapons Center under Northeastern Center for Engineering Education Contract A30327-93.

The author is with the Department of Electrical Engineering, University of Delaware, Newark, DE 19716 USA (e-mail: mills@udel.edu).

IEEE Log Number 9412272.

determines the ordering of events in some given frame of reference or timescale. An oscillator is a generator capable of precise frequency (relative to the given timescale) within a specified tolerance, usually expressed in parts-per-million (ppm). A clock is an oscillator together with a counter which records the number of cycles since being initialized with a given value at a given epoch. The value of the counter at epoch t defines the time of that epoch $T(t)$. In general, time is not continuous and depends on the precision of the counter.

Let $T(t)$ be the time displayed by a clock at epoch t relative to the standard timescale:

$$T(t) = T(t_0) + R(t_0)[t - t_0] + 1/2D(t_0)[t - t_0]^2 + \epsilon(t) \quad (1)$$

where $T(t_0)$ is the time at some previous epoch T_0, $R(t_0)$ is the frequency (rate) and $D(t_0)$ is the drift (first derivative of frequency) per unit time. In the conventional (stationary) model used in the literature, T and R are estimated by some disciplining process and the second-order term D is ignored. The random nature of the clock is characterized by ϵ, usually in terms of phase or frequency spectra or measurements of variance [15].

In this paper the *stability* of a clock is how well it can maintain a constant frequency, the *accuracy* is how well its time compares with UTC and the *precision* is to what degree time can be resolved in a particular timekeeping system. These terms will be given precise definitions when necessary. The *time offset* of clock i relative to clock j is the time difference between them $x_{ij}(t) \equiv T_i(t) - T_j(t)$ at a particular epoch t, while the *frequency offset* is the frequency difference between them $y_{ij}(t) \equiv R_i(t) - R_j(t)$. It follows that $x_{ij} = -x_{ji}$, $y_{ij} = -y_{ji}$ and $x_{ii} = y_{ii} = 0$ for all t. When clear from context, the subscripts i and j will be omitted. In this paper, reference to simply "offset" means time offset, unless indicated otherwise. The term *jitter* refers to differences between the elements of a series $\{y_k\}$; similarly, *wander* refers to differences in $\{y_k\}$ where the peers involved are understood. Finally, the *reliability* of a timekeeping system is the fraction of the time it can be kept connected to the network and operating correctly relative to stated accuracy and stability tolerances.

In order to synchronize clocks, there must be some way to directly or indirectly compare them in time and frequency. In network architectures such as DECnet and Internet, local clocks synchronize to designated *time servers*, which are timekeeping systems belonging to a *synchronization subnet*. At the root of the subnet are the *primary servers*, which synchronize to external sources (e.g., radio clocks) and are assigned a *stratum number* of 1. *Secondary servers*, which synchronize to primary servers and each other, are assigned stratum numbers equal to the minimum subnet hop count from the root. In general, synchronization proceeds in a hierarchical fashion from the root in increasing stratum numbers along the edges of a minimum spanning tree. In this paper to *synchronize frequency* means to adjust the subnet clocks to run at the same frequency, to *synchronize time* means to set them to agree at a particular epoch with respect to UTC and to *synchronize clocks* means to synchronize them in both frequency and time.

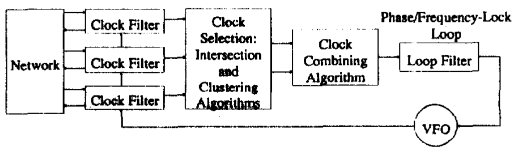

Network
Clock Filter
Clock Filter
Clock Filter
Clock Selection: Intersection and Clustering Algorithms
Clock Combining Algorithm
Phase/Frequency-Lock Loop
Loop Filter
VFO

Fig. 1. Network time protocol.

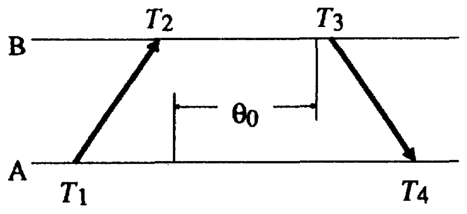


Fig. 2. Measuring delay and offset.

III. Network Time Protocol

The Network Time Protocol (NTP) is used by Internet time servers and their clients to synchronize clocks, as well as automatically organize and maintain the time synchronization subnet itself. NTP and its implementations have evolved and proliferated in the Internet over the last decade, with NTP Version 3 adopted as a Internet Standard (Draft). A detailed description of the architecture and service model is contained in [9], while the current formal protocol specification is defined in RFC-1305 [10].

Fig. 1 shows the overall organization of the NTP time server model. *Timestamps* are exchanged between the client and each of possibly several other subnet peers at intervals ranging from a few seconds to several hours. These are used to determine individual roundtrip delays and clock offsets, as well as provide error estimates. As shown in the figure, the computed delays and offsets for each peer are processed by the clock filter algorithm to reduce incidental jitter.

The clock selection algorithm determines from among all peers a suitable subset capable of providing the most accurate and trustworthy time. This is done using a cascade of two subalgorithms, one based on interval intersections to cast out falsetickers and the other based on clustering and maximum likelihood principles to improve accuracy. The resulting offsets of this subset are first combined on a weighted-average basis and then used to drive the clock-discipline algorithm, which is implemented as a feedback loop. In this loop the combined offset is processed by the loop filter to control the variable frequency oscillator (VFO) frequency. The VFO is implemented as a programmable counter using a combination of hardware and software components. It furnishes the time reference to produce the timestamps used in all timing calculations.

Fig. 2 shows how NTP timestamps are numbered and exchanged between peers A and B. Let T_1, T_2, T_3, T_4 be the values of the four most recent timestamps as shown and, without loss of generality, assume $T_3 > T_2$. Also, for the moment assume the clocks of A and B are stable and run at the same frequency. Let

$$a = T_2 - T_1 \quad and \quad b = T_3 - T_4.$$

If the network delay difference from A to B and from B to A, called *differential delay*, is small, the clock offset θ and roundtrip delay δ of B relative to A at time T_4 are close to

$$\theta = \frac{a+b}{2} \quad \text{and} \quad \delta = a - b. \tag{2}$$

Each NTP message includes the latest three timestamps T_1, T_2 and T_3 , while the fourth T_4 is determined upon arrival. Thus, both peers A and B can independently calculate delay and offset using a single bidirectional message stream. This is a symmetric, continuously sampled, time-transfer scheme similar to those used in some digital telephone networks [6]. Among its advantages are that errors due to missing or duplicated messages can be avoided.

In [11] an exhaustive analysis is presented of the time and frequency errors that can accrue as the data are processed and refined at various levels in the subnet hierarchy. While the analysis is too long to repeat here, the results define the maximum error that can accrue under any operational condition, called the *synchronization distance* λ , and the error expected under nominal operating conditions, called the *dispersion* ϵ. There are several components of ϵ, including:

1) The maximum error in reading the local clock and each peer clock, which depends on the clock resolution and method of adjustment.
2) The maximum error due to the frequency tolerance of the local clock and each peer clock since the time either was last set.
3) The estimated error contributed by each peer clock due to delay variations in the network and statistical latencies in the operating systems on the path to the primary reference source, which depends on differences between successive measurements for each peer clock. This is called the *peer dispersion.*
4) The estimated error contributed by the combined set of peers used to discipline the local clock, which depends upon the differences between individual members of the set. This is called the *select dispersion.*

In practice, errors due to network delays usually dominate ϵ. However, it is not possible to characterize these delays as a stationary random process, since network queues can grow and shrink in chaotic fashion and packet arrivals are frequently bursty. However, the method of calculating ϵ defined in [10] represents a conservative estimate of the errors due to each of the above causes.

In [11] it is shown that, given ϵ calculated as above, $\lambda = \frac{\delta}{2} + \epsilon$ is a good estimate of the maximum error contribution due to all causes. In other words, if θ is the measured offset of the local clock relative to the primary reference source, then the true offset θ_0 relative to that source must with high probability be somewhere in the interval

$$\theta - \lambda \leq \theta_0 \leq \theta + \lambda \tag{3}$$

which is called the *confidence interval.*

The ϵ and λ are used as metrics in the various algorithms presented in following sections. They determine the peers selected by the intersection and clustering algorithms, the weight factors used by the clock combining algorithm, and the calculation of various error statistics. While the basic design of these algorithms is developed using sound engineering and statistical principles, there are a number of intricate details, such as various weights used in the filter and selection algorithms, which can only be determined using simulation and experiment. In general, however, the metrics used are based on the pragmatic observation that the highest reliability is usually associated with the lowest stratum and synchronization distance, while the highest accuracy is usually associated with the lowest stratum and dispersion.

IV. Clock Filter, Combining and Clustering Algorithms

The clock filter, clustering and combining algorithms shown in Fig. 1 operate essentially as described previously in [9], however all three have been refined and defined formally in [10]. In order to understand the other algorithms described in this paper, it will be useful to briefly summarize the operation of these three algorithms.

The clock filter algorithm operates on a moving window of samples to produce three statistical estimates: *peer delay, peer offset* and *peer dispersion.* We will use θ , δ and ϵ for these quantities when their distinction from the previous use is clear. A discussion of the design approach, implementation and performance assessment is given in [9] and will not be repeated here. However, the design described there, which can be described as a *minimum filter*, has been enhanced to include the peer dispersion contributions due to the frequency tolerance of the local clock and the interval between T_1 and the present time, which must be recorded with every data sample.

There are usually some offset variations among the peers surviving the intersection algorithm (described later), due to differential delays, radio clock calibration errors, etc. The clustering algorithm is designed to select the best subset of this population on a maximum likelihood basis. It first ranks the peers by stratum, then by λ. For each peer it computes the select dispersion, defined as the total weighted time offsets of that peer relative to all the others. It then ejects the outlyer peer with greatest select dispersion and repeats the process until either a pre-specified minimum number of peers has been met or the maximum select dispersion is less than or equal to the minimum peer dispersion for all peers in the surviving population.

The termination condition is designed to maximize the number of peers for the combining algorithm, yet to produce the most accurate time. Since discarding more outlyers can neither increase the select dispersion nor decrease the peer dispersion, further discards will not improve the accuracy. As incorporated in NTP Version 3, the increase in dispersion as samples grow old helps to reduce errors resulting from local clock instability.

For each selected peer i the clock combining algorithm constructs a weight

$$w_i = \frac{1}{\epsilon_i \sum_j \frac{1}{\epsilon_j}}$$

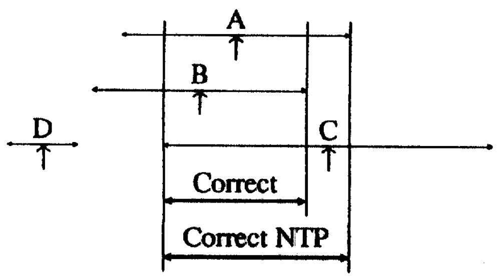


Fig. 3. Confidence and intersection intervals.

where j ranges over all contributors. The algorithm then computes ensemble averages

$$\overline{\theta} = \sum_j w_j \theta_j \quad and \; \epsilon = \sum_j w_j \epsilon_j .$$

V. Intersection Algorithm

When a number of peer clocks are involved as in Fig. 1, it is not clear beforehand which are truechimers and which are falsetickers. In order to provide reliable synchronization, NTP relies on multiple peers and disjoint peer paths whenever possible. Crucial to the success of this approach is a robust algorithm which finds and discards the falsetickers from among these peers. Criteria for evaluation include a suite of sanity checks, consistency checks and the intersection algorithm described in this section.

Recall that the true offset θ_0 of a correctly operating clock relative to UTC must be contained in the confidence interval (3). Marzullo and Owicki [7] devised an algorithm designed to find an appropriate interval containing the correct time given the confidence intervals of m clocks, of which no more than f are considered incorrect. The algorithm finds the smallest *intersection interval* containing points in at least $m - f$ of the given confidence intervals.

Fig. 3 illustrates the operation of this algorithm with a scenario involving four clocks A, B, C and D, with the peer offset θ (shown by the $\uparrow$ symbol) along with the confidence interval for each. For instance, any point in the A interval may represent the actual time associated with that clock. If all clocks are correct, there must exist a nonempty intersection including points in all four confidence intervals; but, clearly this is not the case in the figure. However, if it is assumed that one of the clocks is incorrect (e.g., D), it might be possible to find a nonempty intersection including all but one of the intervals. If not, it might be possible to find a nonempty intersection including all but two of the intervals and so on.

The algorithm used by DEC in DTS is based on these principles. The algorithm finds the smallest intersection containing at least one point in each of $m - f$ confidence intervals, where m is the total number of clocks and f is the number of falsetickers, as long as the $f < \frac{m}{2}$. For the scenario illustrated in Figure 3, it computes the intersection for $m = 4$ clocks, three of which turn out to be correct and one not. The interval marked DTS is the smallest intersection containing points in three confidence intervals, with one interval outside the intersection considered incorrect.

There are some cases where this algorithm can produce anomalistic results. For instance, consider the case where the left endpoints of A and B are moved to coincide with the left endpoint of D, so that $f = 0$. In this case the intersection interval extends to the left endpoint of D, in spite of the fact that there is a subinterval that does not contain at least one point in all confidence intervals. Nevertheless, the assertion that the correct time lies in the intersection interval remains valid.

One problem is that, while the smallest interval containing the correct time may have been found, it is not clear which point in that interval is the best estimate of the correct time. Simply taking the estimate as the midpoint of the interval throws away a good deal of useful statistical data and results in large jitter, as confirmed by experiment. Especially in cases where the network jitter is large, some or all of the calculated offsets (such as for C in Fig. 3 may lie outside the intersection. For these reasons, in the NTP algorithm the DEC algorithm is modified so as to include at least $m - f$ of the peer offsets. The revised algorithm finds the smallest intersection of $m - f$ intervals containing at least $m - f$ peer offsets. As shown in Figure 3, the modified algorithm produces the intersection interval marked NTP and including the calculated time for C.

The algorithm starts with a set of peers which have passed several sanity checks designed to detect configuration errors and defective implementations. In the NTP Version 3 implementation, only the ten peers with the lowest λ are considered to avoid needless computing cycles for candidates very unlikely to be useful. For each peer the algorithm constructs a set of three tuples of the form [*offset, type*]: $[\theta - \lambda, -1]$ for the lower endpoint, $[\theta, 0]$ for the midpoint, and $[\theta + \lambda, +1]$ for the upper endpoint. These entries are placed on a list sorted by increasing *offset*.

The job of the intersection algorithm is to determine the lower and upper endpoints of an interval containing at least $m - f$ peer offsets. As before, let m be the number of entries in the sorted list and f be the number of presumed falsetickers, initially zero. Also, let *lower* designate the lower limit of the final confidence interval and *upper* the upper limit. The algorithm uses *endcount* as a counter of endpoints and *midcount* as the number of offsets found outside the intersection interval.

1) Set both *endcount* and *midcount* equal to zero.
2) Starting from the beginning of the sorted list and working toward the end, consider each entry [*offset, type*] in turn. As each entry is considered, subtract *type* from *endcount*. If *endcount* $\geq m - f$, the lower endpoint has been found. In this case set *lower* equal to *offset* and go to step 3. Otherwise, if *type* is zero, increment *midcount*. Then continue with the next entry.
3) At this point a tentative lower endpoint has been found; however, the number of midpoints has yet to be determined. Set the *endcount* again to zero, leaving *midcount* as is.
4) In a similar way as step 2, starting from the end of the sorted list and working toward the beginning, add the value of *type* for each entry in turn to *endcount*. If *endcount* $\geq m - f$, go to step 5. Otherwise, if *type* is

TABLE I
PEER CONFIGURATION FOR SERVER *pogo*

Code	Server (Location)	Stratum	Source	θ	δ	ϵ	λ	Lower	Upper
*	GPS	0	GPS	0.117	0.0	1.01	1.01	–0.89	1.13
	churchy	2	*pogo*	1.080	0.42	1.36	4.07		
+	*rackety*	1	GPS	0.563	3.83	0.73	2.65	–2.08	3.21
+	*barnstable*	1	GPS	0.618	4.04	0.60	2.62	–2.00	3.24
+	*tick* (USNO)	1	ATOM	0.357	49.84	3.42	28.34	–27.98	28.70
+	*time* (NIST)	1	ACTS	0.635	101.72	4.14	55.00	–54.37	55.64
x	*err* (Switzerland)	1	DCF77	5.420	140.69	18.43	88.78	–83.36	94.20
x	*lucifer* (Germany)	1	GPS	9.863	183.36	36.62	128.30	–118.44	138.16
+	*time1* (Sweden)	1	ATOM	0.544	155.70	124.02	201.87	–201.33	202.41
-	*terss* (Australia)	1	OMEGA	1.088	767.40	69.05	452.75	–451.66	453.84

zero, increment *midcount*. Then continue with the next entry.

5) If $lower \leq upper$ and $midcount \leq f$, then terminate the procedure and declare success with *lower* equal to the lower endpoint and *upper* equal the upper endpoint of the resulting confidence interval. Otherwise, increment f. If $f \geq \frac{m}{2}$, terminate the procedure and declare failure. If neither case holds, continue in step 1.

The original (Marzullo and Owicki) algorithm produces an intersection interval that is guaranteed to contain the correct time as long as less than half the clocks are falsetickers. The modified algorithm produces an interval containing the original interval, so the correctness assertion continues to hold. However, so long as the clock filter produces statistically unbiased estimates for each peer, the new algorithm allows the clustering and combining algorithms to produce unbiased estimates as well.

Table I shows a typical configuration for NTP primary server *pogo*. The data used to construct tables such as this are collected by each server on a regular basis and automatically retrieved by monitoring hosts using scripts and programs designed for the purpose. Using these data, operators can quickly spot trouble in either the servers or the network.

The peers located in Europe, Australia, National Institute of Standards and Technology (NIST) in Boulder, CO, and U.S. Naval Observatory (USNO) in Washington, DC, are identified in the table; the others are located at the University of Delaware. The entry identified as GPS and assigned pseudo-stratum zero is a precision timing receiver synchronized by the Global Positioning System (GPS) and connected to *pogo*. Note that this receiver is treated like any other peer, so that possible malfunctions can be detected and avoided. The synchronization source for each peer is shown by dissemination service if stratum 0 or 1, or by another peer if higher. GPS, DCF77 and OMEGA use radio and satellite, ATOM is a national standard cesium clock ensemble, and ACTS is the Automated Computer Time Service operated by NIST [4].

The offset θ, delay δ, dispersion ϵ and synchronization distance λ for each peer are shown in the table, as well as the lower and upper endpoints used in the clock selection algorithm, all in milliseconds. Peer *churchy* is ineligible for selection because it is operating at a stratum higher than *pogo*, so would not normally provide better time, and in addition, it is synchronized to *pogo*, so would cause a synchronization loop. This peer would be considered for synchronization only if the GPS receiver and all other stratum-1 sources were to fail.

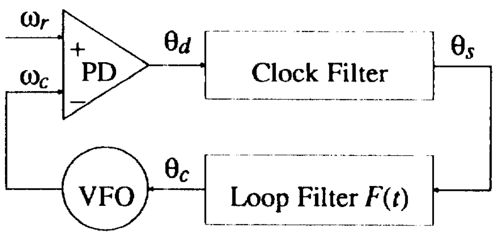


Fig. 4. Disciplined oscillator model.

The remaining peers are eligible for processing by the intersection and clustering algorithms. The synchronization status is shown by the Code column. Those marked "x" have been discarded by the intersection algorithm as falsetickers, while those marked "-" have been discarded by the clustering algorithm as outlyers. Note that the truechimer offsets all fall within the smallest intersection interval, while the falseticker offsets do not. Obviously, the ensemble average is improved by discarding falsetickers and outlyers.

The peers marked "*" and "+" have survived both algorithms and the one marked "*" has been identified as the pick of the litter. All of these peers will be considered by the combining algorithm; however, the NTP Version 3 implementation includes an option: If a designated peer has survived both algorithms, it is the sole source for synchronization and the combining algorithm is not used. This is useful in special cases where known differential delays are relatively severe or when the lowest possible jitter is required.

VI. LOCAL CLOCK MODELS

The local clock is commonly implemented using a hardware counter and room-temperature quartz oscillator. Such oscillators exhibit some degree of temperature-induced frequency instability in the order of 1-2 ppm due to room-temperature variations. The NTP clock discipline continuously corrects the time and frequency of the local clock to agree with the time as determined from the synchronization source(s).

A significant improvement in accuracy and stability is possible by modeling the local clock and its adjustment mechanism as a *disciplined oscillator*. In this type of clock the time and frequency are controlled by a feedback loop with a relatively long time constant, so the frequency is "learned" over some minutes or hours of integration. Besides improving accuracy, a disciplined oscillator can correct for the intrinsic frequency error of the oscillator itself, so that much longer intervals between timestamp messages can be used without significant accuracy degradation.

A disciplined oscillator can be implemented as the feedback loop shown in Fig. 4. The variable ω_r represents the reference signal and ω_c the variable frequency oscillator (VFO) signal, which controls the local clock. The phase detector (PD) produces a signal θ_d representing the instantaneous phase difference between ω_r and ω_c. The clock filter functions as a tapped delay line, with the output θ_s taken at the sample selected by the clock filter algorithm. The loop filter, with impulse response $F(t)$, produces a VFO correction θ_c, which controls the oscillator frequency ω_c and thus its phase. The

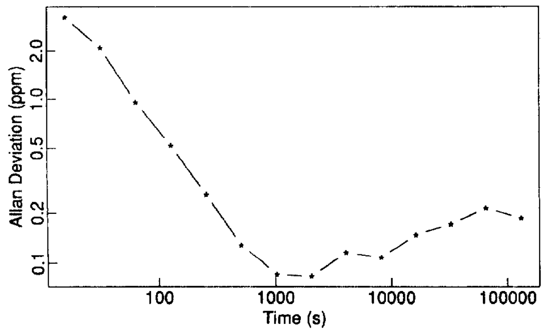


Fig. 5. Allan variance of typical local oscillator.

characteristic behavior of this model, which is determined by the $F(t)$, is studied in many textbooks and summarized in [11].

As reported in [12], the major source of error in most configurations is the stability of the local clock oscillator. The stability of a free-running frequency source is commonly characterized by a statistic called *Allan variance* [1], which is defined as follows. Consider a series of time offsets measured between a local clock and some external standard. Let x_k be the kth measurement and τ_k be the interval since the previous measurement. Define the *fractional frequency*

$$y_k = \frac{x_k - x_{k-1}}{\tau_k} \tag{4}$$

which is a dimensionless quantity. Now, consider a sequence of N independent fractional frequency samples $y_k(k = 0, 1, \cdots, N-1)$. If the averaging interval $\tau = \tau_k$ is the same as the interval between measurements, the 2-sample Allan variance is defined

$$\begin{aligned}\sigma_y^2(\tau) &\equiv \frac{1}{2}\langle(y_k - y_{k-1})^2\rangle \\ &= \frac{1}{2(N-2)\tau^2}\sum_{k=2}^{N-1}(x_k - 2x_{k-1} + x_{k-2})^2.\end{aligned}$$

The Allan variance $\sigma_y^2(\tau)$ (or Allan deviation $\sigma_y(\tau)$) is particularly useful when designing the clock discipline, since it determines the optimum impulse response $F(t)$, time constants and update intervals. Fig. 5 shows the results of an experiment designed to determine the Allan deviation of a typical workstation under normal room-temperature conditions. For the experiment, the local clock was first synchronized to a primary server on the same LAN using NTP to allow the frequency to stabilize, then uncoupled from NTP and allowed to free-run for about seven days. The local clock offsets during this interval were measured at the primary server using NTP. This model is designed to closely duplicate actual operating conditions, including the jitter of the LAN and operating systems involved.

It is important to note that both the x and y scales of Fig. 5 are logarithmic, but the axes are labeled in actual values. Starting from the left at $\tau = 16$, the plot tends to a straight line with slope near -1, which is characteristic of white phase noise [15]. In this region, increasing τ increases the frequency stability in direct proportion. At about $\tau = 1000$ the plot has an upward inflection, indicating that the white phase noise becomes dominated first by white frequency noise (slope -0.5), then by flicker frequency noise (flat slope), and finally by random-walk frequency noise (slope +0.5). In other words, as τ is increased, there is less and less correlation between one averaging interval and the next.

The Allan deviation can be used to determine the best clock discipline method to use over the range of τ likely to be useful in practice. At the lowest τ the errors due to phase noise dominate those due to frequency stability. A phase-lock loop (PLL) clock discipline provides the best performance in such cases. As the PLL time constant increases and with it τ , the PLL low-pass filter characteristic tends to reduce the phase noise, as well as compensate for any systematic (constant) local clock frequency error. However, while the phase averaging interval in a PLL increases directly as the time constant, the frequency averaging interval increases as the square. The price paid for this at the longer τ is an extremely sluggish adaptation to oscillator frequency wander.

On the other hand, at the highest τ, the errors due to frequency stability dominate those due to phase noise. A frequency-lock loop (FLL) clock discipline provides the best performance in such cases. In order to provide the most rapid adaptation to frequency wander, while avoiding spurious disruptions due to phase noise, the best τ would seem from Fig. 1 to be about 1000 s. However, it is apparent from (4) that the FLL can become seriously vulnerable to phase spikes at τ much below this. These conclusions were verified In a series of experiments and simulations using the algorithms developed in the next section.

VII. The NTP Clock Discipline

The Unix 4.3bsd timekeeping functions are implemented using a hardware timer interrupt produced by an oscillator in the 100-1000 Hz range. Each interrupt causes an increment *tick* to be added to the kernel *time* variable. The value of *tick* is chosen so that *time*, once properly initialized, is equal to the present time of day in seconds and microseconds relative to a given epoch. When *tick* does not evenly divide 1 s (1 000 000 μs), an additional increment *fixtick* is added to *time* once each second to make up the difference.

The oscillator can actually run at three different frequencies, one at the intrinsic oscillator frequency, a second slightly higher and a third slightly lower . The adjtime() system call is used to select one of the three frequencies and how long δt to run, in order to amortize the specified offset. The NTP clock discipline uses the adjtime() mechanism to control the VFO and implements the impulse response $F(t)$ using the algorithm described below.

The new clock discipline differs from the one described in the NTP specification and previous reports. It is based on an adaptive-parameter, hybrid PLL/FLL design which gives good performance with update intervals from a few seconds to tens of kiloseconds, depending on accuracy requirements and acceptable costs. As before, let x_k be the time and y_k be the frequency at the kth update. Let $\overline{y}_k$ be the mean oscillator frequency determined from past offsets $\{x_i\}$ and intervals $\{\tau_i\}$. In the most general formulation, an algorithm that corrects for clock time and frequency errors computes a prediction

$$\hat{x}_k = x_{k-1} + \overline{y}_{k-1}\tau. \quad (5)$$

The clock discipline operates as a negative-feedback loop to minimize $\hat{x}_k$ for all k. As each update x_k is measured, the clock time is adjusted by $-x_k$, so that it displays the correct time. In addition, the mean frequency $\overline{y}_k$ is adjusted to minimize the time adjustments in future. Subsequently, the oscillator runs at this frequency until the next update.

Between updates, which can range from seconds to hours, the clock discipline amortizes x_k in small increments at adjustment intervals $t_A = 1s$, in order to prevent timescale discontinuities and to conform to monotonic requirements. At each interval the value

$$ax + \overline{y}_k t_A \quad (6)$$

is added to the clock time, where a is a constant between zero and one ($a = 2^{-6}$ in the current implementation) and x is a variable defined below. In the NTP daemon for Unix, these adjustments are implemented by the adjtime() system call; while, in the modified kernel described in [13], correspondingly scaled adjustments are performed at each timer interrupt. The constant a is used as a gain factor in the following way. Let the value x be the residual in the adjustment whose initial value is x_k. At each interval the time is adjusted by ax and the residual by $-ax$. This provides a rapid adjustment when x is relatively large, together with a fine adjustment (low jitter) when x is relatively small.

In the original type-II PLL design of [9], the frequency is determined as past accumulations of time. In this case,

$$\overline{y}_k = b\sum_{i=1}^{k} x_i\tau_i \quad (7)$$

where b is a constant between zero and one (b=2 $^{-16}$ in the current implementation). In order to understand the dynamics, it is useful to consider the limit as τ approaches zero. In a type-II PLL, the oscillator frequency $y(t)$ is determined by the measured offset $x(t)$:

$$y(t) = ax(t) + b\int_0^t x(t)dt.$$

Since phase is the integral of frequency, the integral of the right hand side represents the overall open-loop impulse response of the feedback loop. Taking the Laplace transform, we get

$$\theta(s) = \frac{x(s)}{s}(a + \frac{b}{s})$$

where the extra pole $\frac{1}{s}$ at the origin is due to the integration which converts the frequency $y(s)$ to phase $\theta(s)$. After some rearrangement, the magnitude of the right hand side can be written

$$\frac{\omega_c^2}{s^2}\left(1 + \frac{s}{\omega_z}\right)$$

where $\omega_z = \frac{b}{a}$ and $\omega_c^2 = b$. From elementary theory, this is the transfer function of a type-II PLL which can control both time and frequency. In practice, the damping factor $\eta = \frac{\omega_c}{2\omega_z} = 4$ for good transient response. In order to simplify the presentation, this model does not include the time constant, which is used to control the loop response. The detailed design and behavior of the PLL is treated in great detail in [11] and will not be repeated here.

The new clock discipline is a hybrid PLL/FLL design in which the original PLL is used for $\tau \leq 1024$ and the FLL used otherwise. The FLL design, adapted from [5], operates in a manner identical to the PLL, except that the mean frequency $\overline{y}(t)$ is determined as an average, rather than an integral. In the FLL, $\overline{y}(t)$ is directly adjusted in order to minimize the time error $x(t)$. While a number of methods could be used to compute $\overline{y}_k$, a convenient one is the weighted average

$$\overline{y}_k = \overline{y}_{k-1} + w(y_k - \overline{y}_{k-1}) \quad (8)$$

where $w = 0.25$ is a weight factor determined by experiment. The goal of the clock discipline is to adjust the clock time and frequency so that $\hat{x}_k = 0$ for all k. To the extent this has been successful in the past, we can assume corrections prior to x_k are all zero and, in particular, $x_{k-1} = 0$. Therefore, from (4) and (8) we have

$$\overline{y}_k = \overline{y}_{k-1} + w\frac{x_k}{\tau}. \quad (9)$$

It may seem strange that the coefficient a in (6) is used in both the FLL and PLL modes. The primary reason is to avoid discontinuities when the offset x_k is very large, e.g., over 100 ms. A secondary reason is to reduce the effects of phase noise, since in the NTP model the local clock of one stratum can be used to discipline clocks at the next higher stratum. While in the PLL $a < 1$ is necessary for stability, its affect on dynamics when the FLL is in use is minor.

A key feature of the NTP design is the selection of τ in response to measured local clock stability. When the PLL is in use, the time constant is directly proportional to τ . At $\tau = 64s$, this results in a 90% time response of about 900 s and a 63% frequency response of about 3600 s, which is a useful compromise under most operating conditions. The time constant is not used when the FLL is in use.

The sum of the peer dispersion and select dispersion is used as a measure of oscillator instability in both the PLL and FLL modes. If $|\theta|$ exceeds this sum, the oscillator frequency is deviating too fast for the clock discipline to follow, so τ is reduced. In the opposite case holds for some number of updates, τ is increased. Under typical network conditions, τ hovers close to the maximum; but, on occasions when the oscillator frequency wanders more than about 1 ppm, τ quickly drops to lower values until the wander subsides.

VIII. Additional Improvements

In a perfect world, the NTP clock discipline would be implemented as an intrinsic feature of the kernel with standardized interfaces for the user and daemon processes and with a precision oscillator available as a standard option. However, during the development and deployment of NTP technology, there was considerable reluctance to intrude on kernel hardware or nonstandard software features, since this would impede portability, maintainability and perhaps reliability. In addition, manufacturers were understandably reluctant to provide a precision oscillator option, since there were not many customers to justify the development expense.

We have explored both the kernel discipline and external oscillator options. A Unix kernel implementation of the discipline has been developed for four popular workstations, the Ultrix kernel for the DEC 5000 series, the OSF/1 kernel for the DEC 3000 series, the SunOS kernel for the Sun SPARCstation series, and the HP-UX kernel for the Hewlett Packard 9000 series. As described in [12], the kernel discipline provides a time resolution of 1 μs and a frequency resolution of parts in 10^{11} (with an appropriately stable external oscillator). In addition, the modified kernels provide new system calls so that applications can learn the local clock status and error estimates determined by the daemon.

A special pulse-per-second (PPS) signal is available from sources such as cesium clocks and precision timing receivers. It generally provides much better accuracy than the serial ASCII timecode produced by an ordinary radio clock. The new kernel software uses a modem control lead of a serial port to produce an interrupt at each PPS pulse. The interrupt captures a timestamp from the local clock and computes the offset modulo 1 s. Assuming the seconds numbering of the clock counter has been determined by a reliable source, such as the ASCII timecode or even other NTP peers, the PPS offset is used to discipline the local clock. Using this feature on a typical workstation with a PPS signal from a GPS receiver, jitter is reduced to few tens of microseconds [12].

Some radio clocks can produce a special IRIG signal, which encodes the day and time as a modulated audio signal compatible with the audio codec native to some workstations. A particularly interesting feature of the NTP design described in [12] is an algorithm that processes codec samples to demodulate the signal, extract the time information and discipline the local clock. The scheme requires very few external components, but achieves a jitter comparable to the PPS signal.

However, neither the PPS or IRIG signals improve the stability of the local clock oscillator itself, since wander-induced time errors usually dominate the error budget. We have experimented with external oscillators, both using commercial bus peripherals and bus peripherals of our own design. An external clock for the Sun SBus has been constructed using FPGA technology. It includes a pair of counters that can be read directly in Unix *timeval* format and an oven-compensated precision oscillator with stability of a few parts in 10^9. In experiments where a host equipped with this device was synchronized to a primary server using NTP, the wander was measured at a few parts in 10^8, about two orders of magnitude better than the original undisciplined oscillator.

Perhaps the most useful and inexpensive approach is an auxiliary feedback loop designed to discipline the oscillator frequency directly to an external PPS signal. In this design, the PPS timestamps are used at intervals τ from 4 to 256 s to calculate a vernier frequency adjustment as in (9). This adjustment is added to the mean frequency $\overline{y}_k$ in (6). The result is that the oscillator frequency is disciplined to the PPS signal

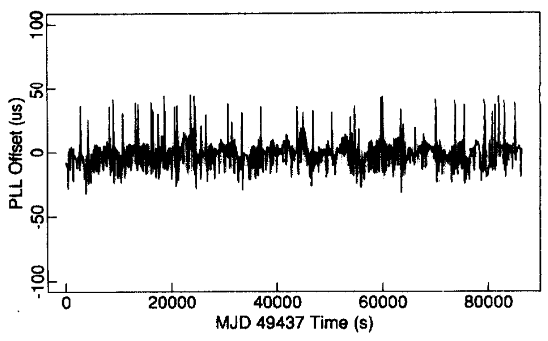


Fig. 6. Offsett with kernel PLL and PPS signal.

and the wander considerably reduced. However, the external corrections provided by NTP continue to function as usual. Measurements show that, using this scheme with a typical workstation and PPS signal from a GPS receiver results in performance comparable to the precision external oscillator.

Fig. 6 shows the performance using the native oscillator, kernel discipline and PPS signal over the Modified Julian Day (MJD) 49437.[1] In this experiment, measurements were made about every 64 s of the local clock offset relative to the PPS signal of a cesium clock and the results graphed. The server involved, a SPARCstation IPC, had about 400 NTP clients on the day of the experiment. The maximum jitter over the day is about 45 μs, primarily due to collisions between the timer interrupt and PPS signal interrupt. This represents probably the best performance possible with this generation of machines.

IX. Present Status and Deployment

Software support for NTP is available for a wide variety of workstations and mainframe computers manufactured by DEC, IBM, Hewlett Packard, Sun Microsystems, Silicon Graphics, Cray Research and many others. One manufacturer (Bancomm) markets a dedicated NTP server integrated with a GPS receiver and another (Cisco) markets a router with integrated NTP support. The software is available for public access or as a standard option in some software products. A client running this software can synchronize to one or more NTP servers or radio timecode receivers and at the same time provide synchronization to a number of dependent clients, in some cases in excess of 500, while requiring only a small fraction of available processor and memory resources.

In the most cherished of Internet traditions, the worldwide NTP synchronization subnet is not engineered in any specific way other than informal, voluntary compliance to a set of configuration rules. To protect the primary servers, potential stratum-2 peers are invited only if they serve a sizable population of stratum-3 and higher peers. Operators are cautioned that reliable service is possible only through the use of redundant servers and diverse network paths. A typical configuration for a campus serving several hundred clients includes three stratum-2 servers, each operating with two different primary

[1] MJD is derived from a scheme invented in the 16th century to number the days since an historically eclectic epoch at noon on the first day of the year 4713 BC.

servers, each of the other campus servers and at least one stratum-2 server at another institution. Department servers then operate with all three campus servers and each other, which simplifies configuration table management. Department servers offer service to client hosts, either individually or using the NTP broadcast mode.

In a previous paper [8] the number of NTP-synchronized peers was estimated at 1000 on the basis of an systematic survey of all known Internet hosts. Today, such a survey would be very difficult and probably be considered rude at best. However, it is known that there are at the time of writing about 100 NTP primary servers located in North America, Europe and the Pacific, almost half of which are advertised for public access. These peers are synchronized to national time standards using all known computer-readable time-dissemination services in the world, including the U.S. (ACTS, WWVB, WWV and WWVH), Canada (CHU), U.K. (MSF), Germany (DCF77) and France (TDF), as well as the GPS, OMEGA and LORAN navigation systems, and the GOES environmental satellite. In addition, NTP primary servers at NIST and USNO, as well as the national time standards laboratories of Norway and Australia, are directly synchronized to national standard clock ensembles.

It is difficult to estimate the number of NTP secondary (stratum-2 and higher) peers in the global Internet. A recent informal estimate puts the total number of Internet hosts over 1.7 million. An intricate check of the monitoring information maintained by some public NTP servers reveals about 8,000 stratum-2 and stratum-3 dependents; however, this survey grossly undercounts the population, since only a fraction of the servers retain this information and many thousands of known dependents are hidden deep inside corporate networks, either independently synchronized or carefully peeking out through access-controlled gateways. Informal estimates based on anecdotal information provided by various network operators suggest the total number of hosts running NTP is probably in excess of 100 000.

The earlier survey presented error measurements for various paths between NTP primary servers in the U.S. and concluded reliable time synchronization could be obtained "...in the order of a few tens of milliseconds over most paths in the Internet of today." As reported in [12], while there are exceptions, this claim remains generally valid in the much larger worldwide Internet of today. With the software and hardware improvements described herein for the NTP Version 3 specification and implementations, and with suitable allowance for differential delays, most places in the worldwide Internet are able to maintain an accuracy better than 10 ms and those on LAN's and high speed WAN's better than 1 ms.

X. Current Work and Future Plans

As time moves on, so do NTP versions. A summary of current work and future plans for Version 4 of the protocol are given in [13]. They include refinement of the broadcast/multicast protocol modes, automated peer discovery and implementation of a new feature called *distributed mode*.

In cases where a moderate loss in accuracy can be tolerated, such as most workstations on a LAN subnet, the NTP broadcast mode greatly simplifies client configuration and network management. In this mode, client workstations automatically survey their environment and configure themselves without requiring pre-engineered configuration files. After joining the subnet, a client listens for broadcasts from one or more servers on the LAN. Upon hearing one, the client exchanges messages with the server in order to determine the best time and calibrate the broadcast propagation delay. When calibration is complete, generally after a few message exchanges, the client again resumes listening for broadcasts. In broadcast mode the NTP filter, selection and combining algorithms operate as in the client/server modes, with resulting accuracy usually in the order of a few milliseconds on an Ethernet.

We have recently extended the NTP broadcast mode to use IP multicast facilities [3] for wide-area time distribution. The NTP multicast mode operates in the same way as the broadcast mode, so that clients can discover servers wherever IP multicast facilities and connectivity to the Internet MBONE are available. At the present time, experimental servers have been established in the U.S., U.K., and Germany, with clients in these and other countries. The accuracies that have been achieved vary widely, depending on the particular server and path. For instance, with typical U.K. servers and clients in the U.S., the accuracies vary from 10 to 100 ms, depending on particular server configuration and ambient network traffic levels.

While we have proof of concept that time distribution using IP multicast is practical, there are many remaining problems to be resolved, such as how to avoid sending messages all over the world from possibly many multicast servers, how to authenticate and select which ones a particular client or client population chooses to believe, and how to allocate and manage possibly many multicast group addresses.

In other future plans, we expect to make use of IP multicast to maintain timekeeping data not only between peers, but between other members of the synchronization subnet as well. This scheme, called distributed mode, will allow additional opportunities to discover potential peers, as well as reduce errors due to differential delays. In addition, we expect to participate in a comprehensive design exercise involving the Domain Name System to discover domain-based time servers and to distribute authentication information.

XI. Summary

This paper has presented an in-depth analysis of certain issues important to achieve accurate, stable and reliable time synchronization in a computer network. These issues include the design of the synchronization protocol, the local clock, and the algorithms used to filter, select and combine the reading of possibly many peer clocks. The intersection algorithm presented in this paper is designed to distinguish truechimers from among a population possibly including falsetickers. The local clock is modeled as a disciplined oscillator and implemented as a hybrid PLL/FLL feedback loop. The behavior of the model is controlled automatically for oscillators of varying stability and network paths of widely varying characteristics.

The NTP Version 3 implementations have been widely deployed to probably over 100 000 installations in the Internet of today. Surveys using previous versions of NTP have found synchronization to UTC can be generally maintained to within a few tens of milliseconds. With NTP Version 3 and the hardware and software improvements described in this paper, synchronization can be generally maintained with some exceptions to within 10 ms on typical Internet paths and within 1 ms on LAN's and WAN's with high speed (over 1 Mb/s) transmission paths. The exceptions are in all known cases due to either severe network congestion or differential path delays, which in principle can be calibrated out.

References

[1] D. W. Allan, Time and frequency (time-domain) estimation and prediction of precision clocks and oscillators. *IEEE Trans. on Ultrasound, Ferroelectrics, and Frequency Contr.* vol. UFFC-34, pp. 647-654, Nov. 1987. Also in "Characterization of clocks and oscillators," D. B. Sullivan, D. W. Allan, D. A. Howe, and F.L. Walls, Eds., U.S. Dep. Commerce, 1990, NIST Tech. Note 1337, pp. 121–128.

[2] *Digital Time Service Functional Specification Version T.1.0.5*, Digital Equipment Corp., 1989.

[3] S. E. Deering and D. R. Cheriton, "Multicast routing in datagram internetworks and extended LAN's," *ACM Trans. Comput. Syst.* , vol. 8, no. 2, pp. 85-100, May 1990.

[4] J. Levine, M. Weiss, D. D. Davis, D. W. Allan, and D. B. Sullivan, "The NIST automated computer time service," *J. Res. Nat. Inst. Standards and Technol. 94,* Sept.–Oct. 1989, vol. 5, pp. 311–321.

[5] J. Levine, "An algorithm to synchronize the time of a computer to universal time," *IEEE Trans. Networks*, vol. 3, pp. 42-50, Feb. 1995.

[6] W. C. Lindsay and A. V. Kantak, "Network synchronization of random signals," *IEEE Trans. Commun.*, vol. COM-28, pp. 1260-1266, Aug. 1980.

[7] K. Marzullo, and S. Owicki. "Maintaining the time in a distributed system," *ACM Oper. Syst. Rev.*, vol. 19, no. 3, pp. 44-54, July 1985.

[8] D. L. Mills, "Measured performance of the network time protocol in the Internet system," *ACM Comput. Commun. Rev.*, vol. 20, no. 1, pp. 65-75, Jan. 1990.

[9] ——, "Internet time synchronization: The network time protocol," *IEEE Trans. Commun.*, vol. 39, pp. 1482-1493, Oct. 1991. Also in *Global States and Time in Distributed Systems*, Z. Yang, and T. A. Marsland, Eds. Los Alamitos, CA: IEEE Press, pp. 91-102.

[10] ——, "Network time protocol (Version 3) specification, implementation and analysis," Univ. Delaware, DARPA Network Working Group Rep. RFC-1305, Mar. 1992, 113 pp.

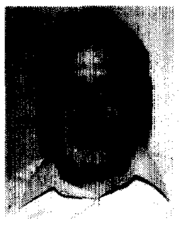
[11] ——, "Modeling and analysis of computer network clocks," Elec. Eng. Dep. Rep. 92-5-2, Univ. Delaware, May 1992, 29 pp.

[12] ——, "Precision synchronization of computer network clocks," *ACM Comput. Commun. Rev.*, vol. 24, no. 2, 16 pp., Apr. 1994.

[13] ——, "Network time protocol version 4 proposed changes," Elec. Eng. Dep. Rep. 94-10-2, Univ. Delaware, Oct. 1994, 46 pp.

[14] *NIST Time and Frequency Dissemination Services* . NBS Special Publication 432 (Revised 1990), Nat. Inst. of Sci. and Technol., U.S. Dep. of Commerce, 1990.

[15] S. R. Stein, "Frequency and time - their measurement and characterization (Chapter 12)," in *Precision Frequency Control, Vol. 2*, E.A. Gerber and A. Ballato Eds. New York: Academic, 1985, pp. 191-232, pp. 399-416. Also in: *Characterization of Clocks and Oscillators*, D. B. Sullivan, D. W. Allan, D. A. Howe, and F. L. Walls, Eds. Nat. Inst. Standards and Technol. Tech. Note 1337, U.S. Government Printing Office, Jan., 1990, TN61-TN119.



David L. Mills (M'86) received the M.S. degree in electrical engineering and the Ph.D. degree in computer science from the University of Michigan in 1963 and 1971, respectively.

He is a Professor of Electrical Engineering at the University of Delaware. His research activities are in the areas of computer network architecture and modeling, protocol engineering and experimental studies using the Internet system. He was formerly a Directorat M/A-COM Linkabit and a Senior Scientist at COMSAT Laboratories, where he worked in the areas of computer network and satellite systems.

Dr. Mills has been a member of the ACM since 1964.