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 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. 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 reconfiguring 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 oscillator.

At the heart of the NTP design is the suite of algorithms that discipline the computer clock to an external source, either an NTP server elsewhere in the Internet or a local radio clock or telephone 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 Microsystems 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 the main topic of this report. It has evolved from simple beginnings to a sophisticated algorithm which automatically adapts to changes in its operating environment without manual configuration or real-time management functions. 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. As described elsewhere [7], 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. While requiring some additional complexity and per-site engineering, this combination has been shown to provide accuracies in the order of a few tens of microseconds [7].

The emphasis in this report is on recent developments in the clock discipline algorithm and its error modelling and evaluation. The current NTP Version 3 design with improvements described in [3] represent the departure point for this report. The goal is the engineering design and evaluation of the algorithm intended for NTP Version 4, which is in the implementation design stage. The performance of the new algorithm has been experimentally determined using a set of experiments described in this report. The experiments include a set of Unix shell scripts which call the NTP simulator program *ntpsim* with varying parameters meant to simulate normal and abnormal conditions found in actual operations. This program produces data files suitable for processing by Matlab programs which produce the data and figures used in this report. It can operate with raw data files collected by the NTP daemon for Unix *xntpd* during regular operation with multiple peers. In this case, it faithfully reproduces the actual NTP operations with a simulated clock discipline algorithm.

This report begins with a brief overview of the NTP Version 4 architecture and algorithms. A comprehensive description of the NTP Version 3 architecture, protocol and algorithms can be found in [6]. Next is a detailed description of the new clock discipline algorithm, including a mathematical analysis of its operation. Folowing this is a discussion of statistics relevant to the analysis of errors, including measures of time accuracy, also called phase accuracy, as well as frequency stability, using a statistic called Allan deviation (wrongly called Allan variance in some literature). Next is a discussion of experiment technique and results of a series of experiments to accurately characterize the Allan deviation statistic for typical computer clock oscillators. Statistical models are developed which characterize this statistic relative to several actual and synthetic sources representing typical Internet peer paths and computer workstation clocks.

Following this discussion, a series of experiments is described which demonstrate the closed-loop behavior of the NTP algorithms using real and synthetic data characteristic of LANs, “nearby” time servers in the same geographic area, and “distant” time servers with Internet paths spanning continents and oceans. The object is to verify the correct operation and quantify the accuracy possible using these servers. The results suggest improved means to adjust the algorithm parameters, in particular, how to manage the selection of clock discipline mode (phase-lock or frequency-lock or some combination of both) and how to adjust the poll interval. The report concludes with a detailed analysis of the improved algorithm performance using sources ranging from precision PPS sources and 64-s poll intervals to telephone modems and poll intervals ranging well over a day.

The software distributions containing the NTP simulator and experiments described in this report are available for distribution via FTP. The location of the distribution directory is given in the NTP web page `www.eecis.udel.edu/~ntp` and/or the author’s home page `www.eecis.udel.edu/~mills`. This directory contains the NTP simulator program `ntpsim.tar.Z`, programs and data to determine the computer clock oscillatory stability `allan.tar.Z`, and programs and data to evaluate NTP performance `ntpeval.tar.Z`. Appendix A contains a description of the program and directions for its use.

2. Network Time Protocol

While not in itself the subject of this report, an brief 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

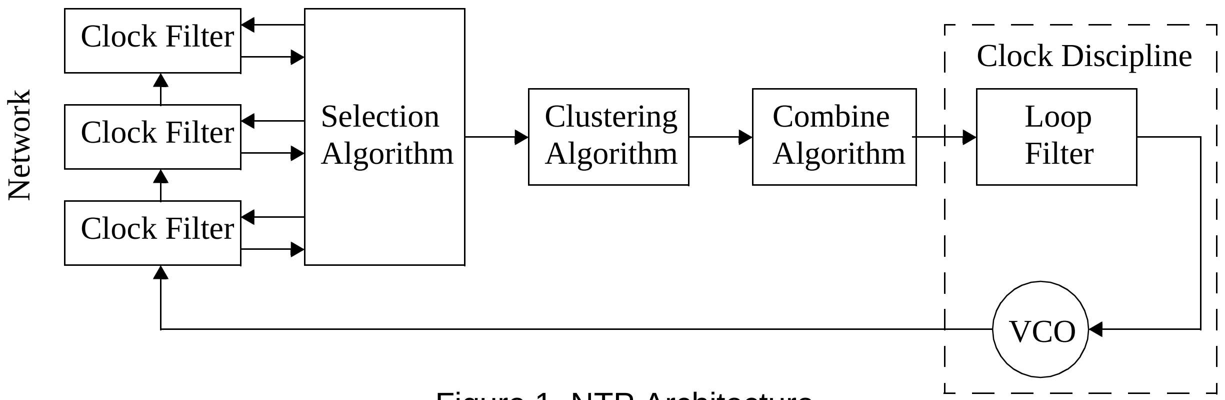

Figure 1. NTP Architecture

telephone networks. The servers at the lowest stratum are synchronized to national standards by radio or telephone 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*. The architecture of the NTP daemon is illustrated in Figure 1. 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, 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 delay and stratum, is used as a metric, called *synchronization* distance, to organize the NTP subnet itself as a shortest-path spanning tree with root at the primary server(s).

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 *peer dispersion* for each server as compared to the ensemble or *select dispersion*. The *survivors* remaining are then weighted by synchronization distance and combined to produce the clock correction used to discipline the computer clock.

In NTP Version 3, a clock correction is produced for each round of messages between a client and each 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 900 s 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. NTP Version 4 Architecture and Algorithms

While a detailed discussion of the new engineering design is beyond the scope of this report, a few details will serve to clarify the operation of the algorithms to be described. The NTP Version 4 architecture differs from Version 3 in subtle ways. For the purposes of this report, the most important difference is the manner in which the computer clock oscillator is adjusted. In both versions, the architecture is organized as a number of semi-autonomous cyclic processes, including a number of peer processes, the system process and the clock adjust process. Reference clocks are supported by specialized peer processes with device drivers specific to each radio or telephone modem. The operations of the peer processes and system process are managed by state machines with defined states, inputs, outputs and transition functions. Each process runs according to a cyclic schedule at defined poll intervals.

In NTP Version 3, the system process is called for every received NTP message to update the system state variables, including the clock time, frequency and error estimates. In NTP Version 4, the system process runs at regular poll intervals determined by the ambient network jitter and clock oscillator wander. As in NTP Version 3, the peer poll intervals are determined as the minimum of the system poll interval and the peer poll interval specified in the latest message received from the peer. Decoupling the system and peer poll interval in this way improves the accuracy and stability of the clock discipline algorithm, as well as simplifies the interactions between the peer processes and the system process. In this report, subsequent reference to poll interval should be interpreted as the system poll interval. Furthermore, the poll interval values used in this report and in the specification and implementation are all in powers of two. The motivation for this will become clear in later discussion in this report.

Each peer process independently polls a remote server or local device driver at intervals specified by the server/driver and system process and updates peer state variables, including clock offset, roundtrip delay and dispersion. The clock filter algorithm for each peer process maintains a list of the eight most recent samples of offset, delay and dispersion and normally selects the sample with minimum delay to represent the peer offset, delay and dispersion values. The values for possibly several peer processes are read by the system process at intervals depending on the network jitter and clock oscillator wander and used to update system time and frequency corrections. The corrections are read by the clock adjust process, which runs independently at intervals of 1 s, and used to adjust the computer clock time using a standard Unix system call which amortizes the adjustments evenly over the second.

This design allows each peer process to operate a polling algorithm specific to each peer type, such as a network peer, radio clock, telephone modem or ISDN connection. Some peer types, such as ISDN connections, incur usage charges, so an economical design must use a relatively large poll interval. However, the NTP design is based on redundancy, diversity and error control through the use of continuous polling. In the ISDN case, for example, an appropriate design would make infrequent ISDN calls, but send a burst of messages at every call, in order that the clock filter algorithm operates at maximum efficiency.

In NTP Version 3, the peer dispersion includes the weighted offset differences between the selected sample and each of the others in order of synchronization distance, plus an increment depending on sample age, in order to model the expected error accumulation due to frequency instability. However, in NTP Version 4, the weighted offset differences, called the peer time dispersion, are calculated for each new message and recorded separately from the accumulation due to sample age, called the peer frequency dispersion. As will become apparent later, this is necessary in order to accurately model the statistical errors due to all noise sources and use these to control the weighting factors used by the clock discipline algorithm.

The NTP Version 4 clock filter algorithm includes a spike detector which measures the ratio between each new clock offset sample and the time dispersion of the previous samples. If this ratio exceeds 10, the new sample is not made visible to the clock selection algorithms of the system process, although it is included in the dispersion calculation[1]. This avoids glitches due to *popcorn noise*, which is commonly found on LANs operating at moderate to heavy loads, and avoids instability at small poll intervals in FLL mode with the clock discipline algorithm.

Another significant difference between the two versions is that the error budgets in NTP Version 4 use root-mean-square (RMS) accumulations, rather than absolute-value accumulations, as in NTP Version 3. The reason for this is to develop accurate noise models for precise predictions of computer clock time and frequency using a hybrid phase/frequency-lock adjustment process, as described later in this report. However, the new design does violate one of the assumptions of the original design, that an implementation not require hardware or software multiply/divide operations, since all arithmetic is done using only add, subtract and shift operations. The new design calls for multiply/divide operations, which are now in general ubiquitous, even on embedded controller systems, as well as square-root operations. It is the design intent that the square-root operation be done using a subroutine and not more than three iterations of the Newton-Raphelson square-root algorithm.

4. Clock Discipline Algorithm

The clock discipline algorithm adjusts the computer clock time as determined by NTP, compensates for the intrinsic frequency error, and adjusts the poll interval and loop time constant dynamically in response to measured network jitter and oscillator stability. In NTP Version 4, the algorithm functions as a true hybrid of two philosophically quite different feedback control systems. In a phase-lock loop (PLL) design, the measured time errors are used to discipline a type-II feedback loop which controls the phase and frequency of the clock oscillator. In the frequency-lock loop (FLL) design, the measured time and frequency errors are used separately to discipline type-I feedback loops, one controlling the phase and the other controlling the frequency. Experiments described later in this report confirm a hybrid combination of both designs can significantly improve the performance of the system under widely varying conditions of network delay variations and clock oscillator instability.

In NTP Version 4, the system process polls the peer processes at intervals from a few seconds to over a day, depending on peer type. When a new sample of offset, delay and dispersion is available in a peer process, a bit is set in its state variables. The system process, upon noticing this bit,

1. In this report, apparently arbitrary constants are used in some cases. Some of these values have been determined by careful simulation and modelling, others on the basis of good engineering judgement tempered by experience, and still others as wild guesses. It may happen that, as experience with the new features accumulates, the values may be changed.

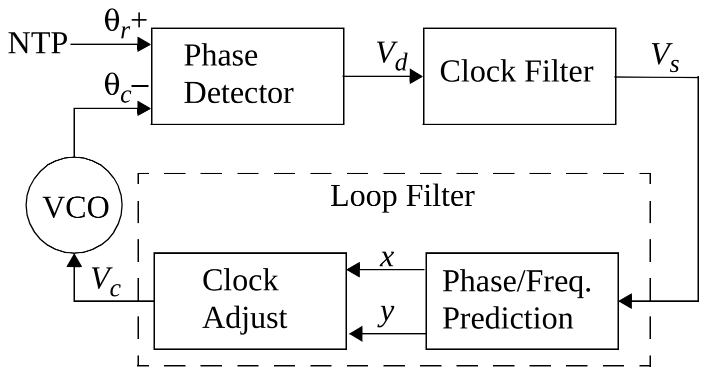

Figure 2. Clock Discipline Algorithm

clears it and calls the clock selection, clustering and combining algorithms, as in the NTP Version 3 design. Normally, this results in a combined sample offset, which is then used by the clock discipline algorithm to control the computer time. This algorithm adjusts the clock oscillator time and frequency with the aid of the clock adjust process, which runs at intervals of one second. A overview of the design is given in following sections. The detailed design of the algorithm, as well as most others used in the peer and system processes, is embodied in the NTP simulator program, which is described in Appendix A.

The clock discipline algorithm is implemented as the feedback control loop shown in Figure 2. The variable θ_r represents the reference phase provided by NTP 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 V_s taken at the sample selected by the algorithm. The clock selection, clustering and combining algorithms (not shown) provide additional processing, as decribed in [3]. The loop filter, with impulse response $F(t)$, produces a correction V_c, which controls the VFO frequency ω_c and thus its phase θ_c. 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].

As described in [6], the original NTP Version 3 clock discipline algorithm, 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, which is normally equal to the poll interval, but could be larger on occasion due to a spike, for instance. If τ is smaller than about one kilosecond, PLL mode is used; otherwise, FLL mode is used. This improves the accuracy and stability in some modes of operation, specifically those involving toll charges, but has proved suboptimal for reasons described later in this report. The new algorithm uses a combined approach with a true hybrid PLL/FLL design which gives good performance with values of τ from a few seconds to well over one day, depending on accuracy requirements and acceptable network overheads or toll charges.

In the new design, the loop filter, shown in Figure 3, is implemented using two subalgorithms, one based on a linear, time-invariant PLL, and the other on a nonlinear, predictive FLL. Both predict a time correction x as a function of phase error θ, represented by V_s in the figure. The PLL predicts

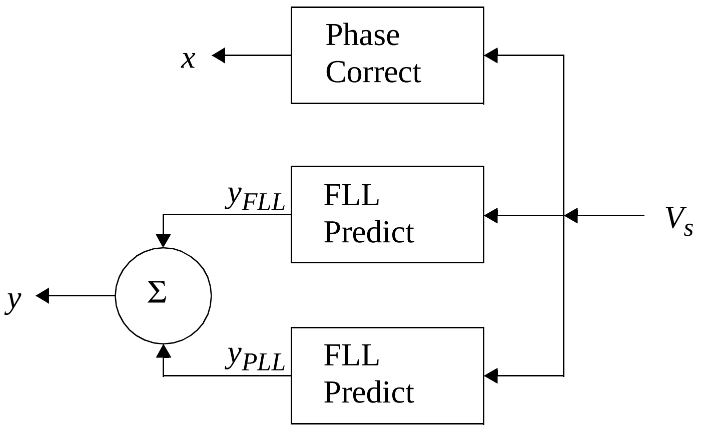

Figure 3. FLL/PLL Prediction Functions

a frequency adjustment y_{PLL} as an integral of past time offsets, while the FLL predicts a frequency adjustment y_{FLL} directly from the difference between the last time correction and the current one. 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 adjust the VCO frequency and close the feedback loop, as shown in Figure 2.

In the following, the index *k* denotes the *k*th iteration of the algorithm, where x_k is the time or phase, y_k the frequency or rate, and τ_k the interval since the last update. The notation $\langle x \rangle_n$ indi-cates the average of the *n* most recent sample measurements of the statistic *x*, and the notation $\hat{x}$ the values predicted for the next update of *x*. In the most general formulation, an algorithm that corrects for clock time and frequency errors computes a prediction $\hat{x}_k = x_{k-1} + y_{k-1}\tau_k$ at the *k*–1th update and then a correction

$$x = x_k - \hat{x}_k \tag{1}$$

at the *k*th update. As each correction is determined, the clock time is adjusted by *x*, so that it displays the correct time, and the clock frequency *y* is adjusted to minimize the corrections in future. Between updates, which can range from seconds to over a day, the clock adjust process amortizes *x* in small increments at adjustment intervals T_A. At the beginning of each adjustment interval, the VCO frequency is

$$f = \frac{\Delta x}{T_A} = ax + y \ , \tag{2}$$

and *x* is multiplied by ,1 − 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 elsewhere [7], 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 *ax* term in (2) is necessary for stability; in both PLL and FLL algorithms, it is also necessary in order to prevent monotonicity violations when the magnitude of adjustment is large

Initially, the frequeny y is zero. At each update, the PLL and FLL algorithms independently calculate a frequency adjustment y_{adj}, which is added to y_{k-1} at the kth update to determine the frequency $y = y_k$ used until the next update. In the PLL algorithm, the frequency at the kth update is determined from the current and all previous updates x_i at intervals τ_i,

$$y_k = b^2 \sum_{i=1}^{k} x_i \tau_i , \tag{3}$$

where b is a constant between zero and one in Hz. Given the frequency at the k–1th update is y_{k-1}, the maximum-likelihood PLL frequency adjustment prediction at the kth update is

$$y_{PLL} = b^2 x_k \tau_k . \tag{4}$$

In order to understand the PLL dynamics, it is useful to consider the limit as τ approaches zero. From (2) and (3), the oscillator frequency is adjusted by

$$f(t) = ax(t) + b^2 \int_0^t x(\tau) d\tau . \tag{5}$$

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,

$$\theta(s) = x(s) \frac{1}{s} \left(a + \frac{b^2}{s} \right) = x(s) G(s) , \tag{6}$$

where the extra pole $1/s$ at the origin is due to the integration which converts the frequency $f(s)$ to phase $\theta(s)$. After some rearrangement, the transfer function $G(s)$ can be written

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

where $\omega_c = b$ is the loop gain and $\omega_z = \frac{b^2}{a}$ is the corner frequency. From elementary theory, this is the transfer function of a linear, time-invariant, type-II PLL which can minimize both time and frequency errors.

The averaging interval is determined by the loop time constant T_c, which depends on the choice of a and b; however, these constants must be chosen so that the damping factor $\xi = \frac{\omega_c}{2\omega_z} = \frac{a}{2b} = 2$, in order to preserve good transient response. For good stability, T_c should be at least eight times the total loop delay which, because of the clock filter delay, is eight times the update interval τ. For values of $a = 2^{-10}$, $b = 2^{-12}$ and $\tau = 2^6$ s for instance[2], the PLL has a risetime in response to a time step of about 53 minutes and a 63% response to a frequency step of about 4.25 hours, which is a useful compromise between stability and network overhead on a LAN. The value $T_c \approx 3000$ s is near the Allan intercept point for the sources considered in the experiments to follow, and in that sense would be considered optimum. An update interval of 2^6 s is appropriate with the above values of *a* and *b*. Values of τ as small as this 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 τ, the desired transient characteristic is preserved if both *a* and *b* vary as $1 / \tau$.

In the FLL algorithm, the frequency at the *k*th update is determined directly from the differences between the current and previous update times. In the previous design, adapted from [2], the FLL operates is the same way as the PLL, except that y_k is determined indirectly from a frequency estimate $\bar{y}_k$ computed as the exponential average

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

with $w = 4$ determined by experiment. The goal of the clock discipline algorithm 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, an appropriate prediction which minimizes future error is

$$y_k = \bar{y}_k + \frac{x_k - x_{k-1}}{\tau_k} = \bar{y}_k + \frac{x_k}{\tau_k}. \tag{9}$$

To simplify the design and to minimize the number of state variables in the NTP Version 4 design, the $\bar{y}_k$ variable is eliminated and the maximum-likelihood frequency adjustment prediction is expressed simply as

$$y_{FLL} = \frac{x_k}{w\tau_k}. \tag{10}$$

This arrangement simplifies implementation without significantly affecting accuracy and stability. In summary, in both FLL and PLL modes, the time correction is *x* at each update. In FLL mode, $y_{adj} = y_{FLL}$ is used for the frequency prediction; while, in PLL mode, $y_{adj} = y_{PLL}$ is used instead.

4.1 Hybrid FLL/PLL Combining Algorithm

The NTP Version 3 design selects either the FLL or PLL mode on the basis of update interval. PLL mode is used at intervals less than 1,024 s and FLL mode used otherwise. While this results in a useful compromise, it does not provide for the automatic selection on the basis of prevailing

2. These numbers were originally expressed as powers of two, since multiply-divide arithmetic was done using only left and right shifts.

network jitter and oscillator frequency 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 averaged over previous predictions. The result is that the PLL prediction is weighted more heavily under conditions of extreme time jitter due, for example, to network congestion, while the FLL prediction is weighted more heavily under conditions of extreme oscillator frequency wander due, for example, to large temperature variations.

As in the original design, T_c is the loop time constant and τ the interval since the previous update. At each update, the time correction is computed from the offset θ determined by the combining algorithm. Recall from the previous section that x is the time correction remaining from the latest update and diminishes at a rate depending on T_c. Also, let y_{adj} be the frequency adjustment computed from the PLL and FLL predictions, as described below. The FLL and PLL prediction errors are computed at each update

$$x_{FLL} = \theta - x - (y_{FLL} - y)\tau \text{ and } x_{PLL} = \theta - x - (y_{PLL} - y)\tau . \tag{11}$$

The RMS prediction errors ε_{FLL} and ε_{PLL} are computed from successive sample squares of x_{FLL} and x_{PLL} saved in separate shift registers. The number of samples n averaged, or equivalently the averaging time T, depends on T_c, which itself depends on the poll interval. The motivation for this can be found in the observation, confirmed by experiments described later in this report, that the optimal averaging time is near the Allan intercept point, which is usually in the range $1000 < T < 4000$ s in typical computer clock oscillators. Thus, averaging times much less than this are likely to be contaminated by excess phase noise, while times much longer than this are likely to be contaminated by excess frequency noise.

In the specification and implementation, it is convenient to implement poll intervals as shifts, primarily to avoid multiply/divide operations, a decision proved optimal in hindsight, according to the experiments described later. It is usually the case that the update interval τ is close to the poll interval; thus, if *poll* is the variable used for this purpose, $\tau \approx 2^{poll}$. Since a new sample is usually produced for every poll, the RMS prediction errors include only n sample squares such that $n2^{poll} < T$:

$$\varepsilon_{FLL} = \sqrt{\langle x^2_{FLL} \rangle_n} \text{ and } \varepsilon_{PLL} = \sqrt{\langle x^2_{PLL} \rangle_n}, \tag{12}$$

Next, the FLL and PLL frequency predictions are computed

$$y_{FLL} = \frac{\theta - x}{w\tau} \text{ and } y_{PLL} = \frac{\theta\tau}{b^2}, \tag{13}$$

where w is the FLL averaging constant in (8) and b is the PLL frequency constant in (3). Finally, the new frequency adjustment is computed

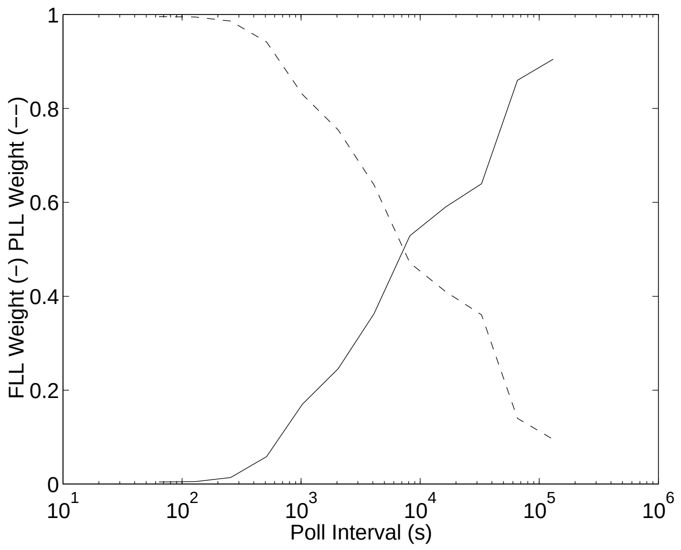

Figure 4. USNO Weight Functions

$$y_{adj} = \frac{y_{FLL}\varepsilon_{PLL} + y_{PLL}\varepsilon_{FLL}}{\varepsilon_{FLL} + \varepsilon_{PLL}}. \tag{14}$$

The averaging constant w in (13) also depends on T_c and poll interval. The intent in this design is to match the averaging time to the Allan intercept point, as described later in this report. A useful formulation is $w = \max(G - poll, 2)$, where $G = 10$ is determined by experiment and the constant 2 is necessary to avoid instability at large poll intervals.

In the hybrid mode of the clock discipline algorithm, the weight assigned the FLL and PLL predictions determines the influence of each one on the actual frequency adjustment. Figure 4 shows the FLL weight factor (solid line) as a function of poll interval for the USNO source used in experiments described later. The PLL weight factor (dashed line) is one minus the FLL weight factor. Clearly, the PLL predictions dominate at small poll intervals, while the FLL predictions dominate at large intervals, as expected. Figure 5 shows the standard error (defined later in this report) with the USNO source as a function of poll interval using the FLL or PLL alone and where both are combined using these weight factors. Recall that these weight factors are computed from the RMS errors previously observed between the predicted offsets and the actual offset. From about 100 s to 1000 s, the hybrid mode is better than either the PLL or FLL modes and above 1000 s is only a little worse than the FLL mode. Below 100 s the hybrid mode is somewhat worse than the PLL mode and a little better than the FLL mode, but still better than the millisecond.

4.2 Poll-Adjust Algorithm

NTP Version 3 time servers and clients operate today using network paths that span the globe. In very many cases, primary (stratum 1) servers operate with several hundred clients or more. It is necessary to explore every means with which the poll interval used by these clients can be increased without significantly degrading clock accuracy or stability. The NTP Version 4 clock discipline algorithm allows a significant increase in poll interval without compromising accuracy,

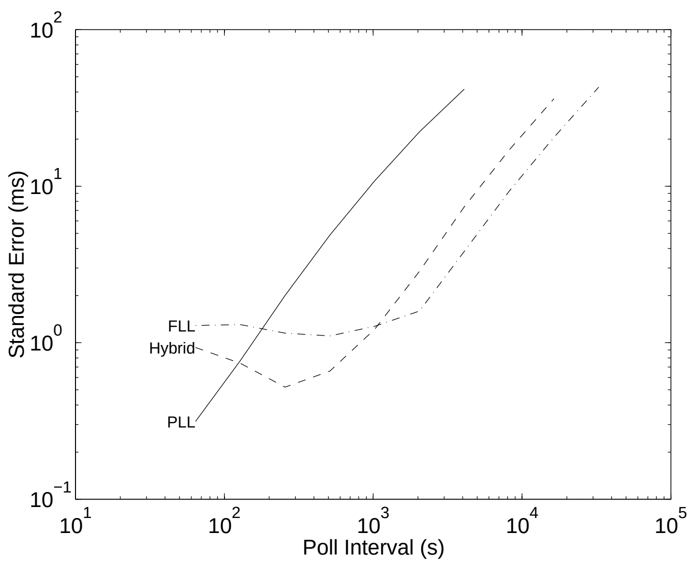

Figure 5. USNO Raw/Filtered Data Comparison

while at the same time adapting dynamically to widely varying network jitter and clock oscillator wander.

The experiments described later in this report show that, in almost all cases, the standard error increases as the poll interval increases using either the FLL or PLL algorithms, due primarily to the effect of clock oscillator instability. Since the overhead decreases as the poll interval increases, a method is needed to select the poll interval as the best compromise between highest accuracy and lowest 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 poll intervals default to values appropriate for almost all network and computer configurations. For network peers these values default to $2^6 = 64$ s and $2^{10} = 1{,}024$ s, respectively, while for telephone modem peers, the values default to 2^{10} and $2^{14} = 16{,}384$ s, respectively. However, in NTP Version 4, these values can be changed to fit special circumstances to as small as $2^4 = 16$ s and as large as $2^{17} = 131{,}072$ s. Within the range between these maximum and minimum values, the clock discipline algorithm automatically manages the poll interval to match prevailing network jitter and oscillator wander. A singular point to be emphasized is that, using the new algorithm, it is not necessary to clamp the poll interval to the minimum when switching among different synchronization sources. In cases of moderate to severe network jitter and where multiple sources exist, frequent *clockhopping* in this way can be a problem in NTP Version 3, but should no longer be a problem in NTP Version 4.

A key statistic in controlling the poll interval is the RMS error measured by the clustering algorithm as it works to sift the best subset of clocks from the current peer population. Called *select dispersion* ε_{SEL}, sample squares of this statistic are held in a shift register. The system dispersion ε_{SYS} is then calculated from the RMS sum of ε_{SEL} and the peer dispersion ε_{PEER} of the selected peer

$$\varepsilon_{SYS} = \sqrt{\langle \varepsilon^2_{SEL} + \varepsilon^2_{PEER} \rangle_n} \qquad (15)$$

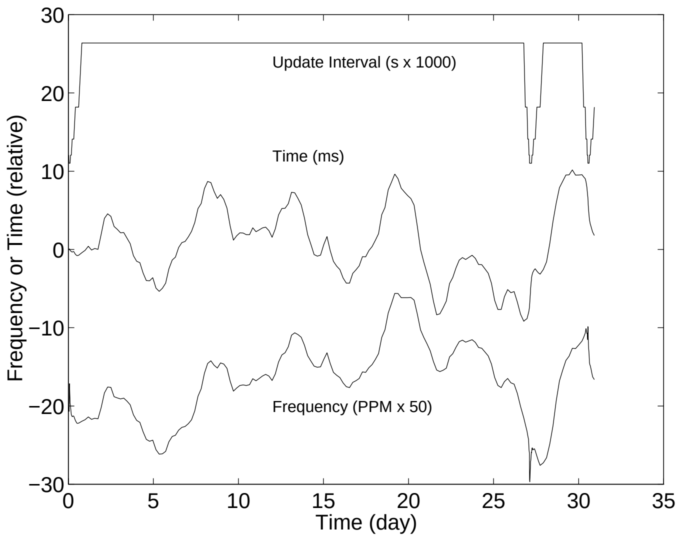

Figure 6. Poll Interval Adjustment

where n = 4 samples is chosen by experiment. If $|\theta| > Y\varepsilon_{SYS}$, where Y = 5 is experimentally determined, the oscillator frequency is deviating too fast for the clock discipline algorithm to follow, so the poll interval is reduced in stages to the minimum. If the opposite case holds for some number of updates, the poll interval is slowly increased in steps to the maximum. A hysteresis mechanism built into the algorithm prevents unnecessary dithering of the interval when not productive. Under typical operating conditions, the interval 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 6 shows the poll intervals of a typical workstation over a 30-day period using synthetic data as described later in this report. In this figure, the baseline is +10 for the poll interval curve, zero for the offset curve, and -20 for the frequency curve. Most of the time is spent at the maximum poll interval, in this case 16,384 s, with brief excursions to lower intervals not less than 1,024 s when the frequency deviates too rapidly for the discipline loop to follow. This particular figure shows the expected behavior for a typical telephone modem, where it is important that the poll interval remain at large values whenever possible. In particular, it is important that the initial frequency adaptation when the clock discipline algorithm is first started be substantially complete within only the first few samples before the poll interval starts to increase. In Figure 6, where the initial time and frequency offsets are zero, this occurs after the eighth call.

4.3 Clock State Machine

The clock discipline algorithm operates over an extremely wide range of network jitter and oscillator wander characteristics. As determined by past experience and experiment, the various 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, linear, time-invariant systems (PLL) and even predictive systems (FLL) may fail to cope with the induced time and/or frequency transients. The results can be frequent time step changes and very large time and frequency errors.

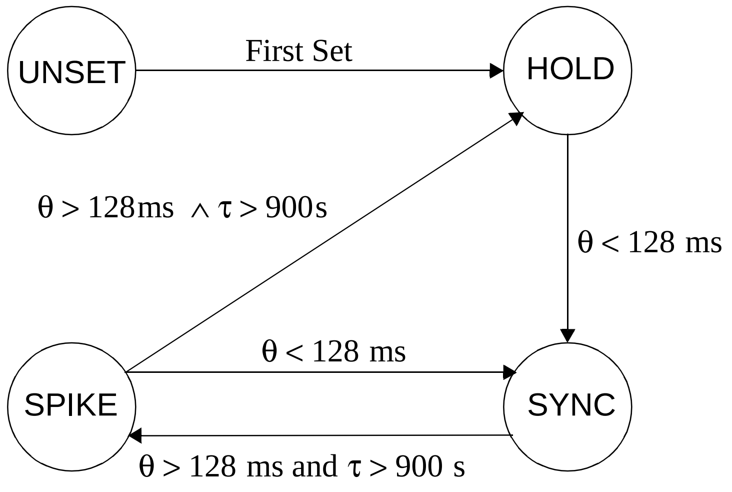

Figure 7. Clock State Macnine

In order to deal with very large transients at and after startup, the clock discipline algorithm is managed by the state machine shown in Figure 7. Each of the four states has defined inputs, outputs and transition functions. Initially, the machine is unsynchronized and in UNSET state. If the minimum poll interval is 1,024 s or greater, 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 the interval is less than 1,024 s, these actions will not occur until after several updates, to allow the synchronization distance to be reduced below 1 s, and allow the various algorithms to accumulate reliable error estimates.

In HOLD state, the various sanity checks, spike detectors and tolerance clamps are disabled, in order to provide rapid adaptation to possibly very large frequency errors up to 500 PPM. In addition, the clock discipline algorithm is forced to operate in FLL mode only, which allows the fastest adaptation to the particular oscillator frequency. Once entering HOLD state, the machine remains in this state for at least five updates, in order to complete, as far as possible, the frequency adaptation process. After this and the nominal clock offset has decreased below 128 ms, the machine transitions to SYNC state and remains there pending unusual conditions.

In SYNC state, the sanity checks, spike detectors and tolerance clamps are operative. To protect against frequency spikes that might occur in FLL predictions at small update intervals, the frequency adjustments are clamped at 1 PPM. To protect against runaway frequency offsets that might occur in FLL predictions at very large update intervals, the frequency estimate is clamped at 500 PPM. To protect against disruptions due to severe network congestion, frequency adjustments are disabled if the system dispersion exceeds 128 ms. These sanity checks are in both the NTP Version 3 and NTP Version 4 algorithms, although the former has no state machine and so cannot react quickly to large frequency excursions.

In NTP Vrsion 4, several mechanisms are built into the various algorithms to cope with less traumatic disruptions. One of these is the spike detector in the clock filter algorithm mentioned previously. A similar spike detector is used by the clock discipline algorithm in SYNC state. If

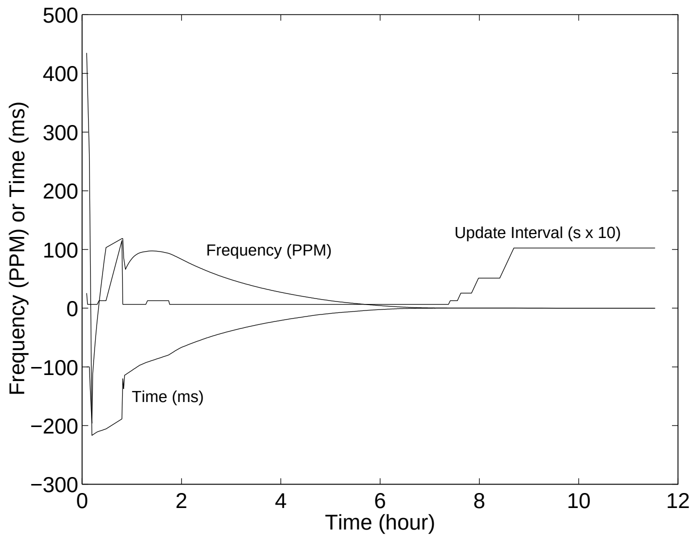

Figure 8. Initial Transient Adaptation - Network Peer

the magnitude of an offset exceeds 10 times the system dispersion, the spike is discarded as if it never existed. However, the increased dispersion due the spike is calculated and included in the dispersion budget. Thus, if a true time step were to occur, the first one or two samples, depending on the magnitude, may be discarded, but eventually the step will be recognized and corrections made.

Special provisions are made in SYNC state if the time offset exceeds 128 ms, which may happen occasionally when the network is congested or when some minor disruption occurs. It can also happen upon the occasion of a leap second, when the local clock has been automatically stepped, but for some reason the remote peer or local radio clock has not implemented the step. In both cases, the optimum response is probably to ignore the spike, unless it persists for some time. Accordingly, if succeeding offsets are less than 128 ms, this behavior is justified. However, if offsets greater than 128 ms persist for a watchdog interval of 900 s, the eventual response should be to believe them. This is the intended behavior of both the NTP Version 3 and NTP Version 4 algorithms.

In the NTP Version 4 algorithm, when the watchdog interval has been exceeded and an update arrives with greater than 128 ms offset, the state machine transitions to SPIKE state, but does not set the clock. If the next update after that has offset less than 128 ms, the machine transitions to SYNC state and adjusts the clock phase and frequency as if in that state. In this case the prior update is considered a spike and ignored. If the next update in SPIKE state has offset greater than 128 ms, the machine transitions to HOLD state and sets the clock. Since the sanity checks are disabled in HOLD state, the clock discipline algorithm can quickly adapt to the new time and frequency as described previously.

Figure 8 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 during the five-sample HOLD interval, 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

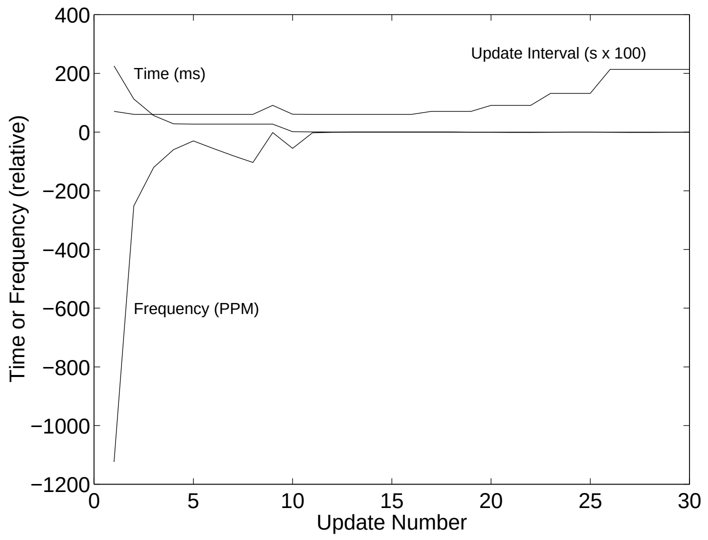

Figure 9. Initial Transient Adaptation - Telephone Modem Peer

in that state. This eventually causes the offset to exceed 128 ms and, after the watchdog interval, the state machine transitions to HOLD state. This process may repeat a time or two, until the residual frequency offset is sufficiently reduced. In Figure 8, after about six hours, the adaptation is complete and the poll interval starts to ramp up from 64 s to 1,024 s. 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.

A critical factor in the case of toll services is the speed with which the initial time and frequency can be adapted and the poll interval increased to the maximum. Figure 8 shows the behavior using network peers, where the minimum poll interval is usually 64 s. Figure 9 shows the behavior using telephone modem services, where the minimum poll interval is usually 1,024 s. Here, the initial time and frequency offsets are as in Figure 8, but the clock reading error has been increased to 1 ms, which is typical of telephone modem services. Even under these extreme conditions, the adaptation is mostly complete and the poll interval begins to increase after 15 calls.

5. Performance Evaluation

The performance of the new clock discipline algorithm is best evaluated using a simulation approach. There are two 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 simulator environment is not burdened by the system infrastructure in which the daemon must operate, such as I/O code and monitoring code. Most of the development and debugging done to develop the simulator uses Microsoft C++, which includes a comprehensive debugging capability far superior to the ordinary Unix development tools. Third, the actual code developed in this way represents a model for the development of the eventual formal specification and implementation, as well as a documentation and specification aid. This is the same approach used in the development of the original NTP Version 3 specification and implementation.

While intended primarily to verify the correct operation of the NTP algorithms under normal and abnormal conditions, the simulator has also been used to explore new algorithms and methods to discipline the computer clock. This includes means to automatically adjust algorithm parameters to match current conditions of phase and frequency noise, which may vary from time to time and place to place. In this way and using real and synthetic data, the parameters can be varied during a series of simulation runs and compromise values can be selected which remain valid over a wide range of conditions. The parameters include architecture constants, such as maximum time and frequency tolerances, outlyer removal and adaptive-parameter estimators. In addition, the experiments have suggested a number of design improvements over the original NTP Version 3 algorithms and identified a few shortcomings. These have resulted in the changes documented in this report and in the eventual NTP Version 4 specification and implementation.

The simulator program operates using real data collected by the existing NTP daemon for Unix and Windows, as well as data synthesized by internal phase and frequency noise generators, or a combination of real and synthetic data. The generators produce white phase noise and random-walk frequency noise, both using zero-mean Gaussian processes. As shown in subsequent discussion, these generators can quite 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 is stabilized, verify the expected operation using real data.

The most visible statistic in evaluating the performance of a timekeeping system is the time error, either in the form of mean absolute error, as in NTP Version 3, or the RMS error, as in NTP Version 4. The original reason for the NTP Version 3 design was to avoid the need to compute squares and square roots, which in most embedded systems would be considered an implementation burden. Whether or not mean absolute error or RMS error is used, the only difference between the two statistics is the arithmetic oprations involved; the statistic is used the same way in both verions. While the RMS error calculations can be a burden in some implementations, it is necessary in order to develop accurate predictions and determine weight factors used by the clock discipline algorithm.

An important strategy in the following experiments is the use of primary (stratum 1) time servers. Since primary servers are independently synchronized to external references, such as GPS receivers or ACTS telephone modems, their time bases are stabilized with presumed zero time and frequency error. This allows the errors due to the network jitter to be separated from the errors due to oscillator stability and for the one-way delays on the outbound and return paths to be measured separately. Since the time offset between primary servers is effectively zero, it is convenient to evaluate the performance using a statistic of RMS error about zero, called in this report *standard error*. The clock stability model is separately derived from a series of measurements involving a free-running clock oscillator and a precision time source. The frequency offset measurements are then collected in a file which is read by the simulation program and integrated with the real or synthetic time offset data.

There are three sets of experiments described in the following. In the first set, the *Allan deviation* characteristics of typical workstations and network paths is investigated to develop profiles which characterize common client-server configurations. In the second set, the behavior of the clock discipline algorithm is explored using synthetic noise sources, in order to determine how the expected errors due to network jitter and oscillator stability scale relative to poll interval. These provide the basis and justification for the new fully hybrid approach in the design. In the third set, the response of the algorithm to real data is investigated with emphasis on the behavior of the automatic poll-adjust scheme to different combinations of network jitter and frequency wander.

5.1 Allan Deviation Experiments

The stability characteristics of typical computer clock oscillators can be experimentally determined through a set of experiments, such as described in this section. The experiments are designed to calculate the Allan deviation for a typical computer clock oscillator synchronized via Internet paths to as many as 19 remote time servers in North and South America, Europe and Asia. Each experiment in the set involves measured one-way delays between NTP primary (stratum 1) time servers synchronized to external sources, such as GPS receivers or cesium oscillators. The experiments are designed to do two things: (a) evaluate the Allan deviation characteristic for typical computer clock oscillators and Internet paths between selected NTP time servers, and (b) validate that the synthetic random noise generators incorporated in the simulator program realistically emulate the real data on a statistical basis.

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 about 1 ms. Each pulse causes 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 or 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 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.

In order to correct for an intrinsic frequency error, adjustments are made at intervals of $T_A = 1$ s. as described previously. 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 the 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 elsewhere [3].

Assuming the clock discipline algorithm 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 kth measurement and τ be the interval since the last measurement. Define the *fractional frequency*

$$y_k \equiv \frac{x_k - x_{k-1}}{\tau}, y_0 = 0, \tag{16}$$

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 τ is the same as the averaging interval, the 2-sample Allan variance is defined

$$\sigma_y^2(\tau) \equiv \langle (y_k - y_{k-1})^2 \rangle_N = \frac{1}{2(N-2)\tau^2} \sum_{k=2}^{N-1} x_k^2 - 2x_k x_{k-1} + x_{k-2}^2 \tag{17}$$

and the Allan deviation 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 reference Allan deviation profiles for common workstation quartz crystal oscillators under typical room-temperature conditions. The Sun SPARC IPC workstations used in the experiments, and most others like them, have rather poor oscillator stability characteristics, compared to some others, such as the Digital Alpha, but are useful indicators of typical performance.

The PPS experiment measured the free-running clock offsets relative to an external precision PPS source. This experiment used data collected over about five days for a SPARC IPC and monitored using the *ppsclock* line discipline [7] connected via a level converter and pulse regenerator to the PPS signal from a cesium oscillator. A special test program measured the PPS offsets at 2-s intervals and recorded them in a data file. The LAN experiment measured the free-running clock offsets relative to a primary (stratum-1) time server on the same network wire. This experiment used data collected over about fifteen days for a SPARC IPC and monitored using NTP. The NTP daemon measured the primary clock offset at 16-s intervals and recorded them in a data file.

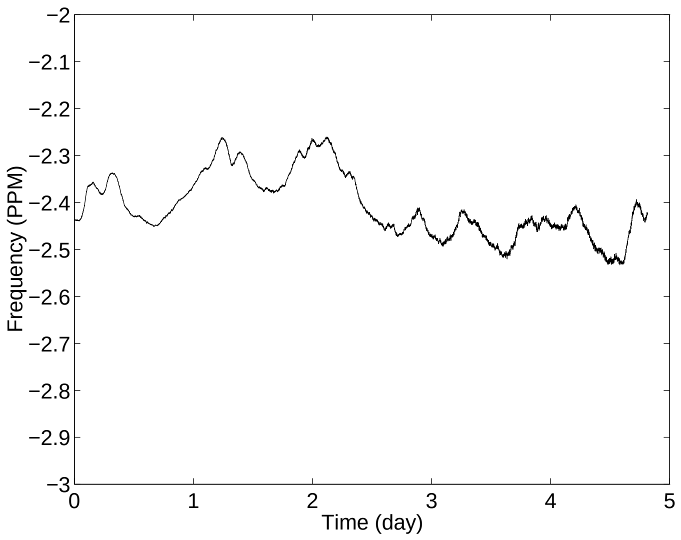

Figure 10. PPS Frequency

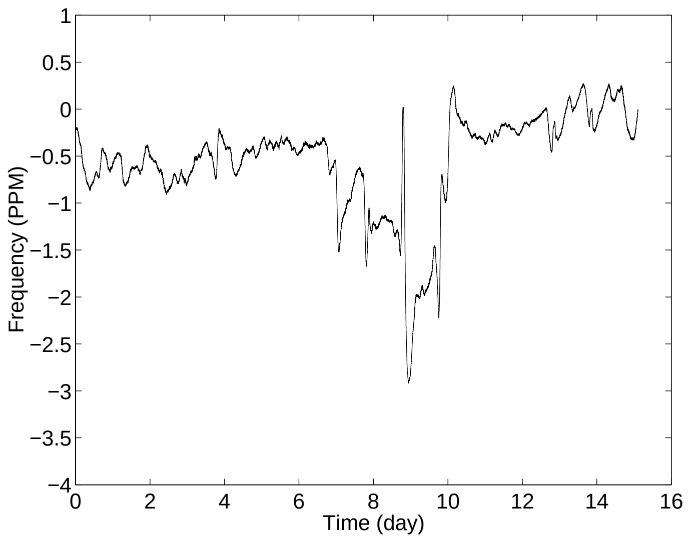

Figure 11. LAN Frequency

The oscillator frequency characteristic in the PPS experiment is shown in Figure 10 over the five-day period. The oscillator frequency varies over about a 0.25 PPM range, which would normally be considered very minor and characteristic of a closely regulated room temperature. The oscillator characteristic in the LAN experiment is shown in Figure 11 over the fifteen-day period. Here, the oscillator frequency varies over a 3-PPM range, sometimes abruptly, which is characteristic of room temperature changes of a few degrees. The PPS experiment was conducted in winter, when the room temperature was thermostatically controlled, and is typical of "good" conditions. The LAN experiment was conducted in spring, when the laboratory windows were open and the temperature allowed to follow the weather, and is typical of "poor" conditions.

The results of the PPS and LAN experiments were processed by Matlab programs to produce graphs plotting Allan deviation σ in parts-per-million (PPM) with respect to time interval τ in sec-

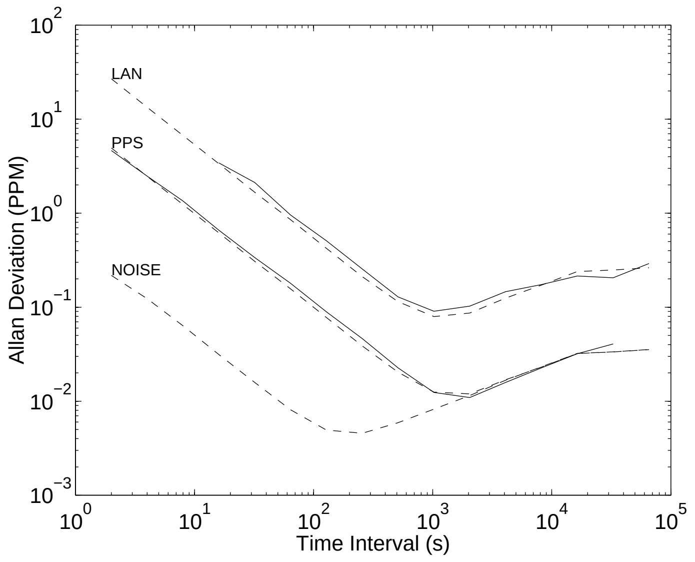

Figure 12. Comparison of PPS and LAN Data

onds in log-log coordinates. The results for the PPS and LAN data are shown in solid-line curves in Figure 12. 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 [8]. 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 point*, characterizes each particular clock oscillator and network path. Note that, as the phase noise decreases, the white-phase line is lowered in direct proportion and, as the frequency noise decreases, the random-walk frequency line is lowered in direct proportion.

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, jitter within the network, which is 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. As apparent from Figure 12, the frequency noise of the LAN experiment is about ten times that of the PPS experiment.

Shown superimposed on Figure 12 as dashed lines are Allan deviation characteristics determined from synthetic noise generated internally by the simulator program. Synthetic noise is generated as the sum of two Gaussian noise processes, a white phase component and a random-walk frequency component, each with specified standard deviation. The synthetic PPS noise use a phase parameter of $5.7x10^{-6}$ and frequency parameter $3.5x10^{-9}$ calculated at 64-s intervals. The LAN parameters are $3.1x10^{-5}$ and $2.6x10^{-8}$, respectively. For comparison, the NOISE curve on the figure was generated by the simulator using a phase parameter of zero and frequency parameter of $3.5x10^{-9}$. Ordinarily, it would be expected that the curve continue downwards toward the left; however, the curve inflects upwards with apparent phase parameter of $2.9x10^{-7}$. This is due to the simulated clock precision of 1 μs, which models the common Unix microsecond clock. The ulti-

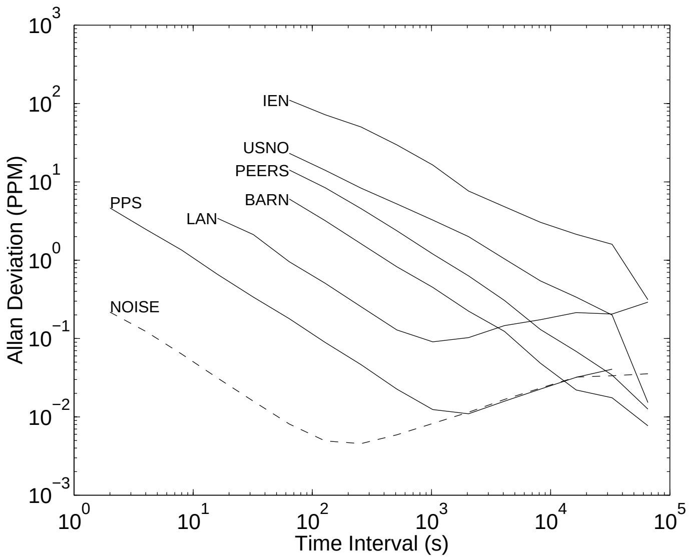

Figure 13. Allan Deviation - Combined Data

mate phase noise in any case is limited by the clock hardware, which is 1 μs for the SPARC IPC, 0.5 μs for the UltraSPARC, and a few nanoseconds for the Digital Alpha kernel[3].

The PPS and LAN data are considered representative of timekeeping accuracy expectations when using directly connected external sources, such as radio clocks or PPS sources. However, the characteristic curves shown in Figure 12 are not representative of accuracy expectations when using remote time sources at considerable distances as the Internet packet flies. Accordingly, experiments were done to evaluate the Allan deviation using the NTP algorithms and paths to 19 remote primary time servers located in North and South America and in Europe, Japan and Australia. In all experiments, the NTP clock discipline algorithm was logically disconnected, so that the simulated local clock free-runs at constant frequency. Since all peers used were primary time servers and their local clocks were disciplined to external sources, this allows direct measurement of one-way network delays and separation of phase and frequency noise.

The data were collected over a ten-day period in fall, 1996, using the *filegen* facility of the NTP daemon for Unix and NTP primary time server *pogo.udel.edu*, which is connected via a lightly loaded T1 tail circuit to the Internet service provider point of presence at College Park, MD. The clock offset and one-way delay data are derived from messages exchanged between pogo and each of its peers at 64-s intervals. Each exchange results in four timestamps, one at the times of transmission and reception for each outbound and return message. The four raw timestamps collected for each exchange, as well as a host identifier, were recorded in the raw data file, henceforth identified as the *rawstats* data.

The Allan deviation characteristics for all sources considered are shown in Figure 13. The NOISE, PPS and LAN results duplicate the data of Figure 12 for comparison. The BARN,

3. Prior to Digital Unix 4.0, the kernel clock resolution was limited to about 1 ms. Digital Unix 4.0 incorporates the author's kernel modifications [7], which provide a resolution of 1 μs. In principle, the kernel clock resolution is limited only by the CPU clock period.

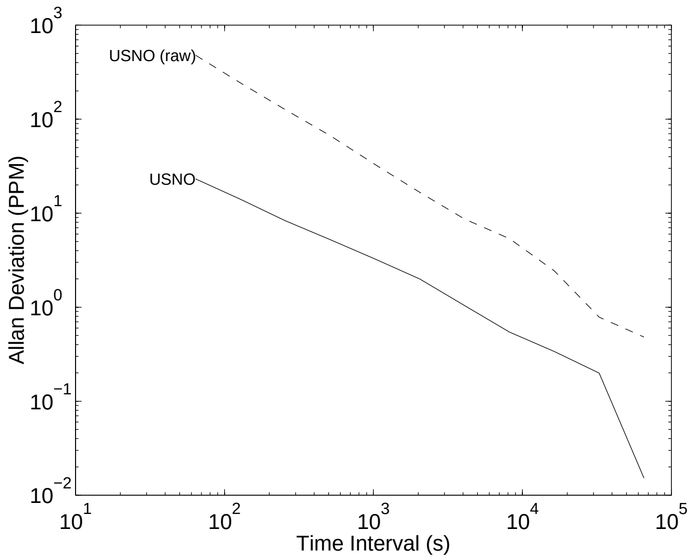

Figure 14. Comparision of Raw and Filtered USNO Data

PEERS, USNO and IEN results use the rawstats data collected from a selected subset of the 19 remote time servers. 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, in this case at the U.S. Naval Observatory, in Washington, DC. The IEN data represent a “distant” server, in this case 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.

Since all the peers in these experiment are synchronized to external sources, the phase noise of each remote source can be determined independent 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. Note that the PEERS curve, which represents the combination of all remote peers, including USNO and IEN, is actually lower (better quality) than any of the other remote peers separately. Of all the remote peers, USNO is the “best” in the sense of lowest phase noise. This confirms that the NTP intersection, clustering and combining algorithms do in fact deliver time more accurate than available from any single peer separately.

Figure 14 shows the phase noise of the USNO source under two conditions, one using the raw data from the rawstats file (dashed line) and the other using the data processed by the clock filter algorithm (solid line). As described in the protocol specification, this algorithm measures the clock offset and roundtrip delay for the last eight message volleys and selects the measurement with the lowest delay as representing the best offset sample. Fitting straight lines with slope -1 to the two curves and measuring the difference on the *y* axis shows a noise reduction of over ten times due to the clock filter algorithm.

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 13 were measured using the NTP algorithms and open-loop conditions, the question remains as to how faithfully the

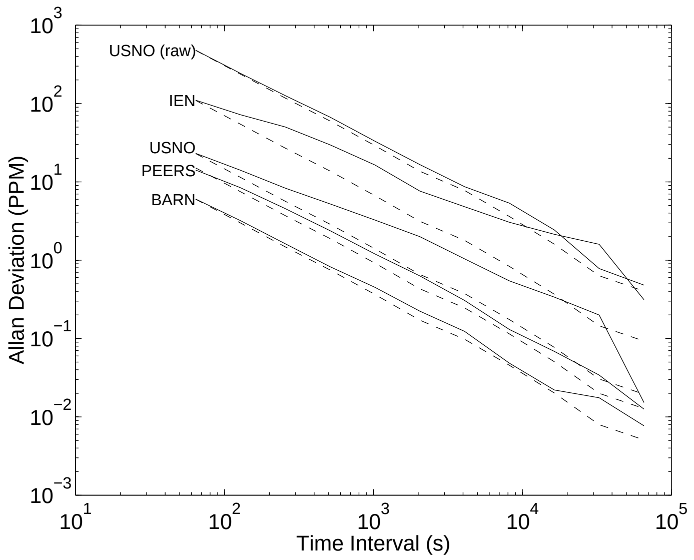

Figure 15. Comparison of Real and Synthetic Phase Noise

white-phase/random-walk frequency model applies to the remaining peers. The solid lines in Figure 15 show the measured phase noise 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 τ = 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 peers. Since the phase modelling is most important at small values of τ, the disagreement at large values is not considered significant.

The results demonstrated here suggest that a useful predictive model for ordinary computer clock oscillators and real networks can be specified by determining the Allan intercept point (τ, σ) and assuming white phase noise at τ less than the *x*-intercept and random-walk frequency noise above that point. Furthermore, a working approximation for the frequency noise for a “poor” clock oscillator is a straight line with slope +0.5 passing through the point (1000 s, 0.1 PPM), which corresponds to a frequency parameter of about 2.6×10^{-8} in the synthetic noise generator. A working approximation for a “good” clock oscillator is the line passing through the point (1000 s, .01 PPM), which corresponds to a frequency parameter of 3.5×10^{-9}.

In this model, the Allan variance is completely determined by the white phase noise characteristic of the network and operating system and the assumption of “good” or “poor” clock oscillators. This can be determined directly from plots similar to Figure 13 or determined by direct measurement. A convenient *x* intercept for graphical analysis is τ = 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 τ = 64 s is particularly useful, since this is the standard minimum poll interval used in the NTP protocol. Figure 16 shows the *y* intercept at this value and the equivalent phase 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.

Source	y Intercept	p Parameter
NOISE	$8.09 \text{x} 10^{-3}$	$2.93 \text{x} 10^{-7}$
PPS	0.179	$5.70 \text{x} 10^{-6}$
LAN	0.955	$3.10 \text{x} 10^{-5}$
BARN	6.04	$2.19 \text{x} 10^{-4}$
PEERS	15.2	$5.50 \text{x} 10^{-4}$
USNO	23.1	$8.38 \text{x} 10^{-4}$
IEN	110	$4.00 \text{x} 10^{-3}$
USNO raw	480	$1.74 \text{x} 10^{-2}$

Figure 16. Allan Deviation Phase Noise Parameters

5.2 Performance Using Synthetic Noise Sources

The primary emphasis in the Allan deviation experiments is on the characterization of the noise sources contributing to the time and frequency errors of the clock discipline algorithm. In particular, the experiments provide a basis for an accurate modelling of the noise sources relative to real network peers, in particular, the characterization of each configuration using only two parameters corresponding to the amplitude of the prevailing phase and frequency noise.

As described previously, the algorithm can operate in three modes, phase-lock loop (PLL), frequency-lock loop (FLL) and a hybrid combination of both modes. In hybrid mode, the weight factors used in combining the FLL and PLL predictions are determined from the prediction errors, which normally change in response to the prevailing phase and frequency noise and poll interval. The following experiments are designed to explore the response of the algorithm as these parameters are varied over normal and abnormal ranges. In each experiment, the NTP simulator program is presented with varying combinations of phase and frequency noise parameters and the standard error determined as a function of poll interval. In all experiments, the simulator is operated in closed-loop mode with all sanity checks, spike detectors and tolerance clamps in place.

The experiments were performed using each of the sources identified in the Allan deviation experiment and selected poll intervals from 64 s to 131,072 s. Each experiment run used 30 days of synthetic phase and frequency noise with parameters for each source determined as in the Allan deviation experiments. The first experiment uses the USNO data to determine the standard error as a function of synthetic phase and frequency noise and poll interval. This is done first using synthetic phase noise only, then using synthetic frequency noise only, in order to determine how the standard error scales relative first to the noise amplitude and then to the poll interval. Figure 17 shows the phase and frequency noise characteristics for both PLL mode (solid lines) and FLL mode (dashed lines) as a function of poll interval. When the poll interval is varied with phase noise only, the standard error is approximated by the two nearly horizontal lines, as shown in the figure. In this case, the PLL outperforms the FLL by a factor up to ten times. When the poll interval is varied with frequency noise only, the standard error is approximated by the two lines with slope near 1.4, as shown in the figure. In this case, the FLL outperforms the PLL by a factor of

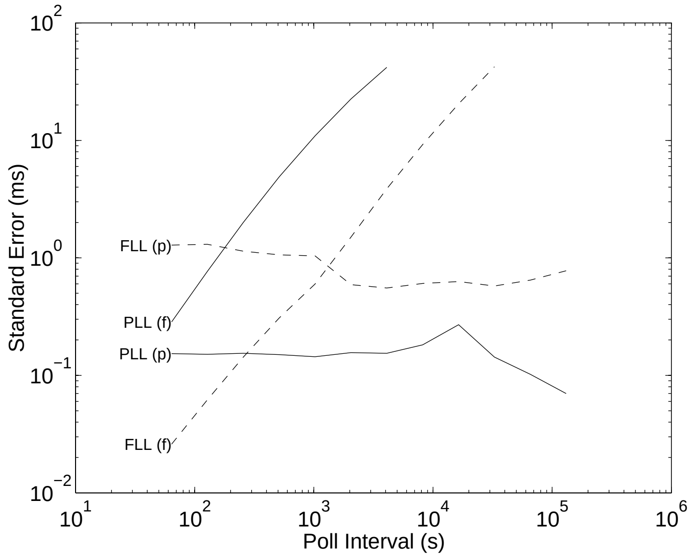

Figure 17. Phase and Frequency Noise Comparisons With PLL and FLL

about ten. In fact, the PLL becomes unstable at poll intervals above 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.

As evident from Figure 17, 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 peer 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 for the clock oscillators considered in the experiment, the selection of which clock discipline mode to use depends only on the phase noise characteristics of the particular network path involved.

It may seem that a simple measurement of phase noise would suffice to determine the optimum point to switch between FLL and PLL modes, as suggested in the Allan deviation experiments. As in [2], the phase noise could be measured prior to regular operation and the optimum intercept determined. However, as determined by experiment, this is not practical in the current Internet. Figure 18 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 US. The phase noise amplitude 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 intersection 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 predictions are determined in real time from the measured prediction errors.

Figure 19 shows the standard error in PLL mode with the various sources used in the Allan deviation experiments as a function of poll interval. In this case, the data for each source was simulated using the parameters developed in the Allan deviation experiments. Except for the extreme phase

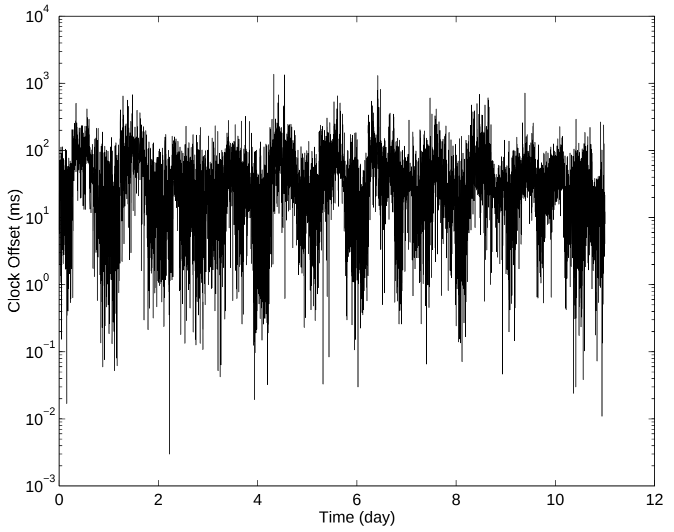

Figure 18. IEN Phase Noise

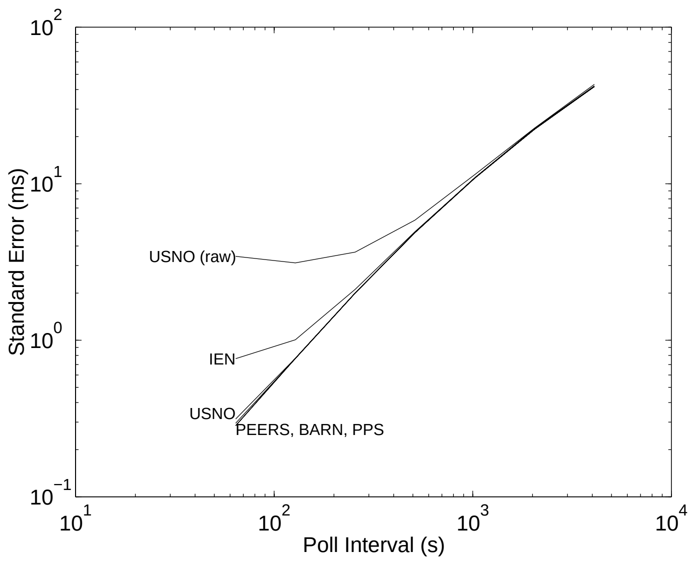

Figure 19. PLL Mode Standard Error by Source - Synthetic Noise

noise represented by the IEN and raw-USNO data, the results scale very near the same asymptote line. This is a reflection on the fact that the PLL has in effect a highly integrative response that deals with phase noise by massive averaging. Note that the curves corresponding to the USNO, IEN and raw-USNO sources do not extend above 4,096 s, for the reasons explained in conjunction with Figure 17.

Figure 20 shows a composite of the standard errors using hybrid mode and synthetic noise parameters for each source (solid lines). Overlaying these lines are the composite PLL from Figure 19. For most sources over most of the poll-interval range, the errors in hybrid mode are about ten times less than in PLL mode. As expected, the exceptions are the regimes where the phase noise overwhelms the frequency noise. Ideally, the hybrid mode should weight the PLL mode more than the FLL mode as the poll interval becomes smaller, in which case none of the hybrid curves for each source should exceed the corresponding PLL curve. According to the figure, as the phase

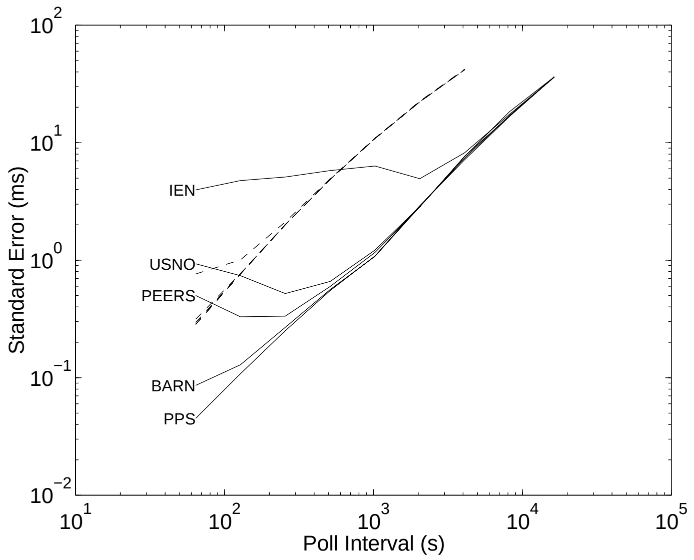

Figure 20. Hybrid Mode Standard Error by Source - Synthetic Noise

noise increases to exceptional levels, as in the IEN case, this is not always the case and the errors in hybrid mode can exceed those in PLL mode by up to four times.

Comparing Figure 20 with Figure 13, it is apparent the inflection point where the curves corresponding to each source depart the asymptote on Figure 20 are about a factor of ten below the Allan intercept point on Figure 13. 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 20 shows data collected under closed-loop conditions, while Figure 13 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 clock discipline algorithm performance.

5.3 Performance Using Real Data

The next series of experiments is designed to test the clock discipline algorithm performance using real data corresponding to each source in the Allan deviation experiments. In this case, the clock offsets are computed from the rawstats file mentioned previously, while the frequency offsets are determined at 64-s intervals using the data for the LAN curve shown on Figure 11. Figure 21 shows the standard error for the IEN, USNO, PEERS and BARN sources considered in the synthetic data experiments and using PLL mode. The asymptote, considered as the BARN curve, is nearly the same as with the synthetic data shown in Figure 19; however, as with synthetic data, the PLL mode with real data and some sources becomes unstable above 4,096 s, so only the data at smaller poll intervals are comparable. In addition, the PLL does not converge with real USNO raw data for any poll interval, so this does not appear on the figure. Remember that the criterion for convergence is that no step-change adjustments occur over the lifetime of the experiment. This may be unfair in some applications, as there may be only one or two step changes of 128 ms over the ten-day duration of the experiment, and this may be acceptable in some applications. Also, note from Figure 11 that there are relatively large frequency discontinuities in the LAN data, which could very likely create a step change, unless the poll interval is allowed to decrease and

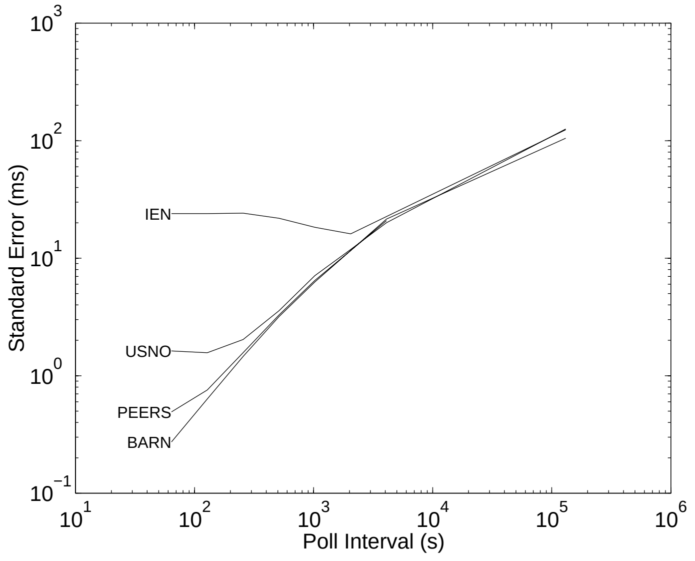

Figure 21. PLL Mode Standard Error by Source - Real Data

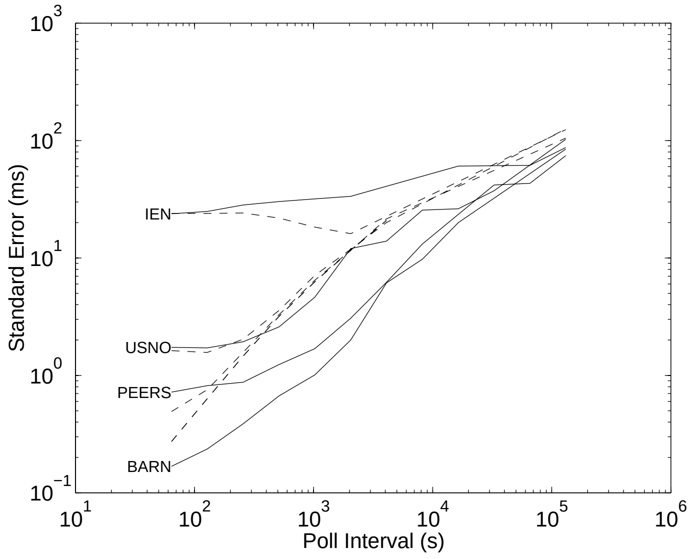

Figure 22. Hybrid Mode Standard Error by Source - Real Data

the PLL frequency to quickly adapt. As shown in other experiments, the behavior in PLL mode, where the poll interval adapts to prevailing conditions, is stable with all sources.

The major difference between the real and synthetic data in PLL mode is that the standard errors with real data become up to five times worse than with synthetic data in the regime for which PLL mode would naturally be favored. In Figure 21, this only occurs with USNO and IEN data and then only with poll intervals of 128 s and 64 s. Figure 22 shows the same sources using hybrid mode and should be compared with Figure 20, which shows the same sources using synthetic noise. As in the Figure 20, Figure 22 shows the data for PLL mode superimposed with dashed lines for comparison. As expected, the curves for real data are somewhat jagged, due to the spiky frequency characteristicshown in Figure 11, but in general follow the same characteristic as the synthetic data. As evident in the figures, the performance with real data is not quite as good as

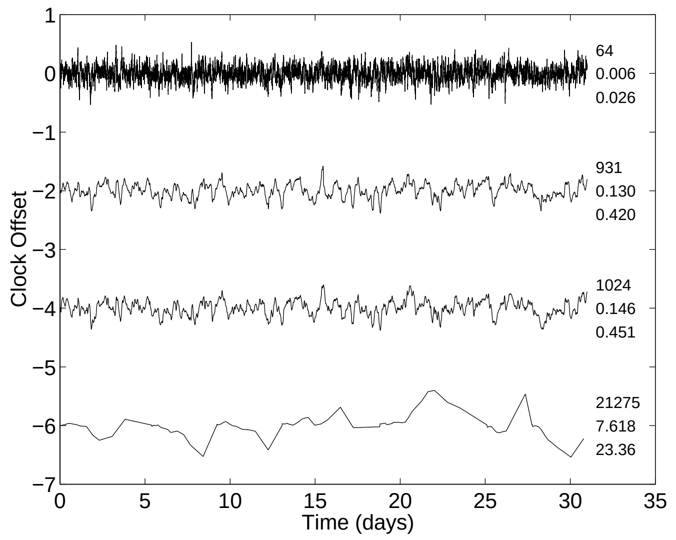

Figure 23. Clock Offsets for PPS Reference Source

with synthetic data; however, the performance in hybrid mode is almost always better than PLL mode, which represents the NTP Version 3 algorithms.

5.4 Performance with Real Data over Time

An important design goal in the development of the NTP Version 4 algorithms is to attain the smallest standard error and largest poll interval consistent with the machine architecture, network paths and synchronization sources. As demonstrated previously, this requires a compromise between poll interval and error margin. Figure 23 illustrates typical cases involving a PPS signal with suitable interface, as described in [7]. Each of the four curves is labelled with three numbers, reading from the top: mean poll interval, standard error in milliseconds and maximum absolute error in milliseconds. For the top curve, the poll interval is clamped at 64 s, representing a typical configuration with a cesium oscillator or GPS receiver connected via the DCD line of a serial port to a SPARC IPC workstation with assumed clock reading error of 40 μs. The standard error in this case is 6 μs, which is in the order of the interrupt latency of this machine. Modern workstations have clock reading errors in the order of one microsecond, in which case presumably the standard error could be reduced to the order of nanoseconds.

The second curve from the top in Figure 23 represents the case with a GPS receiver connected via a serial port, but with the poll interval allowed to vary according to the algorithm described earlier. In this case, the mean poll interval 931 s is close to the maximum 1,024 s, but the standard error has increased to 130 μs. For most purposes, this degree of accuracy would be acceptable in all but the most demanding applications. For comparison purposes, the third curve from the top represents the performance with the poll interval clamped at 1,024 s. As expected, the performance is about the same. The bottom curve in the figure shows the performance with the poll interval clamped to the range 1,024 s to 32,768 s, which is representative of telephone modem services. Here, the clock precision is assumed 1 ms, in order to model modern microprocessor-based modems, which can present considerable delay variations between calls. In this case, the maximum error is about 23 ms at a mean call interval of 21,275 s. In another experiment, the maximum poll interval was set at 131,072, or about 1.5 days. Over the 30-day simulation period, 43 calls were made, resulting in standard error about 24 ms and maximum error about 59 ms.

6. Conclusion

The results demonstrated in this report show a substantial improvement in the performance of the NTP Version 4 algorithms over the previous NTP Version 3 algorithms of roughly a factor of ten in most cases. Perhaps the most striking result is that the new clock discipline algorithm is effective with poll intervals well over one day, which is an attractive feature when telephone toll charges are involved. In addition, the new algorithm can automatically select the optimum combination of FLL and PLL data and the poll interval over a wide range of network jitter and oscillator wander while in regular operation and not requiring initial calibration.

While only minor changes are required in the NTP specification and reference implementation, a major difference between the NTP Version 3 and 4 architecture is the process decomposition, where the clock update process operates on a polling basis, rather than being called directly, as in the Version 3 algorithms. Besides simplifying the interactions with the peer processes and the clock adjust process, this allows the polling strategies of the peer processes to adapt to sources of varying types, such as network peers, radio clocks or telephone modems.

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 report has focussed on the nature of the simulation process, design and analysis of the synthetic noise models and validation of the algorithm performance in systematic simulation exercises. Many times during these exercises interesting things happened which resulted in significant improvements in the algorithm design. While this report has not dwelled on 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 allowed some simplifications which would not ordinarily be considered, like the elimination of the frequency estimator variable in the original FLL algorithm.

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 report require a good deal of machine time. A full suite of all simulations, including the simulator program itself and Matlab programs, requires an hour or two on a Sun UltraSPARC, which is by comparison one of the fastest workstations available today.

7. 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.

Appendix A. NTP Simulator

The programs and data in the ntpsim.tar.Z distribution operate as a simulator for the Network Time Protocol (NTP) protocol and algorithms. The file ntpsim.c is a portable C program designed to faithfully simulate the clock filtering, selection and discipline algorithms. It compiles and runs in Unix and Windows environments with generic C or C++ compilers. The program reads history files in two formats created using the “filegen” facility of the xntpd daemon for Unix and Windows/NT. It then simulates the behavior of the NTP algorithms and produces traces and summary statistics as directed.

The program operates in one of four modes, as specified by the -i command-line option (command-line options are not available when compiled for Windows). The input data file is specified by the -f option or the first argument of the command line. The default mode 0 uses the “rawstats” file produced by xntpd from the four timestamps determined at the transmit and receive times of the outbound and return messages at each NTP measurement volley. The timestamps are determined before processing by the engineered algorithms defined in the specification. Mode 1 uses the “loopstats” file produced by the daemon from the final corrections used to adjust the local clock. The formats of the rawstats and loopstats files are described in the comments in the program text.

Modes 2 and 3 can be used to generate random phase and frequency variations characteristic of real data. Mode 2 is used to produce the actual simulation, while mode 3 is used to generate data files which are later processed by a Matlab program in the allan.tar.Z distribution to verify the generators faithfully replicate the statistics of real data. The generators can also be used in modes 0 and 1 to introduce synthetic phase and frequency variations to the real data files. Alternately, a frequency history file can be specified with the -W option. The file, which can be produced by another program or experiment, contains frequency-offset samples at 64-s intervals. A maximum of about ten days of samples can be read, after which the simulator starts over from the beginning.

The generated phase and frequency variations use samples from a Gaussian distribution with standard deviations specified by the -p and -w command-line options, respectively. A random phase sample is used directly or added to the clock offset determined from the data file, depending on input mode. A random frequency sample is multiplied by the time since the last sample and added to a frequency adjustment. The adjustment is then used directly or added to the simulated frequency determined from the data file, depending on input mode.

In all modes, the initial phase and frequency of the local clock are specified with the -T and -F command-line options, repectively. The number of days to include in the simulation is limited to the duration of the data file or the number of days specified by the -D command-line option (default 30), whichever is smaller. Some experiments may require the clock-discipline loop be opened and the clock allowed to free-run. The -u option is provided for this purpose. The clock reading error can be specified by the -R option, which is useful for evaluating the FLL/PLL weighting and poll interval adjustment algorithms.

Example rawstats and loopstats files are included for testing and evaluation. The rawstats data were collected over a ten-day period involving NTP primary server pogo.udel.edu and two dozen NTP primary servers located in the US, Europe, Asia and South America. At the present time, they probably represent the most extreme cases of dispersive network delays and congestion on existing Internet paths. Since these data involve only primary servers, which are controlled by external means, frequency errors should be very small. These data are most useful in evaluating phase-lock loop (PLL) clock discipline algorithms.

The loopstats data were collected over about 15 days using a SPARC IPC with free-running local clock compared to a precision pulse-per-second (PPS) signal using the ppsclock line discipline. These data demonstrate clock oscillator variations due to temperature changes, etc., but have very low phase variations. These data are most useful in evaluating frequency-lock loop (FLL) clock discipline algorithms.

In the case of rawstats data, the selection of which peers to use in the simulation is determined by two command-line options. The -l option suppresses peers on the same IP network as the host generating the rawstats data. The -r option adds an IP address to a restriction list. As each rawstats sample is processed, the source address is compared to each entry in the list. If the source address is not in the list, the sample is discarded. If no -r options are present, all peers are used, except those excluded by the -l option. In any case, updates from a local discipline source, such as a pulse-per-second (PPS) signal, are suppressed.

The program produces output in three formats, as specified by the -o command-line option. The default format 0 includes variables useful for processing by various statistics and plotting packages, such as S and Matlab. Format 1 consists of a trace which gives details of the various simulated events, as well as the values of state variables at each local clock update. A sample of a typical trace is as follows:

```
7 22059.554    192.5.41.40 3  ff       88      1024    0.562   0.076   0.667   1.454   10
7 22995.567    192.5.41.40 7 source outlyer 0.014439 0.002824 0.000624
7 22995.567    18.145.0.30 3 new clock source 0.000624 0.000568 0.000624
7 22995.567    8.145.0.30 3   ff       936     1024    0.531   0.076   1.193   1.467   20
7 23868.680    frequency 0.005
7 24019.620    18.145.0.30 3 spike -0.000584 0.023787 0.001193
7 24403.747    frequency -0.032
7 25043.573    18.145.0.30 3  ff       2048    1024    0.652   0.050   5.838   1.466   30
```

The first number on each line is the day number, which begins at 0, followed by the seconds and fraction past midnight of that day. For all but frequency changes, the third field is the IP address of the currently selected peer, while the fourth is the number of peers surviving the selection and clustering algorithms. As per specification, these are combined in order to generate the actual clock update. For frequency changes, the value following the “frequency” string is in parts-per-million (PPM). The above trace shows a source change due to the current source being discarded by the clustering algorithm, followed by a switch to a new source. Later a spike was detected in an update, which was then discarded. As the example rawstats file contains a relatively large number of peers, most with large dispersive delays, the example data shows a rather large number of these events.

If the remainder of the line consists of an alpha string followed by other data, the line is one of many informative messages about events internal to the simulator. The best way to decode these is to grep the program source and read the comments in the text. The remaining fields represent the actual clock update. The first field following the number of survivors is the reachability register for the given peer in hex format followed by the actual interval since the previous update followed by the poll interval determined by the local discipline algorithm.

If the remainder of the line is not an alpha string, the next four numbers following the poll interval show the local clock offset (ms) and frequency (PPM), followed by the noise and phase error estimates, both in ms. The next shows the standard deviation of the first-order frequency differences, originally intended as an aid in the local clock algorithm. The last number is the poll-update counter, which is used to determines the poll interval.

In output format 1, when the data file is completely processed, the program produces summary statistics similar to:

```
ID     IP Src          IP Dst         Samples    Mean       StdDev     Max
Local Clock                           1277       0.258      1.444      4.850
0      192.43.244.18   128.4.1.20     1497       -5.338     8.105      31.780
1      129.132.2.21    128.4.1.20     1473       2.750      80.690     1110.294
2      192.36.143.150128.4.1.20       1560       0.056      4.625      44.778
3      131.107.1.10    128.4.1.20     1503       47.949     129.488    321.024
4      18.145.0.30     128.4.1.20     1557       0.050      3.735      23.666
5      128.252.19.1    128.4.1.20     1536       1.061      4.450      32.398
6      204.123.2.5     128.4.1.20     1535       -7.566     9.098      50.951
7      192.5.5.245     128.4.1.20     1531       0.362      8.194      64.464
8      128.115.14.97   128.4.1.20     1469       -38.083    24.929     81.723
9      192.67.12.101   128.4.1.20     1464       24.400     64.070     414.240
10     128.250.36.2    128.4.1.20     1505       -17.308    65.031     14874.870
11     133.100.9.2     128.4.1.20     1362       -2.671     5.744      36.539
12     192.5.41.40     128.4.1.20     1520       0.019      4.643      28.403
13     204.34.198.41   128.4.1.20     1400       -9.291     7.309      41.659
14     131.188.2.75    128.4.1.20     1488       17.104     16.494     72.309
15     129.20.128.2    128.4.1.20     1521       13.000     23.308     65.790
16     193.204.114.1   128.4.1.20     1515       19.761     29.983     86.775
17     132.163.135.130 128.4.1.20     1525       -10.856    8.153      34.198
18     146.83.8.200    128.4.1.20     1366       -18.543    34.829     120.607
```

The mean, standard deviation and maximum are all in milliseconds. The first line after the header line represents the actual local clock, with the mean relative to the actual time. In other words, if this were a real scenario and the local clock was controlled by the given peers using the same algorithms, the local clock would have a mean error of 0.258 us relative to the actual time. The remaining lines represent the individual peer data as collected and displayed with the ntpq program in real life. Obviously, some of these critters are doubtful as providers of precision time, but these are real data for the real Internet where congestion is a fact of life. The results invite the conclusion that the algorithms are doing an excellent job under very demanding conditions.

The minimum and maximum poll intervals are specified in powers of two seconds by the -M and -M command-line options, respectively. The -P option is used to specify the method used to select how the FLL and PLL frequency predictions affect the acutal frequency adjustment. A nonzero value specifies the poll interval at or above which the FLL is used, rather than the FLL. If the value is zero, the FLL and PLL prediction errors control the weighting of the FLL/PLL frequency predictions. When automatic poll-interval adjustment is in effect, the summary information includes a table showing for each value of poll interval the number of polls at that interval and the total time (seconds) spent at that value. A summary line shows the number of clock updates in phase-lock mode, the number of clock updates in frequency-lock mode, the number of spikes discarded and the number of step phase-change adjustments.

The third output format is produced only in mode 3. It designed for processing by Matlab programs which construct an Allan deviation plot used to verify the random generators operate as intended. The format of the file produced in this mode is documented in the program text.

Command-Line Format

There is one optional argument, which is the filename to use for input.

The options, which work only in Unix, are as follows:

-d	Select debug output format.
-D<days>	Set maximum number of days in simulation run, with default 30.
-f<filename>	Read input from specified file.
-F<frequency>	Set the initial frequency offset, with default 0.
-i<mode>	Select input mode_0 rawstats_1 loopstats_2 synthetic phase and frequency variations_3 data for Allan deviation
-l	Suppress peers on the same LAN as the host generating the rawstats data.
-m<interval>	Set minimum poll interval specified in log2 units from 4 (16 s) to 14 (16384 s), with default 6 (64 s).
-M<interval>	Set maximum poll interval specified in log2 units from 4 (16 s) to 14 (16384 s), with default 10 (1024 s).
-o<format>	Set output format_0 Matlab_1 Trace_2 Summary
-p<parameter>	Set the phase noise parameter, represented as the standard deviation of a Gaussian distribution, with default 0.
-P<interval>	Set PLL/FLL mode switch poll interval, specified in log2 units from 4 (16 s) to 14 (16384 s). The value 0, which is the default, specifies automatic weight.
-r<address>	Add an IP address to the restriction list.
-R<parameter>	Set clock reading error, with default 40e-6.
-T<time>	Set the initial time offset, with default 0.
-u	Open the clock discipline loop; that is, disconnect the simulated local clock from control by NTP and allow it to free-run. This is useful for testing.
-w<parameter>	Set the frequency noise parameter, represented as the standard deviation of a random-walk Gaussian distribution, with default 0.
-W<filename>	Read frequency data from specifed file.