Precision Synchronization of Computer Network Clocks[1]

David L. Mills
Electrical Engineering Department
University of Delaware

Abstract

This paper builds on previous work involving the Network Time Protocol, which is used to synchronize computer clocks in the Internet. It describes a series of incremental improvements in system hardware and software which result in significantly better accuracy and stability, especially in primary time servers directly synchronized to radio or satellite time services. These improvements include novel interfacing techniques and operating system features. The goal in this effort is to improve the synchronization accuracy for fast computers and networks from the tens of milliseconds regime of the present technology to the submillisecond regime of the future.

In order to assess how well these improvements work, a series of experiments is described in which the error contributions of various modern Unix system hardware and software components are calibrated. These experiments define the accuracy and stability expectations of the computer clock and establish its design parameters with respect to time and frequency error tolerances. The paper concludes that submillisecond accuracies are indeed practical, but that further improvements will be possible only through the use of temperature-compensated local clock oscillators.

Keywords: disciplined oscillator, computer clock, network time synchronization.

1. Introduction

This is one of a series of reports and papers on the technology of synchronizing clocks in computer networks. Previous works have described The Network Time Protocol (NTP) used to synchronize computer clocks in the Internet [MIL91a, MIL92a], modeling and analysis of computer clocks [MIL92b], the chronology and metrology of network timescales [MIL91b], and measurement programs designed to establish the accuracy, stability and reliability in service [MIL90]. This paper, which is a condensation of [MIL93], presents a series of design improvements in interface hardware, input/output driver software and Unix operating system kernel software which improve the accuracy and stability of the local clock, especially when directly synchronized via radio or satellite to national time standards. Included are descriptions of engineered software refinements in the form of modified driver and kernel code that reduce jitter relative to a precision timing source to the order of a few tens of microseconds and timekeeping accuracy for workstations on a common Ethernet to the order of a few hundred microseconds.

This paper begins with an introduction describing the NTP architecture and protocol and the local clock, which is modeled as a disciplined oscillator and implemented as a phase-lock loop (PLL). It describes several methods designed to reduce clock reading errors due to various causes at the hardware, driver and operating system level. Some of these methods involve new or modified device drivers which reduce latencies well below the original system design. Others allow the use of special PPS and IRIG signals generated by some radio clocks, together with the audio codec included in some workstations, to avoid the latencies involved in reading serial ASCII timecodes. Still others involve surgery on the timekeeping software of three different Unix kernels for Sun Microsystems and Digital Equipment machines.

The paper continues with descriptions of several experiments intended to calibrate the success of these improvements with respect to accuracy and stability. They establish the latencies in reading the local clock, the errors accumulated in synchronizing one computer clock to another and the errors due to the intrinsic instability of the local clock oscillator. The paper concludes that it is indeed possible to achieve reliable synchronization to within a few hundred microseconds on an Ethernet or FDDI network using fast, modern workstations, and that the most important factor in limiting the accuracy is the stability of the local clock oscillator.

2. Network Time Protocol

The Network Time Protocol (NTP) is used by Internet time servers and their clients to synchronize clocks, as

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

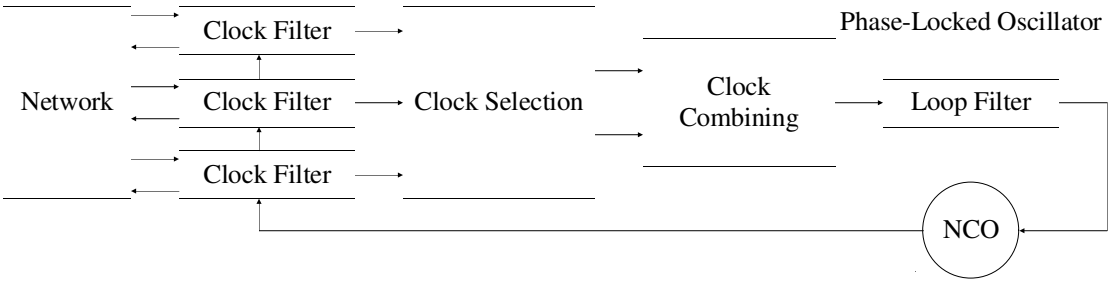

Figure 2. Network Time Protocol

well as automatically organize and maintain the time synchronization subnet itself. It is evolved from the Time Protocol [POS83] and the ICMP Timestamp Message [DAR81b], but is specifically designed for high accuracy, stability and reliability, even when used over typical Internet paths involving multiple gateways and unreliable networks. This section contains an overview of the architecture and algorithms used in NTP. A detailed description of the architecture and service model is contained in [MIL91a], while the current protocol specification, designated NTP Version 3, is defined by RFC-1305 [MIL92a]. A subset of the protocol, designated Simple Network Time Protocol (SNTP), is described in RFC-1361 [MIL92c].

NTP and its implementations have evolved and proliferated in the Internet over the last decade, with NTP Version 2 adopted as an Internet Standard (Recommended) [MIL89] and its successor NTP Version 3 adopted as a Internet Standard (Draft) [MIL92a]. NTP is built on the Internet Protocol (IP) [DAR81a] and User Datagram Protocol (UDP) [POS80], which provide a connectionless transport mechanism; however, it is readily adaptable to other protocol suites. The protocol can operate in several modes appropriate to different scenarios involving private workstations, public servers and various subnet configurations. A lightweight association-management capability, including dynamic reachability and variable poll-interval mechanisms, is used to manage state information and reduce resource requirements. Optional features include message authentication based on crypto-checksums and provisions for remote control and monitoring.

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

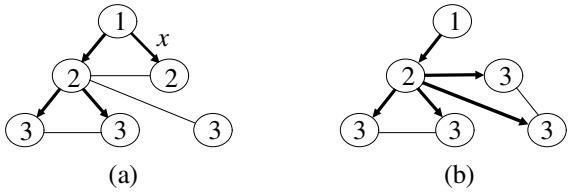

Figure 1. Subnet Synchronization Topologies

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

The clock selection algorithm determines from among all peers a suitable subset capable of providing the most accurate and trustworthy time using principles similar to those described in [VAS88]. This is done using a cascade of two subalgorithms, one based on interval intersections to cast out faulty peers [MAR85] and the other based on clustering and maximum likelihood principles to improve accuracy [MIL91a]. The resulting offsets of this subset are first combined on a weighted-average basis using the algorithm described in [MIL92a] and then processed by a phase-lock loop (PLL) using the algorithms described in [MIL92b]. In the PLL the combined

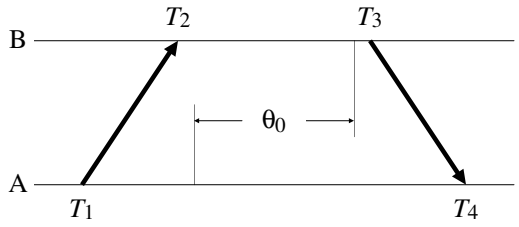

Figure 3. Measuring Delay and Offset

effects of the filtering, selection and combining operations are to produce a phase correction term, which is processed by the loop filter to control the numeric-controlled oscillator (NCO) frequency. The NCO is implemented as an adjustable rate counter using a combination of hardware and software components. It furnishes the phase (timing) reference to produce the timestamps used in all timing calculations.

Figure 3 shows how NTP timestamps are numbered and exchanged between peers *A* and *B*. Let T_1, T_2, T_3, T_4 be the values of the four most recent timestamps as shown and, without loss of generality, assume $T_3 > T_2$. Also, for the moment assume the clocks of *A* and *B* are stable and run at the same rate. Let

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

If the delay difference from *A* to *B* and from *B* to *A*, called *differential delay*, is small, the roundtrip delay δ and clock offset θ of *B* relative to *A* at time T_4 are close to

$$\delta = a - b \quad \text{and} \quad \theta = \frac{a+b}{2}.$$

Each NTP message includes the latest three timestamps T_1, T_2 and T_3, while the fourth timestamp T_4 is determined upon arrival of the message. Thus, both peers *A* and *B* can independently calculate delay and offset using a single bidirectional message stream. This is a symmetric, continuously sampled, time-transfer scheme similar to those used in some digital telephone networks [LIN80]. Among its advantages are that errors due to missing or duplicated message are avoided (see [MIL92b] and [MIL93] for an extended discussion of these issues and a comprehensive analysis of errors).

2.1. The NTP Local Clock Model

The Unix 4.3bsd clock model requires a periodic hardware timer interrupt produced by an oscillator in the 100-1000 Hz range. Each interrupt causes an increment *tick* to be added to the kernel *time* variable. The value of the increment is chosen so that the counter, plus an initial offset established by the settimeofday() call, is equal to the time of day in seconds and microseconds. When the tick does not evenly divide the second in microseconds,

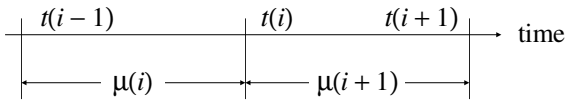

Figure 4. Update Nomenclature

an additional increment *fixtick* is added to the kernel time once each second to make up the difference.

The Unix clock can actually run at three different rates, one at the intrinsic oscillator frequency, another at a slightly higher frequency and a third at a slightly lower frequency. The adjtime() system call can be used to adjust the local clock to a given time offset. The argument is used to select which of the three rates and how long to run at that rate in order to amortize the specified offset.

The NTP local clock model described in [MIL92b] incorporates the Unix local clock as a disciplined oscillator controlled by an adaptive parameter, type-II phase-lock loop. Its characteristics are determined by the transient response of the loop filter, which for a type-II PLL includes an integrator with a lead network for stability. As a disciplining function for a computer clock, the NTP model can be implemented as a sampled-data system using a set of recurrence equations. A capsule overview of the design extracted from [MIL92b] may be helpful in understanding how the model operates.

The local clock is continuously adjusted in small increments at fixed adjustment intervals σ. The increments are computed from state variables representing the frequency offset *f* and phase offset *g*. These variables are determined from the timestamps in messages received at nominal *update intervals* μ, which are variable from about 16 s to over 17 minutes. As part of update processing, the *compliance h* is computed and used to adjust the *time constant* τ. Finally, the *poll interval* ρ for transmitted NTP messages is determined as a multiple of τ. Details on how τ is computed from *h* and how ρ is determined from τ are given in [MIL92a].

Updates are numbered from zero, with those in the neighborhood of the *i*th update shown in Figure 4. All variables are initialized at $i = 0$ to zero. After an interval $\mu(i) = t(i) - t(i-1)$ $(i > 0)$ from the previous update the *i*th update arrives at time $t(i)$ including the time offset $v_s(i)$. When the update $v_s(i)$ is received, the frequency error $f(i+1)$ and phase error $g(i+1)$ are computed:

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

The factor τ in the above determines the PLL time constant and thus the bandwidth as a function of the prevailing network dispersions determined by the NTP

daemon. When the dispersion has been low over some relatively long period, τ is increased and the bandwidth is decreased. In this mode small timing fluctuations due to jitter in the subnet are suppressed and the PLL attains the most accurate phase estimate. On the other hand, if the dispersion becomes high due to network congestion or a systematic frequency change, for example, τ is decreased and the bandwidth is increased. In this mode the PLL is most adaptive to transients due to these causes and others due to system reboot or missed timer interrupts.

The NTP daemon simulates the above recurrence relations and provides offsets to the kernel at intervals of $\sigma = 1$ s using the adjtime() system call. However, provisions have to be made for the additional jitter which results when the timer interval does not divide the second in microseconds. Also, since the adjustment process must complete within 1 s, larger adjustments must be parceled out in a series of adjtime() calls. Finally, provisions must be made to compensate for the roundoff error in computing Δt. These factors add to the error budget, increase system overhead and complicate the daemon implementation.

3. Hardware and Software Interfaces for Precision Timekeeping

It has been demonstrated in previous work cited that it is possible using NTP to synchronize a number of hosts on an Ethernet or a moderately loaded T1 network within a few tens of milliseconds with no particular care in selecting the timing sources or configuring the time servers on the network. This may be adequate for the majority of applications; however, modern workstations and high speed networks can do much better than that, generally to within some fraction of a millisecond, by taking special care in the design of the hardware and software interfaces. The following sections discuss issues related to the design of hardware and software interfaces for external time sources such as radio clocks and associated timing signals.

3.1. Interfaces for the ASCII Timecode

Most radio clocks produce an ASCII timecode with a resolution of 1 ms. Depending on the system implementation, the maximum reading errors range from one to ten milliseconds. For systems with microsecond-resolution local clocks, this results in a maximum peak-to-peak (p-p) jitter of 1 ms. However, assuming the read requests are statistically independent of the clock update times, the average over a large number of readings will make the clock appear 0.5 ms late. To compensate for this, it is only necessary to add 0.5 ms to the reading before further processing by the NTP algorithms. For example, Figure 5 shows the time offsets between a WWVB receiver and the local clock over a typical day. The

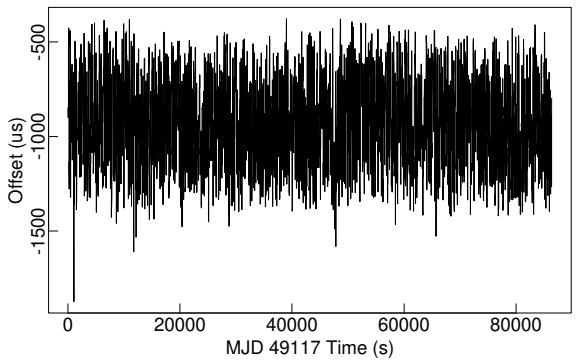

Figure 5. Time Offsets with Serial ASCII Timecode

readings are uniformly distributed over the approximate interval -400 to -1400 µs, with mean about -900 µs; thus, the true offset of the radio clock is -400 µs.

Radio clocks are usually connected to the host computer using a serial port operating at a speed of 9600 bps. The on-time reference epoch for the timecode is usually the start bit of a designated character of the timecode. The UART chip implementing the serial port most often has a sample clock of eight to 16 times the basic bit rate. Assuming the sample clock starts midway in the start bit and continues to midway in the first stop bit and there are eight bits per character, this creates a processing delay of 9.5 bit times, or about 1 ms relative to the start bit of the character. The jitter contribution is usually no more than a couple of sample clock periods, or about 26 µs p-p. This is small compared to the clock reading jitter and can be ignored. Thus, the UART delay can be considered constant, so the hardware contribution to the total mean delay budget is 0.5 + 1.0 = 1.5 ms.

In some kernel serial port drivers, in particular, the Sun zs driver, an intentional delay is introduced when characters are received after an idle period. A batch of characters is passed to the calling program when either (a) a timeout in the neighborhood of 10 ms expires or (b) an input buffer fills up. The intent in this design is to reduce the interrupt load on the processor by batching the characters where possible. Obviously, this can cause severe problems for precision timekeeping. Judah Levine of the National Institute of Science and Technology (NIST) has developed patches for the zs driver which fixes this problem for the native serial ports of the Sun SPARCstation [personal communication].

Good timekeeping depends strongly on the means available to capture an accurate timestamp at the instant the stop bit of the on-time character is found; therefore, the code path delay between the character interrupt routine and the first place a timestamp can be captured is very important, since on some systems, such as Sun SPARCstations, this path can be astonishingly long. The Unix scheduling mechanisms involve both a hardware

interrupt queue and a software interrupt queue. Entries are made on the hardware queue as the interrupt is signaled and generally with the lowest latency, estimated at 20-30 μs for a Sun SPARCstation IPC. Then, after minimal processing, an entry is made on the software queue for later processing in order of software interrupt priority. Finally, the software interrupt unblocks the NTP daemon, which then calculates the current local clock offset and introduces corrections as required.

Opportunities exist to capture timestamps at the hardware interrupt time, software interrupt time and at the time the NTP daemon is activated, but these involve various degrees of kernel trespass and hardware gimmicks. To gain some idea of the severity of the errors introduced at each of these stages, measurements were made using a Sun IPC and a test setup that results in an error between the local clock and a precision timing source (calibrated cesium clock) no greater than 0.1 ms. The total delay from the on-time epoch to when the NTP daemon is activated was measured at 8.3 ms in an otherwise idle system, but increased on rare occasion to over 25 ms under load, even when the NTP daemon was operated at a relatively high software priority level. Since 1.5 ms of the total delay is due to the hardware, the remaining 6.8 ms represents the total code path delay accounting for all software processing from the hardware interrupt to the NTP daemon.

On Unix systems which include support for the SIGIO facility, it is possible to intervene at the time the software interrupt is serviced. The NTP daemon code uses this facility, when available, to capture a timestamp and save it along with the timecode data in a buffer for later processing. This reduces the total code path delay from 6.8 ms to 3.5 ms on an otherwise idle system. This design is used for all input processing, including network interfaces and serial ports.

By far the best place to capture a serial-port timestamp is right in the kernel interrupt routine, but this generally requires intruding in the kernel code itself, which can be intricate and architecture dependent. The next best place is in some routine close to the interrupt routine on the code path. There are two ways to do this, depending on the ancestry of the Unix operating system variant. Older systems based primarily on the original Unix 4.3bsd support *line discipline* modules, which are hunks of code with more-or-less well defined interface specifications that can get in the way, so to speak, of the code path between the interrupt routine and the remainder of the serial port processing. Newer systems based on System V STREAMS can do the same thing using *streams* modules.

Both approaches are supported in the NTP daemon implementation. The CLK line discipline and streams module operate in the same way. They look for a designated character, usually <CR>, and stuff a Unix *timeval* timestamp in the data stream following that character whenever it is found. Eventually, the data arrive at the clock driver, which then extracts the timestamp as the actual time of arrival. In order to gain some insight as to the effectiveness of this approach, measurements were made using the same test setup described above. The total delay from the on-time epoch to the instant when the timestamp is captured was measured at 3.5 ms. Thus, the net code path delay is this value less the hardware delay 3.5 - 1.5 = 2.0 ms. This represents close to the best that can be achieved using the ASCII timecode.

3.2. Interfaces for the PPS Signal

Many radio clocks produce a 1 pulse-per-second (PPS) signal of considerably better precision than the ASCII timecode. Using this signal, it is possible to avoid the 1-ms p-p jitter and 1.5 ms hardware timecode adjustment entirely. However, a device called a *gadget box* is required to interface this signal to the hardware and operating system. The gadget box includes a level converter and pulse generator that turns the PPS signal on-time transition into a valid character. Although many different circuit designs could be used, a typical design generates a single 26-μs start bit for each PPS signal on-time transition. This appears to the UART operating at 38.4K bps as an ASCII DEL (hex FF).

The character resulting from each PPS signal on-time transition is intercepted by the CLK facility and a timestamp inserted in the data stream. Since the timestamp is captured at the on-time transition, the seconds-fraction portion is the offset between the local clock and the on-time epoch less the UART delay. If the local clock is within ±0.5 s of this epoch, as determined by other means, such as the ASCII timecode, the local clock correction is taken as the offset itself, if between zero and 0.5 s, and the offset minus one second, if between 0.5 and 1.0 s.

The baseline delay between the on-time transition and the timestamp capture was measured at 400±10 μs on an otherwise idle Sun IPC. As the UART delay at 38.4K bps is about 270 μs, the difference, 130 μs, must be due to the hardware interrupt latency plus the time to capture a timestamp, perform register window and context switching, and manage various incidental system operations.

An interesting feature of this approach is that the PPS signal is not necessarily associated with any particular radio clock and, indeed, there may be no such clock at all. Some precision timekeeping equipment, such as cesium clocks, VLF receivers and LORAN-C receivers produce only a precision PPS signal and rely on other mechanisms to resolve the second of the day and day of the year. It is possible for an NTP-synchronized host to

derive the latter information using other NTP peers, presumably properly synchronized within ±0.5 second, and to remove residual jitter using the PPS signal. This makes it quite practical to deliver precision time to local clients when the subnet paths to remote primary servers are heavily congested. This scheme is now in use at the Norwegian Telecommunications Research Establishment in Oslo, Norway.

For the ultimate accuracy and lowest jitter, it would be best to eliminate the UART entirely and capture the PPS on-time transition directly using an appropriate interface. This has been done using a modified serial port driver and modem status lead in the interface cable. In this scheme, described in detail in the NTP Version 3 distribution[2], the PPS source is connected via a gadget box to the carrier-detect lead of a serial port. When a transition is detected, a timestamp is captured and saved for later retrieval using a Unix ioctl() system call. The NTP daemon uses the timestamp in a way similar to the scheme described above. Figure 6 shows the offsets between the local clock and the PPS signal from a Global Positioning System (GPS) receiver measured using this implementation.

However, this scheme is specific to the SunOS 4.1.x kernel and requires a special streams module. Except for special-purpose interface modules, such as the KSI/Odetics TPRO IRIG-B decoder and the modified audio driver for the IRIG-B signal to be described, this scheme provides the most accurate and precise timing . There is essentially no latency and the timestamp is captured within 10-30 μs of the on-time epoch, depending on the system architecture.

3.3. Interfaces for the IRIG Signal

The PPS schemes have the disadvantage that two interfaces are required, one for the PPS signal and the other for the ASCII timecode. There is another signal produced by some radio clocks that can be used for both of these purposes, the Inter-Range Instrumentation Group (IRIG) signal, which was developed to synchronize instrumentation recorders in the early days of the U.S. space program. There are several radio clocks that can produce IRIG signals, including those made by Austron, TrueTime, Odetics and Spectracom, among others.

Among the several IRIG formats is one particularly suited for computer clock synchronization and designated IRIG-B. The signal modulation encodes the day of year and time of day in binary-coded decimal (BCD) format, together with a set of control functions used for housekeeping. Roy LeCates at the University of Delaware designed and implemented an IRIG driver with which the IRIG-B signal can be connected to the audio

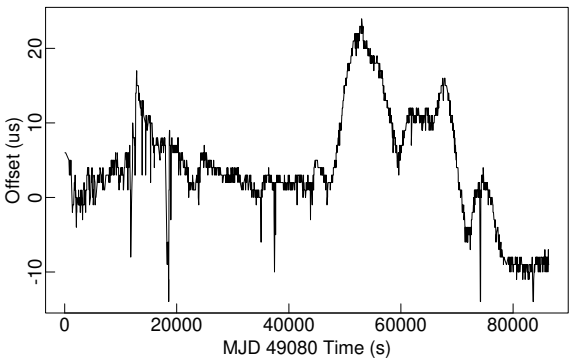

Figure 6. Time Offsets with PPS Signal

codec of some workstations, demodulated and used to synchronize the local clock [MIL93]. There are two components of the IRIG facility, a set of modifications to the BSD audio driver for the Sun SPARCstation, which was designed and implemented by Van Jacobson and collaborators at Lawrence Berkeley National Laboratory, and a companion clock driver for the NTP daemon. This scheme does not require any external circuitry other than a resistor voltage divider, but can synchronize the local clock in principle to within a few microseconds.

In operation, the 1000-Hz modulated IRIG signal is sampled at an 8-kHz rate by the audio codec and processed by the IRIG driver, which synchronizes to the carrier and frame phase. The driver then demodulates the symbols and develops a raw binary timecode and decoded ASCII timecode. A good deal of attention was paid in the software design to noise suppression and efficient demodulation technique. A matched filter is used to synchronize the frame and the zero crossing determined by interpolation to within a few microseconds. An automatic gain control function is implemented in order to cope with varying IRIG signal levels and codec sensitivities.

The NTP clock driver converts the ASCII timecode returned by the read() system call to Unix *timeval* format and subtracts it from the kernel timestamp included in the structure. The result is an adjustment that can be subtracted from the kernel time, as returned in a gettimeofday() call, to correct for the deviation between IRIG time and kernel time. The result can always be relied on to within 128 μs, the audio codec sampling interval, and ordinarily to within a few microseconds, as determined by the interpolation algorithm.

4. Unix Kernel Modifications for Precision Timekeeping

The following sections describe modifications to Unix kernel routines that manage the local clock and timer

2 Information on how to obtain the NTP Version 3 distribution can be obtained from the author.

functions. They provide improved accuracy and stability through the use of a disciplined-oscillator model for use with NTP or another synchronization protocol such as DTSS [DEC89]. There are three versions of this software, one each for the Sun SPARCstation with the SunOS 4.1.x kernel, Digital Equipment DECstation 5000 with the Ultrix 4.x kernel and Digital Equipment 3000 AXP Alpha with the OSF/1 V1.x kernel. The software involves minor changes to the local clock and interval timer routines and includes interfaces for application programs to learn the local clock status and certain statistics of the time-synchronization process. A detailed description of the programming model, including data structures and calling sequences, is given in [MIL93]. Detailed installation instructions are given in the software distributions. However, the software distributions are provided only by special arrangement, since they involve changes to licensed code.

The principal feature added to the kernels is to change the way the local clock is controlled, in order to provide precision time and frequency adjustments. Another feature of the Ultrix and OSF/1 kernel modifications improves the local clock resolution to 1 μs. This feature can in principle be used with any Ultrix-based or OSF/1-based machine that has the required hardware counters, although this has not been verified. Other than improving the local clock resolution, the addition of these features does not affect the operation of existing Unix system calls which affect the local clock, such as gettimeofday(), settimeofday() and adjtime(). The NTP daemon automatically detects the presence of these features and changes behavior accordingly.

In the original Unix design a hardware timer interrupts the kernel at a fixed rate: 100 Hz in the SunOS kernel, 256 Hz in the Ultrix kernel and 1024 Hz in the OSF/1 kernel. Since the Ultrix timer interval (reciprocal of the rate) does not evenly divide one second in microseconds, the kernel adds 64 microseconds once each second, so the timescale consists of 255 advances of 3906 μs plus one of 3970 μs. Similarly, the OSF/1 kernel adds 576 μs once each second, so its timescale consists of 1023 advances of 976 μs plus one of 1552 μs. In this design and with the original NTP daemon design, the adjustment interval σ is fixed at 1 s.

In the original NTP design, the NTP daemon provides offset adjustments to the kernel at periodic adjustment intervals of 1 μs using the adjtime() system call. However, this process is complicated by the need to parcel out large adjustments and to compensate for roundoff error. In the new software this scheme is replaced by another that represents the local clock as a multiple-word, precision time variable in order to provide very precise clock adjustments. At each timer interrupt a precisely calibrated quantity is added to the time variable and carries propagated as required. The quantity is computed as in the NTP local clock model, which operates as a type-II phase-lock loop. This PLL can provide precision control of the local clock oscillator phase to within ±1 μs and frequency to within ±5 ns per day.

The type-II PLL model is identical to the one implemented in the NTP daemon, except that the daemon needs to call the kernel only as each new update is received at update intervals μ, not at the much smaller adjustment intervals σ required by the original scheme. In addition, the need to parcel large updates, account for odd timer rates and compensate for roundoff error is completely avoided.

In the new scheme, a system call ntp_adjtime() operates in a way similar to the original adjtime(), but does not affect the original system call, which continues to operate in its traditional fashion. It is the intent in the design that settimeofday() or adjtime() be used for changes in system time greater than ±128 ms. It has been the Internet experience that the need to change the system time in increments greater than ±128 ms is extremely rare and is usually associated with a hardware or software malfunction or system reboot. The easiest way to do this is with the settimeofday() system call; however, this can under some conditions cause the clock to jump backward. If this cannot be tolerated, adjtime() can be used to slew the clock to the new value without running backward or affecting the frequency discipline process.

Once the local clock has been set within ±128 ms, the ntp_adjtime() system call is used to provide periodic updates including the time offset, maximum error, estimated error and PLL time constant. With NTP the update interval depends on the measured dispersion and time constant; however, the scheme is quite forgiving and neither moderate loss of updates nor variations in the polling interval are serious.

For the Digital Equipment DECstation 5000/240 and possibly other machines of that family, there is an undocumented IOASIC hardware register that counts system bus cycles at a rate of 25 MHz. The new microtime() routine for the Ultrix kernel uses this register to interpolate system time between hardware timer interrupts. This results in a precision of ±1 μs for all time values obtained via the gettimeofday() and ntp_gettime() system calls. For the Digital Equipment 3000 AXP Alpha, the architecture provides a hardware Process Cycle Counter and a machine instruction *rpcc* to read it. This counter operates at the fundamental frequency of the CPU clock or some submultiple of it, 133.333 MHz for the 3000/400 for example. The new microtime() routine for the OSF/1 kernel uses this counter in the same fashion as the Ultrix routine uses the IOASIC counter.

The SunOS kernel already includes a time-of-day clock with microsecond resolution; so, in principle, no microtime() routine is necessary. There is in fact an existing kernel routine uniqtime() which implements this function, but it is coded in the C language and is rather slow. A replacement microtime() routine coded in assembler language is available in the NTP Version 3 distribution and is much faster at about 2 μs per call. In both the Ultrix and OSF/1 kernels the gettimeofday() and ntp_gettime() routines call the microtime() routine, which returns the actual interpolated value.

In order to evaluate how well the kernel modifications work, it is useful to compare the operation over a typical day in the life of a DECstation 5000/240 both with and without the kernel PLL and microtime() routines. Figure 7 shows the cumulative probability distribution of time offsets between a primary server on the same Ethernet segment and the local clock. These curves show the probability that a randomly selected sample offset exceeds a particular value. The upper curve shows the clock behavior without the kernel modifications; the maximum sample offset is 7.95 ms in this case. The lower curve shows the clock behavior with the modifications; the maximum sample offset is 0.933 ms in this case. Clearly, the modifications do significantly improve timekeeping performance.

It is a design feature of the NTP architecture that the local clocks in a synchronization subnet are to read the same or nearly the same values before during and after a leap-second event, as declared by national standards bodies. The kernel software is designed to implement the leap event upon command by an ntp_adjtime() argument. The intricate and sometimes arcane details of the model and implementation are discussed in [MIL91b] and [MIL93].

Most Unix programs read the local clock using the gettimeofday() system call, which returns only the system time and timezone data. For some applications it is useful to know the maximum error of the reported time due to all causes, including clock reading errors, oscillator frequency errors and accumulated latencies on the path to a primary reference source. The user application interface includes a new system call ntp_gettime(), which returns the system time, as well as the maximum error, estimated error and clock status.

5. Errors in Time and Frequency

In preceding sections a number of improvements in driver software and hardware are described, along with modifications to the SunOS, Ultrix and OSF/1 Unix kernels. These provide increased accuracy and stability of the local clock without requiring an external precision oscillator. However, there is only so much improvement possible when the clock oscillator is not specifically

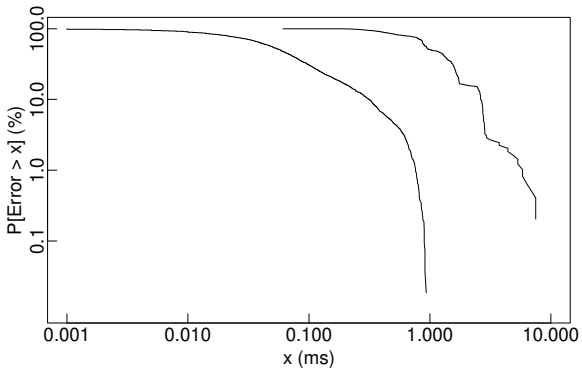

Figure 7. Probability of Error with Ultrix Kernel

engineered for good stability. The basic question to answer is: can the residual sources of error be systematically controlled so that the dominant factor remaining is the stability of the oscillator itself? The software and hardware previously described are designed to do just that.

There are two major components of error remaining in the local clock itself, the timing accuracy, or precision, with which it can be set and read and the frequency accuracy, or stability, it can maintain after being set. For the following experiments, the PPS signal from a GPS radio clock was used as the disciplining source and all hardware and software improvements and kernel modifications described previously were in place. To give some idea of the precisions attainable, the residual latency to capture a timestamp from a PPS transition has been measured at less than 10 μs on a similarly equipped Sun IPC [Craig Leres - personal communication].

5.1. Clock Reading Errors

In order to calibrate the error in reading the local clock from an application, the delay to cycle through the kernel and retrieve a timestamp using the gettimeofday() system call was measured on each of three Unix workstations: a SPARCstation IPC (4/65) SPARC processor running SunOS 4.1.1, a Digital Equipment DECstation 5000/240 MIPS 3000 processor running Ultrix 4.2a and a Digital Equipment 3000/400 AXP Alpha 21064 processor running OSF/1. For the purposes of these measurements, the workstations were performing no tasks other than routine system maintenance and the application task making the measurements.

The experiment involves first touching up to 250,000 words (64 bits in the OSF/1 kernel, 32 bits in the others) of a main-memory array in turn. Since for this experiment the workstations were otherwise idle, this insures that the virtual memory pages are in main memory and that old data are flushed from the various caches and lookaside buffers. Following this, up to 250,000 calls on gettimeofday() are made and the timestamps returned are

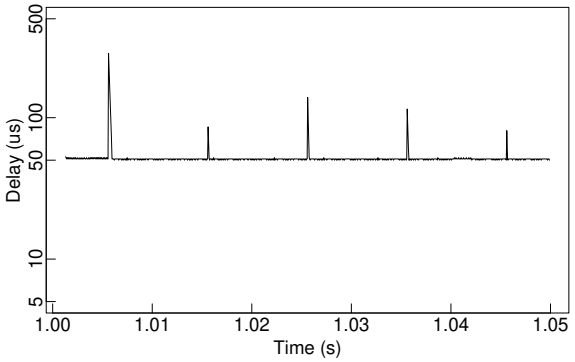

Figure 8. Kernel Latency for SPARCstation IPC - 1

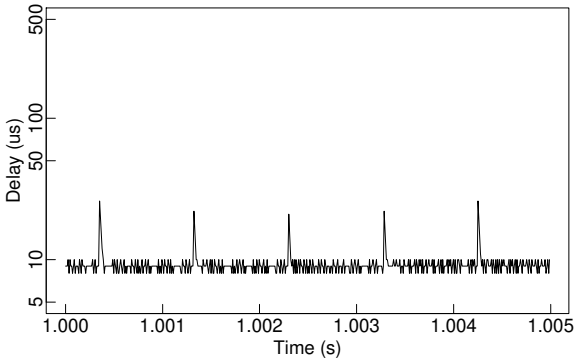

Figure 9. Kernel Latency for DEC 3000/400 AXP

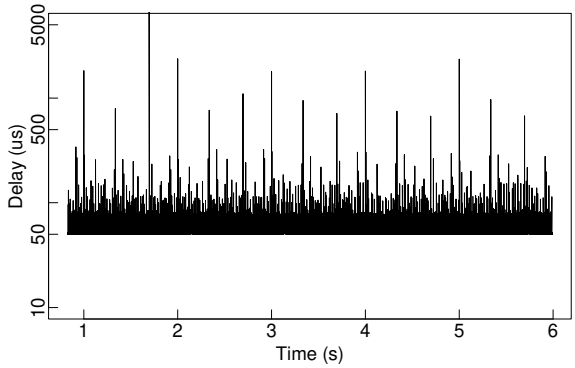

Figure 10. Kernel Latency for SPARCstation IPC - 2

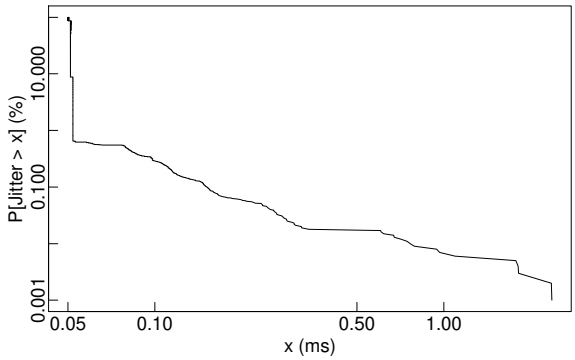

Figure 11. Probability of Error for SPARCstation IPC

saved in the array. Finally, the array is saved in a file for later processing and display.

Figures 8 and 9 show the latencies of the gettimeofday() system call on the SunOS and OSF/1 kernels, respectively. The figures are basically similar and reflect the architecture of the processor and memory system and kernel code paths. Characteristic of these figures is a constant baseline, representing the minimum latency in microseconds of the code path through the kernel, interrupted at intervals with spikes up to several times the baseline. These spikes are due to the timer interrupt routine which is called at each tick of the local clock and reflect the time to update the clock itself and perform certain scheduling and statistics tasks.

The apparent regularity evident in these figures is belied by Figure 10, which shows the latencies of the Sun IPC over a much longer interval than in the previous figure. The fine structure evident in this figure is due to the characteristics shown in Figure 8, but the much larger excursions up to a millisecond or more are most likely due to system daemon housekeeping functions. The figure suggests a basic heartbeat of three beats per second with somewhat longer latencies beating on the second. Obviously, the time intervals of the previous figures were selected to avoid these beats. The on-second beats are in part due to the NTP daemon, which is activated once per second for housekeeping purposes. The cause(s) of the other beats are undetermined, but very likely due to housekeeping functions on the part of other system daemons.

Note that the characteristics shown in these figures are specific to an application process running at a relatively low system priority. It would ordinarily be expected that real-time processes are assigned a higher priority, so that latencies could be controlled with respect to other application and daemon processes. In fact, this is the case with the NTP daemon. In this way at least some the spikes evident in Figure 10 could probably be avoided. Nevertheless, the measurements reported previously in this paper reveal delay excursions over 25 ms on rare occasions, even for the NTP daemon.

In many real-time applications it is more important to assign a precision timestamp to an event than it is to launch it at an exact prearranged epoch. In fact, in all three workstations considered here, internally timed events can be launched only as the result of a timer interrupt, and this limits the timing precision to that of the interval timer itself, which is in the range 1-10 ms. However, when it is necessary to derive a precision timestamp before launching an event, a simple trick can suffice to insure a precision approaching the minimum latency shown in the above figures. The algorithm con-

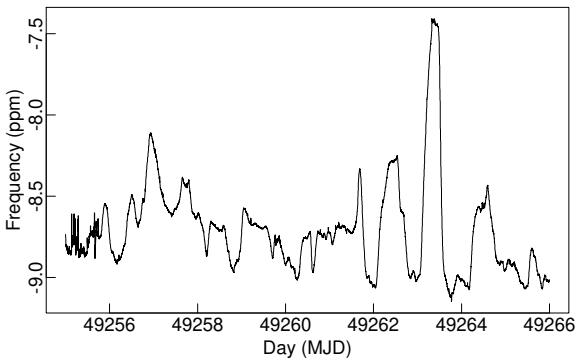

Figure 12. Wander of Typical Clock Oscillator

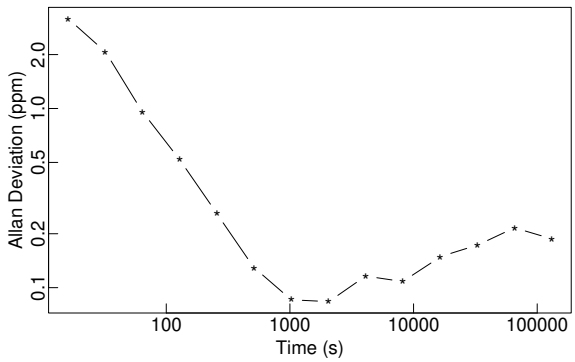

Figure 13. Allan Variance of Typical Local Oscillator

sist of calling gettimeofday() repeatedly until the interval between two calls is less than a prescribed value. There is of course a tradeoff between the precision achievable in this way and the overhead of repeated calls, since the smaller values will cause the calls to be repeated more times.

Figure 11 illustrates the results of this technique using the Sun IPC. The graph shows the cumulative probability distribution for the gettimeofday() latency over one full day, from which a conclusion can be drawn that the probability of exceeding even a threshold as low as about 60 μs is about 0.5 percent, or about the probability of colliding with a timer interrupt on a random request. Note that the probability of an exceeding this value is roughly a straight line on log-log coordinates and that only a few of some 100,000 samples show latencies greater than 1 ms.

5.2. Clock Frequency Stability

With respect to computer network clocks, such as those discussed in this paper, three components of frequency error can be identified: noise, with characteristic intervals less than a minute or so, short-term stability (wander), with intervals from a minute to an hour or so, and long-term stability (mean frequency error), with intervals greater than an hour. The noise component depends on such things as power supply regulation and vibration, but is not ordinarily a problem. The wander component depends primarily on the ambient temperature and is the major source of timing errors in the quartz oscillators used in modern computers. In a type-II PLL the mean frequency error is minimized by the discipline imposed by the PLL and is normally not significant, except for an initial transient while the intrinsic frequency offset of the local clock oscillator is being learned.

Since the major contribution to frequency error is due to temperature fluctuations, it would make sense to stabilize the operating temperature of the circuitry. While the oscillator stability of modern workstations is typically within a couple of ppm in normal office environments, stabilities one or two orders of magnitude better than this are necessary to reliably reduce incidental timing errors to the order of a few tens of microseconds. However, a good temperature-compensated quartz oscillator can be a relatively expensive component not likely to be found in cost-competitive workstations. Therefore, it is assumed in this paper that the time synchronization system must accept what is available, and this is what the following experiments are designed to evaluate.

For an example of the frequency wander in a typical workstation, consider Figure 12, which shows the frequency of the clock oscillator over a ten-day period for a DECstation 5000/240. The figure shows variations over a 2.5 ppm range, with marked diurnal variations on MJD days 49262 and 49263. These happened to be pleasant Fall days when the laboratory windows were open and the ambient temperature followed the climate. Nevertheless, the conclusion drawn from this figure is that frequency variations up to a couple of ppm must be expected as the norm for typical modern workstations.

The stability of a free-running frequency source is commonly characterized by a statistic called *Allan variance* [ALL74], which is defined as follows. Consider a series of time offsets measured between a local clock oscillator and some external standard. Let $\theta(i)$ be the ith measurement and T be the interval between measurements. Define the *fractional frequency* $y(i) \equiv \frac{\theta(i) - \theta(i-1)}{T}$. Consider a sequence of n independent fractional frequency samples $y(j)$ ($j = 1, 2, \ldots, n$). Let τ be the nominal integration interval over which these samples are averaged (not to be confused with the use of τ for the PLL time constant). The 2-sample Allan variance is defined

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

The Allan variance $\sigma_y^2(\tau)$ or Allan deviation $\sigma_y(\tau)$ are particularly useful when comparing the intrinsic stability of the local clock oscillator used in typical workstations,

as it can be used to refine the PLL time constants and update intervals. Figure 13 shows the results of an experiment designed to determine the Allan deviation of a DECstation 5000/240 under typical room-temperature conditions. For the experiment the oscillator was first synchronized to a primary server on the same LAN using NTP to allow the frequency to stabilize, then uncoupled from NTP and allowed to free-run for about seven days. The local clock offsets during this interval were measured using NTP and the primary server. This model is designed to closely duplicate actual operating conditions, including the jitter of the LAN and operating systems involved.

It is important to note that both the *x* and *y* scales of Figure 13 are logarithmic. The characteristic falls rapidly from the lowest τ to a minimum of 0.1 ppm and then rises again to about 0.2 ppm at the highest. The conclusion to be drawn is that adjusting the integration interval much below or much above $\tau = 1000$ s does not improve the oscillator stability.

The PLL time constant is directly related to the integration interval. With the default PLL parameters specified in [MILL92c], for a PLL time constant of 4 the integration interval is about 900 s, or near the optimum. However, while a type-II PLL can in principle eliminate residual timing errors due to a constant frequency offset, the PLL is quite sensitive to changes in frequency, such as might occur due to the room temperature variations illustrated in Figure 12. Figure 14 shows the timing errors induced by a 2-ppm step change in frequency as determined by a simulation model. The error reaches a peak of 600 μs, which is large in comparison with other sources of error considered in this paper. The amplitude of this characteristic scales directly with the temperature change.

The PLL characteristics shown in Figure 14 are calculated for a time constant $\tau = 4$, which requires the update interval $\mu \le 64$ s. For subnet paths spanning a WAN, such frequent updates are impractical and much longer update intervals are appropriate. The design of the NTP PLL allows μ to be increased in direct proportion to τ while preserving the PLL characteristics. To do this, the optimum value of τ is determined on the basis of measured network delays and dispersions. For the longer network paths with higher delays and dispersions, this allows τ to be increased and with it μ. However, a large τ limits the PLL response to temperature-induced frequency changes. Analysis confirms the *x* and *y* axes of the characteristic shown in Figure 14 scale directly as τ, which means the timing errors will scale as well.

6. Timekeeping in the Global Internet

The preceding sections suggest that submillisecond timekeeping on a primary server connected directly to a precision source of time is possible most of the time, where the exceptions are almost always due either to system disruptions like reboot or such things as kernel error messages or large temperature surges. As a practical matter, it is useful to explore just how well the timekeeping function can be managed in an ordinary LAN workstation and in WAN paths of various kinds. In this section are presented the results from several experiments meant to calibrate the expectations in accuracy. As before, all hardware and software improvements and kernel modifications have been done on the LAN workstations, although this is not the case on systems outside the LAN.

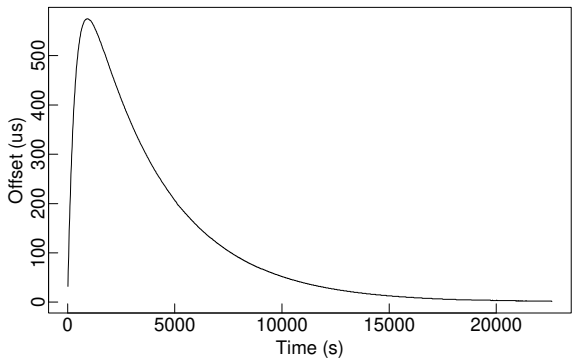

Figure 14. Transient Response of NTP PLL - Frequency

It is important to note that, in all measurements reported in this section, time offsets relative to the local clock are measured at the output of the clock filter on Figure 2, so include the smoothing effect of that filter. However, the local clock itself is controlled by that output and others and processed further by the clock selection and combining algorithms before processing by the local clock algorithm. The local clock algorithm acts as a low-pass filter to suppress transients, so that solitary spikes shown in the data are almost always suppressed. Thus, while it is not possible to infer the exact local clock offset between two NTP time servers, it is certain that the actual offsets tend to the mean as shown in the figures.

6.1. NTP Timekeeping in LANs and WANs

Figure 15 shows the timekeeping behavior of a primary server synchronized to the PPS signal of a GPS radio clock over one full day. The data from which this figure was generated consist of measured offsets between the PPS signal and the local clock, where the measurements were taken every 16 s. This particular machine is a dedicated, primary server with both GPS and WWVB radio clocks and supporting over 400 clients, some of which use the computationally intense cryptographic authentication procedures outlined in the NTP Version 3 specification RFC-1305. Both noise and wander components are apparent from the figure, as well as a 400-μs

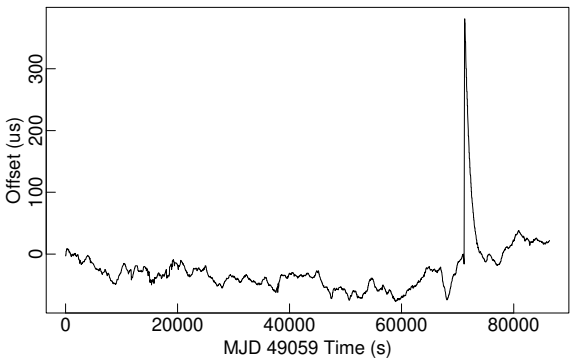

Figure 15. Time Offsets of a Primary Server

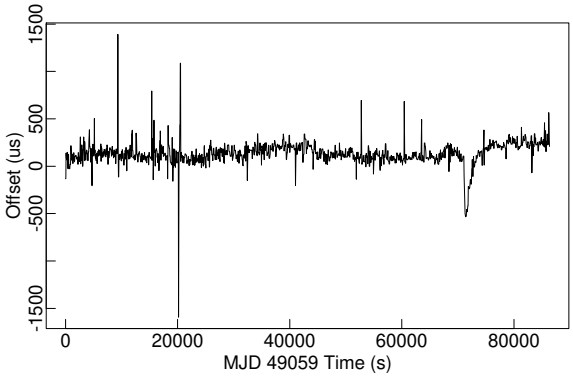

Figure 16. Time Offsets Between Primary Servers

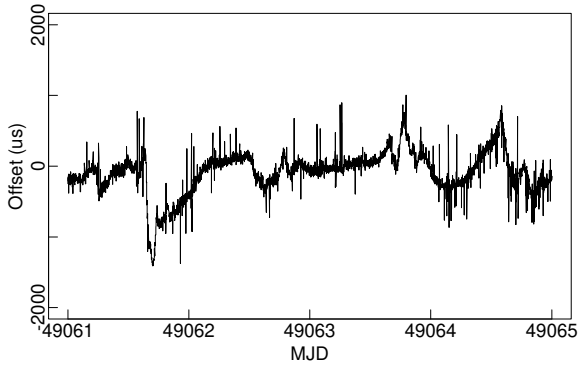

Figure 17. Time Offsets of a LAN Secondary Server

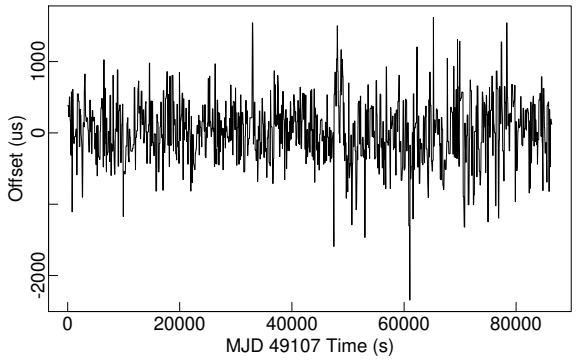

Figure 18. Time Offsets of a NSFnet Secondary Server

glitch that may be due to something as arcane as a daemon restart or radio glitch, for example. The behavior shown in Figure 15 should be contrasted with the behavior shown in Figure 6, which is for a similarly configured primary server restricted to a lessor number of private clients. These data suggest a conclusion that, even with over 400 clients and two radio clocks, the local clock can be stabilized to well within the millisecond.

Figure 16 shows the time offsets between two primary servers, each synchronized to the same PPS signal and connected by a moderately loaded Ethernet. One of these servers is the dedicated primary server mentioned above, while the other is both a primary time server and a file server for a network of about two dozen client workstations, so the experiment is typical of working systems. The jitter apparent in the figure is due to queueing delays, Ethernet collisions and all the ordinary timing noise expected in a working environment. There are occasional spikes of 1 ms or more due to various causes, but these are suppressed by the PLL. Note the 400-μs spike near second 72,000, which matches the spike of Figure 15 taken on the same day, and the occurrence of an apparent systematic offset of 50-100 μs, which is believed due to the difference in cable lengths connecting each server to the GPS radio clock.

Figure 17 shows the time offsets between a primary server and a secondary (stratum 2) client on the DCnet, a multi-segment LAN with Ethernet and FDDI segments [MIL93], over five full days. The PLL time constant $\tau = 4$ and the update interval $\mu = 64$ s. Clearly, the wander due to ambient temperature variations has increased; there is a hint of diurnal variation as well. This particular machine is located in a room with a window air conditioner, so is subject to relatively large and sudden temperature changes. Similar graphs were obtained using several LAN workstations of various manufacture and comparable speeds using both Ethernet and FDDI transmission media.

Figure 18 shows the result of a similar experiment between a primary server and secondary client over a 1.544 Mbps circuit and several routers. The primary server is one of the DCnet public servers mentioned previously, while the secondary client is an IBM RS6000 which is a component of the NSFnet backbone node at College Park, MD. There are three routers on two Ethernets, the T1 circuit and a token ring on the path between the two machines, but the T1 circuit is loaded to only a fraction of its capacity. Again, note that, while the jitter evident in the figure ranges over about 2.5 ms, the PLL at the client is effectively a low-pass filter and removes much of the apparent jitter. It would not be adventurous to

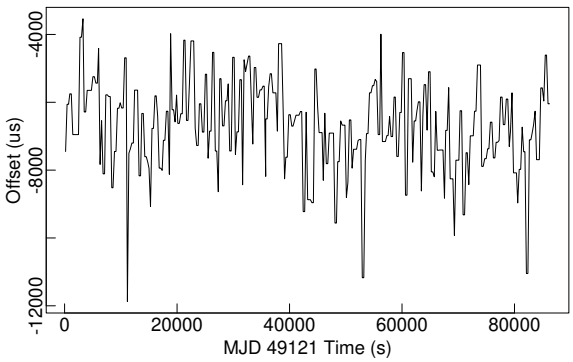

Figure 19. Time Offsets of a NIST Primary Server

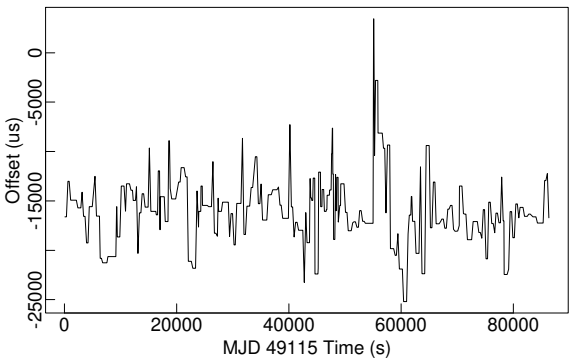

Figure 20. Time Offsets of an Australian Primary Server

suggest the actual discrepancies between the two clocks are not much worse than shown in Figure 17.

However, timekeeping accuracy using much longer paths spanning the globe can be uneven. Figure 19 illustrates a path between a DCnet primary server and a client at the National Institute of Science and Technology (NIST) at Boulder, CO, which is directly synchronized to the U.S. national standard cesium clock ensemble. Note the apparent bias of about -5.5 ms, which is due to the differential delays on the outbound and return legs of the network path. The outbound leg enters the NIST agency network at Gaithersburg, MD, while the return leg enters the NSFnet backbone at National Center for Atmospheric Research (NCAR) at Boulder, CO. These two legs have different transmission delays, undoubtedly due to different network speeds.

Finally, in a search to determine from among about 100 NTP primary time servers the one that was (a) independently synchronized directly to national standard time and (b) as far away as possible in the globe from the DCnet machines, a primary server in Sydney, Australia, was found. This is a truly heroic test, since the transmission facilities are partially by satellite, partially by undersea cable and the intervening networks sometimes slow and seriously congested. Figure 20 shows an apparent -15-ms bias due to differential delays on the outbound and return legs, as well as spikes up to a few milliseconds due to ordinary network queueing, plus a few larger and longer spikes probably due to a circuit outage and network reroute.

6.2. NTP System Performance

The above experiments have used data collected over only one day or a few days. In order to gain some insight in the behavior over longer periods, a number of experiments were conducted over periods spanning up to several months. Table 1 shows a selection of client-server paths typical of the DCnet primary servers described in [MIL93]. During the lifetime of these experiments various software and hardware were in repair, machines were rebooted, the network was reconfigured and different versions of the NTP protocol daemon and Unix kernel were in development. In the table, the first column identifies the peer or radio clock and the second the number of days in which data were collected (data were not collected on days when the local clock offset of the monitoring machine was greater than 1 ms relative to its radio clock). The next two columns give the mean and RMS error over all days of collection, while the next gives the maximum absolute error relative to the mean on the day of collection. Finally, the last four columns give the number of days on which the maximum absolute error exceeds 1 ms, 5 ms, 10 ms and 50 ms, respectively.

The results of these experiments are a mixed bag. The performance of the GPS radio clock was excellent, as expected, but that of the WWVB radio clock is disappointing. There is some evidence to suggest the problem with the latter is due to local receiving conditions, since receivers elsewhere in the country do not experience errors nearly as large as shown in the table. The performance of the clients on the DCnet Ethernet (subnet 1) and FDDI ring (subnet 5) confirm the claim that they can maintain submillisecond accuracy relative to the primary server, but all of them show errors greater than a millisecond on at least some occasions. This is a strong claim, since all it takes is one delay spike over 1 ms and the day is marked accordingly.

The performance of the global primary servers was somewhat better than expected. These servers are near Washington, D.C. (umd1.umd.edu), San Diego (fuzz.sdsc.edu), NIST Boulder (time_a.timefreq.bldrdoc.gov), Norway (fuzzy.nta.no), Switzerland (swisstime.ethz.ch), Germany (ntps1-0.uni.erlangen.de and Australia (swifty.dap.csiro.au). All of these servers are independently synchronized to a local source of standard time, either by a radio clock or calibrated cesium clock. Most of these servers are many Internet hops distant, where the networks involved are not particularly fast and are frequently congested. For example, the Aus-

NTP Server	Days	Mean	RMS Error	Max Error	>1	>5	>10	>50
Radio Clocks								
Spectracom WWVB	71	-0.974	2.179	57.600	18	4	1	1
Austron GPS	91	0.000	0.012	1.000	0	0	0	0
DCnet Servers								
rackety.udel.edu	95	-0.066	0.053	2.054	11	0	0	0
mizbeaver.udel.edu	17	-0.150	0.171	1.141	2	0	0	0
churchy.udel.edu	42	-0.185	0.227	3.150	15	0	0	0
pogo.udel.edu	88	0.091	0.057	1.588	8	0	0	0
beauregard.udel.edu	187	0.016	0.108	2.688	30	0	0	0
pogo-fddi.udel.edu	113	0.001	0.059	1.643	1	0	0	0
cowbird.udel.edu	63	-0.098	0.238	2.071	13	0	0	0
Global Servers								
umd1.umd.edu	78	-4.266	2.669	35.893	29	29	28	0
fuzzy.nta.no	22	0.015	5.328	70.470	2	2	2	1
swisstime.ethz.ch	37	3.102	4.533	97.291	14	14	13	4
swifty.dap.csiro.au	84	2.364	56.700	3944.471	27	27	27	13
ntps1-0.uni.erlangen.de	70	0.810	10.861	490.931	12	12	12	6
time_a.timefreq.bldrdoc.gov	85	-1.511	1.686	80.567	28	19	11	2
fuzz.sdsc.edu	77	-3.889	2.632	47.597	27	27	23	0
DARTnet Routers								
la.dart.net	83	-0.650	0.771	17.849	28	8	3	0
lbl.dart.net	72	0.103	0.214	15.729	20	8	1	0
isi.dart.net	79	-0.819	0.740	8.564	21	9	0	0
NSFnet Routers								
enss136.t3.ans.net	88	-0.657	1.203	32.659	38	23	10	0
enss141.t3.ans.net	87	-6.285	1.846	20.174	37	29	15	0

Table 1. Characteristics of Typical NTP Peers

tralian server is 20 Internet hops distant from the DCnet monitoring machine and the Switzerland server is 17 hops distant. The mean offsets shown are undoubtedly due to differential path delays; however, the rather large maxima are probably due to network congestion.

The data for the DARTnet routers suggest a claim of submillisecond accuracy on a transcontinental network composed of 1.544 Mbps T1 circuits may be adventurous, since there were at least some days when the offsets exceeded 10 ms. However, some experiments involving DARTnet are designed to stress the network to extremes and likely lead to large variations in delay; it is likely that at least some data were recorded during these experiments and account for some of the spikes. Note that lbl.dart.net is independently synchronized to a GPS radio clock and that there is only one path between any two DARTnet routers, so the mean offset shown represents true measurement error and should be compared with the mean offsets shown for DCnet servers rackety.udel.edu and pogo.udel.edu, which are also synchronized directly to a GPS radio cock.

Two of about two dozen NSFnet backbone routers are shown in the table. The College Park router enss136.t3.ans.net is the same one shown on a smaller interval in Figure 18. The path between this router and the monitoring machine is the main connecting link between the DCnet and other national and regional backbones. Since it carries one of the “multicast tunnels” involved in the MBONE multimedia conferencing network, it is subject to relatively heavy loads on occasion, which explains at least a few of the spikes evident in the data. This site and the other shown operate as secondary servers (stratum 2), with each server configured to use different primary servers. Since these primary servers are located in regional networks some number of hops distant from the NSFnet point of presence, there are differential path delays which account for the mean time offsets shown.

7. Summary and Conclusions

In the several years over which the NTP versions evolved, the accuracy, stability and reliability expectations have increasingly become more ambitious. As each new version was developed, a particular crop of error

sources was found and remedial algorithms devised. This work led to the clock filter algorithm, intersection algorithm, clustering algorithm and combining algorithms of the NTP Version 3 specification and implementation. In parallel, the NTP local clock model was refined and tuned for best performance on existing Internet paths, some of which have outrageous delay variations due to gross overload. Previous experience has suggested that timekeeping accuracies in most portions of the Internet can be achieved to some tens of milliseconds.

This paper discusses issues in precision time synchronization of computer network clocks. The primary emphasis in the discussion is on achieving accuracies better than a millisecond on a network with a one or more primary servers and a number of modern workstation clients. Networks with which this goal has been demonstrated include Ethernet, FDDI and light to moderately loaded 1.544-Mbps T1 circuits. As evident from measurements reported herein, accuracies in the order of 10 ms can usually be achieved on heroic paths of the global Internet, including paths to Australia and Europe. However, to do this reliably may require prior knowledge of differential delays that are unfortunately common in some portions of the Internet.

Much of the discussion in this paper is on methods to improve the accuracy of primary servers and their clients using engineered hardware and/or kernel software modifications. These include mechanisms to capture a precision timestamp from the PPS or IRIG signals generated by some radio clocks. It is apparent, however, that accumulated latencies over 8 ms can accrue in some Unix kernels, unless means are taken to capture timestamps early in the code path between the interrupt and the synchronization daemon. Most of the latency burden can be avoided without kernel modifications, but some workstations will require additional hardware or kernel software to achieve submillisecond accuracy.

This paper describes a number of experiments designed to calibrate performance in various LAN and WAN configurations. The results show, as expected, that timekeeping accuracy depends on the calibration of differential network path delays between the time server and its clients. However, there is no way other than using outside references to determine these delays. In the experiments, measurements between servers independently synchronized to national time standards were used to calibrate them. It is in principal possible to compensate for them using information broadcast from designated time servers, for example.

One striking fact emerging from the experiment program is the observation that the limiting factor to further accuracy improvements is the stability of the local clock, which is usually implemented by an uncompensated quartz oscillator. The stability of such oscillators varies in the order of 1 ppm per degree Celsius. With normal room-temperature variations, the timing error can reach a large fraction of a millisecond. While it is possible to reduce these errors by more frequent updates, this is practical only in primary servers where the radio clock can be read more frequently without imposing additional traffic on the network.

8. Acknowledgments

This research was made possible with equipment grants and loans from Sun Microsystems, Digital Equipment, Cisco Systems, Spectracom, Austron and Bancomm Divisions of Datum and the U.S. Coast Guard Engineering Center. Thanks are due especially to David Katz (Cisco Systems), James Kermitz (U.S. Coast Guard), Judah Levine (NIST) and Jeffery Mogul (Digital Equipment). Acknowledgement is also due to the over two dozen contributors to the NTP Version 3 implementation, especially Dennis Ferguson (Advanced Network Systems), and Lars Mathiesen (University of Copenhagen).

9. References

[ALL74] Allan, D.W., J.H. Shoaf and D. Halford. Statistics of time and frequency data analysis. In: Blair, B.E. (Ed.). *Time and Frequency Theory and Fundamentals*. National Bureau of Standards Monograph 140, U.S. Department of Commerce, 1974, 151-204.

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

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

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

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

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

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

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

[MIL91a] Mills, D.L. Internet time synchronization: the Network Time Protocol. *IEEE Trans. Communications 39, 10* (October 1991), 1482-1493.

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

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

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

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

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

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

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

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

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