Improved Algorithms for Synchronizing Computer Network Clocks[1,2,3]

David L. Mills
Electrical Engineering Department
University of Delaware

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 ensemble from among a set of peer clocks and combine their differences to produce a 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 message-exchange intervals 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 software in order to realize submillisecond accuracies with fast workstations and networks.

Keywords: computer network synchronization, clock synchronization, distributed protocol, disciplined oscillator.

1. 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), 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 [NIS90].

Computer clocks can be synchronized to within a few tens of milliseconds in the global Internet of today [MIL90]. 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, 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 [MIL92b] and [MIL93].

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 Synchroni-

Copyright (c) 1994, Association for Computing Machinery.

1 Sponsored by: Advanced Research Projects Agency under NASA Ames Research Center contract NAG 2-638, National Science Foundation grant NCR-93-01002 and U.S. Navy Surface Weapons Center under Northeastern Center for Engineering Education contract A30327-93.

2 Author's address: Electrical Engineering Department, University of Delaware, Newark, DE 19716; Internet mail: mills@udel.edu.

3 Reprinted from: Mills, D.L. Improved algorithms for synchronizing computer network clocks. *Proc. ACM SIGCOMM 94 Symposium* (London UK, September 1994), 317-327.

zation Service (DTSS) [DEC89] could be used as well. After a review of terms and notation in Section 2, Section 3 gives an overview of NTP. Section 4 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 5 and 6 Section 5 describes the intersection algorithm, which is used to separate the *truechimers*, which represent correct clocks, from *falsetickers*, which may not. Section 6 contains a detailed analysis of the local clock model, which functions as a disciplined oscillator and is implemented as a phase-locked loop. These algorithms are primarily responsible for the increased accuracy and reliability of the NTP Version 3 protocol compared to previous versions.

Section 7 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 three different Unix operating system kernels which provide extremely precise control of the oscillator time and frequency. Section 8 discusses the present status of NTP in the Internet, Section 9 outlines future plans, and Section 10 is a summary of this paper.

2. 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 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:

$$T(t) = T(t_0) + R(t_0)[t - t_0] + \tfrac{1}{2}D(t_0)[t - t_0]^2 + x(t) , \qquad (1)$$

where $T(t_0)$ is the time at some previous epoch t_0, $R(t_0)$ is the frequency and $D(t_0)$ is the drift (first derivative of frequency) per unit time. It is conventional to represent both absolute and relative (offset) values for T and R using the same letters, where the particular use is clear from context. 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 x, usually in terms of frequency or phase spectra or measurements of variance.

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 $T_{ij}(t) \equiv T_i(t) - T_j(t)$ at a particular epoch t, while the *frequency offset* is the frequency difference between them $R_{ij}(t) \equiv R_i(t) - R_j(t)$. It follows that $T_{ij} = -T_{ji}$, $R_{ij} = -R_{ji}$ and $T_{ii} = R_{ii} = 0$ for all t. In this paper reference to simply "offset" means time offset, unless indicated otherwise. The term *jitter* refers to differences in subsequent time offset measurements, while the term *wander* refers to differences in successive frequency offset measurements. 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 are synchronized to designated *time servers*, which are timekeeping systems belonging to a *synchronization subnet*, in which each server disciplines its local clock to other clocks in the subnet. 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.

3. 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. It is evolved from the Time Protocol [POS83] and the ICMP Timestamp Message [DAR81b], but is specifically designed for high accuracy, stability and reliability, even when used over typical Internet paths involving multiple gateways and unreliable networks. This section contains an overview of the architecture and algorithms used in NTP. A detailed description of the architecture and service model is contained in [MIL91], while the current protocol formal specification, designated NTP Version 3, is defined in RFC-1305 [MIL92a]. A subset of the protocol, designated Simple Network Time Protocol (SNTP), is presented in RFC-1361 [MIL92c]. A security analysis of NTP is presented in [BIS90].

NTP and its implementations have evolved and proliferated in the Internet over the last decade, with NTP Version 2 adopted as an Internet Standard (Recommended) [MIL89] and its successor NTP Version 3 adopted as a Internet Standard (Draft). NTP is built on the Internet Protocol (IP) [DAR81a] and User Datagram Protocol (UDP) [POS80], which provide a connectionless transport mechanism; however, it is readily adaptable to other protocol suites. The protocol can operate in several scenarios involving unicast and broadcast modes, private workstations, public servers and various subnet configurations. A lightweight association-management capability, including dynamic reachability and

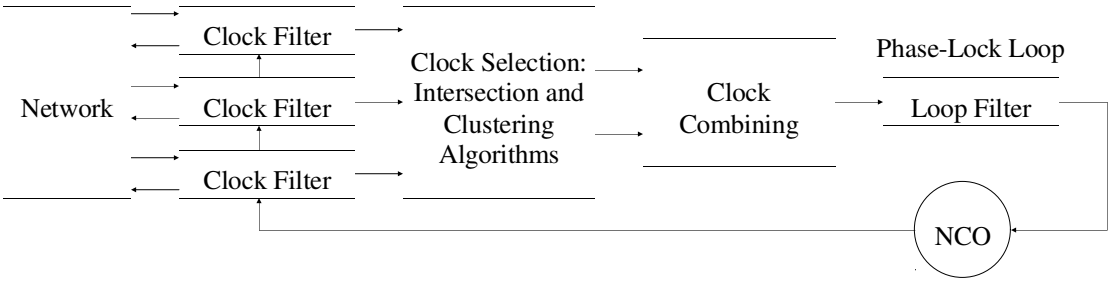

Figure 2. Network Time Protocol

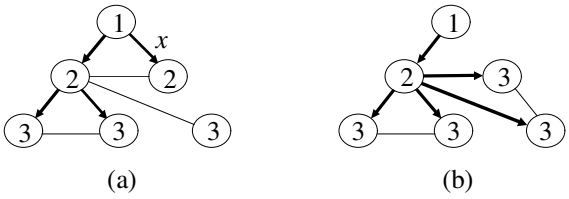

Figure 1. Subnet Synchronization Topologies

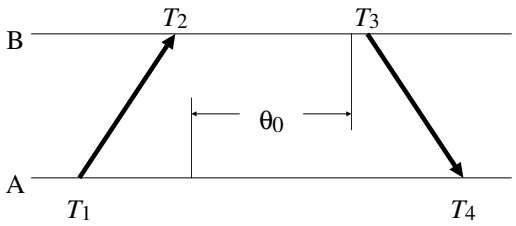

Figure 3. Measuring Delay and Offset

variable poll-interval mechanisms, is used to manage state information and reduce resource requirements. Optional features include message authentication based on DES [DES77] and MD5 [RSA90] algorithms, as well as provisions for remote control and monitoring.

In NTP one or more primary servers synchronize directly to external reference sources such as radio clocks. Secondary time servers synchronize to the primary servers and others in the synchronization subnet. A typical subnet is shown in Figure 1a, in which the nodes represent subnet servers, with normal level or stratum numbers determined by the hop count from the primary (stratum 1) server, and the heavy lines the active synchronization paths and direction of time information flow. The light lines represent backup synchronization paths where time information is exchanged, but not necessarily used to synchronize the local clocks. Figure 1b shows the same subnet, but with the line marked *x* out of service. The subnet has reconfigured itself automatically to use backup paths, with the result that one of the servers has dropped from stratum 2 to stratum 3. In practice each NTP server synchronizes with several other servers in order to survive outages and Byzantine failures using methods similar to those described in [SHI87].

Figure 2 shows the overall organization of the NTP time server model, which has much in common with the phase-lock methods summarized in [RAM90]. *Timestamps* exchanged between the server and each of possibly many other subnet peers at intervals ranging from one to 17 minutes are used to determine individual roundtrip delays and clock offsets, as well as provide reliable error bounds. As shown in the figure, the computed delays and offsets for each peer are processed by the clock filter algorithm to reduce incidental jitter. As described in [MIL92a], this algorithm selects from among the last several samples the one with minimum delay and presents the associated offset as the output.

The clock selection algorithm determines from among all peers a suitable subset capable of providing the most accurate and trustworthy time using principles similar to those described in [VAS88]. This is done using a cascade of two subalgorithms, one based on interval intersections to cast out faulty peers 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 using the algorithm described in [MIL92a] and then processed by a phase-lock loop (PLL) using the algorithms described in [MIL92b]. In the PLL the combined offset is processed by the loop filter to control the numeric-controlled oscillator (NCO) frequency. The NCO is implemented as an adjustable-rate counter using a combination of hardware and software components. It furnishes the phase (timing) reference to produce the timestamps used in all timing calculations.

Figure 3 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 rate. Let

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

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 [LIN80]. Among its advantages are that errors due to missing or duplicated messages can be avoided.

In [MIL92b] 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, as defined in the NTP Version 3 specification, represents a conservative estimate of the errors due to each of the above causes.

In [MIL92b] it is shown that, given the ε calculated as above, $\lambda \equiv \frac{\delta}{2} + \varepsilon$ represents 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 be somewhere in the interval

$$\theta - \lambda \le \theta_0 \le \theta + \lambda , \qquad (3)$$

which is called the *confidence interval.*

The ε and λ are used as metrics in the various algorithms presented in following sections. They determine which clocks are selected by the clock selection and clustering algorithms, the weight factors used by the clock combining algorithm, and in calculating 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 by cut-and-try. 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.

4. Clock Filter, Combining and Clustering Algorithms

The clock filter, clustering and combining algorithm shown in Figure 2 operate essentially as described previously in [MIL91], however all three have been refined and defined formally in [MIL92a]. 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* $\hat{\delta}$, *peer offset* $\hat{\theta}$ and *peer dispersion* $\hat{\varepsilon}$. A discussion of the design approach, implementation and performance assessment is given in [MIL91] and will not be repeated here. However, the design 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, 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 synchronization distance λ. For each peer it computes the select dispersion, defined as the total weighted time differences 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 prespecified 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.

The clock combining algorithm averages the time offsets of the peers selected by the clustering algorithm using a system of weights, where the weight of each contributing peer is determined by its dispersion as a fraction of the total dispersion of all peers. As incorporated in NTP Version 3, the dispersions are augmented in the same way as in the clock

filter algorithm and the system dispersion determined as the sum of the weighted dispersions.

5. Intersection Algorithm

When a number of peer clocks are involved as in Figure 2, 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 [MAR85] 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.

Figure 4 illustrates the operation of this algorithm with a scenario involving four clocks A, B, C and D, with the peer offset $\hat{\theta}$ (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 DTSS 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 4, it computes the intersection for $m = 4$ clocks, three of which turn out to be correct and one not. The interval marked DTSS 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. 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

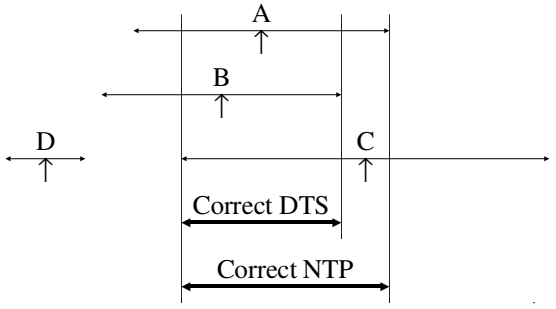

Figure 4. Confidence Intervals and Intersections

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 Figure 4) 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 4, 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 misconfigurations and defective implementations. In the NTP Version 3 implementation only the ten peers with the lowest synchronization distance 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 *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 falseticker clocks, 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 confidence 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.

Code	Server (Location)	Stratum	Source	Offset	Delay	Dispersion	Distance
*	GPS	0	GPS	-0.01	0.00	0.00	0.00
+	barnstable	1	GPS	-0.27	2.53	0.00	1.27
+	rackety	1	GPS	-0.30	3.07	0.37	1.91
+	mizbeaver	1	GPS	-0.38	4.06	0.00	2.03
	churchy	2	pogo	-0.66	3.83	4.53	6.45
	porkypine	2	pogo	0.16	2.27	15.66	16.80
	baldwin	3	ACTS	-0.42	2.70	16.59	17.01
x	time_a (Boulder, CO)	1	ATOM	7.97	61.92	2.11	33.07
x	er-gw (Switzerland)	1	DCF77	-3.99	117.14	22.37	80.94
-	lucifer (Germany)	1	GPS	-1.10	174.71	20.65	108.00
-	grundoon	1	WWV	-0.80	220.67	0.85	111.19
x	swifty (Australia)	1	ATOM	-13.71	276.57	13.72	152.00

Table 1. Peer Configuration for Server pogo

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 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 1 shows a typical configuration for NTP primary server *pogo*. The peers are located in Europe, Australia and the National Institute of Standards and Technology (NIST) in Boulder, CO, as shown; the others are located at the University of Delaware. The server identified as GPS and assigned pseudo-stratum zero is a precision timing receiver synchronized by the Global Positioning System (GPS) directly connected to pogo. Since pogo is operating at stratum 1, the servers marked stratum 2 and higher would be considered for synchronization only if the GPS receiver and all other stratum-1 sources fail. The synchronization source for each server is shown by dissemination service if stratum 0 or 1 or by another server if higher. GPS, DCF77 and WWV use radio and satellite, ATOM is a national standard clock ensemble and ACTS is the Automated Computer Time Service operated by NIST [LEV89].

The offset, delay, dispersion and synchronization distance for each peer are shown in the table, all in milliseconds. The synchronization status is shown by the Code column. The peers with no symbol in that column are disallowed by the sanity checks, since they are at a lower stratum than the server. Those servers with a symbol are eligible for processing by the intersection algorithm and then the clustering algorithm; however, those marked "x" have been discarded by the intersection algorithm as falsetickers, while the peers marked "-" have been discarded by the clustering algorithm as 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.

6. The Local Clock Algorithm

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 one or two ppm due to room temperature variations. The NTP local clock algorithm 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 modelling the local clock and its adjustment mechanism as a *disciplined oscillator*. In this type of oscillator 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 messages can be used without degrading accuracy.

A disciplined oscillator can be implemented as a type-II, phase-lock loop (PLL) as shown in Figure 5. The variable ω_r represents the reference signal and ω_c the numeric-controlled oscillator (NCO) 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 transfer function $F(s)$, produces a NCO correction θ_c, which controls the oscillator frequency ω_c and thus its phase. The characteristic behavior of this PLL model, which is determined by the $F(s)$, is studied in many textbooks and summarized in [MIL92b].

The Unix 4.3bsd clock model requires a periodic 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 the tick does not evenly divide the second in microseconds, an additional increment *fixtick* is added to *time* once each second to make up the difference.

The Unix clock can actually run at three different rates, one at the intrinsic oscillator frequency, another at a slightly higher frequency and a third at a slightly lower frequency. The adjtime() system call is used to adjust the local clock to a given time offset. The argument is used to select which of the three rates and how long Δt to run at that rate, in order to amortize the specified offset. The NTP local clock uses the Unix clock model as the NCO and implements the transfer function $F(s)$ using a set of recurrence equations described below. A capsule overview of the design extracted from [MIL92b] may be helpful in understanding how the model operates.

In the NTP local clock algorithm, the Unix clock is continuously adjusted in small increments at fixed *adjustment intervals* $\sigma = 1$ s. The increments are computed from state variables representing the *frequency offset f* and *phase offset g* of the local clock. These variables are in turn determined from NTP messages received at nominal *update intervals* μ, which are variable from 16 to 1024 s. The *time constant* τ is the adaptive parameter of the model; it is adjusted as a function of μ and the dispersion ε determined by the clock filter and clock selection algorithms.

In the following, successive generations of the algorithm are numbered from zero and shown in parentheses. All state variables are initialized at $i = 0$ to zero. After an interval $\mu(i) = t(i) - t(i-1)$ $(i > 0)$ from the previous update, the ith update arrives at time $t(i)$ and the time offset $\theta_d(i)$ is determined as above. Then, new values for the $f(i+1)$ and $g(i+1)$ state variables are computed:

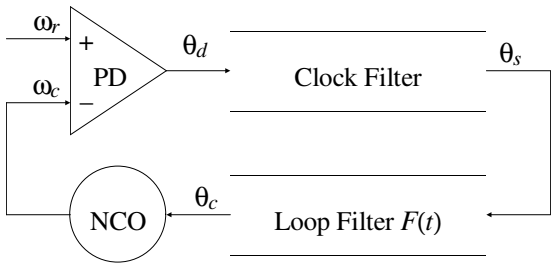

Figure 5. Phase-Lock Loop Model

$$f(i+1) = f(i) + \frac{\mu(i)\theta_d(i)}{\tau^2}, \quad g(i+1) = \frac{\theta_d(i)}{\tau},$$

It is convenient to set the temporary variable $a = g(i+1)$. At each adjustment interval σ the quantity

$$\frac{a}{K_g} + \frac{f(i+1)}{K_f} \tag{4}$$

where K_g and K_f are fixed constants, is added to the local clock time and the quantity $\frac{a}{K_g}$ is subtracted from a. For convenience, let n be the greatest integer in $\frac{\mu(i)}{\sigma}$; that is, the number of adjustments that occur in the ith interval. Thus, at the end of the ith interval just before the i+1th update, the local clock offset is:

$$\theta_c(i+1) = \theta_c(i) + [1 - (1 - \frac{1}{K_g})^n]\, g(i+1) + \frac{n}{K_f} f(i+1).$$

As described in [MIL92b], the NTP daemon simulates the PLL loop filter and NCO using the above recurrence relations. At each adjustment interval, the offset (4) is provided to the kernel using the adjtime() system call. With the parameters $K_g = 2^6$ and $K_f = 2^{16}$ used in the NTP Version 3 implementation and default $\tau = 4$, the PLL converges to a step change in phase in about 900 s with less than 7 percent overshoot. Through the use of carefully chosen parameter values and arithmetic procedures, almost all multiply and divide operations are done with economical shifts.

However, using the Unix clock model, the residual jitter can exxceed 100 μs when the timer interval does not evenly divide the second in microseconds. Also, since the adjustment process must complete within 1 s, larger adjustments must be parceled out in a series of adjtime() calls. Finally, provisions must be made to compensate for the roundoff error in computing Δt. These factors add to the error budget, increase system overhead and complicate the daemon implementation. A solution to these problems is presented in a later section.

As reported in [MIL93], the major source of error in most configurations is the stability of the local clock oscillator. For example, a typical uncompensated quartz oscillator varies 1 ppm for each degree Celsius and has a short term stability in

the range 0.1 ppm to 1 ppm. A key feature of the NTP design is the behavior of τ in response to local oscillator stability. When operated with a relatively small τ, the PLL adapts quickly to changes in the local oscillator frequency, but has poor long term stability. When operated with a relatively large τ, the PLL produces the most accurate time, but adapts slowly to changes in the local oscillator frequency.

For the best accuracy and reliability, it is necessary to adjust τ on a continuous basis as a function of measured stability. The stability of a free-running frequency source is commonly characterized by a statistic called *Allan variance* [ALL87], which is defined as follows. Consider a series of time offsets measured between an oscillator and some external standard. Let $\theta(i)$ be the ith measurement and T be the interval between measurements. Define the *fractional frequency*

$$y(i) \equiv \frac{\theta(i) - \theta(i-1)}{T} . \quad (5)$$

Now, consider a sequence of n independent fractional frequency samples $y(j)$ ($j = 1, 2, \ldots, n$). If the averaging interval T is the same as the interval between measurements, the 2-sample Allan variance can be defined

$$\sigma_y^2(T) \equiv \frac{1}{2(n-1)} \sum_{j=1}^{n-1} [y(j+1) - y(j)]^2 .$$

The Allan variance $\sigma_y^2(T)$ (or Allan deviation $\sigma_y(T)$) is particularly useful when designing the local clock algorithm, since it determines the optimum PLL time constants and update intervals. Figure 6 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 oscillator 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 Figure 6 are logarithmic. The characteristic falls rapidly from the lowest T, where the errors are due primarily to phase jitter, to a minimum of 0.1 ppm and then rises again to about 0.2 ppm at the highest, where the errors are due primarily to random-walk frequency variations. The conclusion to be drawn is that using integration intervals much below or much above $T = 1000$ s do not improve the oscillator stability.

In the NTP design the PLL time constant τ, update interval μ and integration interval T are directly proportional to each other. The default $\tau = 4$ corresponds to $\mu = 64$s and $T = 900$s, which is close to optimum under most operating conditions. In this design τ can be varied from 1 to 64 and μ and T scale in direct proportion. In order to minimize network load, it is ordinarily desirable to operate with the largest μ

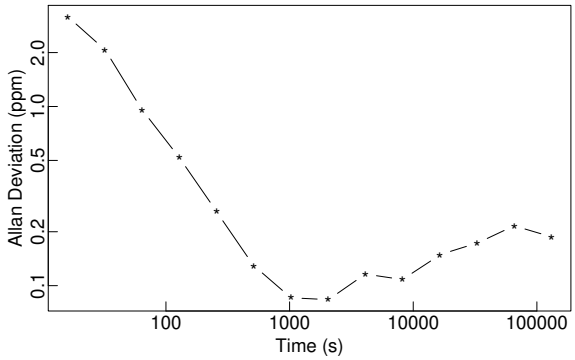

Figure 6. Allan Variance of Typical Local Oscillator

consistent with good accuracy. However, while the PLL can in principle eliminate residual timing errors due to a constant frequency offset, it is quite sensitive to changes in frequency, such as might occur due to room temperature surges. For instance, a 2-ppm step change in frequency causes a surge of 600 μs at $\tau = 4$. In addition, the amplitude of the surge scales directly with τ and the temperature change. Therefore, in order to avoid occasional large errors, it is necessary to adjust τ automatically to match prevailing conditions.

In the NTP Version 3 implementation, the product $\varepsilon\tau$ is used as a measure of oscillator instability. If the absolute offset $|\theta|$ exceeds $\varepsilon\tau$ by an experimentally determined threshold, the oscillator frequency is deviating too fast for the PLL to follow. so τ is reduced. In the opposite case holds for some number of updates, τ is increased. The threshold is adjusted so that, under typical 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.

7. Additional Improvements

In a perfect world the NTP PLL model would be implemented as an intrinsic feature of the kernel with standardized interfaces for the user and daemon processes and with a precision local clock 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 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 PLL and external oscillator technology. A Unix kernel implementation of the PLL has in fact been developed for three popular workstations, the Ultrix kernel for the DECstation 5000 series, the OSF/1 kernel for the 3000 AXP Alpha series and the SunOS 4 kernel for the SPARCstation series. As described in [MIL93], the kernel PLL 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, maximum error and estimated error 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 precision 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 the PPS signal transition. The software captures a timestamp at each transition and computes the residue 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 an NTP peer, the PPS offset is used to control the local clock via the NTP or kernel PLL. Using this feature on a typical workstation with a PPS signal from a GPS receiver, jitter is reduced to few tens of microseconds [MIL93].

Some radio clocks can produce a special IRIG signal, which encodes the day and time as a modulated audio signal that is compatible with the audio codec native to some workstations. A particularly interesting feature of the NTP design described in [MIL93] is an algorithm that processes codec samples to demodulate the signal, extract the time information and control the local clock via the usual NTP algorithms. 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 less than the original undisciplined oscillator.

Perhaps the most novel and useful approach is an auxiliary feedback loop designed to discipline the local clock oscillator frequency to an external PPS signal. In this design the time difference between a timestamp captured at a PPS interrupt and the hardware microsecond counter is computed at intervals from 4 to 256 s. If $\theta(i)$ is the difference at interval i, the fractional frequency $y(i)$ is calculated as in (5) and used to update a frequency estimate $\bar{y}$:

$$\bar{y}(i+1) = \bar{y}(i) + \frac{\bar{y}(i) - y(i)}{K_y} ,$$

where $K_y = 4$ is an experimentally determined averaging factor. Equation (4) above is then modified to include the frequency estimate $\bar{y}$,

$$\frac{a}{K_g} + \frac{f(i+1)}{K_f} + \bar{y}(i+1) .$$

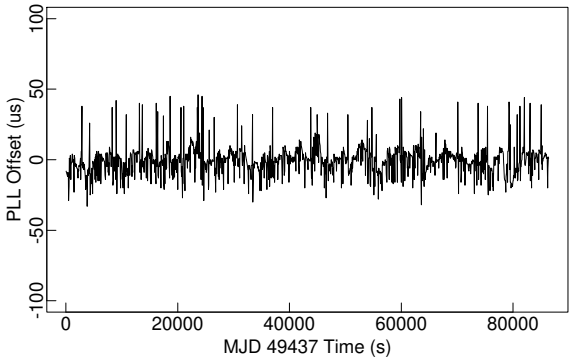

Figure 7. Offset with Kernel PLL and PPS signal

The result is that the oscillator frequency is disciplined to the 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 timing receiver results in performance comparable to the precision external oscillator.

Figure 7 shows the performance using the kernel PLL and PPS discipline, but no external clock, over the UTC day 23 March 1994, corresponding to Modified Julian Day (MJD) 49437. 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 particular machine.

8. Present Status and Deployment

Software support for NTP is available for a wide variety of workstations and mainframe computers manufactured by Digital, 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 400, 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 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 [MIL90] the number of NTP-synchronized peers was estimated at 1,000 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 Rim, about a third 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. (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 Geosynchronous Orbiting Environmental Satellite (GOES). In addition, NTP primary servers at the national time standards laboratories of the U.S., 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 synchronized 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 [MIL93], while there are exceptions, this claim remains generally valid in the 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 LANs and high speed WANs better than 1 ms.

9. Future Plans

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 configure themselves without requiring pre-engineered network configurations or client configuration files. Upon joining the subnet, a client listens for broadcasts from one or more servers on the subnet. Upon hearing one, the client enters client/server mode in order to calibrate the one-way delay between the server and client. When calibration is complete, generally after a few messages, the client resumes listen-only mode. 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 [DEE90] 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 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, the accuracies vary from 10 to 100 ms.

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 multicasting to maintain timekeeping data not only between peers, but between other members of the synchronization subnet as well. This 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.

10. 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 correct peer clocks from among a population possibly including faulty ones. The local clock is modelled as a disciplined oscillator and implemented as an adaptive-parameter, phase-lock 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 LANs and WANs with high speed (over 1 Mbps) 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.

11. References

[ALL87] Allan, D.W. Time and frequency (time-domain) estimation and prediction of precision clocks and oscillators. *IEEE Trans. on Ultrasound, Ferroelectrics, and Frequency Control UFFC-34, 6* (November 1987), 647-654. Also in: Sullivan, D.B., D.W. Allan, D.A. Howe and F.L. Walls (Eds.). *Characterization of Clocks and Oscillators*. NIST Technical Note 1337, U.S. Department of Commerce, 1990, 121-128.

[BIS90] Bishop, M. A security analysis of the NTP protocol. Report to the Privacy and Security Research Group. Department of Mathematics and Computer Science, Dartmouth College, June 1990.

[DAR81a] Defense Advanced Research Projects Agency. Internet Protocol. DARPA Network Working Group Report RFC-791, USC Information Sciences Institute, September 1981.

[DAR81b] Defense Advanced Research Projects Agency. Internet Control Message Protocol. DARPA Network Working Group Report RFC-792, USC Information Sciences Institute, September 1981.

[DEC89] Digital Time Service Functional Specification Version T.1.0.5. Digital Equipment Corporation, 1989.

[DEE90]Deering, S.E., and D.R. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Trans. Computing Systems 8, 2* (May 1990), 85-100.

[DES77]*Data Encryption Standard*. Federal Information Processing Standards Publication 46. National Bureau of Standards, U.S. Department of Commerce, 1977.

[LEV89] Levine, J., M. Weiss, D.D. Davis, D.W. Allan, and D.B. Sullivan. The NIST automated computer time service. *J. Research National Institute of Standards and Technology 94, 5* (September-October 1989), 311-321.

[LIN80] Lindsay, W.C., and A.V. Kantak. Network synchronization of random signals. *IEEE Trans. Communications COM-28, 8* (August 1980), 1260-1266.

[MAR85] Marzullo, K., and S. Owicki. Maintaining the time in a distributed system. *ACM Operating Systems Review 19, 3* (July 1985), 44-54.

[MIL89] Mills, D.L. Network Time Protocol (version 2) - specification and implementation. DARPA Network Working Group Report RFC-1119, University of Delaware, September 1989.

[MIL90] Mills, D.L. Measured performance of the Network Time Protocol in the Internet system. *ACM Computer Communication Review 20, 1* (January 1990), 65-75.

[MIL91] Mills, D.L. Internet time synchronization: the Network Time Protocol. *IEEE Trans. Communications COM-39, 10* (October 1991), 1482-1493. Also in: Yang, Z., and T.A. Marsland (Eds.). *Global States and Time in Distributed Systems*, IEEE Press, Los Alamitos, CA, 91-102.

[MIL92a] Mills, D.L. Network Time Protocol (Version 3) specification, implementation and analysis. DARPA Network Working Group Report RFC-1305, University of Delaware, March 1992, 113 pp.

[MIL92b] Mills, D.L. Modelling and analysis of computer network clocks. Electrical Engineering Department Report 92-5-2, University of Delaware, May 1992, 29 pp.

[MIL92c] Mills, D.L. Simple Network Time Protocol (SNTP). DARPA Network Working Group Report RFC-1361, University of Delaware, August 1992, 10 pp.

[MIL93]Mills, D.L. Precision synchronizatin of computer network clocks. Electrical Engineering Department Report 93-11-1, University of Delaware, November 1993, 66 pp.

[NIS90] *NIST Time and Frequency Dissemination Services*. NBS Special Publication 432 (Revised 1990), National Institute of Science and Technology, U.S. Department of Commerce, 1990.

[POS80] Postel, J. User Datagram Protocol. DARPA Network Working Group Report RFC-768, USC Information Sciences Institute, August 1980.

[POS83] Postel, J. Time protocol. DARPA Network Working Group Report RFC-868, USC Information Sciences Institute, May 1983.

[RAM90] Ramanathan, P., K.G. Shin and R.W. Butler. Fault-tolerant clock synchronization in distributed systems. *IEEE Computer 23, 10* (October 1990), 33-42.

[RSA90] MD5 Message-Digest Algorithm, RSA Data Security, Inc., 1990.

[SHI87] Shin, K.G., and P. Ramanathan. Clock synchronization of a large multiprocessor system in the presence of malicious faults. *IEEE Trans. Computers C-36, 1* (January 1987), 2-12.

[VAS88] Vasanthavada, N., and P.N. Marinos. Synchronization of fault-tolerant clocks in the presence of malicious failures. *IEEE Trans. Computers C-37, 4* (April 1988), 440-448.