1. Introduction

This report covers recent work in refining and extending the Network Time Protocol (NTP). The present version 3 of the protocol and its predecessors have been deployed in an estimated 100,000 time servers and clients in the Internet for over a decade. The definitive specification of NTP Version 3 is RFC-1305 [MIL90], while a subset of it, the Simple Network Time Protocol (SNTP) is described in RFC-1361 [MIL93].

NTP has been in a state of continuous evolution since its origins in the Hello Protocol of the late 1970s until Version 3 of the protocol appeared in 1992. Several papers, technical reports and RFCs have described the architecture [MIL91a], analysis [MIL91b], [MIL92a], [MIL94a], design [MIL94b], and implementation [MIL94c] of the various algorithms utilized in its operation, as well as measurements of its performance in the Internet [MIL89].

Most of the recent work in NTP has focussed on the continued development of the NTP daemon and utility programs for Unix. Recent improvements include the capability to operate in IP multicast mode with automatic calibration for transmission delays across the Internet. Others include the addition of several new radio clock drivers and a new driver interface, re-engineered mitigation rules for clock selection, and a re-engineered local clock discipline algorithm. These will be discussed in subsequent sections of this report.

A particularly useful tool used during the development and refinement of the NTP algorithms has been a simulator which implements most of the features required of a generic, multi-peer time server. A program listing of the simulator, written in the C language, is given in Appendix A. It can be used to test the operation of the various NTP algorithms, either with synthetic timing data or actual data collected by the statistics facilities of the NTP daemon implementation for Unix.

Mention is made often of the NTP Version 3 daemon for Unix systems. A distribution of this daemon and related components is available for anonymous FTP from louie.udel.edu in the pub/ntp directory as the compressed tar archive xntp3.4h.tar.Z. Note that in future the sub-version identifier 4h may change.

2. NTP Local Clock Discipline Algorithm

The NTP local clock discipline algorithm, or clock discipline, has evolved over several years as the result of analysis, simulation and experiment. However, since in the traditional NTP model all peers are interconnected by the Internet in real time, there was little incentive to explore the operation of NTP at update intervals much longer than 1000 s. Implementation of a reference clock driver for the NIST Automated Computer Time Service (ACTS) caused a re-examination of this model. ACTS requires telephone calls to Boulder, CO, on a regular basis, making calls at 1000-s intervals a potentially expensive business. The redesign also fixed a number of problems caused by very large system clock oscillator frequency errors and very large network delay variations.

2.1. Typical Oscillator Characteristics

The model described in the specification RFC-1305 is based on a phase-lock loop (PLL) design, which is optimum or near optimum for the update intervals used with NTP peers and radio clocks, ordinarily in the range 16-1024 s. However, the ACTS driver must operate with update intervals much longer than 1024 s, where the performance of the PLL model deteriorates. As suggested by Judah Levine of NIST and used in his “lockclock” algorithm, a frequency-lock loop (FLL) gives better performance at the longer update intervals up to a maximum depending on the acceptable

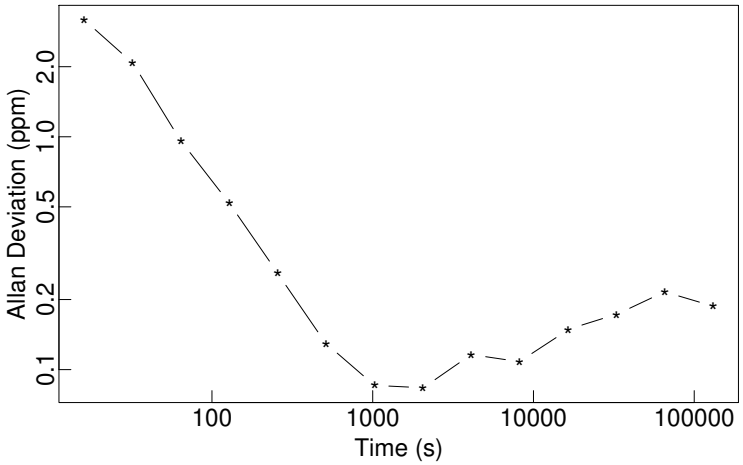


Figure 1. Allan Variance of Typical Local Oscillator

error bound. Prior experiments and simulations have shown that reliable timekeeping using this FLL is possible with call intervals up to tens of kiloseconds.

The new design uses a hybrid of the PLL and FLL models. The design parameters are based on an analysis of the stability characteristics of room-temperature quartz oscillators found in typical computing equipment. Following the accepted notation used in the literature of precise time and frequency control [ALL90], let $x(t)$ be the time offset and $y(t)$ be the frequency offset of the disciplined local clock oscillator relative to an external standard oscillator. Where no confusion exists, we will drop the word offset when dealing with time and frequency values. We will use the functional notation $x(t)$ when dealing with continuous functions of t and subscript notation x_k when dealing with discrete sample values.

Let the time average of a variable be indicated by a bar "$\bar{\ }$" over the symbol, the (possibly infinite) average of an expression be indicated by brackets "< >" around the expression, and the estimate produced by the clock discipline by a caret "$\hat{\ }$" over the symbol. Consider a sequence of N time offsets $\{x_k\}$ $(k = 0, 1, ..., N - 1)$ measured at epoches $\{t_k\}$ between the local clock and a reference clock of high quality. If $x(t)$ is continuous and differentiable over the interval $[t_{k-1}, t_k]$, the average frequency during the interval ending at the kth sample is

$$y_k \equiv \frac{1}{t_k - t_{k-1}} \int_{t_{k-1}}^{t_k} y(t)\, dt = \frac{x_k - x_{k-1}}{\tau_k}, \tag{1}$$

where $\tau_k = t_k - t_{k-1}$. For a sequence of N measurements spaced at nominal intervals $\tau = \tau_k$, the (two-sample) Allan variance is defined

$$\sigma_y^2(\tau) \equiv \frac{1}{2} <(y_k - y_{k-1})^2> = \frac{1}{2(N-2)\tau^2} \sum_{i=2}^{N-1} (x_i - 2x_{i-1} + x_{i-2})^2.$$

and the Allan deviation as the square root of the variance.

Figure 1 shows a plot of Allan deviation in log-log coordinates for NTP client *baldwin*, which is connected to the same Ethernet segment as NTP server *pogo*. The local clock on *pogo* is synchronized to a GPS receiver, while the local clock on *baldwin* was allowed to free-run. The plot

is calculated from a series of time offsets measured between the two machines over a period of about two weeks using NTP. Starting from the left at $\tau = 16$ s, the plot tends to a straight line with slope near -1, which is characteristic of white phase noise [STE85]. In this region, increasing τ increases the frequency stability in direct proportion. At about $\tau = 1000$ s the plot has an upward inflection, indicating that the white phase noise becomes dominated first by white frequency noise (slope -0.5) and then by flicker frequency noise (flat slope) and, finally by random-walk frequency noise (slope +0.5). In other words, as τ is increased, there is less and less correlation between one averaging interval and the next.

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

On the other hand, at the highest τ, the errors due to frequency stability dominate those due to phase noise. A FLL-based clock discipline provides the best performance in such cases. In order to provide the most rapid adaptation to frequency wander, while avoiding spurious disruptions due to phase noise, the best τ would seem from Figure 1 to be about 1000 s. Since the frequency estimate in the FLL is calculated as the difference between the time offsets at the beginning and end of the averaging interval divided by the interval length, the FLL can become seriously vulnerable to phase spikes at τ much below this.

In a series of experiments and simulations, it was verified that the PLL model provides the best performance at $\tau \leq 1000$ s, where the phase noise predominates, while the FLL model provides the best performance at intervals above that, where the frequency stability predominates. In the PLL described in [MIL94c], a frequency averaging interval of 1000 s corresponds to a time constant of about unity and a poll interval of 16 s, which would normally be considered at the lower end of the acceptable polling range. The point of crossover between the two models is from Figure 1 evidently in the vicinity of $\tau = 1000$ s, although this point can vary over a considerable range, depending on the delay variations in the network and operating system and the local clock oscillator stability.

2.2. Controlling Time and Frequency

The above model has been used as the basis of an adaptive-parameter, hybrid PLL/FLL loop design which gives good performance with update intervals from a few seconds to tens of kiloseconds, depending on accuracy requirements and acceptable costs. The model on which the design is based is described in this section. The algorithm itself, which is described in a later section, has been implemented in the NTP daemon for Unix, along with certain other refinements described below. The resulting performance using ACTS and call intervals in the 1000-16000-s range result in an accuracy of about 1 ms RMS with occasional surges of a few milliseconds. While the new design was intended primarily for ACTS, it can be used with any NTP peer or radio clock, such as in a public data network where per-packet charges are made.

The development of the model proceeds as follows. As before, let x_k be the time and y_k be the frequency at the kth update. Let $\bar{y}_k$ be the mean frequency of the local clock oscillator determined

from past time offsets $\{x_i\}$ and intervals $\{\tau_i\}$. In the most general formulation, an algorithm that corrects for the intrinsic local clock oscillator time and frequency errors will compute a prediction

$$\hat{x}_k = x_{k-1} + \bar{y}_{k-1}\tau_k \ . \tag{2}$$

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

Between updates the clock discipline periodically adjusts the phase in small increments in order to prevent discontinuities in the timescale and to conform to monotonic requirements. Thus, at each adjustment interval the value

$$ax + \bar{y}_k t_A \tag{3}$$

is added to the local clock phase, where a is a constant between zero and one, but yet to be determined, x is a variable defined below, and t_A is the interval between adjustments. The constant a is used as a gain factor in the following way. Let the value x be the residual in the adjustment whose initial value is x_k. At each adjustment interval the clock is adjusted by ax and the residual by $-ax$. This provides a rapid adjustment when x is relatively large, together with a fine adjustment (low jitter) when x is relatively small.

The following two models work in a similar way, with the only substantial difference between them the way the frequency $\bar{y}_k$ is determined. Consider first the FLL, in which the mean frequency $\bar{y}(t)$ is adjusted in order to minimize the time error $x(t)$. While a number of methods could be used to compute $\bar{y}_k$, a convenient one is the weighted average

$$\bar{y}_k = \bar{y}_{k-1} + w(y_k - \bar{y}_{k-1}) \ , \tag{4}$$

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

$$\bar{y}_k = \bar{y}_{k-1} + w\frac{x_k}{\tau_k} \ . \tag{5}$$

In contrast to the FLL model, in the PLL model the frequency is determined as past accumulations of phase. In this case,

$$\bar{y}_k = b\sum_{i=1}^{k} x_i\tau_i \ . \tag{6}$$

The PLL operates the same as the FLL, except that the frequency is determined as an integral, rather than an average. In order to understand how it works, it is useful to consider the limit as the time intervals approach zero. In a type-II PLL, the oscillator frequency $y(t)$ is determined by the measured phase $x(t)$:

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

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

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

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

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

where $\omega_z = \frac{b}{a}$ and $\omega_c^2 = b$. From elementary theory, this is the transfer function of a type-II PLL which can control both phase and frequency. By convention, the order of the PLL is the order of the loop filter, not that of the overall transfer function; therefore, the above PLL is classed as first order. The behavior of this type of loop is treated in great detail [MIL92a] and will not be repeated here.

In the new NTP local clock discipline, the above FLL and PLL models are incorporated in a hybrid phase/frequency-lock loop. The discipline minimizes the prediction error as in (2) according to the offset x_k measured at the kth update. As a new update arrives, the variable x mentioned above is initialized $x = x_k$ and the frequency $\overline{y}_k$ computed by either the FLL model or PLL model, depending on the interval τ_k since the last update. Between updates, which can range from seconds to hours, the phase is adjusted as in (3). In the NTP daemon for Unix, these adjustments are implemented by the adjtime() system call; while, in the modified kernel described in [MIL94b], correspondingly scaled adjustments are performed at each timer interrupt.

In the new design it is a requirement that the discipline operate seamlessly with varying update intervals including some instances using the PLL model for updating the frequency, and others using the FLL model. Operating experience has demonstrated the hybrid loop does in fact operate seamlessly over a range from 16 s to 16384 s. Normally, the PLL model is used for update intervals greater than 1024 s and the PLL model used otherwise. However, the PLL model shown here does not include the effect of the time constant, which is varied directly with the poll interval, normally equal to the update interval. The implications of this are discussed in [MIL94c].

It may seem strange that the coefficient a in (3) is used in both the FLL and PLL modes. The primary reason for doing this is to avoid discontinuities when the phase x_k is very large; e.g., over 100 ms. A secondary reason is to reduce the effects of measurement errors on the local clock phase, since in the NTP model the local clock of one stratum can be used to discipline clocks at higher strata. However, the affect on the FLL dynamics is minor, since when the FLL mode is in use, the intervals between updates (usually over 20 minutes) is much longer than the time constant provided by a (about one minute).

2.3. Avoiding Old Data

Another modification to the NTP clock discipline is to avoid stale timekeeping data, which can be a significant source of error. From a study of the stability characteristics of typical computer clock oscillators using both experiment and simulation, it was determined that aggregated data used in the clock filter algorithm to discipline the PLL are not generally useful if older than about 1000 s. This corresponds roughly to the knee in the Allan deviation characteristic measured for a typical workstation oscillator. The NTP clock filter algorithm was modified to limit the number of stages sorted in the clock filter shift register, so that only samples less than about 1000 s old are used to determine the offset and delay. For the most robust error estimation, all samples are used to determine the dispersion.

In addition, the clock filter samples are not sorted at all when the roundtrip delay is very small, as in the case of most reference clock drivers. In these cases only the first sample in the filter is used. In cases where the delay is very small, the synchronization distance, defined in the NTP specification as the sum of the dispersion plus half the delay, becomes an unreliable metric to determine the best sample. In addition, where the peer is a reference clock driver, the standard interface already includes a median filter, so that the minimum filter algorithm used by the clock filter is useful only to provide a long term error estimate.

2.4. Increasing the Frequency Tolerance

A problem which has recurred on every occasion a leap second has been inserted in the UTC timescale is that not all radio clocks detect and implement the leap event. As a result, some radios sail along without noticing the event, become confused for periods up to 15 minutes, then re-acquire correct synchronization. However, due to prior warning of the event, the NTP clock discipline executes the leap as indicated at the proper time. Thus, for periods up to 15 minutes there can be up to a 1-s discrepancy between the radio time and NTP time.

In order to cope with this problem, as well as other occasions where atypically large offsets occur, the discipline switches to unlocked state upon receiving an offset exceeding 128 ms. The discipline switches to the locked state and resets the time upon arrival of the first update offset less than 128 ms or the first update received after a 15-minute timeout. The discipline has also been modified so that the first update received when unsynchronized will cause the clock to be immediately reset, as well as reset the poll interval to its lowest value. Both of these modifications improve the loop response when first started and in cases of very large network delay variations.

It has been the experience of some users that, when first installed in a system, the NTP clock discipline fails to reliably lock to other peers and servers as configured. The indications are that the loop locks for some period of time, but is unable to stabilize the frequency correction. As a result, the time offsets eventually climb above 128 ms and the discipline unlocks. After the 15-minute timeout, the PLL locks again and the cycle repeats. Formerly, the only remedy in cases where this happened was a somewhat painful manual process where the nominal oscillator frequency offset is measured by some other means, such as eyeball-and-wristwatch, and a specific drift file (ntp.drift) crafted with this value. There are two contributing factors exacerbating this problem: (a) insufficient PLL frequency tolerance and (b) delay in the clock filter between the time the system clock is set and the time the first update is processed.

In the design of the NTP clock discipline, an upper bound on frequency tolerance must be set in order that correctness assertions about the errors inherent in the time measurement process remain

valid. In the original design, 100 ppm was selected as a compromise between reasonable correctness assertions and the tolerances found in typical computer clock oscillators. As additional encouragement to adopt this limit, DEC engineers have established this figure as the absolute worst case scenario in order that correctness assertions not be violated in the Digital Time Synchronization Service (DTSS) [DEC89] specified for DCE systems. In any case, where the inherent oscillator error is greater than 100 ppm, it is usually possible to adjust the frequency in 100-ppm steps by adjusting the value of the Unix kernel variable tick using the tickadj utility program supplied with the NTP distribution.

Unfortunately, many machines other than DEC routinely display intrinsic frequency errors far greater than 100 ppm, up to 250 ppm in some cases. Accordingly, the discipline was redesigned to operate with much larger tolerances exceeding 300 ppm at an update interval of 64 s, depending on the values of tick and tickadj selected by the tickadj utility. However, even when the PLL is first brought up in a system with errors that large, the behavior described above may still occur, as described in the next section.

2.5. Improving Stability at Initial Startup

The second factor affecting the PLL response is that, while the tolerance of the PLL can be made greater than 300 ppm, the delay between the time the system clock is first set until a following update is processed can be several minutes. The delay results from the requirement that the clock filter algorithm must accumulate a sufficient number of samples to allow reliable estimates of offset, delay and dispersion. At 300 ppm oscillator frequency error, for example, the system clock drift is about 20 ms per 64-s update interval. The delay, generally four update intervals, allows the system clock to drift up to 80 ms before time and frequency corrections can be effective. With the present loop constants (see below) and an update interval of 64 s, the maximum phase and frequency correction per update are only about 1 ms and 0.5 ppm, respectively. Under these conditions it does not take long before the offset shoots above 128 ms, the maximum allowed before the PLL unlocks, and the cycle repeats.

One approach to solve this problem is to measure the frequency during periods when the PLL is not disciplined to another server or radio clock, such as during the scenario described previously. When lock is reestablished, the working frequency used by NTP can be initialized with the measured value and the local clock stepped to the correct time. However, an experiment using this scheme proved it vulnerable to large network delay variations. Under these conditions the PLL frequency can surge drastically, leading in some cases to chaotic PLL behavior without permanent lock ever being achieved.

The scheme eventually adopted works as follows. Initially, the dispersion measured by the clock filter algorithm works down from 16 s to nominals as determined by the clock precision, interval between updates, differences between successive updates, etc. When the dispersion drops below 1 s, which occurs after about four updates, the local clock phase. but not the frequency, is disciplined to the selected source. When the dispersion falls below 128 ms, the local clock phase and frequency are disciplined in the normal manner. This scheme provides a phase correction up to 128 ms per update before the PLL is fully engaged and allows the PLL to start with only 20 ms offset, instead of 80 ms as in the above scenario. This is sufficient to allow the loop to converge to nominal time and frequency without straying outside the 128-ms limit.

2.6. Specification of the Local Clock Discipline Algorithm

The detailed operation of the new hybrid PLL/FLL loop is defined by the following algorithm, which is executed as each new update is received from the system source; that is, the peer or radio clock currently selected for synchronization. Following the accepted notation previously established, let θ, δ and ε represent the measured time offset, roundtrip delay and dispersion (error), respectively, produced by the clock filter and clock selection algorithms defined in the NTP specification. Following the NTP specification, the synchronization distance is defined $\lambda = \frac{\delta}{2} + \varepsilon$. The poll interval ν is described in the next section.

Let $x_k = \theta$ be the time and $\bar{y}_k$ be the mean frequency at the end of the kth interval of length τ_k. The algorithm computes a phase adjustment x and a frequency adjustment y to discipline the local clock. The discipline is implemented by means of an adjustment process that runs at specified intervals, usually 1 s. Updates are produced and this algorithm is executed only if there is a synchronization source with $\lambda < 1$ s. The system clock is initially unsynchronized and starts in the unlocked state. (To clarify the following steps, consider them to be the clauses in a C-language *if* statement.)

1. If $|x_k| > 1000$ s, something must be very wrong. The best behavior is probably to abandon the protocol and request external intervention. A message is written to the system log advising the operator to set the system time manually and the daemon exits.

2. If the above is not the case and $|x_k| > 128$ ms, either an extreme outlyer has occurred or a leap second has occurred and the radio clock or peer has not yet recognized it. If the system clock is synchronized and $\tau_k < 900$ s, do nothing; otherwise, adjust the phase $x = x_k$ and frequency $y = 0$. In this case only, the local clock is set using the Unix settimeofday() system call, which either steps the clock or slews it rapidly to the correct offset before the next update arrives. This is the only occasion where settimeofday() is used; all other adjustments are made using the Unix adjtime() system call, which slews the clock gradually to the correct time before the next update arrives.

3. If the above is not the case and the kernel has been modified to implement the NTP PLL and the kernel PLL has been enabled, the ntp_adjtime() system call is used to pass the offset to the kernel, which processes it directly. In this case the following steps of this algorithm are implemented in the kernel.

4. If the above is not the case and $\varepsilon > 128$ ms, either the loop has not completely converged or the clock filter samples are extremely noisy. Under these conditions, the best action is to set $x = x_k$ and $y = 0$.

5. If the above is not the case and ε has increased by a substantial factor since the last update, consider x_k a spike and ignore it completely. If x_k on the next update is near the same value, the ε produced by the clock filter will not increase further and the latest x_k will be processed as any other update. Using a factor of 8, a small number of apparently "good" samples at the shorter update intervals are discarded; however, "bad" spikes of 10 ms or more are easily detected and suppressed in ACTS updates.

6. If the above is not the case, most errors due to gross noise spikes have been suppressed and x_k can safely be used to calculate the new phase x and frequency y adjustments. Let t_A be the point

on the Allan deviation plot where the phase and frequency noise contributions are about equal, 1000 s on Figure 1. Since directly connected radio clocks generally have considerably less phase noise than NTP peers, assume $t_A = 800s$ for radio clocks and $t_A = 1600s$ for ordinary NTP peers, although the exact value is not critical. If $\tau_k \le t_A$, the PLL model is used to adjust the time and frequency. In this case, the PLL time constant $\tau = \frac{\nu}{4}$, in order to maintain good transient response. Note the use of τ for PLL time constant and τ_k for update interval; the uses will be clear from context.

The new time and frequency offset adjustments are, respectively,

$$x = \frac{x_k}{\tau} \quad \text{and} \quad y = \frac{x_k \tau_k}{K_f \tau^2},$$

where the $K_g = 2^6$ and $K_f = 2^{16}$ are chosen for optimum transient response for $2^0 \le \tau \le 2^6$, corresponding to $2^4 \le \nu \le 2^{10}$ s. With these parameters and with typical $\tau = 2^2$, the discipline reaches 63 percent response to a phase step in about 900 s and reaches 63 percent response to a frequency step in about 3600 s.

Justification of this model as simulating the behavior of a first-order, type-II, adaptive parameter PLL is given in previously published reports [MIL92a].

7. If $\tau_k > t_A$, the FLL model is used to adjust the time and frequency. The new time and frequency offset adjustments are, respectively,

$$x = x_k \quad \text{and} \quad y = \frac{x_k}{K_h \tau_k},$$

where $K_h = 2^2$ is an experimentally determined constant.

The behavior of this model is analyzed and performance assessed in a paper by Judah Levine accepted for publication in *IEEE Transactions on Networks*. The model used here is slightly different from his model in that the averaging constant $K_h = 2^2$ is fixed, rather than adjusted as a function of μ. It was found through simulation that this gives more agile response in cases when the oscillator frequency takes a jump.

The values x and y are then used to discipline the local clock phase and frequency. First, the frequency is updated

$$\bar{y}_k = \bar{y}_{k-1} + y,$$

which ends the update procedure. Once each second the adjustment procedure is executed. First, the local clock is adjusted by adding a quantity $\frac{x}{K_g}$ to account for the phase offset, then the same quantity is subtracted from x. Finally, the local clock is adjusted by adding a quantity $\bar{y}_k$ to account for its intrinsic frequency offset.

2.7. Determining the Poll Interval

The above algorithm determines the local clock time and frequency adjustments as a function of offset θ, delay δ and dispersion ε It remains to determine the poll interval ν, which indirectly affects the update interval, here called μ for convenience. In the case of a radio clock, every poll results in an update, so $\mu = \nu$ by construction. In the case of a NTP peer, there are two poll intervals involved, one for the local peer and the other for the remote peer, as provided in the NTP update message. Both the local and remote peer select ν independently, but the actual poll interval used by both the local and remote peers is the minimum of the two intervals, as required by the NTP specification.

The intent of the scheme is to adjust ν to drive μ to the largest value consistent with the required accuracy and acceptable cost. Normally, this results in an averaging time at or above the knee of the Allan deviation characteristic t_A as described above. For the NTP PLL model described above, this results in $\mu \approx 16$ s. However, for most applications, it is possible to increase μ in order to reduce network overhead, while accepting only a moderate loss in accuracy and stability.

On the assumption that the time error contribution due to phase noise is independent of μ, while the contribution due to frequency instability becomes prominent only for averaging times t_A, a useful approach is to increase ν until θ, representing the prediction error, exceeds some constant times ε, representing the measurement error. For this purpose, ε includes the contributions determined by the clock filter algorithm plus the contributions determined by the clock select algorithm, as defined by the NTP specification.

In practice, the implementation is complicated by the very noisy data involved. In a design evolved from a number of prior experiments, if $|\theta| > \varepsilon$, a counter is decreased by $2\log_2(\nu)$ counts; otherwise, the counter is increased by $\log_2(\nu)$ counts. If the counter increases above 30, $\nu \leftarrow 2\nu$ and the counter is reset to zero. If the counter decreases below -30, $\nu \leftarrow \frac{\nu}{2}$ and the counter is reset. The intent in this scheme is to only slowly increase ν in good times, but quickly reduce it in bad times, and also to provide a good deal of hysteresis. The dependency on $\log_2(\nu)$ serves to quicken the response for relatively large ν. Note that, in the NTP specification and implementation, some variables, including μ, ν and τ, are maintained in $\log_2$ units, so that multiply/divide operations can be done with simple shifts.

The above parameters were determined by experiment so that, under typical conditions with lightly loaded networks, ν increases to the maximum; but, under conditions of extreme phase noise or oscillator instability, ν temporarily decreases, so that the oscillator can more quickly adjust to nominal frequency variations.

To take advantage of the new design for other than the ACTS driver, where it is automatic, note that the default minimum poll interval is 64 s and default maximum poll interval 1024 s (for the ACTS driver the default minimum is 1024 s and default maximum 16384 s. However, using the *minpoll* and/or *maxpoll* parameters of the *server* or *peer* commands in the configuration file, it is possible to set *minpoll* as low as 16 s and *maxpoll* as high as 16384. Poll intervals less than 64 s are useful if an exceptionally quick lock is required, like in real-time or portable systems. With clock drivers other than ACTS, poll intervals above 1024 s may be useful to reduce traffic in some situations, such as when charges are made on a per-packet basis.

3. Mitigation Rules

Experience has shown that radio clocks often can't be trusted, since they occasionally lose contact with the transmitters in the respective services or develop a hardware fault. In order to provide robust backup sources, primary (stratum-1) servers are usually operated in a diversity configuration, in which the server operates with a number of remote peers in addition to one or more radio clocks operating as local peers. In these configurations, the suite of algorithms used in NTP to refine the data from each peer separately and to select and combine the offsets from a number of peers is used with the entire ensemble of remote peers and local radios. However, because of small but significant systematic time offsets between the peers, it is in general not possible to achieve the lowest jitter and highest stability in these configurations. In addition, there are a number of special configurations involving auxiliary radio clock signals, telephone backup services and other special cases. In such cases it is necessary to define a set of mitigation rules that provide an engineered choice among the several opportunistic synchronization sources.

The mitigation rules are based on the concept of *prefer peer* together with the special characteristics of the various reference clock drivers configured on the server. Note that it is possible to designate any peer, radio or otherwise, as preferred by including the *prefer* keyword with the *server* or *peer* command in the configuration file. All other things being equal, this peer will be selected to synchronize the system clock exclusive of other eligible candidates surviving the clock selection process. The precise characterization of the prefer peer, when used with various clock drivers, is described later.

It is convenient to establish a few naming conventions:

1. When the PPS driver (type 22) is configured in the server, a pulse-per-second (PPS) signal is connected and operating properly, and an associated prefer peer is within a nominal time offset relative to the server, the PPS driver is designated the *PPS peer*.
2. When the local clock driver (type 1) is configured in the server, it is designated the *local peer*.
3. When the Automated Computer Time Service (ACTS) driver (type 18) is configured in the server, it is designated the *ACTS peer*.
4. When the PPS kernel support is available, as described elsewhere, it is designated the *kernel peer*. Note that PPS kernel support operates independently of the kernel phase-lock loop (PLL) support, which is an alternative to the PLL provided by the NTP daemon.

In detail, the mitigation rules operate in two stages. The first stage is designed to determine whether the local peer and/or ACTS peer is eligible to become the *system peer*. Ordinarily, the local peer is eligible only when all other peers have failed, in order to provide a consistent reference time among the peers of an isolated subnet. However, it may happen that the system clock is controlled by some source other than NTP, in which case the local clock driver is configured with the *prefer* keyword. Due to the large difference in update intervals, using both ACTS and ordinary NTP peers at the same time is somewhat unstable. Ordinarily, ACTS is eligible only when all other peers have failed; however, if it is necessary to use ACTS as the primary source and use others only if ACTS fails, then the ACTS driver is configured with the *prefer* keyword. How the prefer designation affects clock selection is described below.

The second stage of operation acts upon the survivors of the sanity checks and intersection algorithm. The survivors at this point represent the subset of all peers which can provide the most accurate,

stable time. The rules establish among the survivors the system peer, which determines the stratum, reference identifier and several other system variables which are visible to clients of the server. In addition, the rules establish which source or combination of sources controls the system clock. In the general case with no designated prefer peer, PPS peer or local peer, the rules require all survivors be averaged according to a weight depending on the reciprocal of the dispersion, as provided in the NTP specification. The rules are as follows:

1. If there is a prefer peer and it is the local peer or the ACTS peer, choose it as the system peer and assign its θ, δ and ε to the system clock.

2. If the above is not the case and there is a PPS peer, then choose it as the system peer and assign its θ, δ and ε to the system clock. Note that there can be no PPS peer, unless there is a prefer peer other than the local peer or ACTS peer.

3. If the above is not the case and there is a prefer peer (neither the local peer nor ACTS peer in this case), then choose it as the system peer and assign its θ, δ and ε to the system clock.

4. If the above is not the case and the peer previously chosen as the system peer is in the surviving population, then choose it as the system peer and average its θ along with the other survivors to the system clock. Also, assign its δ and ε to the system clock. This behavior is designed to avoid excess jitter due to "clockhopping," when switching the system peer would not materially improve the time accuracy.

5. If the above is not the case, then choose the first candidate in the list of survivors ranked in order of synchronization distance and average its θ along with the other survivors to the system clock. Also, assign its δ and ε to the system clock. This is the default case and the only case considered in the current NTP specification.

The specific interpretation of the prefer peer and PPS peer require some explanation, which is given in following sections.

3.1. The Prefer Peer

The concept of prefer peer provides an intelligent mitigation metric which results in the best quality time without compromising the normal operation of the NTP algorithms. This scheme in its present form is not an integral component of the NTP specification. but is likely to be included in future versions of the specification. A prefer peer is designated by including the *prefer* keyword with the *server* or *peer* command in the configuration file. This keyword can be used with any peer or server, but is most commonly used with a radio clock server.

The prefer scheme operates on the set of peers that have survived the sanity and intersection algorithms of the clock select procedures. Ordinarily, the members of this set can be considered truechimers and any one of them can in principle provide reliable time. The job of the clustering algorithm, which is invoked at this point, is to select the best subset of the survivors providing the least variance in the combined ensemble time offset, compared to the variance in each member separately. However, due to various systematic errors (generally not more than a few milliseconds), the particular survivor chosen to discipline the system clock can change from time to time as the result of nominal timing errors and network delay variations, commonly called "clockhopping." The detailed operation of the clustering algorithm, which is described in the specification, is not

important here, other than to point out it operates in rounds, where a survivor, presumably the worst of the lot, is discarded in each round until one of several termination conditions is met.

In the prefer scheme, the clustering algorithm is modified so that the prefer peer is never discarded; on the contrary, its potential removal becomes a termination condition. If the original algorithm were about to toss out the prefer peer, the algorithm terminates right there. The prefer peer can still be discarded by the sanity and intersection algorithms, of course, but it will always survive the clustering algorithm. A prefer peer retains that designation as long as it survives the sanity and intersection algorithms. If for some reason the prefer peer fails to survive these algorithms, either because it was declared a falseticker or became unreachable, it loses that designation and the clock selection re-mitigates as described above.

Along with this behavior, the clock select procedures are modified so that the combining algorithm is not used when a prefer or PPS peer is present. Instead, the offset of the prefer or PPS peer is used exclusively as the synchronization source. In the usual case involving a radio clock and a flock of remote stratum-1 peers, and with the radio clock designated a prefer peer, the result is that the high quality radio time disciplines the system clock as long as the radio itself remains operational and with valid time, as determined from the remote peers, sanity algorithm and intersection algorithm. However, if the radio fails or becomes very noisy, the prefer peer drops out of the survivor population and the algorithm re-mitigates as described above.

While the model does not forbid it, it does not seem useful to designate more than one peer as the prefer peer, since the additional complexities to mitigate among them do not seem justified from on-the-air experience.

3.2. The Pulse-per-Second (PPS) Signal

Most radio clocks are connected using a serial port operating at speeds of 9600 bps or lower. The accuracy using serial ASCII timecode formats, where the resolution is limited to 1 ms and the on-time epoch is established by a designated character, is limited to a millisecond at best and a few milliseconds in most cases. However, some radios produce a precision pulse-per-second (PPS) signal which can be used to improve the accuracy in typical workstation servers to the order of a few tens of microseconds. The details of the hardware and software interface are given in the documentation in the NTP distribution. The following discusses how this signal is processed and mitigated by the NTP clock discipline.

First, it should be pointed out that the PPS signal is inherently ambiguous, in that it provides a precise seconds epoch, but does not provide a way to number the seconds. In principle and most commonly, another source of synchronization, either the timecode from an associated radio clock, or even a set of remote NTP peers, is available to perform that function. In all cases a specific, configured peer or server must be designated as associated with the PPS signal. This is done as above by using the *prefer* keyword with the *server* or *peer* command in the configuration file. The PPS signal can be associated in this way with any peer or server, but is most commonly used with the radio clock generating the PPS signal.

The PPS signal is processed by a special PPS driver (type 22). This driver automatically determines the hardware configuration and selects the appropriate software interface. This scheme replaces the former scheme based on conditional compilation and the PPS, CLK and PPSCLK compile-time switches. Regardless of method, the driver, like all other drivers, is mitigated in the manner described above. However, in the case of the PPS peer, the behavior is slightly more complex.

First, in order for the PPS peer to be considered at all, its associated prefer peer must have survived the sanity and intersection algorithms and have been designated the prefer peer. This insures that the radio clock hardware is operating correctly and that, presumably, the PPS signal is operating correctly as well. Second, the absolute time offset from that peer must be less than 128 ms, well within the 0.5-s unambiguous range of the PPS signal itself. Finally, the PPS time offsets measured by the PPS peer are propagated via the clock filter to the clock selection procedures just like any other peer. Should the PPS time offsets pass the sanity and intersection algorithms, they will show up along with the offsets of the prefer peer itself. Under these conditions, the mitigation rules defined above will choose the PPS peer to control the system clock. Note that, unlike the prefer peer, the PPS peer samples are not protected from discard by the clustering algorithm. These complicated procedures insure that the PPS offsets developed in this way are the most accurate, reliable available for synchronization.

Like any other clock driver, the PPS clock driver continuously evaluates the quality of the PPS signal itself. This is done using a median filter, which also provides a dispersion estimate. If for some reason the signal fails or the dispersion becomes excessive, the PPS peer will either become unreachable or stray out of the survivor population. In this case the clock selection re-mitigates as described above. Under these conditions, the mitigation rules defined above will choose the prefer peer, assuming it continues to operate within nominal bounds.

3.3. The PPS Kernel Discipline

Code to implement the PPS kernel discipline is a special feature that can be incorporated in the kernel of some workstations as described in the NTP distribution and in recent papers and reports. The discipline controls the system clock time and/or frequency by means of an external PPS signal connected via the carrier-detect (CD) modem control lead. The kernel serial port routines are modified so that a timestamp is captured at each on-time transition of this lead and the hardpps() kernel routine is called with the timestamp as argument.

As the PPS signal is derived from external equipment, cables, etc., which sometimes fail, a good deal of error checking is done in the hardpps() routine to detect signal failure and excessive noise. This includes checking for excessive jitter and wander, as well as lost interrupts and spurious interrupts due to noise. Error bits are maintained in a status word, which is returned in the ntp_gettimeofday() and ntp_adjtime() system calls, which are included in the modifications. The ntp_adjtime() system call can be used to read and write the status word and thus control the operation and determine the health of the PPS signal.

There are two functions of the PPS kernel discipline, one to provide a frequency discipline independent of the PLL and externally derived corrections, the other to provide a phase offset to discipline the PLL itself. These functions can be separately enabled by means of the ntp_adjtime() system call. In order to operate, the PPS kernel discipline must be enabled by an explicit configuration command and the PPS signal must be present and within nominal jitter and wander tolerances. In the NTP daemon the PPS frequency discipline is enabled by an explicit configuration command and runs continuously if enabled.

While the PPS phase discipline is enabled by an explicit configuration command, it operates only when the prefer peer is among the survivors of the clustering algorithm and operating within 128 ms of server time, as described above. Under these conditions the mitigation rules defined above will choose the prefer peer (or PPS peer if it is used) to control the system clock; however, when

the PPS phase discipline is operating, the kernel disregards updates produced by ntp_adjtime() and uses its internal PPS source instead. The kernel maintains a watchdog timer for the PPS signal; if the signal has not been heard or is out of tolerance for more than some interval, currently two minutes, the kernel discipline is declared inoperable and clock selection re-mitigates as described above.

4. New Radio Clock Drivers

A system of radio clocks is the primary means of synchronization for the worldwide NTP subnet of today. These clocks, represented by several different manufacturers and located in the U.S., Canada, Europe and Australia, are used to synchronize primary (stratum-1) time servers. Over 100 of these servers, including those operated by the U.S. National Institutes of Standards and Technology and U.S. Naval Observatory, now provide synchronization to thousands of secondary (stratum-2) time servers scattered throughout the world. These secondary servers in turn provide synchronization to other servers and hosts in a hierarchical organization several additional layers deep.

Over the years, a number of drivers have been implemented for many commercial radio clocks. While not addressed specifically by the specification, most of these drivers conform to a standard interface implemented in the current NTP distribution. An exhaustive list of radio clock drivers is shown below. Where an accuracy figure is quoted and the radio produces a precision PPS signal, the accuracy is relative to that signal.

1. Undisciplined Local Clock - used to provide synchronization to other hosts when the local clock is controlled by an external clock, such as the laboratory constructed module described in the last quarterly report, the TPRO/Odetics TPRO/S SBus interface, or the NIST “lockclock” implementation of the Automated Computer Time Service.

2. TRAK 8820 GPS Receiver - a receiver for the Global Positioning System (GPS) satellites, with rated accuracy of 300 ns.

3. PSTI/Traconex WWV/WWVH Receiver - a receiver for the U.S. high-frequency radio transmitters in Colorado and Hawaii, with measured accuracy of about 5 ms.

4. Spectracom WWVB Receiver - a receiver for the U.S. low-frequency transmitter in Colorado, with rated accuracy of 100 μs..

5. TrueTime GPS/GOES Receivers - a receiver for generic TrueTime receivers for GPS, the NIST Geosynchronous Orbit Environment Satellite (GOES), and the U.S. Navy OMEGA navigation system, with rated accuracies in the range 250 ns to 1 ms.

6. IRIG Audio Decoder - a driver/decoder using the SPARCstation audio codec and the Inter-Range Instrumentation Group (IRIG) signal widely used in timekeeping systems, with measured accuracy of about 20 μs.

7. Scratchbuilt CHU Receiver - a decoder for the Canadian high-frequency radio transmitter in Ottawa, Ontario. Used with an external modem, it has a measured accuracy of about 5 ms.

8. Generic Reference Clock Driver - a system interface for a number of client clocks - see below.

9. Magnavox MX4200 GPS Receiver - a GPS receiver with measured accuracy of about 20 μs.

10. Austron 2201A GPS Receiver - a receiver for a full-featured timing receiver using both GPS and the LORAN-C navigation system. Using time provided by GPS and oscillator discipline steered by LORAN-C, this receiver has an accuracy of about 50 ns as determined by U.S. Naval Observatory and portable atomic clock.

11. TrueTime OM-DC OMEGA Receiver - a receiver for the U.S. Navy OMEGA radionavigation system, with rated accuracy of 1 ms.

12. KSI/Odetics TPRO/S IRIG Interface - a decoder/interface for an SBus peripheral, as well as a GPS receiver equipped with IRIG, with a rated accuracy of 1 μs relative to the IRIG timecode.

13. Leitch CSD 5300 Master Clock Controller - a system to control and monitor an atomic clock.

14. EES M201 MSF Receiver - a receiver for the U.K. low-frequency transmitter near Rugby.

15. TrueTime GPS/TM-TMD Receiver - a GPS receiver with a rated accuracy of 250 ns.

16. Bancomm GPS/IRIG Receiver - a GPS receiver with IRIG output (note: this driver is not yet fully functional).

17. Datum Precision Time System - a GPS receiver and controller.

18. NIST Automated Computer Time Service - a service operated by NIST and providing synchronization to U.S. national time standards by telephone modem. Operated with telephone redial intervals in the 1000-4000 s range, it has a measured accuracy of about 1 ms.

19. Heath WWV/WWVH Receiver - a receiver for the U.S. high-frequency radio transmitters in Colorado and Hawaii, with a rated accuracy of 100 ms under favorable propagation conditions.

20. Generic NMEA GPS Receiver - an inexpensive GPS receiver.

21. Motorola Six Gun GPS Receiver - an inexpensive GPS receiver (note: this driver is not fully functional yet).

22. PPS Clock Discipline - an auxiliary driver using the pulse-per-second (PPS) signal generated by some timekeeping equipment. This signal is used in conjunction with another radio clock or NTP peer and provides considerably better accuracy than that achieved with the radio clock or NTP peer.

In addition to the above, there are a number of other radio clocks supported by the “parse” clock driver. Currently, the following nine clock types are supported.

1. Meinberg PZF535 receiver - a DCF77 receiver using FM demodulation and a temperature-compensated quartz oscillator for an accuracy of about 50 us.

2. Meinberg PZF535 receiver - a DCF77 receiver using FM demodulation and an oven-compensated quartz oscillator for an accuracy of about 50 us.

3. Meinberg DCF U/A 31 receiver - a DCF77 receiver using AM demodulation for an accuracy of about 4 ms.

4. ELV DCF7000 - a DCF77 receiver using AM demodulation for an accuracy of about 50 ms.

5. Walter Schmid DCF receiver Kit - a DCF77 receiver using AM demodulation for an accuracy of about 1 ms.
6. Conrad DCF77 receiver module - a DCF77 receiver using 100/200 ms pulses for an accuracy of about 5 ms.
7. TimeBrick DCF77 receiver module - a DCF77 receiver using 100/200 ms pulses for an accuracy of about 5 ms.
8. Meinberg GPS166 receiver - a GPS receiver providing better than 1 μs accuracy when used with the PPS signal.
9. Trimble SV6 GPS receiver - a GPS receiver providing better than 1 μs accuracy when used with the PPS signal.

The drivers above were developed by a number of different implementers in the U.S., U.K., Germany and Australia. A selection of the more unusual are described in more detail below.

4.1. NIST Automated Computer Time Service

This driver supports the NIST Automated Computer Time Service (ACTS). It periodically dials a pre-specified telephone number, receives the NIST timecode data and calculates the local clock correction. It designed primarily for use when neither a radio clock nor connectivity to Internet time servers is available. For the best accuracy, the individual telephone line/modem delay needs to be calibrated using outside sources.

The ACTS is located at NIST Boulder, CO, telephone 303 494 4774. A toll call from Newark, DE, costs between three and four cents, although it is not clear what carrier and time-of-day discounts apply. The modem dial string will differ depending on local telephone configuration, etc., and is specified by the phone command in the configuration file. The argument to this command is an AT command for a Hayes compatible modem.

The accuracy produced by this driver should be in the range of a millisecond or two, but may need correction due to the delay characteristics of the individual modem involved. Ordinarily, the propagation time correction is computed automatically by ACTS and the driver. For undetermined reasons, some modems work with the ACTS echo-delay measurement scheme and some don't. When this is not possible or erratic due to individual modem characteristics, a configurable switch can be set to disable the ACTS echo-delay scheme in which case this driver tries to do the best it can with what it gets. Initial experiments with a Practical Peripherals 9600SA modem suggest an accuracy of a millisecond or two can be achieved without the scheme by using a correction factor of 65.0 ms. In either case, the dispersion for a single call involving ten samples is about 1.3 ms.

The driver can operate in either of three modes, as determined by the *mode* parameter in the *server* configuration command. In automatic mode the driver operates continuously at intervals depending on the prediction error measured by the driver, usually in the order of several hours. In backup mode the driver operates as in automatic mode only when no other source of synchronization is available and when more than one hour has elapsed since last synchronized by other sources. In manual mode the driver operates only when enabled using a configurable switch, as described below.

For reliable call management, this driver requires a 1200-bps modem with a Hayes-compatible command set and control over the modem data terminal ready (DTR) control lead. Present restrictions require the use of a POSIX-compatible programming interface, although other interfaces

may work as well. The ACTS telephone number and modem setup string are hard-coded in the driver and may require changes for nonstandard modems or special circumstances.

Since ACTS will be a toll call in most areas of the U.S., it is necessary to carefully manage the call interval, which is determined in one of three ways. In manual mode a call is initiated by setting a configurable call switch using the *xntpdc* utility, either manually or via a cron job. Since *xntpdc* works across the network, this does not have to be done in the same machine as the server. In automatic mode the call switch is set by the peer timer, which is controlled by the system poll variable in response to measured offset and dispersion. In backup mode the driver is ordinarily asleep, but awakes (in automatic mode) if all other synchronization sources are lost. In either automatic or backup modes, the call interval increases as long as the offset does not exceed the measured error, as determined by the clock discipline described above.

When the call switch is set, the ACTS call program is activated. This program dials each number listed in the phones command in turn. If a call attempt fails, the next number in the list is dialed. This provides diversity paths, perhaps using different long distance carriers or alternate numbers on the ACTS rotary. The call switch and counter are reset and the call program terminated when (a) a valid clock update has been determined, (b) no more numbers remain in the list, (c) a device fault or timeout occurs, or (d) the call switch is reset manually using *xntpdc*.

In automatic and backup modes, the driver determines the call interval using a procedure depending on the measured prediction error and a configurable threshold. If the error exceeds the threshold for a number of times depending on the current interval, the interval is decreased, but not less than about 1000 s. If the error is less than the threshold for some number of times, the interval is increased, but not more than about 18 h. Setting the threshold to 0 results the best accuracy, but decreases the call interval to the minimum. Setting the threshold to a large value, like 100 ms, results in degraded accuracy, but increases the call interval. A reasonable compromise may be a value of 20 ms, which typical results in an average call interval of one call per hour. The exact value for each configuration will depend on the modem and operating system involved, so some experimentation may be necessary.

The mitigation rules are altered in a subtle way when this driver is in use. The algorithm is designed so that, in the default case, this driver will never be selected, unless no other discipline source is available. This can be overridden using the *prefer* keyword with the *server* configuration command for this driver, in which case only this driver will be selected for synchronization and all other discipline sources will be ignored. Ordinarily, the *prefer* keyword is used only in automatic mode when primary time is to be obtained via ACTS and backup NTP peers used only when ACTS fails or to provide calibration of the ACTS driver itself.

4.2. PPS Clock Discipline

Some radio clocks and related timekeeping gear have a pulse-per-second (PPS) signal that can be used to discipline the local clock oscillator to a high degree of precision, typically to the order less than 50 μs in time and 0.1 ppm in frequency. The PPS signal can be connected in either of two ways, either via the data leads of a serial port or via the modem control leads. Either way requires conversion of the PPS signal, usually at TTL levels, to RS232 levels, which can be done using a circuit such as described in the documentation included in the NTP distribution.

The data leads interface requires regenerating the PPS pulse and converting to RS232 signal levels, so that the pulse looks like a legitimate ASCII character. The *tty_clk* module included in the NTP

distribution inserts a timestamp following this character in the input data stream. The driver uses this timestamp to determine the time of arrival of the PPS pulse to within 26 μs at 38.4 kbps while eliminating errors due to operating system queues and service times. In order to use the *tty_clk* module, the distribution must be compiled with CLK defined.

The modem control leads interface requires converting to RS232 levels and connecting to the carrier-detect (CD) lead of a serial port. The *ppsclock* module included in the distribution is used to capture a timestamp at each on-time transition of the PPS signal. The driver reads the latest timestamp with a special ioctl() system call to determine the time of arrival of the PPS pulse to within a few tens of microseconds. In order to use the *ppsclock* module, the distribution must be compiled with PPS defined.

Both of these mechanisms are supported by the PPS reference clock driver. This driver is ordinarily used in conjunction with another clock driver that supports the radio clock that produces the PPS signal. This driver furnishes the coarse timecode used to disambiguate the seconds numbering of the PPS signal itself. The NTP daemon mitigates between the radio clock driver and the PPS driver in order to provide the most accurate time, while respecting the various types of equipment failures that could happen.

5. Alternate Modes for NTP

This section describes alternate modes for NTP which may be suitable for coarse synchronization of hosts in the Internet. It may be particularly applicable for synchronizing PC's, workstations on LANs and extended LANs. These modes usually result in much less traffic than ordinary modes with previous versions of NTP.

In the common bidirectional modes, NTP peers exchange timestamps in continuous rounds where each local peer collects four timestamps: T_1, the time it transmitted its last outbound message, T_2, the time the remote peer received this message, T_3, the time the remote peer transmitted the response, and T_4, the time the local peer received the response. From these four values, the local peer calculates the clock offset of its partner and the roundtrip delay between the peers. A suite of algorithms as described in the NTP version 3 specification RFC-1305, is then used to select the presumed best measurements in a series of rounds with each individual peer and the presumed best ensemble of these peers collectively. The metric used in these algorithms is a function of the roundtrip delay and the dispersion accumulated by these algorithms.

In the broadcast (also called multicast) mode however, only two timestamps are collected: T_1, the time the server transmitted the message, and T_2, the time the local peer received it. Thus it is not possible to determine the propagation delay between the server and the client; moreover, the synchronization distance, which depends on measured delay and dispersion, is unavailable for use in mitigating between succeeding samples and between multiple peers. Nevertheless, it has been demonstrated that U.S. multicast clients can synchronize to European multicast servers typically within the order of 100 ms. Protocol modifications which provide this level of accuracy are described below.

The current NTP model is intentionally hierarchical, generally with primary (stratum 1) servers synchronized by radio or satellite to national standards and to each other using symmetric modes (active and passive) for reciprocal redundancy. Secondary (stratum 2 and above) servers are synchronized to a subset of the primary servers and to subsets of each other. Client file servers and

workstations in local area nets are synchronized to subsets of these secondary servers, often using broadcast mode. At the present stage of development, engineering the worldwide subnet configuration requires intricate analysis of network topology, a little experimentation and a good deal of black art.

We believe that the basic model for NTP must be hierarchical, even when an automatically configured multicast, self-pruning spanning tree is available. The primary reason for this is that synchronization accuracy degrades as the number of spanning-tree hops increase. At some point in the global hierarchy, the accumulated jitter must be removed by retiming the signal at selected nodes of the spanning tree, so that downstream clients listen to that node and not necessarily its parents. We are currently conducting experiments with using NTP in a distributed mode, an abstract of which can be found in a later section.

In the estimated global population of over 100,000 servers and clients in the Internet of today, it is highly likely that a significant fraction of them could use multicast mode without significant reduction in accuracy. Thus, it is absolutely essential for a multicast client to recognize suitably certified and authenticated servers from among the background noise. We believe that this can be done using modern cryptographic methods, including the use of public-key cryptosystems, message digests and key-distribution protocols. These issues are of concern not only to multicast time clients, but also to many other protocols based on similar paradigms.

This authenticated, hierarchical model was motivated during initial experimentation with the NTP multicast feature. A number of NTP mavens connected to the Multicast Backbone (MBone) were brought up with the new Unix *xntpd* implementation and put into operation using the assigned NTP multicast group address 224.0.1.1. These included hosts both in the U.S. and the U.K., some of which enjoyed something less than completely stable network connections. When a client registered itself as a member of the 224.0.1.1 group, messages from all other time servers using the same group address immediately landed on the client. Hence, a mechanism to filter out traffic from unnecessary, inaccurate and unreliable sources became necessary.

5.1. Reasons for Multicasting Time

There are provisions in NTP for providing broadcast service. This feature was intended for use in LANs and networks which did not require the high degree of synchronization available from the use of the symmetric or client/server modes. In broadcast mode, the servers transmit messages periodically and hosts can synchronize their clocks from these messages. It has been the common experience that synchronization to the millisecond level is possible in an extended LAN and hosts closely associated (as in an Ethernet or FDDI ring) to the server can synchronize to within the millisecond.

Most networks limit the scope of a broadcast to the subnet that a host is attached to; hence, it is not possible to use broadcast mode to disseminate messages across several subnets or networks. Multicasting permits the distribution of a message to multiple recipients in Wide Area Networks (WANs). In multicast mode, the servers send out time signals periodically and hosts listening to these chimes synchronize their clocks using the range of algorithms developed for broadcast clients. This technique invites some degradation in accuracy, since there is no way to determine and compensate for the propagation delay across the network(s). Multicasting time signals also enables efficient distribution of the messages over a network, since these messages are delivered only to hosts that are tuned to the specific multicast group address. This leads to a more efficient delivery

mechanism in comparison to broadcast where every host receives a copy of the message, whether it wants to receive the time signals or not.

The number of messages sent out on any network is also vastly reduced using multicast mode. The client server model requires two messages to be exchanged before obtaining a single estimate of the delay and offset. If there are *N* hosts on a network synchronizing to a single server, the number of messages exchanged will be 2*N*. If the *N* hosts were to be synchronized using multicast mode, only a single message is required from the server.

The multicast mode of operation also lays the foundation for peer-discovery and a plug-and-play type of a system. The ultimate goal is that, when a host is newly attached to a network, it listens for time signals on the wire and, using a set of protocols, decides which servers to synchronize to, without any need for any preconfiguration.

5.2. NTP Protocol Refinements:

The suite of algorithms used by NTP to discard faulty clocks and select the best from among the survivors is only partially effective in broadcast mode. The problem lies in accurately determining the roundtrip delay and dispersion, which are used in the client/server mode to determine error bounds and quality weights on a dynamic basis (note that the roundtrip delay can be estimated only using a bidirectional mode). Experiments were carried out using default values for the delay, but the results were not very encouraging. This implies that there is no way to dynamically select which broadcasters are more desirable than others without compromising the very useful feature that no preconfiguration of any kind is necessary, except knowledge of the NTP multicast group address (e.g. 224.0.1.1.), possibly augmented by a cryptographic key identifier and key.

In order to provide the most accurate time using multicast mode, it is necessary to estimate the message transmission delay between the client and server. In principle, this can be done once at the beginning of the session, since this delay normally changes only in small amounts during the session lifetime. This can be achieved using a bidirectional mode temporarily at the time that the association is mobilized. The problem with this technique however, is that the multicast spanning tree path and the unicast paths may be quite different resulting in an inaccurate estimate of the transmission delay.

Recently, provisions for multicasting were introduced in the Unix implementation for NTP. In addition, a hybrid mode of operation was implemented in which a client, having overheard a multicast NTP message, initiates a client/server association in order to determine a systematic correction factor between the time offsets determined using the unicast and multicast modes. This factor arises due to differences in the path geometry between the multicast spanning tree and the reciprocal delays on the unicast path between the client and server. Once this factor has been reliably determined, the client abandons the client/server mode and switches to the broadcast client mode for regular operation. This hybrid mode of operation has the potential to drastically reduce the load on the network and the primary servers in the U.S. and abroad, while improving the accuracy over the old scheme with preconfigured delays.

5.3. Protocol State Machine

NTP broadcast and multicast messages are identical, except in the destination IP address, which is a local IP broadcast address for broadcast messages and an IP multicast group address for multicast messages. By convention, the local IP address has the same subnet address field as the NTP peer with ones in the host address field. In most cases the IP multicast group address used is that assigned

to NTP, i.e. 224.0.1.1. In both cases the media access address conforms to the media specification, usually all ones. By convention, subnet routers do not forward IP broadcast packets, but they do forward IP multicast packets according to a defined architecture and model.

No specific changes are required in the stock Unix kernel or Unix NTP daemon to operate in broadcast mode (5). The existing specification defines the NTP protocol state machine operating in that mode, although the specification as it stands is not strictly complete. The stock Unix kernel however, must be modified to support the Internet Group Membership Protocol (IGMP), to enable communication with other multicast-capable systems.

In principle, there is no change in the NTP protocol state machine when running with either broadcast or multicast messages, although the internal operation of the daemon itself is different in subtle ways. The propagation delays, on the other hand can be far different between local peers in the same LAN and remote peers in a different subnet. For local broadcast peers, the delay can be assumed to be a constant in the low milliseconds without significant loss in accuracy; however, for remote multicast peers it is necessary to determine the propagation delay separately for each remote peer. This is the basis for the modifications to the NTP protocol state machine described below.

The NTP version 3 daemon has been modified to operate in the following way. If the line

broadcast <address> <poll interval> <time-to-live>

is present in the configuration file, the peer sends a NTP broadcast message to the specified address using the specified poll interval and time-to-live. The address can be any local broadcast address or a multicast group address. The time-to-live value provides a crude mechanism for controlling the scope of delivery of the multicast messages; it is fixed at 1 for broadcasts. For each such command, the daemon creates a persistent association; if there is more than one such association, the daemon sends messages for each one separately; however, the daemon does not send broadcast messages unless it is properly synchronized to another NTP peer or radio clock.

Unless the NTP daemon is set up for broadcast or multicast, it does not listen for broadcast or multicast messages and does not respond to these messages. If the line

broadcastclient <broadcast address>

is present in the configuration file, the peer listens on the local broadcast address given. If the line

multicastclient <multicast address>

it listens on both the local broadcast address and, in addition, notifies the subnet multicast router to forward IP multicast messages to the specified multicast group (defaults to the assigned NTP multicast group address 224.0.1.1.)

The NTP daemon and protocol state machine operate in the following way. Note the distinction between the broadcast server, which sends the broadcast/multicast messages, and the broadcast/multicast client, which receives them. Also note the distinction between the broadcast client, which is an NTP peer, and operation in broadcast client mode, which is a state in the protocol machine. The distinction will be clear from context.

1. When specified in the configuration file or the NTP utility *xntpdc*, a broadcast/multicast client listens on the local broadcast address and/or joins the specified multicast group.

2. Upon receiving a NTP broadcast or multicast message for the first time, the broadcast client mobilizes a client association in the normal way with the broadcast server IP address as the source address in the association. In this case, the association responds to messages from the broadcast server to both the unicast address of the client and multicast group address that the broadcast client is a member of.

3. When operating in the client-server mode, the broadcast client sends NTP client messages to the broadcast server, which responds with NTP server messages as required by the protocol. The broadcast client processes these messages in the normal way and accumulates offset, delay and dispersion data to eventually synchronize the local clock. In client mode, the host poll interval is set as the peer poll interval (provided in the broadcast message) divided by eight, in order to speed the initial training process.

   If after eight client messages, the broadcast server has not responded, the association is switched from client mode to broadcast client mode and operation continues in that mode. Presumably, in this case ether no upstream path is available to the server, or the server does not respond to NTP client messages. In broadcast client mode the host poll interval is set as the peer poll interval. The host poll variable is used for reachability determination.

4. Simultaneously as in step 3, the broadcast client collects NTP broadcast messages from the broadcast server and updates its propagation delay estimate from that server to the client via the multicast spanning tree. Note that broadcast messages can be sent over quite different paths than unicast messages and may have far different propagation delays. Therefore, these preliminary delay estimates are not strictly valid until the local clock has been set by this server or by some other means. However, if the client times out as above, operation will continue in broadcast client mode with degraded accuracy.

5. After some number of client/server messages (typically eight), the local clock is set in the normal way. If there are multiple broadcast servers operating, the normal NTP filtering, selection and mitigation algorithms will operate to cast out falsetickers and cherish truechimers. When the local clock is set, each association will switch from client mode to broadcast client mode, operating by only receiving messages from the broadcast server. It is to be noted that the propagation delay from server to client has still not yet been accurately determined.

6. Upon receipt of the first broadcast message after the local clock is set, each broadcast client association computes the final propagation delay estimate and continues indefinitely with this determination.

7. As a further refinement to this scheme, the client may optionally send a client message from time to time in order to refine the propagation delay estimate as the network delays change over the course of the day or week. This feature has not yet been implemented in the Unix NTP daemon.

As implemented in the Unix NTP daemon, the full suite of filtering, selection and combining algorithms remain for use by a broadcast client, which may be operating simultaneously with two or more other servers in both multicast and bidirectional unicast modes. This preserves the diversity and redundancy characteristics of the original model, and with it the capability to recognize falsetickers and select the best truechimers from among an ensemble. This is regarded as a valuable

feature supporting ubiquitous, minimal-intrusion backup paths for primary servers to maintain collective sanity.

From continuous observation of hosts synchronized using the multicast mode of operation, it is seen that hosts which have a small round trip time estimate to the servers synchronize better in comparison to those that have larger values. A large round trip time typically indicates several hops between the host and the synchronizing server and multicast traffic exhibits higher jitter between the hosts that are far apart. Another reason for the higher dispersion values using multicast mode can be attributed to difference in paths between the unicast and multicast modes of operation. Since multicast is not yet fully deployed over the Internet, there are several "tunnels" that have been set up to forward multicast traffic between multicast-capable networks. This can cause a significant error in determining the multicast transmission delay using the unicast path.

Currently, work is in progress to determine suitable bounds for the roundtrip times which would still allow for low dispersion values. Techniques to reduce the jitter in the multicast traffic are also being studied as part of ongoing research.

5.4. Hierarchical Subnets

It does not seem generally useful for all or even most primary servers to use multicast mode directly to client workstations, since the paths across the backbone and regional networks can accumulate severe jitter. A better approach is for secondary servers located relatively "close" to the workstations to synchronize using either the client-server mode or the multicast mode to a defined subset of primary servers chosen to minimize the load on the networks and maximize the accuracy and stability of the delivered service. These secondary servers can then disseminate local or regional service using multicast or broadcast mode. Assuming this hierarchical model, it would be appropriate to integrate the NTP servers with the subnet routers in a network. The routers would then provide synchronization via unicast or broadcast modes and would optionally disable forwarding of NTP messages at strata lower than the router.

While this does divide the problem, it does not conquer it, since the scaling issue must still be addressed. With the above model it does not seem useful for the spanning trees rooted at each primary server to completely cover all or even most of the Internet. What is needed is a robust algorithm where the spanning trees of different primary servers are not allowed to overlap, but annihilate each other at various seams automatically determined by the protocol.

One approach to do this is based on the assumption that many of the IP multicast routers based on Unix already run NTP, along with some based on proprietary systems, e.g., Cisco. Following the NTP model, these routers would synchronize to other low stratum peers using either unicast or multicast modes; however, once a stratum is established, the router would modify the routing tables so that NTP messages of lower strata are not forwarded. This modification could be implemented as a component of the access controls already implemented in many routers. Another approach is to disable forwarding of NTP messages altogether, on the assumption that inbound NTP multicast messages will land on the router NTP daemon anyway and that daemon can generate NTP messages at its own stratum for downstream distribution. Either of these approaches may cause other routers to lose backup paths and become vulnerable to a loss of diversity.

The problem of forwarding NTP multicast messages can be resolved by careful choice of the receivers. Multicast traffic is forwarded along links serving members of the particular multicast group. It is possible to choose the receivers and groups in a manner such that the desired effect is

obtained. It might be possible to tap into the resources of the *sd* (Session Director) service to achieve this result.

There is still the issue of hierarchical subnets of high stratum servers and their clients. One approach to this issue is to enforce "subnets," where the multicast group address space is subdivided based on hierarchy. The routers cum NTP servers can then enforce the rule that the messages of one subnet multicast server cannot poach on the subnets of another lower stratum server. This in fact is the *hello* protocol model used in the original NSFnet backbone routers.

It may still be useful for the low-stratum servers to listen to each other's broadcasts, if only to monitor other server's state of health and as a backup for the ordinary client/server backup servers. In fact, the ultimate refinement of this approach may be for a server to automatically switch from client/server, for the most accurate and stable associations, to multicast for the rest, switching each association back and forth as a function of estimated error (dispersion).

5.5. Multicast Group Address Assignment and Group Address Reuse

If, as expected, there will be a generous number of broadcasters willing to honk the time in multicast mode, a real concern is that a large client population may develop, putting strain on the backbone and burden on the multicast routers themselves. As there are presently over 100 primary (stratum 1) time servers and a client population estimated over 100,000, with each client chiming with three or more servers, the situation can have serious scaling consequences.

An alternate approach requires each distinct spanning tree to be assigned a unique multicast group address, where the size of the spanning tree is determined by the usual administrative procedures now used in the MBone. How these addresses are assigned is a topic for further study; however, there are a number of ways in which this can be done. The most obvious is to associate an addressing hierarchy that parallels that of the domain hierarchy. For instance, group addresses could be assigned to the DNS domains, with the understanding that all servers using a single address would conform to the designated security and service model of that domain. With appropriate thresholds at the junctions between the providers and between the providers and the backbone, these addresses could be reused as necessary. The particular server addresses for each provider could be obtained via DNS using host names such as ntp-1.sura.net for NTP stratum-1 service in SURAnet, for example. This scheme has in fact been implemented for NTP servers by the academic community in the U.K.

This approach, while not hard to implement using existing tools, suffers from the fact that the address assignments which have to be engineered may be difficult to manage and wasteful of precious address real estate. Yet again, the facilities of *sd* now used to initiate multicast sessions may be put to use. The *sd* implements a service for coordinating the assignment of globally unique multicast group addresses. A multicast time server can in principle interact with an *sd* peer to associate a named session with a unique group address. Clients would then discover this address using local *sd* peers and avoid having to tickle the DNS. However, this is not a universal solution, since not all clients in the Internet are likely to operate *sd* peers.

5.6. Authentication

It is a fundamental requirement that a wide-area time synchronization service employs some means of authenticating the timekeeping information exchanged between a client and a server. For example, the Digital Time Synchronization Service (DTSS) runs in an authenticated environment, although the protocol itself does not include provisions for authentication. On the other hand, the

Network Time Protocol (NTP) includes specific provisions for an authentication scheme using either the Data Encryption Standard (DES) or RSA Message Digest (MD5). Although this scheme has been in use for several years, it is considered a temporary expedient until generic authentication features become universally available. Note in passing that the use of DES and its cumbersome export controls is being phased out. Subsequently, it is expected that DES support in NTP will be withdrawn.

In the current NTP scheme, the primary purpose of authentication is for a client to verify that a chosen server is a member of a trusted subset of time servers that share a common secret. The secret is implemented in the form of a message digest, as computed by the DES or MD5 algorithms, and included in each transmitted NTP message. The secret is maintained as long as the transmitter and receiver (and none other) share the same cryptographic key, which is identified by a field in the message. The NTP specification describes the way in which the message digest is computed and checked, but not the manner in which the keys are distributed among the participants.

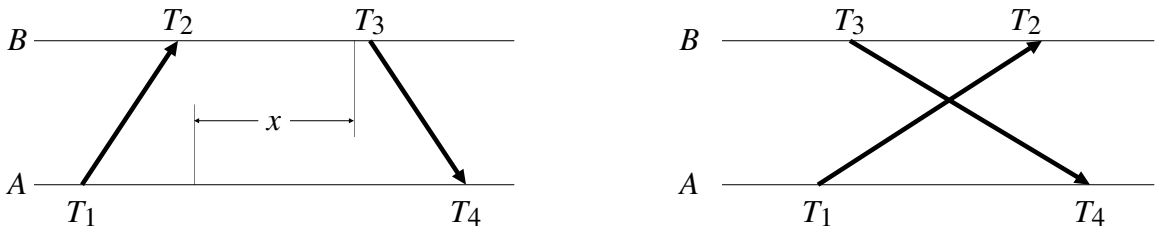
However, the current authentication scheme is inadequate for wide-area multicasting, where hundreds of servers and thousands of clients are scattered all over the globe and connected by sometimes rickety networks. The purpose of this discussion is to provoke thinking, discussion and resolution on engineering changes to the protocol and infrastructure, including directory services such as DNS and authentication services such as Kerberos. The result of these changes should greatly reduce the degree of black art necessary to manage the informal NTP subnet and, in most cases, completely eliminate the need for clients to know anything about the NTP subnet prior to raising antenna.

5.6.1. Authentication in Client-Server Mode

An NTP peer can operate in any of several modes, among them client/server and broadcast modes. In the client/server mode, a client sends a message including the time of origin and message digest to a selected server. The message digest is constructed using a particular initialization vector (presently all zeros) and key, which are presumed known to both the client and server. The server then returns a message including the time of origin and message digest using the same initialization vector and key. The client recomputes the message digest to verify that the data are trusted and then compares the time of origin included in the received message with that saved when the last message was sent. The protocol is inherently tolerant of replay messages and resistant to old replays; that is, a replay of an old client message once a new one has been sent.

A feature of this model (some would consider it a fault) is that only one key is required for each synchronization subnet, not separate keys for each client-server pair, which would become awkward for the common case of several hundred clients per server. In this case, each of possibly several members of the subnet are considered equally trustable and a client can reliably synchronize to any member of that subnet. This model has worked well in the usual case where the subnet peers synchronize only with each other or another trusted subnet of lower stratum.

This scheme was originally designed for use in point-to-point (unicast) modes, since a client (a) selects explicitly which servers to communicate with and (b) can verify a response returned by a server as trusted and associated with a particular message previously sent to that server. This is necessary in order to maintain distinct protocol state machines and related time data separately for each of possibly several servers. Highly evolved algorithms in the NTP specification select from



(a) Nonoverlapping Messages (b) Overlapping Messages

Figure 2. Measuring Delay and Offset

among these servers, the ones with the best time and combine them to produce an accurate update for the local clock.

5.6.2. Authentication in Broadcast Mode

While the best accuracy can be obtained only using multiple servers, in which the roundtrip delay, dispersion and clock offset are individually measured, the NTP specification provides a broadcast mode, in which a client listens for periodic broadcasts from possibly several servers in the "vicinity" and does not ordinarily transmit. With respect to local-net broadcasting, the vicinity extends only to the same subnet, in which the roundtrip times are relatively low and subject to only moderate jitter. In this case, the roundtrip delay, and any additional delay for the message digest to be computed, can be assumed fixed, either by default or explicitly determined by an entry in a configuration file or command line, for instance.

In contrast to the ordinary client/server mode, where the client can verify authenticity on a pairwise basis with each of possibly several servers, it is not possible in broadcast mode to select which broadcasters to listen to, other than assigning each a different key and then trust only those keys for the selected servers or subnet. Owing to the promiscuous nature of the broadcast mode, it seems prudent to require the use of authentication in all cases involving broadcasting. In fact, the current Unix implementation requires authentication if a server has not been previously specified in the configuration file, regardless of mode. Again for emphasis, a subnet client can assume all the servers on the same subnet are equally trustable, as long as all share a common key.

This application obviously suggests the use of a public-key cryptosystem; however, in broadcast mode the NTP scheme serves the same purpose, but with far less computational load, as long as the key is kept confidential. When operating in the client/server mode the scheme is used only to authenticate the server reply. In any case, the processing delay to calculate a public-key cryptosum would severely compromise the accuracy expected of most platforms.

However, in the present NTP authentication scheme, a broadcast client is vulnerable to spoofing, unless the keys can be held in confidence. There is no reliable way to differentiate a server using a particular key from another using the same key, since the server host addresses are not included in the message digest. This is a design weakness that should be addressed and corrected before widespread use of multicast service is initiated. In principle, it is easily solved by the simple expedient of using the server host address (as returned by DNS) as the initialization vector for the message-digest algorithm. By assumption, the server host address is difficult to spoof without intricate kernel modifications and not likely to result along with the correct key as the result of host misconfiguration.

It is in fact not difficult to implement a source-authentication scheme such as this. A new key type and key identifier is defined for use with the MD5 message digest algorithm and retained in the keys data structure by the NTP daemon *xntpd*. This provides backwards compatibility with the current scheme, in which the initialization vector is always zero. The broadcaster, upon recognizing this key type, uses its own host address as the initialization vector. The broadcast client has previously learned the host address, key identifier and key from trusted sources. If the key (and key identifier) is stolen and used by another server, or if the NTP message is modified or replayed by another host, the message digest checksum will fail and the message will be discarded.

5.7. Strawman Model

The ultimate goal in engineering a globally ubiquitous NTP service would be to remove completely the need to identify in advance the names and host addresses of candidate servers and whatever key material is required. Considering the special cases that develop involving common points/paths of failure, especially in low-stratum servers, this goal may not be completely achievable. However, a strawman approach which permits a casual workstation to participate in a self-configuring system could operate in the following way.

The first step in moving to a self-configuring system is for a host to listen for NTP messages or to obtain information from a well-known source. The first approach involves using multicast and the simple act of joining a well-known multicast group (e.g. 224.0.1.1), which causes NTP multicast traffic from various sources to land on the host. The second approach involves querying the DNS to obtain a list of time servers in its "locality". The host can then begin client/server operations and proceed in the normal way with these servers. Both of these approaches have their limitations, especially when they involve security issues implicit in the choice and trustability of servers.

A third approach which provides a greater degree of security is a combination of the two approaches. After the host receives the initial volley of multicast NTP traffic, as in the first approach, a criteria for authenticating the various time servers is necessary. The DNS could then be queried to obtain a public key for a particular server (the public key for the server could be made available as a separate field in the DNS tables). The NTP messages can then be cross-checked using this key and the authenticity of the messages can be determined. This method therefore shields the host from accidental misconfiguration and hostile attack from hackers, for example. However, the issue of distribution of the various keys needs to be addressed.

Another step towards a plug-and-play system is to avoid pre-configuration with respect to retiming of the time signal between different strata. Currently, clients and servers parse a configuration file which contains explicit information on which servers to listen to/prefer, which clients to give time to, etc. Future versions of NTP will need to do avoid pre-configuration to the greatest extent possible; clients and servers will need to apply a range of algorithms to reliably detect faulty/malicious servers and clients, determine the right spots in the network for retiming the signal, and operate in multicast/broadcast mode when the dispersion error is low. These issues are currently being studied.

6. Future Improvements

There are several areas in which the model described in this report can be extended. One of these involves redistributing timestamps received from neighbor peers operating in symmetric or client/server modes to a larger population. This allows peers to calculate not only the conditions relative to their neighbor peers, but between the neighbors of their neighbors and so on. Obviously, this can require substantial protocol changes, which is a topic for further research. While the quantity of data

exchanged in a synchronization subnet of even moderate size can be considerable, the data need not be exchanged at the rate the basic NTP protocol operates, since the data will probably be most useful in determining systematic asymmetrical delays, clock faults, outages and various maintenance activities.

6.1. NTP Distributed Mode

The current NTP protocol model operates in both unicast and broadcast modes, as described in the specification RFC-1305. In unicast mode, synchronization information is exchanged between neighbor peers allowing them to synchronize their clocks to each other. In broadcast mode, a server provides synchronization information to possibly many clients, but insufficient information for the clients to calculate the propagation delay from the server. In both of these modes the data exchanged is between neighbor peers about their own clocks, but only indirectly about the clocks of neighbors of their other neighbors.

In the following, scalars will be represented in light face, vectors in bold face, lower case, and matrices in bold face, upper case. An element of vector $\boldsymbol{v}$ is denoted v_i, while the set of all its elements is $\boldsymbol{v} = [v_i]$. An element of matrix $\boldsymbol{M}$ is denoted m_{ij}, while the set of all its elements is $M = [m_{ij}]$. Let $\boldsymbol{m}_i$ be the ith row vector of $\boldsymbol{M}$ and $\boldsymbol{m}_j$ be the jth column vector; the distinction between the two uses will be clear from context. The transpose of a vector $\boldsymbol{v}$ or matrix $\boldsymbol{M}$ is denoted $\boldsymbol{v}_T$ or $\boldsymbol{M}_T$, respectively. Note that the transpose of a column vector is a row vector, while the transpose of a row vector is a column vector.

In NTP distributed mode, a number of peers in a designated subnet exchange messages among themselves, with each message providing data about all peers in the subnet, including the sender. Ordinarily, this information is exchanged in broadcast or multicast modes; however, the same information could in principle be distributed using unicast modes, but at the cost of a considerably higher message frequency. The manner in which these data are processed is given below. Note that every reference to broadcast should be interpreted to include multicast as well.

Consider a subnet of NTP peers in which a broadcast sent by one of them is delivered with fair reliability to all of the others in the subnet. It is not strictly necessary that delivery be free of loss or duplication or in strict sequence.

Figure 2 shows how NTP messages are exchanged between two peers A and B in NTP distributed mode. Peer A captures a transmit timestamp T_1 just before transmitting a message to B and includes it in the message. Peer B captures a receive timestamp T_2 just after reception and records the difference b between its receive and transmit timestamps. In a similar manner, peer A records the difference a between its receive timestamp T_4 and transmit timestamp T_3. Let x_B be a random variable representing the stochastic delay variation on the network path from A to B and x_A represent the delay variation on the reciprocal path. Define $b = T_2 - T_1 + x_B$ as the propagation delay from A to B and $a = T_4 - T_3 + x_A$ as the propagation delay from B to A.

In NTP distributed mode, all subnet peers periodically broadcast their propagation delays measured as above. For example, in Figure 2 all peers learn the propagation delays a and b in broadcast messages from B and A, respectively. Note that the order of measurements by A and B illustrated in (a) and (b) of Figure 2 cannot be determined from the broadcast data, since there is no way to determine whether T_3 is earlier or later than T_2, or whether T_4 is earlier or later than T_1. Nevertheless, in either case the clock offset measured for peer A is

$$\theta_A = \frac{(T_2 - T_1 + x_B) - (T_4 - T_3 + x_A)}{2} = \frac{b - a}{2} ,$$

while the clock offset measured for *B* is

$$\theta_B = \frac{(T_4 - T_3 + x_A) - (T_2 - T_1 + x_B)}{2} = \frac{a - b}{2} = -\theta_A .$$

The error in the expectation of the above quantities is distributed as the difference between x_B and x_A, which is ordinarily dominated by systematic asymmetries in path routing. Let x be the actual clock offset of *B* relative to *A*. Then the roundtrip delay measured for *A* is

$$\delta_A = (T_2 - T_1 + x_B - x) + (T_4 - T_3 + x_A + x) = b + a ,$$

while the roundtrip delay measured for *B* is

$$\delta_B = (T_4 - T_3 + x_A + x) + (T_2 - T_1 + x_A - x) = a + b = \delta_A .$$

The error in the expectation of the above quantities is distributed as the sum of x_B and x_A, which can be moderate to large on Internet paths. Note that neither the actual clock offset x nor the order of messages in the above computations matters, as long as the local clocks maintain substantially the same frequency.

The NTP distributed mode can be implemented as follows. Each subnet peer *k* maintains a vector $\boldsymbol{r} = [r_i]$, where *i* ranges over the *n* peers of the subnet. Let q_i be the timestamp generated when the last message from *i* was heard by *k*, p_i be the timestamp generated when this message was sent, and $r_i = q_i - p_i$ be the difference between them. Thus, each new message heard from *i* causes r_i to be overwritten with new data in an atomic operation. Note that q_i is generated by the *k* clock upon reception, while p_i is generated by the *i* clock upon transmission and included in the message itself. Since by convention *k* does not usually transmit to itself, r_k is set to a code indicating not-a-number (NAN). Entries that are unavailable or have timed out are also indicated by NAN.

At designated poll intervals each subnet peer *k* sends a NTP message using the standard NTP broadcast-mode header followed by a copy of the $\boldsymbol{r}$ vector in a defined order and format. The vector is followed by the authenticator, if used. Each subnet peer *k* maintains a matrix $\boldsymbol{T} = [t_{ij}]$, where the row index *i* and column index *j* range over the *n* peers of the subnet. When a broadcast message from *i* is received, the *i*th row vector $\boldsymbol{t}_i$ is overwritten as an atomic operation by the $\boldsymbol{r}$ vector included in the message.

Note that, since only the transmit timestamp is valid in the NTP broadcast message, it is not possible to verify the authenticity of the sender or whether the message has been modified or artificially delayed. Also, as mentioned previously, there is no way to determine whether the propagation delay in a broadcast message sent by peer *i* is earlier or later than the delay in a message sent by *j*, unless additional data are included in the message. This is in contrast to the standard NTP message format, where T_1, T_2 and T_3 shown in Figure 2 are included in the message sent by *B* to *A*.

After subnet peer *k* has received NTP messages from all other peers, its transmitted $\boldsymbol{r}$ vector contains only valid data. When *k* has received these vectors from all other peers, its $\boldsymbol{T}$ matrix contains only valid data. From the $\boldsymbol{T}$ matrix, *k* constructs two matrices, $\boldsymbol{\Theta} = [\theta_{ij}]$, where θ_{ij} represents the clock offset of *j* relative to *i* as measured by *i*, and $\boldsymbol{\Delta}\boldsymbol{j} = [\delta_{ij}]$, where δ_{ij} represents the roundtrip delay

between j and i as measured by i. In principle, the $\boldsymbol{T}$ and thus the Θ and Δ matrices are substantially identical and differ only due to minor differences in sending times, network delay variations and operating system latencies.

Referring to Figure 2 and the discussion above, let t_{ij} be the r_j component in the most recent $\boldsymbol{r}$ vector received from peer i and t_{ji} be the r_i component in the most recent $\boldsymbol{r}$ vector received from peer j. Making no assumptions about the order of the messages,

$$\theta_{ij} = \frac{t_{ji} - t_{ij}}{2} \quad \text{and} \quad \delta_{ij} = t_{ji} + t_{ij} ,$$

Therefore, in the general case,

$$\Theta = \frac{\boldsymbol{T}_T - \boldsymbol{T}}{2} \quad \text{and} \quad \Delta = \boldsymbol{T}_T + \boldsymbol{T} .$$

Assume for the moment there is some algorithm which processes the elements of $\boldsymbol{T}$ to produce the best estimates $\hat{\Theta}$ and $\hat{\Delta}$ of the nominal clock offset and roundtrip delay, respectively, between all peers of the subnet. Ordinarily, this would include the clock filter algorithm defined in the NTP specification, where a separate shift register with offset and delay estimates would be required for the $n \times n$ elements of $\boldsymbol{T}$. Using the clock discipline described earlier in this report, or others suggested in [BAR87] and [LIN80], relative phase and frequency characteristics could be established between all peers in the subnet. The development of appropriate algorithms is an important topic for further research.

7. References

[ALL90] Allan, D., H. Hellwig, P. Kartaschoff, J. Vanier, J. Vig G.M.R. Winkler and N.F. Yannoni. Standard terminology for fundamental frequency and time metrology. *Proc. 42nd Annual Frequency Control Symposium* (1988), 419-425. 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), TN139-TN145.

[BAR87] Barnes, J.A., and S.R. Stein. Application of Kalman filters and ARIMA models to digital frequency and phase lock loops. *Proc. 19th Annual Precise Time and Time Interval (PTTI) Applications and Planning Meeting* (December 1987, Redondo Beach, CA), 311-322.

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

[MIL89] Mills, D.L. Measured performance of the Network Time Protocol in the Internet system. DARPA Network Working Group Report RFC-1128, University of Delaware, October 1989. Also published as: Electrical Engineering Department Report 89-9-3, University of Delaware, September 1989.

[LEV] Levine, J. An algorithm to synchronize the time of a computer clock to universal time. *IEEE Trans. Networks* (to appear).

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

[MIL90] Mills, D.L. Network Time Protocol (Version 3) specification, implementation and analysis. DARPA Network Working Group Report RFC-1305, University of Delaware, March 1992, 113 pp. Previous draft published as: Electrical Engineering Department Report 90-6-1, University of Delaware, June 1990.

[MIL91a] 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 Computer Society Press, Los Alamitos, CA, 1994, 91-102.

[MIL91b] Mills, D.L. On the chronology and metrology of computer network timescales and their application to the Network Time Protocol. *ACM Computer Communications Review 21, 5* (October 1991), 8-17.

[MIL92a] Mills, D.L. Modelling and analysis of computer network clocks. Electrical Engineering Department Report 92-5-2, University of Delaware, May 1992, 29 pp.

[MIL92b] Mills, D.L. Simple Network Time Protocol (SNTP). DARPA Network Working Group Report RFC-1361, University of Delaware, August 1992, 10 pp.

[MIL94a] Mills, D.L. Precision synchronization of computer network clocks. *ACM Computer Communication Review 24, 2* (April 1994). 16 pp.

[MIL94b] Mills, D.L. A kernel model for precision timekeeping. ARPA Network Working Group Report RFC-1589, University of Delaware, March 1994. 31 pp.

[MIL94c] Mills, D.L. Improved algorithms for synchronizing computer network clocks. *Proc. ACM SIGCOMM 94 Symposium* (London UK, September 1994), 317-327.

[STE85] 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.