Adaptive Hybrid Clock Discipline Algorithm for the Network Time Protocol[1,2]

David L. Mills, *Member ACM, IEEE*[3]

Abstract

This paper describes the analysis, implementation and performance of a new algorithm engineered 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 algorithm is intended for the Network Time Protocol (NTP), which is in widespread use to synchronize computer clocks in the global Internet, or with another functionally equivalent protocol such as DTSS or PCS. It controls the computer clock time and frequency using an adaptive-parameter, hybrid phase/frequency-lock feedback loop. Compared with the current NTP Version 3 algorithm, the new algorithm developed for NTP Version 4 provides improved accuracy and reduced network overhead, especially when per-packet or per-call charges are involved. The algorithm has been implemented in a special purpose NTP simulator, which also includes the entire suite of NTP algorithms. The performance has been verified using this simulator and both synthetic data and real data from Internet time servers in Europe, Asia and the Americas.

Keywords: computer network time synchronization, clock discipline algorithm, oscillator error modeling, feedback control loop

1. Introduction

General purpose workstation 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 with 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 [4]. 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 230 public primary servers are available in this network, each connected to a external source, such as a GPS receiver 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. NTP is designed to do this in Unix, Windows and VMS 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.

At the heart of the NTP design are the algorithms that synchronize the computer clock to NTP servers elsewhere in the Internet or an external source. These include the algorithms that mitigate among multiple servers to provide the most accurate and reliable time, together with the algorithm that disciplines the computer clock with respect to this time. The clock discipline algorithm, shortened to *discipline* in the following, is the main topic of this paper. It has evolved from simple beginnings to a sophisticated design which automatically adapts to changes in operating environment without manual configuration or real-time management functions. The algorithm has been implemented both in the NTP software itself and, for the highest accuracy, in the operating system kernel.

The current NTP Version 3 discipline, together with improvements already implemented and described in [4], represent the departure point for this paper. The new algorithm is intended for NTP Version 4, which is in

---

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, and Digital Equipment Corporation Research Agreement 1417.

2. Reprinted from: *IEEE/ACM Trans. Networking 6, 5* (October 1998), 505-514.

3. Author's address: Electrical and Computer Engineering Department, University of Delaware, Newark, DE 19716, mills@udel.edu, http://www.eecis.udel.edu/~mills.

final testing, but could be used in principle with any protocol that provides periodic time corrections. A key feature in the new design is improved accuracy to the order of a few microseconds at the application program interface with fast, modern workstations. 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 Microsystems SPARC to less than 1 μs today on an UltraSPARC. Another valuable feature is that message intervals can be increased well over the current maximum without significant impact on accuracy.

While not in itself the subject of this paper, a brief overview of the NTP design will be helpful in understanding the algorithms involved. A comprehensive description of the NTP architecture, protocol and algorithms is in [4] and citations found there. The global NTP time synchronization subnet is a hierarchical tree of time servers, and clients organized in much the same way as digital telephone synchronization networks. The primary servers at the root of the tree are synchronized to national standards by radio or telephone modem. In order to provide the most accurate and reliable service, secondary servers and clients typically operate with several redundant servers over diverse network paths.

The NTP software operates in each server and client as an independent process. At designated intervals, a client sends a request to each in a set of configured servers and expects a response at some later time. The exchange results in four clock readings, one at the sending and receiving times for each round. From these, the client calculates the clock offset and roundtrip delay relative to each server separately and presents these data to a set of grooming algorithms. The client also calculates a distance metrics which is used to organize the NTP subnet itself as a shortest-path spanning tree with root at the primary servers. The grooming algorithms select the best samples in the data stream from each server, exclude provably incorrect servers, and combine the data from the survivors to produce time correction, *update* in the following, which drives the clock discipline algorithm.

This paper continues with a detailed description of the new discipline, including its state machine and subalgorithms which adjust the clock time and frequency and manage the various parameters. The paper ends with a suite of experiments designed to validate the design and calibrate its performance using both synthetic noise generators and real data from Internet paths spanning continents and oceans.

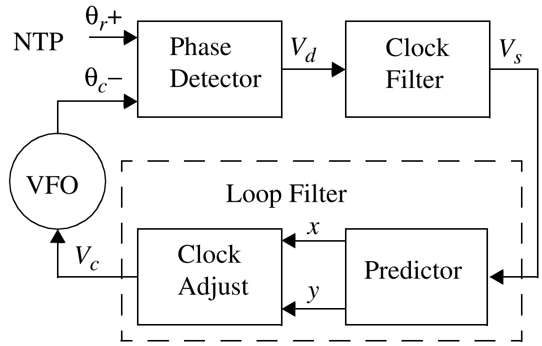

Figure 1. Clock Discipline Algorithm

2. The NTP Clock Discipline Algorithm

In both the current NTP Version 3 and new Version 4 designs, the discipline corrects the computer clock time, compensates for its intrinsic frequency error, and adjusts the various parameters dynamically in response to measured network latency variations or *jitter* and oscillator frequency stability or *wander*. The algorithms function as a combination of two philosophically quite different feedback control systems. In a phase-lock loop (PLL) design, the updates are used directly to minimize the phase error and indirectly to minimize the frequency error. In a frequency-lock loop (FLL) design, the updates are used directly to minimize the frequency error and indirectly to minimize the phase error. As shown later, a PLL usually works better when network jitter dominates, while a FLL works better when the oscillator wander dominates.

The discipline is implemented as the feedback control system shown in Figure 1. The variable θ_r represents the reference phase produced by the update and θ_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 θ_r and θ_c. The clock filter functions as a tapped delay line, with the output taken at the tap selected by the algorithm. The clock selection, clustering and combining algorithms (not shown) combine the data from multiple filters to produce the signal V_s. The loop filter, with impulse response $F(t)$, produces the signal V_c which controls the VFO frequency ω_c and thus its phase θ_c. The V_c signal is generated by an adjustment process which runs at intervals of one second. 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 [4].

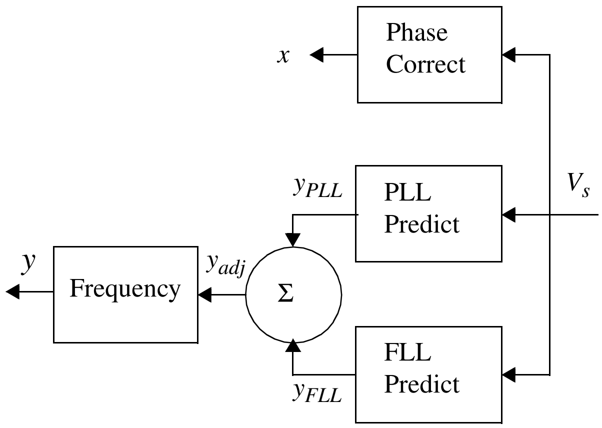

Figure 2. FLL/PLL Prediction Functions

The original NTP Version 3 discipline, which is based on a conventional PLL, has been improved to include a FLL capability. The selection of which mode to use, FLL or PLL, is made on the basis of *update interval* τ. In the improved design, PLL mode is used for τ less than 1024 s and FLL mode is used from 1024 s to 4.5 h, which is the maximum possible in this design. This improves the accuracy and stability in some modes of operation, specifically those involving large τ, but has proved suboptimal for reasons described later in this paper. The new Version 4 discipline uses a combined approach with a true hybrid PLL/FLL design which gives good performance with τ from a few seconds to well over a day, depending on accuracy requirements and acceptable network overhead.

2.1 Time and Frequency Discipline

In the new design, the loop filter of Figure 2 is implemented using two subalgorithms, one based on a conventional PLL and the other on a FLL design suggested in [2]. Both predict a time correction x as a function of V_s. The PLL predicts a frequency adjustment y_{PLL} as an integral of $V_s\tau$, while the FLL predicts an adjustment y_{FLL} as an exponential average of V_s/τ. The two adjustments are combined and added to the current clock frequency y, as shown in the figure. The x and y are then used by the clock adjust process to control the VFO frequency and close the feedback loop shown in Figure 1.

An overview will suffice to describe how the correction and prediction algorithms work. The details, which are given in [5], are beyond the scope of this paper. At each update, a phase correction

$$x = V_s \tag{1}$$

is initialized, then exponentially amortized in V_c by the clock adjust process at rate $1/T_c$, where T_c is the loop time constant. This is necessary to avoid step changes in time, as well as to implement the lag network required for a stable PLL.

In PLL mode, y is a time integral over all past values of V_s, so

$$y_{PLL} = \frac{V_s\tau}{T_c^2}. \tag{2}$$

From [4], this results in a transfer function

$$F(s) = \frac{\omega_c^2}{s^2}\left(1 + \frac{s}{\omega_z}\right), \tag{3}$$

where $\omega_c = \frac{1}{T_c}$ is the loop gain and $\omega_z = \frac{1}{2T_c}$ is the corner frequency. From elementary theory, this is the transfer function of a linear, time-invariant, PLL which can drive both time and frequency errors to zero.

For good stability, T_c must be at least several times the total loop delay which, because of the clock filter delay, is several times τ. When the discipline is first started, a relatively small $\tau = 64$ s is required, in order to achieve the required discipline capture range of 500 parts-per-million (PPM). Selecting a compromise $T_c = 1024$ s, the PLL response to a time step has a risetime of 53 minutes, an overshoot of 5% and a 63% response to a frequency step of 4.25 hours. Ordinarily, τ increases substantially once the frequency has stabilized and T_c increases in proportion.

In FLL mode, y_{FLL} is an exponential average of past frequency changes, as computed from V_s and τ. The goal of the algorithm is to reduce V_s to zero; so, if this has been successful in the past, previous values can be assumed zero and the exponential average becomes simply

$$y_{FLL} = \frac{V_s}{w\tau}, \tag{4}$$

where w is a constant determined so that the averaging time is close to the Allan intercept, as described later.

2.2 Hybrid FLL/PLL Combining Algorithm

The NTP Version 3 discipline selects either the PLL or FLL mode on the basis of τ. In general, PLL mode is used for $\tau < 1024$ s, which is appropriate for network servers, and FLL mode is used at $\tau > 1024$ s, which is appropriate for modem servers. While this results in a useful compromise, it does not provide for the automatic selection on the basis of prevailing network jitter and oscillator wander. It also results in a moderate transient when switching between modes. In the NTP Version 4 design, the feedback loop is modified to operate as a true hybrid, where FLL and PLL frequency predictions are computed separately and combined on the basis of RMS error over previous predictions. The result is that the PLL prediction is weighted more heavily under conditions of extreme network jitter due, for example, to network congestion, while the FLL prediction is weighted more heavily under conditions of extreme oscillator wander due, for example, to large local temperature variations.

As in the original design, T_c is the loop time constant and τ the interval since the previous update. In the new design, x is the residual time correction and y_{adj} is the frequency adjustment at the next update. First, the x_{FLL} and x_{PLL} prediction errors are computed from these values and the frequency predictions at the last update:

$$x_{FLL} = V_s - x - (y_{FLL} - y_{adj})\tau \text{ and}$$
$$x_{PLL} = V_s - x - (y_{PLL} - y_{adj})\tau . \qquad (5)$$

The RMS prediction errors ε_{FLL} and ε_{PLL} are computed from successive samples of x_{FLL} and x_{PLL}. The number of samples averaged, or equivalently the averaging time, depends on T_c, which itself depends on τ. The intent of this design is to keep the averaging time near the Allan intercept.

Next, new values for x, y_{PLL} and y_{FLL} are determined using (1), (2) and (4). The prediction errors determine the proportional gain for the PLL and FLL frequency adjustments used to compute y_{adj}:

$$y_{adj} = \frac{y_{FLL}\varepsilon_{PLL} + y_{PLL}\varepsilon_{FLL}}{\varepsilon_{FLL} + \varepsilon_{PLL}} . \qquad (6)$$

Figure 3 shows typical standard error (RMS) curves as a function of τ in FLL mode (solid), PLL mode (dash-dot) and hybrid mode (dash). For this particular network path, the optimum choice is PLL mode below 200 s and FLL mode otherwise; however, the optimum choice varies widely, depending on the particular network path involved and degree of congestion. As a compromise, the original NTP Version 3 discipline used PLL mode below 1024 s and FLL mode otherwise. In the new hybrid mode, PLL predictions are usually better at small τ, while FLL predictions are better at large τ. From about 100 s to 1000 s, hybrid mode is better than either PLL or FLL modes and above 1000 s is only a little worse than FLL mode. Below 100 s hybrid mode is somewhat worse than PLL mode and a little better than FLL mode, but still better than 1 ms.

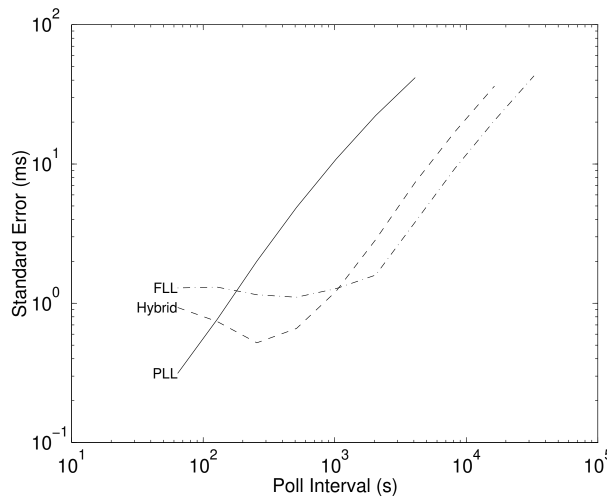

Figure 3. Raw/Filtered Data Comparison

2.3 Poll-Adjust Algorithm

NTP Version 3 time servers and clients operate today using network paths that span the globe. In very many cases, primary servers operate with several hundred clients or more. It is necessary to explore every means with which τ can be increased without significantly degrading clock accuracy or stability. The NTP Version 4 discipline allows a significant increase in τ without compromising accuracy, while at the same time adapting dynamically to widely varying network jitter and oscillator wander regimes.

The experiments described later in this paper show that, in almost all cases, the standard error increases as τ increases, due primarily to the effect of oscillator wander. Since the overhead decreases as τ increases, a method is needed to select τ as the best compromise between required accuracy and acceptable overhead. This is most important in configurations where a toll charge is incurred for each poll, as in ISDN and telephone modem services.

In the NTP Version 3 design, the minimum and maximum τ default to values appropriate for almost all network and computer configurations. For network servers, τ can range from 64 s to 1024 s, while for telephone modem servers, it can range from 1024 s to 16,384 s. However, in NTP Version 4, the upper limit can range up to 131,072 s, or well over a day. The discipline automatically manages τ within these ranges in response to

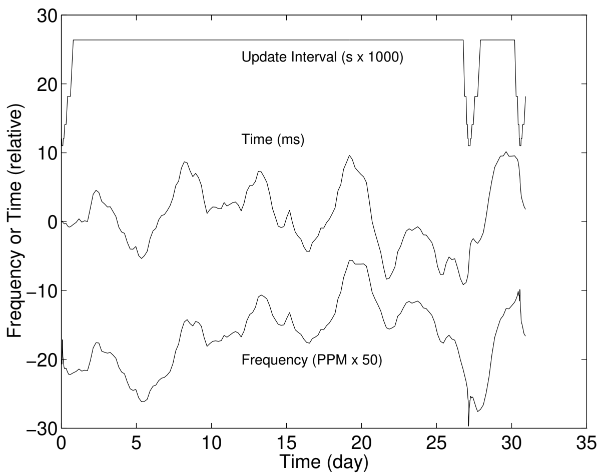

Figure 4. Poll Interval Adjustment

the prevailing network jitter and oscillator wander. An important point is that, using the new discipline, it is not necessary to clamp τ to the minimum when switching among different synchronization sources as in NTP Version 3. In cases of moderate to severe network jitter and with multiple sources, this sometimes causes frequent *clockhopping* which in turn degrades accuracy.

A key statistic in controlling τ is the *system dispersion,* which is a determined from the RMS errors measured by the grooming algorithms [5]. If $|V_s|$ is greater then the system dispersion by an experimentally determined constant factor, the oscillator frequency is deviating too fast for the discipline to follow. In this case, τ is reduced in stages to the minimum. If the opposite case holds for some number of updates, τ is slowly increased in steps to the maximum. A hysteresis mechanism built into the algorithm prevents unnecessary dithering of τ. Under typical operating conditions, τ 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.

Figure 4 shows the time and frequency offsets and update interval for a typical workstation over a 30-day period. In this figure, the baseline is +10 for the τ curve, zero for the time curve, and −20 for the frequency curve. Most of the time is spent at the maximum interval, in this case 16,384 s, with brief excursions to lower intervals not less than 1024 s when the frequency deviates too rapidly for the discipline to follow. This particular figure shows the expected behavior for a typical telephone modem, where it is important that τ remain at large values whenever possible. In particular, it is important that the initial frequency adaptation when the discipline is first started be substantially complete within only the first few samples before τ starts to increase. In Figure 4, where the initial time and frequency offsets are zero, this occurs after the eighth call.

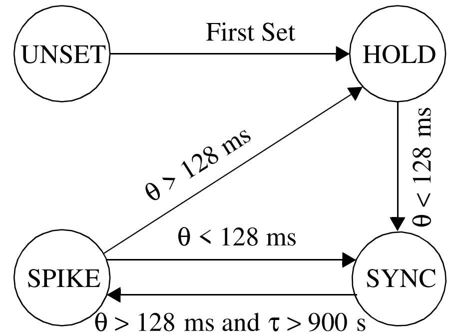

Figure 5. Clock State Machine

2.4 Clock State Machine

The discipline must operate over an extremely wide range of network jitter and oscillator wander characteristics without manual intervention or configuration. As determined by past experience and experiment [4], the various data grooming algorithms work well to sift good data from bad, especially under conditions of light to moderate network and server loads. However, under conditions of extreme network or server congestion, operating system latencies, and oscillator wander, any discipline may perform poorly and even become unstable.

In order to deal with very large transients at and after startup, the discipline is managed by the state machine shown in Figure 5. Each of the four states has defined inputs, outputs and transition functions. Initially, the machine is unsynchronized and in UNSET state. If the minimum τ is greater than 1024, the first update received sets the clock and transitions to HOLD state. This behavior is designed for toll services with long intervals between calls. If τ is less than 1024 s, these actions will not occur until after several updates, to allow the data grooming algorithms to accumulate sufficient data for reliable synchronization.

In HOLD state the discipline is locked in FLL mode, in order to provide rapid adaptation to possibly very large clock oscillator frequency errors. Once entering HOLD state, the machine remains in this state for several updates, in order to complete, as far as possible, the frequency adaptation process. After the correction θ has decreased below 128 ms, the machine transitions to SYNC state and remains there pending unusual conditions.

In SYNC state, a suite of sanity checks, spike detectors and tolerance clamps are operative. To avoid disruptions

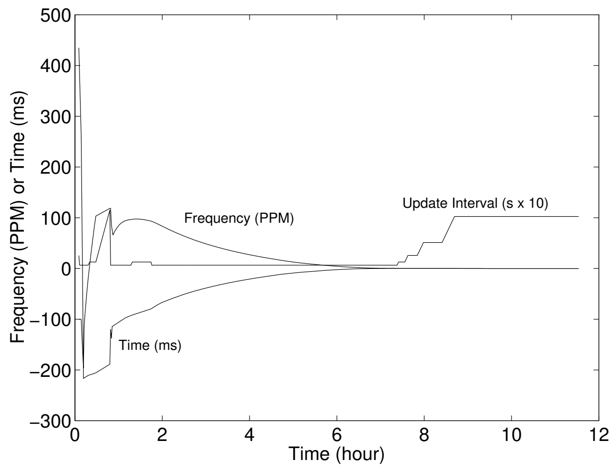

Figure 6. Initial Transient Adaptation

due to frequency spikes, the frequency is clamped not to exceed 500 PPM and frequency adjustments not to exceed 10 PPM. In addition, frequency adjustments are disabled if the system dispersion exceeds 128 ms. To avoid disruptions due to occasional network delay transients, spike detectors reject updates greater than ten times the dispersion. However, the spike does increase the dispersion; so, if a true time step were to occur, the first one or two samples, may be discarded, but eventually the step will be recognized and corrections made.

Special provisions are made for updates larger than 128 ms, which may happen occasionally when the network is badly congested or when some major disruption occurs. It can also happen upon the occasion of a leap second, when the clock has been automatically stepped by the protocol, but for some reason the source has not implemented the step. In both cases, the best response is to ignore large updates, unless they persist for some time.

However, if no updates less than 128 ms arrive after a timeout interval, currently 15 m, the eventual response should be to believe any that are received, regardless of value. The first large update after the timeout causes a state transition to SPIKE state, but does not set the clock. If the next update after that is less than 128 ms, the machine transitions to SYNC state and proceeds normally. In this case the prior update is considered a spike and ignored. If the next update in SPIKE state is greater than 128 ms, the machine transitions to HOLD state and sets the clock. Since the sanity checks are disabled in HOLD state, the discipline can quickly adapt to the new time and frequency as described previously.

Figure 6 shows an extreme example, where the initial time offset is 100 ms and initial frequency offset is 500 PPM. Most of the frequency adaptation occurs in HOLD state, as the FLL wrangles the frequency to approximate agreement. The initial time offset is purposely chosen so that the residual offset after this initial adaptation results in the state machine first transitioning to SYNC state, but with residual frequency offset beyond the loop capture range in that state. This eventually causes the offset to exceed 128 ms and, after the timeout interval, the state machine transitions to HOLD state. This process may repeat a cycle or two, until the residual frequency offset is sufficiently reduced. In Figure 6, after about six hours, the adaptation is complete and τ starts to ramp up. In the original NTP Version 3 design, the loop capture range was only 100 PPM and adaptation to even that unambitious value took well over a day.

3. Performance Evaluation

The performance of the new discipline has been evaluated using a simulation approach. There are three reasons for this, rather than building and testing the algorithms in the context of the existing NTP Version 3 implementation. First, evaluation of these algorithms can take long wallclock times, since the intrinsic time constants are often quite long - several hours to days. Simulation time runs much faster than real time, in fact by several orders of magnitude. Second, the simulation environment is not burdened by the infrastructure in which the real software must operate, such as I/O and monitoring code. Third, the simulator code itself, written in portable C, represents a model for the eventual formal specification and implementation, as well as a documentation aid. This is the same approach that proved highly successful in the development of the original NTP Version 3 specification and implementation.

The simulator faithfully mimics the operation of the implementation, including the entire suite of data grooming algorithms, as well as the discipline itself. It can use timestamp data collected by the existing software operating over real Internet paths, as well as data synthesized by internal phase and frequency noise generators. The generators produce white phase noise and random-walk frequency noise, both using zero-mean Gaussian processes. As shown in subsequent discussion, these generators closely emulate the real data for most statistical purposes. The general philosophy is to evolve the algorithms using repeatable, synthetic noise streams and, once the design has stabilized, verify the expected operation using real data.

An important strategy in the experiments with real data is the exclusive use of 19 primary servers in Europe, Asia and the Americas. Since primary servers are independently synchronized to external references, such as GPS receivers or ACTS telephone modems, their clocks have presumed zero time and frequency error[4]. This allows real phase data to be collected without contamination by errors due to frequency variations. Real frequency data are separately collected using a series of measurements with a free-running clock oscillator and a precision time reference source. The phase and frequency data are collected in data files which are read by the simulator and integrated with synthetic noise as required.

There are three sets of experiments described in the following. In the first set, the behavior of typical Internet paths and computer clocks is modeled using a special statistic, in order to develop synthetic noise profiles which characterize common client-server configurations. In the second set, the behavior of the model is explored using these profiles, in order to predict how the errors due to network jitter and oscillator wander scale relative to τ. These provide the basis and justification for the new fully hybrid discipline design. In the third set, the model is evaluated using real phase and frequency data, in order to verify the model and discipline behave as predicted.

3.1 Computer Clock Modeling

The first set of experiments is designed to develop a model characterizing a typical computer clock oscillator synchronized via Internet paths. The experiments are designed to do two things: (a) evaluate the error characteristics for typical oscillators and Internet paths between a University of Delaware site and selected remote sites and (b) validate that the synthetic random noise generators incorporated in the simulator realistically emulate these characteristics on a statistical basis.

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 frequency error was 78 PPM [3], with some hosts showing errors over 500 PPM. Since the clock oscillator is not temperature stabilized, its frequency may vary over a few PPM in the normal course of the day.

The traditional characterization of oscillator stability is a plot of *Allan variance* [1], which is defined using a series of time differences measured between a computer clock and some external standard. Let x_k be the kth measurement and τ be the interval between measurements. Define the *fractional frequency*

$$y_k \equiv \frac{x_{k+1} - x_k}{\tau}, \; k > 0, \qquad (7)$$

which is a dimensionless quantity. Now, consider a sequence of N independent fractional frequency samples for $k = 1, 2, ..., N$. If the measurement interval is the same as the averaging interval, the 2-sample Allan variance is defined

$$\sigma_y^2(\tau) \equiv \left\langle \frac{1}{2}(y_{k+1} - y_k)^2 \right\rangle$$

$$= \frac{1}{2\tau^2(N-2)} \sum_{k=1}^{N-2} (x_{k+2} - 2x_{k+1} + x_k)^2 \qquad (8)$$

and the *Allan deviation* σ_y as the square root of this quantity. The results are commonly displayed as a curve plotted in log-log coordinates as described below.

Two experiments were designed to establish Allan deviation characteristics for typical workstation clocks under typical room-temperature conditions. The PPS experiment measured time differences between a free-running SPARC IPC clock and a directly connected cesium clock. Data were collected at 2-s intervals over five days and recorded in a data file. The LAN experiment measured differences between a free-running SPARC IPC clock and a primary server on the same network wire. Data were collected at 16-s intervals over fifteen days using NTP and recorded in a data file.

The oscillator frequency computed from the LAN data is shown in Figure 7. The frequency varies over a 3-PPM range, sometimes abruptly, which is characteristic of room temperature changes of a few degrees. This experiment was conducted in spring, when the laboratory windows were open and the temperature allowed to follow the weather, and is typical of “poor” stability. The oscillator frequency computed from the PPS data (not shown) is similar to the LAN characteristic, but varies over a much smaller 0.25 PPM range. This experiment was conducted in winter, when the room temperature was thermostatically controlled, and is typical of “good” stability.

The results of the PPS and LAN experiments were processed to produce plots of $\sigma_y(\tau)$ in PPM against τ in seconds in log-log coordinates. The results for the PPS

4. In practice, the errors relative to the reference source are rarely greater than 1 ms and often lower than 100 µs.

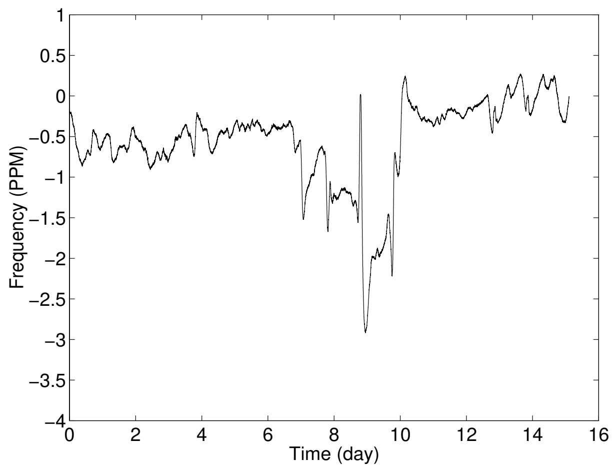

Figure 7. LAN Frequency

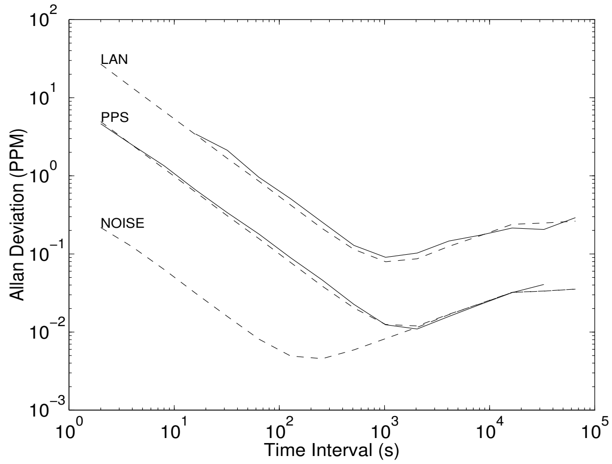

Figure 8. Allan Deviation

and LAN data are shown as solid lines in Figure 8. The shape of each curve depends on the phase and frequency variations specific to the particular clock oscillator and network path involved. The phase variations, which are dominated by white phase noise, produce straight lines with slope −1 on the plots [6]. The frequency variations, which are dominated by random-walk frequency noise, produce straight lines with slope +0.5 on the plots. The intersection of these two lines, called here the *Allan intercept*, characterizes each particular clock oscillator and network path.

The Allan intercept is the primary statistic influencing the design of the discipline. For τ below this point, there is an underlying frequency tendency, so increased averaging can improve the accuracy; however, above this point increased averaging actually degrades the accuracy. This various time constants used in the discipline are engineered with these facts in mind.

The primary contributions to the phase noise evident in the results for the PPS and LAN data are jitter due to the clock precision and reading errors, operating system latencies and, for the LAN experiment, latencies contributed by a lightly loaded 10-Mbps Ethernet. In both the PPS and LAN experiments, the frequency variations are due to nondeterministic wobbles of the oscillator frequency, which is affected primarily by ambient temperature variations.

The dashed lines on Figure 8 show the Allan deviations determined from a random sequence where each value consists of the sum of a white phase noise sample and a random-walk frequency noise sample. White phase noise is generated directly from a Gaussian distribution. Random-walk frequency noise is generated by integrating samples from a Gaussian distribution. The standard deviation parameter of each noise generator was chosen by experiment so that the synthetic characteristics closely match the measured characteristics. For comparison, the NOISE curve on the figure was generated using a phase parameter of zero and the frequency parameter of the PPS experiment. This models the case where the only errors are due to the simulated clock precision of 1 μs, which represents the best performance possible with the common Unix microsecond clock.

The PPS and LAN results represent cases involving external sources, either directly connected or connected by a LAN. However, these results do not represent cases where long Internet paths are involved. In the next set of experiments, the simulator used timestamps measured at 64-s intervals over a ten-day period in fall, 1996, between server *pogo.udel.edu* and the 19 remote servers, individually and in combination. In these experiments the NTP data grooming algorithms were connected, but the feedback loop was disconnected, so the clock oscillator ran at constant frequency determined by the external source.

The Allan deviations for representative sources are shown in Figure 9. The NOISE, PPS and LAN results duplicate the data of Figure 8 for comparison. The BARN data represent a “local” server, in this case on the same network wire as pogo, while the USNO data represent a “nearby” server at the U.S. Naval Observatory in Washington, DC, and the IEN data represent a “distant” server at the IEN Galileo Ferraris in Torino, Italy. The USNO data fairly well represent a path between two servers in the U.S., where the path is only lightly congested, while the IEN data represent moderate to heavy congestion typical of paths spanning the Atlantic.

The curves described so far involve only a single server, which is the only source used to synchronize the clock.

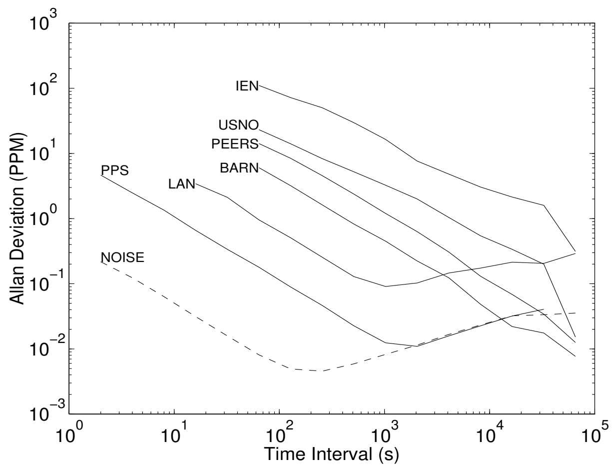

Figure 9. Allan Deviation - Combined Data

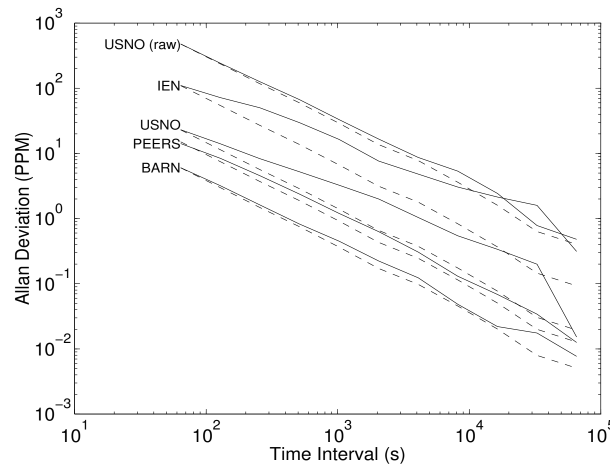

Figure 10. Comparison of Real and Synthetic Allan Deviation

The PEERS curve represents the case where all servers, except those on the local LAN, are used, so that the intersection and clustering algorithms can select the best subset of sources to synchronize the clock. Note that the PEERS curve is actually lower (better quality) than the "best" remote server, in this case USNO, which suggests that the data grooming algorithms do in fact deliver time more accurate than available from any single server separately.

Since all the servers in these experiment are synchronized to external sources, the phase noise of each remote source can be determined independently of the frequency noise. All the curves appear as slightly wiggly straight lines with slope −1 at low τ values, which is indicative of white phase noise, as described previously. On the other hand, the NOISE, PPS and LAN curves, which represent free-running clock oscillators, inflect upward, as expected with random-walk frequency noise. If the remote servers were not externally synchronized, their curves would inflect upwards in the same manner as the PPS and LAN curves.

So far, synthetic-noise models for only the NOISE, PPS and LAN sources have been shown to closely approximate the real noise characteristics. As the data shown in Figure 9 were measured using the NTP simulator and open-loop conditions, the question remains as to how faithfully the white phase/random-walk frequency model applies to the other sources under these conditions. The solid lines in Figure 10 show the measured Allan deviation for each source, while the dashed lines show the synthetic noise generated by the NTP simulator program. For this purpose, the frequency parameter was set to zero, while the phase parameter was set to agree with the actual phase noise at $\tau = 64$ s.

The results show general agreement in all cases at the lower values of τ, but some disagreement at the larger values for some servers. Since the phase modeling is most important at small values of τ, the disagreement at large values is not considered significant. It is of interest in passing to compare the USNO curve, which includes the data grooming algorithms, with the USNO (raw) curve, which used the same data, but with these algorithms disabled. With these algorithms in place, the errors are reduced by well over a factor of ten.

The results suggest that a useful predictive model for ordinary computer clock oscillators and real networks can be determined from the Allan intercept point ($\tau = x$, $\sigma = y$) and assuming white phase noise for $\tau < x$ and random-walk frequency noise for $\tau > x$. In this model, the Allan deviation is completely determined by the white phase noise characteristic of the network and operating system and the assumption of "good" (PPS) or "poor" (LAN) clock oscillators. A convenient x intercept for graphical analysis is $\tau = 64$ s, which can be used to predict the y intercept point at any other τ by simply multiplying by the ratio of the selected value to 64. The value $\tau = 64$ s is particularly useful, since this is the default minimum used in the NTP protocol. Figure 11 shows the y intercept at this value and the equivalent phase

Source	y Intercept (PPM)	p Parameter
NOISE	8.09×10^{-3}	2.93×10^{-7}
PPS	0.179	5.70×10^{-6}
LAN	0.955	3.10×10^{-5}
BARN	6.04	2.19×10^{-4}
PEERS	15.2	5.50×10^{-4}
USNO	23.1	8.38×10^{-4}
IEN	110	4.00×10^{-3}

Figure 11. Allan Deviation Phase Noise Parameters

noise generator parameter. In this table, the y intercept is determined directly from the Allan deviation plots. The p parameter of the synthetic PPS data is varied until the y intercept matches that of the measured PPS data. In a similar manner, the p parameter of the synthetic LAN data is varied until the y intercept matches that of the measured LAN data. The remaining p parameters are determined by scaling proportionally to the PPS value.

3.2 Performance Using Real and Synthetic Noise Sources

The next set of experiments is designed to explore the performance of the NTP data grooming and discipline algorithms with synthetic noise generators and the parameters of Figure 11 over a simulated interval of 30 days. In each experiment, random timestamps are generated according to the statistical model for each of the sources and the standard error of the discipline determined for τ from 64 s to 131,072 s. The intent of each experiment is to determine how the errors scale with τ using both PLL and FLL modes. This is done first using synthetic phase noise only, then using synthetic frequency noise only.

Figure 12 shows the results with the USNO source for both PLL mode (solid lines) and FLL mode (dashed lines) as a function of τ. When τ is varied with phase noise only (p), the standard error is approximated by the two nearly horizontal lines. In this case, the PLL outperforms the FLL by a factor of about ten. When τ is varied with frequency noise only (f), the standard error is approximated by the two lines with slope near 1.4. In this case, the FLL outperforms the PLL by a factor of about ten. In fact, the PLL becomes unstable at $\tau > 4{,}096$ s, as evidenced by the absence of plotted points in the figure. In this and other cases, the criterion for instability is the occurrence of one or more step corrections of 128 ms or more during the simulation run.

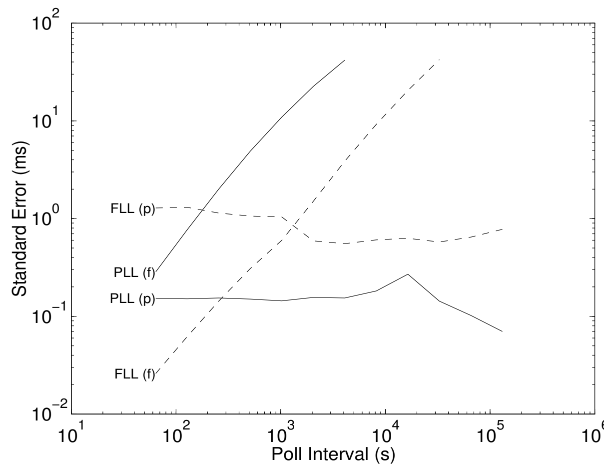

Figure 12. USNO Phase and Frequency Noise Comparisons with PLL and FLL

As evident from Figure 12, the PLL alone usually performs better under conditions of high phase noise and low frequency noise, while the FLL alone usually performs better under conditions of high frequency noise and low phase noise. While the generator parameters used in constructing the figure are typical of a network path consisting of two LANs connected by a relatively uncongested T1 network, the y coordinates of the lines scale directly as the noise parameters, but retain the slopes shown. Thus, since the frequency noise is assumed fixed in the experiment, the selection of which discipline mode to use depends only on the phase noise of the particular network path involved.

Assuming the frequency noise is fixed by oscillator type, it would seem that a simple measurement of phase noise would suffice to determine the Allan intercept and optimum point to switch between FLL and PLL modes. As in [2], the phase noise could be measured for each Internet server prior to regular operation and the optimum point determined. However, as confirmed by experiment, this is not practical in the current Internet. Figure 13 shows the absolute phase noise in log-y coordinates measured for the IEN path, which is typically congested during working hours in Europe and the U.S. The phase noise varies over three decades during the hours and days represented in the figure. An NTP client synchronized via this path would have to periodically measure the phase noise and recompute the optimum point. In principle, this could be done at intervals throughout the working day and a profile developed. This is still only an approximate solution and does not allow for minute-by-minute adjustment of the optimum point. This is in fact the motivation for the hybrid mode, in which the weight factors for the FLL and PLL predic-

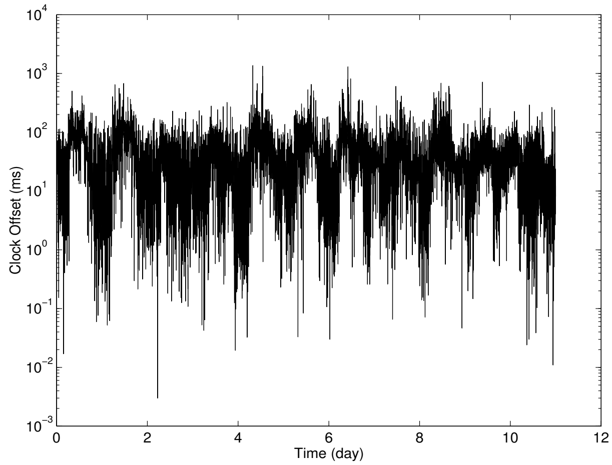

Figure 13. IEN Phase Noise

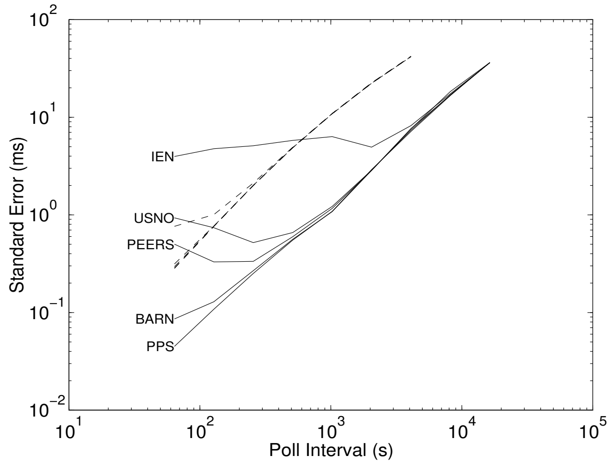

Figure 14. Hybrid Mode Standard Error by Source - Synthetic Noise

tions are determined in real time from the measured prediction errors.

Figure 14 shows a composite of the standard errors with noise parameters for each source in PLL mode (dashed lines) and hybrid mode (solid lines). For most sources over most of the range, the errors in hybrid mode are about ten times less than in PLL mode. The exceptions are when the phase noise overwhelms the frequency noise, which occurs for the most distant servers at small τ.

Comparing Figure 14 with Figure 9, it is apparent the inflection point where the curves corresponding to each source depart the asymptotes on Figure 14 are roughly a factor of ten above the Allan intercept point on Figure 9. This is an interesting comparison, in spite of the fact that the former shows standard error, which is a measure of time differences, while the latter shows Allan deviation, which is a measure of frequency differences. In addition, Figure 14 shows data collected under closed-loop conditions, while Figure 9 shows data collected under open-loop conditions. Nevertheless, the results suggest that an Allan deviation characterization of each network path and clock oscillator can be an accurate predictor of discipline performance.

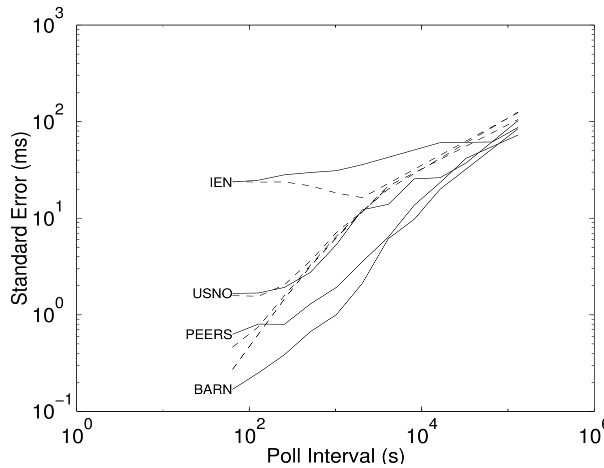

Figure 15. Hybrid Mode Standard Error by Source - Real Data

The final set of experiments is designed to determine how closely the discipline with synthetic noise emulates the discipline with real data. Figure 15 shows the standard error for real data with PLL mode (dashed lines) and hybrid mode (solid lines). Each curve shows the results with the phase data of Figure 9 and the frequency data of Figure 7. Compared to Figure 14, the curves for real data are somewhat jagged, mostly due to the spiky frequency characteristic evident in Figure 7, but in general follow the same characteristic as synthetic noise. As evident in the figures, the performance with real data is not quite as good as with synthetic data; however, the performance in hybrid mode is almost always better than PLL mode.

4. Conclusions

The results demonstrated in this paper show a substantial improvement in the performance of the NTP Version 4 discipline over both the original NTP Version 3 algorithm and the improved algorithm described in [4]. In addition, the new algorithm automatically selects the optimum combination of FLL and PLL data over a wide range of network jitter and oscillator wander while in regular operation and not requiring initial calibration. Perhaps the most striking result is that the new discipline is effective with τ well over one day, which is an attractive feature when telephone toll charges are involved.

As the complexity represented by the suite of crafted NTP algorithms have grown, it has become necessary to test and validate their performance by analysis and simulation. A good deal of the discussion in this paper has focused on the nature of the simulation process, design and analysis of the synthetic noise models and validation of the discipline performance in systematic simulation exercises. Many times during these exercises interesting things happened which resulted in significant improvements in the algorithm design. While this paper has not revealed them, there were in fact a number of fruitless experiments and dead ends which gave some insight into the relative merit of some design features, in particular the sanity checks, spike detectors and tolerance clamps. In addition, the insights gained suggested modifications which considerably simplified the FLL and PLL design.

Finally, the refinement of the NTP simulator program, while guiding the development of the various algorithms, may be most valuable as a specification vehicle, implementation tool and documentation aid for actual NTP Version 4 implementation. It should be mentioned in passing that the simulations demonstrated in this paper require a good deal of machine time. A full suite of all simulations, including the simulator program itself and Matlab analysis programs, requires an over two hours on a 300-MHz workstation.

5. References

References 3-5 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. Networking 3, 1* (February 1995), 42-50.

3. Mills, D.L., A. Thyagarajan and B.C. Huffman. Internet timekeeping around the globe. *Proc. Precision Time and Time Interval (PTTI) Applications and Planning Meeting* (Long Beach CA, December 1997).

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

5. Mills, D.L. Clock discipline algorithms for the Network Time Protocol Version 4. Electrical Engineering Department Report 97-3-3, University of Delaware, March 1997, 35 pp.

6. 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.