# The Network Computer as Precision Timekeeper[1,2]

David L. Mills[3]
Electrical Engineering Department
University of Delaware

**Abstract**

This paper describes algorithms to discipline a computer clock to a source of standard time, such as a GPS receiver or another computer synchronized to such a source. The algorithms are designed for use in the Network Time Protocol (NTP), which is used to synchronize computer clocks in the global Internet. They have been incorporated in the NTP software for Unix and Windows and, for the highest accuracy, in the operating system kernels for Sun, DEC and HP workstations. RMS errors on LANs are usually less than 10 µs and on global Internet paths usually less than 5 ms. However, rare disruptions of one kind or another can cause error spikes up to 100 µs on LANs and 100 ms on Internet paths.

Keywords: computer network time synchronization, clock discipline algorithm, pulse-per-second steering

## 1. Introduction

General purpose workstation n computers are becoming faster each year, with processor clocks now operating at 300 MHz and above. Computer networks are becoming faster as well, with speeds of 622 Mbps available now and 2.4 Gbps being installed. Using available technology and existing workstations and Internet paths, it has been demonstrated that computers can be reliably synchronized to better than a millisecond in LANs and better than a few tens of milliseconds in most places in the global Internet [3]. This technology includes the Network Time Protocol (NTP), now used in an estimated total of over 100,000 servers and clients in the global Internet. Over 220 primary time servers are available in this network, each connected to an external source of time, such as a GPS radio clock or ACTS telephone modem.

Reliable network synchronization requires crafted algorithms which minimize jitter on diverse network paths between clients and servers, determine the best subset of redundant servers, and discipline the computer clock in both time and frequency. The Network Time Protocol (NTP) is designed to do this in Unix and Windows operating systems. The NTP architecture, protocol and algorithms have evolved over almost two decades, with the latest NTP Version 3 designated an Internet (draft) standard [6]. Among the goals of this design are:

1. Optimize the computer clock accuracy and stability, subject to constraints of network overheads and/or telephone toll charges, relative to local and/or remote sources of time.
2. Enhance the reliability by detecting and discarding misbehaving local and/or remote sources and reconfigure network paths as necessary.
3. Automatically adjust algorithm parameters in response to prevailing network delay/jitter conditions and the measured stability of the computer clock.

At the heart of the NTP design are the algorithms that discipline the computer clock to an external source, either an NTP server elsewhere in the Internet or a local radio or modem. A key feature in this design is improved accuracy to the order of a few microseconds at the application program interface (API). The need for this becomes clear upon observing that the time to read the computer clock via a system call routine has been reduced from 40 µs a few years ago on a Sun Microsys-

1. Sponsored by: DARPA Information Technology Office Contract DABT 63-95-C-0046, NSF Division of Network and Communications Research and Infrastructure Grant NCR 93-01002, Northeastern Center for Electrical Engineering Education Contract A303 276-93, Army Research Laboratories Cooperative Agreement DAA L01-96-2-002, and Digital Equipment Corporation Research Agreement 1417.

2. Reprinted from: Mills, D.L. The network computer as precision timekeeper. *Proc. Precision Time and Time Interval (PTTI) Applications and Planning Meeting* (Reston VA, December 1996, 96-108

3. Author's address: Electrical Engineering Department, University of Delaware, Newark, DE 19716; Internet mail: mills@udel.edu; URL: www.eecis.udel.edu/~mills.

tems SPARC IPC to less than 1 μs today on an UltraSPARC.

The computer clock discipline algorithm, which is at the heart of the design, is described in this paper. It is implemented as an adaptive-parameter, type-II, hybrid phase/frequency-lock loop. Portions of the algorithm are implemented in the NTP software that runs the protocol and provides the computer clock corrections. The remaining portions have been implemented in this software and in the operating system kernel. For greater accuracy, a stable oscillator and counter delivering a pulse-per-second (PPS) signal can be used to steer the computer clock frequency, while an external NTP server or local radio provides the UTC time. For the highest accuracy, a PPS signal synchronized to UTC can be used directly to discipline the frequency and time within the second, while an external source, such as an NTP server or radio, provides the UTC seconds numbering.

## 2. Network Time Protocol

While not in itself the subject of this paper, an overview of the NTP design will be helpful in understanding the algorithms involved. As described in [4], a *synchronization subnet* is a hierarchical set of time servers and clients organized by *stratum*, in much the same way as in digital telephone networks. The servers at the lowest stratum are synchronized to national standards by radio or modem. In order to provide the most accurate, reliable service, clients typically operate with several redundant servers over diverse network paths.

The NTP software operates in each server and client as an independent process or *daemon*. At designated intervals, a client sends a request to each configured server and expects a response at some later time. The exchange results in four clock readings, or *timestamps*, one at the sending time (relative to the sender) and another at the receiving time (relative to the receiver), for the request and the reply. The client uses these four timestamps to calculate the clock offset and roundtrip delay relative to each server separately. The *clock filter algorithm* discards offset outlyers associated with large delays, which can result in large errors. As a byproduct, a statistical accuracy estimate called *dispersion* is produced which, combined with the stratum, is used as a metric, called *synchronization distance*, to organize the NTP subnet itself as a shortest path spanning tree.

The clock offsets produced by the clock filter algorithm for each server separately are then processed by the *intersection algorithm* in order to detect and discard misbehaving servers called *falsetickers*. The *truechimers* remaining are then processed by the *clustering algorithm* to discard outlyers on the basis of dispersions for each server as compared to the ensemble dispersion. The *survivors* remaining are then weighted by dispersion and combined to produce a correction used to discipline the computer clock.

A clock correction is produced for each round of messages between a client and a survivor. Corrections less than 128 ms are amortized using the NTP clock discipline algorithm, which is the main topic of this paper. Those greater than 128 ms cause a step change in the computer clock, but only after a sanity period of 15 minutes while these large values persists. Corrections of this magnitude are exceedingly rare, usually as the result of reboot, broken hardware or missed leap-second event.

Primary servers sometimes operate with more than one synchronization source, including multiple radios and other primary servers, in order to provide reliable service under all credible failure scenarios. The same NTP algorithms are used for all sources, so that malfunctions can be automatically detected and the NTP subnet reconfigures according to the prevailing synchronization distances.

## 3. Computer Clock Oscillator Characterization

The time-of-day (TOD) function in modern workstations is commonly implemented using an uncompensated quartz crystal oscillator and counter, which delivers a pulse train with period ranging from 10 ms to less than 1 ms. Each pulse causes a a timer interrupt, which increments a software logical clock variable by a fixed value *tick* scaled in microseconds or nanoseconds. Conventional Unix systems represent the TOD as two 32-bit words in seconds and microseconds/nanoseconds from UTC midnight, 1 January 1970, with no provision for leap seconds. Thus, the clock reading precision is limited to the tick interval; however, many systems provide an auxiliary counter with reading precision of a microsecond or less, which can be used to interpolate between timer interrupts.

That typical computer clocks behave in ways quite counterproductive to good timekeeping should come as no surprise. There are no explicit means to control crystal ambient temperature, power level, voltage regulation or mechanical stability. For instance, in a survey of about 20,000 Internet hosts synchronized by NTP, the median intrinsic frequency error was 78 PPM, with some hosts as much as 500 PPM. Since the clock oscillator is not temperature stabilized, its frequency may vary over a few PPM in the normal course of operation.

In order to correct for an intrinsic frequency error, adjustments must be made at intervals depending on the accuracy and jitter requirements. At a typical clock period of 10 ms and a frequency tolerance of 500 PPM, for example, the TOD function must add or subtract 5 μs at each timer interrupt and complete the entire 500-μs adjustment within a 1-s adjustment interval. The residual error thus has a sawtooth characteristic with maximum amplitude 500 μs, which can be reduced only by reducing the intrinsic frequency error or by reducing the adjustment interval as described later in this paper.

Assuming the clock discipline can learn the nominal frequency error of each clock oscillator separately and correct for it, the primary characteristic affecting the clock accuracy is the oscillator stability. The traditional characterization of oscillator stability is a plot of *Allan variance* [1], which is defined as follows. Consider a series of time offsets measured between a computer clock and some external standard. Let $x_k$ be the $k$th measurement and $\tau_k$ be the interval since the last measurement. Define the *fractional frequency*

$$y_k \equiv \frac{x_k - x_{k-1}}{\tau_k}, \qquad (1)$$

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

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

and the *Allan deviation* as the square root of this quantity. Figure 1 shows the results of an experiment designed to determine the Allan deviation of a typical workstation (Sun SPARC IPC) under normal room-temperature conditions over about five days. The data used to generate this plot were obtained using the PPS signal of a GPS receiver captured by a special interface described in [7].

It is important to note that both the $x$ and $y$ scales of Figure 1 are logarithmic, but the axes are labelled in actual values. Starting from the left at $\tau = 2$ s, the plot tends to a straight line with slope near $-1$, which is characteristic of white phase noise [8]. In this region, increasing $\tau$ increases the frequency stability in direct proportion. At about $\tau = 1000$ s the plot flattens out, indicating that the white phase noise becomes dominated first by white frequency noise (slope $-0.5$), then by flicker frequency

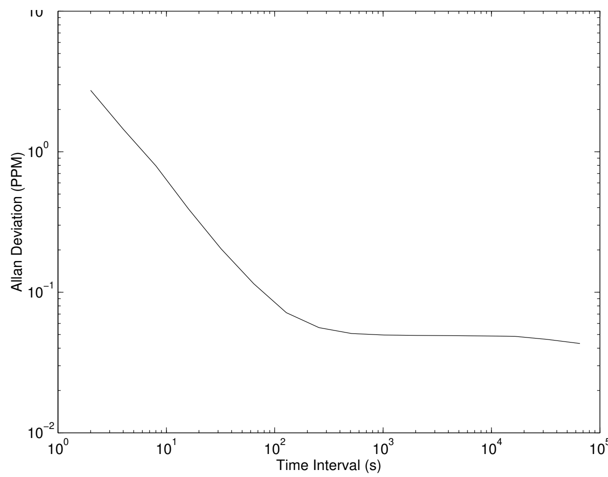

Figure 1. Allan Deviation Plot

noise (flat slope). In other words, as $\tau$ is increased, there is less and less correlation between one averaging interval and the next. The inflection point between these two regions is important in the design of the clock discipline algorithm, as described later.

## 4. The NTP Clock Discipline

The clock discipline algorithm adjusts the computer clock time as determined by NTP, compensates for the intrinsic frequency error, and adjusts the server update interval and loop time constant dynamically in response to measured network jitter and oscillator stability. A comprehensive description of the algorithm is given below (an outline of the algorithm appeared previously in [3]). The algorithm is implemented as the feedback loop shown in Figure 2. The variable $\theta_r$ represents the reference phase provided by NTP and $\theta_c$ the control phase produced by the variable frequency oscillator (VFO), which controls the computer clock. The phase detector produces a signal $V_d$ representing the instantaneous phase difference between $\theta_r$ and $\theta_c$. The clock filter functions as a tapped delay line, with the output $V_s$ taken at the sample selected by the algorithm. The loop filter, with impulse response $F(t)$, produces a correction $V_c$, which controls the VFO frequency $\omega_c$ and thus its phase. The characteristic behavior of this model, which is determined by $F(t)$ and the various gain factors, is studied in many textbooks and summarized in [5].

The new clock discipline differs from the one described in the NTP specification and previous reports. It is based on an adaptive-parameter, hybrid phase-lock/frequency-lock loop (PLL/FLL) design which gives good performance with update intervals $\tau$ from a few seconds to tens of kiloseconds, depending on accuracy requirements and acceptable network overheads. In the most

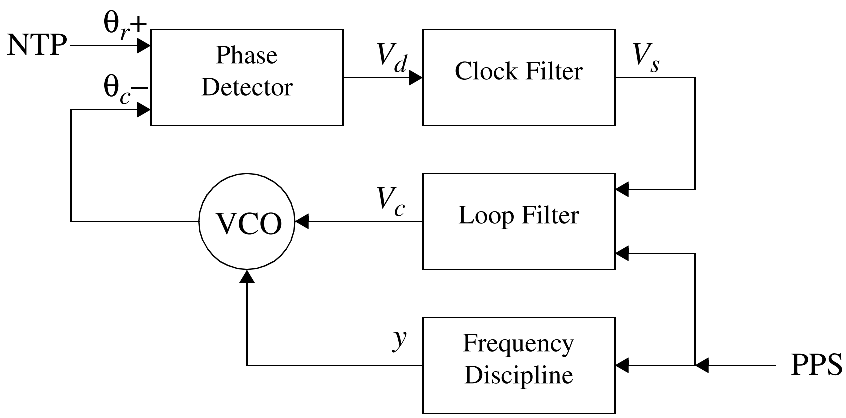


Figure 2. NTP Clock Discipline

general formulation, an algorithm that corrects for clock time and frequency errors computes a prediction $\hat{x}_k = x_{k-1} + y_{k-1}\tau_k$ and then a correction $x = \hat{x}_k - x_k$. As each correction is determined, the clock is adjusted by $x$, so that it displays the correct time, and the frequency $y_k$ is adjusted to minimize the corrections in future. Between updates, which can range from seconds to hours, the algorithm amortizes $x$ in small increments at adjustment intervals $t_A$. At each adjustment interval the the frequency is changed by

$$\Delta y = \frac{\Delta x}{t_A} = ax + y_k, \qquad (3)$$

and $x$ is multiplied by $1 - a$, where $a$ is a constant between zero and one in Hz. In the NTP daemon for Unix and Windows, $t_A$ is one second; while, in the modified kernel described later, $t_A$ is one clock tick. This model provides rapid adjustment (fast convergence) when $x$ is relatively large, together with fine adjustment (low jitter) when $x$ is relatively small. In PLL mode, the first term in (3) is necessary for stability; in both PLL and FLL modes, it is also necessary in order to prevent monotonicity violations when the magnitude of adjustment is large

The PLL mode is used in configurations with remote NTP servers or local radios, where the averaging interval is usually below the knee of the Allan deviation plot. In this mode the frequency at the $k$th update is determined directly from the summation

$$y_k = b \sum_{i=1}^{k-1} x_i \tau_i, \qquad (4)$$

where $b$ is a constant between zero and one in $Hz^2$. In order to understand the PLL dynamics, it is useful to consider the limit as $\tau_i$ approach zero. From (3) and (4), the oscillator frequency is adjusted by

$$y(t) = ax(t) + b\int_0^t x(\tau)d\tau. \qquad (5)$$

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,

$$\theta(s) = x(s)\frac{1}{s}\left(a + \frac{b}{s}\right) = x(s)G(s), \qquad (6)$$

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 transfer function G(s) can be written

$$\frac{\omega_c^2}{s^2}\left(1 + \frac{s}{\omega_z}\right), \qquad (7)$$

where $\omega_c^2 = b$ is the loop gain and $\omega_z = \frac{b}{a}$ is the corner frequency. From elementary theory, this is the transfer function of a type-II PLL which can control both time and frequency. The averaging interval is determined by the loop time constant, which depends on the choice of $a$ and $b$; however, these constants must be chosen so that the damping factor $\xi = \frac{\omega_c}{2\omega_z} = \frac{a}{2\sqrt{b}} = 2$, in order to preserve good transient response. For good stability, the time constant should be at least eight times the total loop delay which, because of the clock filter delay, is eight times the update interval. For values of $a = 2^{-10}$, $b = 2^{-24}$ and $\tau = 64$ s for instance, the PLL has a risetime in response to a phase step of about 53 minutes and a 63% fresponse to a frequency step of about 4.25 hours, which is a useful compromise between stability and network overhead on a LAN. Values of $\tau$ as low as 64 s are necessary to achieve the required capture range of ±500 PPM; however, much larger values are appropriate on long paths in the Internet. For other values of $\tau$, $a$ varies as $\frac{1}{\tau}$, while $b$ varies as $\frac{1}{\tau^2}$.

The FLL mode is used in configurations with modem services, such as those operated by NIST, USNO, PTB and NPL, where the averaging interval ($\tau$ in this case) is usually above the knee. The FLL, adapted from [2], operates is the same way as the PLL, except that the frequency $y_k$ is determined indirectly from the exponential average

$$\bar{y}_k = \bar{y}_{k-1} + w\left(\frac{x_k}{\tau_k} - \bar{y}_{k-1}\right), \qquad (8)$$

with $w = 0.25$ determined by experiment. The goal of the clock discipline is to adjust the clock time and frequency so that $x_k = 0$ 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 (1) and (8) we have

$$y_k = \bar{y}_k + \frac{x_k}{\tau_k}. \qquad (9)$$

In PLL mode, (4) is used for $y_k$ in (3); while, in FLL mode, (9) is used instead.

A key feature of the NTP design is the automatic selection of $\tau$ in response to measured network jitter and oscillator stability. The ensemble dispersion is used as a measure of oscillator stability in both the PLL and FLL modes. If the correction $|x|$ exceeds this value, the oscillator frequency is deviating too fast for the clock discipline to follow, so $\tau$ is reduced in stages to the minimum. If the opposite case holds for some number of updates, $\tau$ is slowly increased in steps to the maximum. Under typical operating conditions, $\tau$ hovers close to the maximum; but, on occasions when the oscillator frequency wanders more than about 1 PPM, it quickly drops to lower values until the wander subsides.

## 5. Operating System Kernel Modifications

Previous experience has justified the claim that an ordinary workstation running the algorithms described above can reliably maintain time accurate to a millisecond or two relative to a server on the same LAN. However, with a 1-s adjustment interval, 500-PPM frequency tolerance and $\tau$ at the knee of the Allan deviation plot, it is not possible to improve the accuracy much better than this, primarily due to the instability of the clock oscillator and also due to the sawtooth error. Both of these problems can be addressed in the form of operating system kernel modifications, which in effect move the clock discipline algorithm to the kernel, as described in [7]. This provides a smaller adjustment interval, which reduces the sawtooth error and also provides more precise phase and frequency control. Without the kernel modifications, the adjustment interval is limited by practical considerations to 1 s; with the kernel modifications, the adjustments occur at every timer interrupt.

The modifications have been implemented and tested on Sun, DEC and HP workstations. They areidistributed in Digital Unix 4 for the DEC Alpha and planned for early release in Solaris 2 for the Sun SPARC. They include two system functions, one to read the system clock and related status indicators and error bounds, and another to adjust the clock phase and frequency. The clock discipline algorithm operates as shown in Figure 2, with phase corrections provided at each NTP update. The oscillator frequency is preset when the NTP daemon is first started, in order to reduce the startup transient, after which the frequency is controlled by the algorithm.

The PPS signal is connected using a pulse generator and level converter. Each on-time transition causes an interrupt to the serial port driver, which latches the current seconds offset and disciplines the clock oscillator, as shown in Figure 2. The signal can be used in two ways, to discipline the oscillator frequency and to discipline the phase. Frequency discipline is used when a stable PPS signal is available, but not synchronized to UTC time. In this case, the floor of the Allan deviation plot moves downward and the knee moves to the right. Thus, $\tau$ can be made much larger, increasing the averaging time and improving the accuracy. Frequency and phase discipline is used when a PPS signal synchronized to UTC time is available.

Since noise problems on the PPS signal could lead to serious errors, the kernel routines carefully grade and groom the data. Three-stage median filters are used to discard outlyers and provide quality metrics for jitter and wander. The nominal frequency offset is computed from the time difference between the beginning and end of a calibration interval and added directly to the frequency variable, as shown in Figure 2. These operations are complicated by the requirement that all values must depend only on the clock hardware. When relatively small tick values are involved, less than a millisecond with DEC Alpha, and large frequency errors, as much as 500 PPM, this requires the initial calibration interval to be not more than 4 s. If the stability metric exceeds a threshold, the length of the calibration interval is reduced by half. If this is not the case for several consecutive intervals, the interval is doubled up to a maximum of 256 s, which corresponds to a frequency resolution of a few parts in $10^9$.

The NTP daemon performs a number of sanity checks to insure the integrity of the radio or modem ASCII timecode and the PPS signal itself. The sanity checks are implemented by a suite of mitigation algorithms which identify improperly operating hardware or software, cast out the truants, and continue operating with the remaining sources, even if this means casting out a radio or modem and demoting the stratum. For instance, before the PPS signal can be considered valid, the computer clock must be within 128 ms of the offset associated with the source of the signal. In addition, the source must remain among the survivors of the intersection and clustering algorithms. In practice, failures of this kind are not uncommon with WWVB radio clocks in our part of the country, since a combination of poor signal strength and local interference sometimes cause relatively large receiver errors.

Considerable effort was made in the implementation of the NTP software and kernel modifications to reduce hardware and operating system delay variations; however, not all machines make good timekeepers. Unpredictable delay variations occur in the hardware, interrupt routines, buffering operations and system scheduling policies. In the case of a network interface, the network driver captures a receive timestamp in the interrupt routine. In the case of a radio or modem, this is done using a *line discipline*, which is invoked by the serial port driver. It inspects for one of a designated set of intercept characters, usually the one designated *on-time* in the ASCII timecode string sent by the radio. Upon finding an intercept character, it captures a receive timestamp and stuffs the bits in the input buffer following the intercept character. The NTP daemon captures a transmit timestamp before computing the cryptographic message digest used to verify the server authenticity. Fortunately, the MD5 algorithm used for this purpose has an almost constant running time independent of the message contents. The daemon measures this time and then advances the transmit timestamp by a like amount in the following message.

## 6. Performance Analysis

In order to assess the performance of the NTP algorithms in the global Internet, recordings of raw timestamp data were made over an 11-day period involving three paths selected to represent extreme cases with presumed large delays and delay variations. The three paths are between the University of Delaware (pogo.udel.edu) and (WUSTL) Washington University in St. Louis (navobs1.wustl.edu, 15 router hops), (IEN) IEN Galileo Ferraris in Torino, Italy (time.ien.it, 19 hops), and (OZ) University of Melbourne in Australia (ntp.cs.mu.oz, 22 hops). Each server is synchronized to GPS, although only pogo has the modified kernel and PPS support.

A common assumption is that network delays are reciprocal; that is, the statistics exploited by the various NTP algorithms on each direction of transmission are the same. In order to test this assumption, the raw data collected on all three paths were processed by a simulator program which faithfully models the NTP algorithms and includes provisions to adjust the various parameters and graph the results. The propagation delay can be estimated as the mean of the ten lowest delays on each direction. The IEN outbound path delay was 93.0 ms and the return 97.9 ms, while the figures for the OZ path were 124.6 and 138.4 ms, and for the WUSTL path are 19.6 and 19.5 ms. The offset errors due these nonreciprocal delays are half the differences, 2.45, 6.9 and 0.05 ms, respectively. These errors are surprisingly small, considering the number of router hops and the great distances involved.

In addition to the fixed propagation delays, there are variable delays due to queueing in routers along the path. For instance, the mean roundtrip delays on the IEN, OZ and WUSTL paths are 345.4, 359.9 and 65.4 ms, respectively, leaving 154.5, 96.9 and 26.3 ms as the mean queueing delays. Thus, between a quarter and a half of the mean roundtrip delays are due to queueing delays. However, after processing by the NTP algorithms, The degree to which the NTP algorithms clean up the raw data can be seen in the following: The mean offset errors for the raw data are 15.2, 9.8 and 4.8 ms for the IEN, OZ and WUSTL paths, respectively, while the RMS errors are 64.0, 470.0 and 19.5 ms. However, after processing by the NTP algorithms, the mean offset errors are reduced to .045, .004 and .003 ms, respectively, while the RMS errors are reduced to 2.9, 3.0 and 0.15 ms. These values should be added to the nonreciprocal path errors in the total error budget. Figure 3

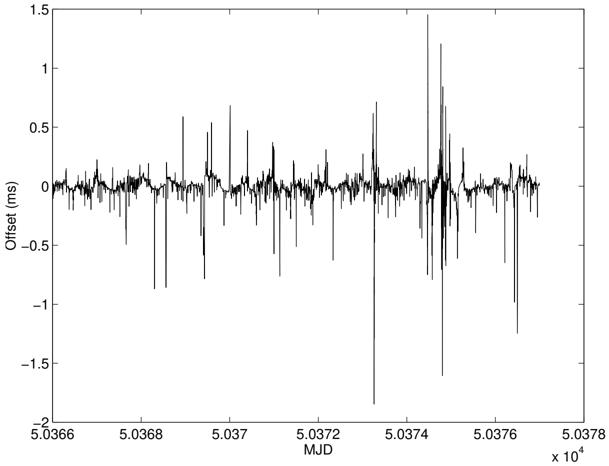


Figure 3. Processed Data Offsets

shows the results for the WUSTL path; the other paths behave in similar ways.

The graph has a spikey characteristic shared by the other paths and suggests further processing might eliminate most or all of the spikes, especially if the PPS signal were used to stabilize the clock oscillator and the length of the clock filter increased a substantial amount. While the emphasis in this paper is on heroic paths in the Internet, a similar experiment involving pogo and another machine on the same Ethernet LAN shows negligible mean error, RMS error .007 ms and maximum error .078 ms, the latter due to a single spike of unknown origin during a run of 15 days,

## 7. Summary and Conclusions

The significance of this work is confirmation that general purpose workstations with appropriate software support can deliver timekeeping accuracies in the low millisecond range over global distances using the Internet shared with many other users and applications. This may be especially useful for astronomy, oceanography, manufacturing and process control applications.

The key to achieving accuracies of this order is through carefully crafted algorithms and the use of a stable external oscillator and counter to produce a PPS signal. The signal is processed by algorithms embedded in the operating system kernel and used to discipline the frequency of the computer clock at every timer interrupt.

The package of NTP software and kernel modifications has been implemented for several families of Unix and Windows workstations and is available for public distribution. The kernel modifications have been incorporated in the standard Digital Unix operating system and are planned for early release in the Solaris operating system.

## 8. References

References 3-7 are available from Internet archives in PostScript format. Contact the author for location and availability.

1. 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.

2. Levine, J. An algorithm to synchronize the time of a computer to universal time. *IEEE Trans. Networks 3, 1* (February 1995), 42-50.

3. Mills, D.L. Improved algorithms for synchronizing computer network clocks. *IEEE/ACM Trans. Networks* (June 1995), 245-254.

4. 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.

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

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

7. Mills, D.L. Unix kernel modifications for precision time synchronization. Electrical Engineering Department Report 94-10-1, University of Delaware, October 1994, 24 pp.

8. Stein, S.R. Frequency and time - their measurement and characterization (Chapter 12). In: E.A. Gerber and A. Ballato (Eds.). *Precision Frequency Control, Vol. 2*, Academic Press, New York 1985, 191-232, 399-416. Also in: Sullivan, D.B., D.W. Allan, D.A. Howe and F.L. Walls (Eds.). *Characterization of Clocks and Oscillators*. National Institute of Standards and Technology Technical Note 1337, U.S. Government Printing Office (January, 1990), TN61-TN119.