1. Introduction

This report discusses the analysis and design of computer clocks. A computer clock is an ensemble of hardware and software components used to provide an accurate, stable and reliable time-of-day function to the operating system and its clients. In order that multiple distributed computers sharing a network can synchronize their operations with each other, it is necessary that some means be provided to exchange time information and synchronize their clocks. If these computers are to agree with Universal Coordinated Time (UTC), a means must be provided to synchronize the network time to UTC as disseminated by various means [NIS90].

The computer clocks of present operating systems such as Unix and Fuzzball are normally synchronized to within a few tens of milliseconds in the Internet of today [MIL90]. However, as workstations and networks become faster, there is every expectation that future applications will require timekeeping to the submillisecond regime. This requires in essence a complete reexamination of all elements of the timekeeping apparatus, including the clock design and its synchronization mechanism. This report examines in detail the various design issues necessary to achieve that goal.

In order to synchronize computer clocks, a time-synchronization protocol and appropriate operating-system support are required. In this report the Network Time Protocol (NTP) developed for the Internet is used as an example, but others, such as the Digital Time Synchronization Service (DTSS) could be used as well. Section 3 gives an overview of NTP. The interested reader is directed to [DEC89] for a description of DTSS.

A *local clock* is used in each computer in order to maintain the time. It includes an oscillator, clock counter and software support to provide the time in some format to the operating system and client processes. It must include provisions to adjust the time and, in some systems, the frequency of the oscillator in response to corrections computed by the time-synchronization protocol. Section 4 describes hardware and software models based on two different operating systems and presents an analysis of their operating characteristics and performance envelopes.

Section 5 contains an detailed analysis of the generic local-clock model, which is described as a disciplined oscillator. This analysis is based on the theory of phase-locked loops and allows predictions of stability, convergence time and accuracy based on various parameters of the design. The analysis considers the dynamic management of various intrinsic parameters in order to achieve the best accuracy and stability with various oscillator types and statistical network delays.

It may happen that the accuracy and reliability of the timekeeping system as a whole can be improved when each computer exchanges timekeeping information with two or more *peers*. In these cases the information from multiple peers can be combined to produce an accuracy better than any one of them. Techniques to accomplish this are described in Section 6. Section 7 contains an extensive analysis of errors and develops absolute and statistical error bounds originating in the local clock and network, while Section 8 is a summary of this report.

2. Terms and Notation

In this report the terms *time, timescale, oscillator, tolerance, clock, epoch, timestamp, calendar* and *date* are used in a technical sense. Strictly speaking, the *time* of an event is an abstraction which determines the ordering of events in some given frame of reference or *timescale*. An *oscillator* is a generator capable of precise frequency (relative to the given timescale) within a specified *tolerance*, usually expressed in parts-per-million (ppm). A *clock* is an oscillator together with a counter which records the number of cycles since being initialized with a given value at a given time. The value of the counter at any given time t is called its *epoch* and recorded as the *timestamp* $T(t)$ of that epoch. In general, epoches are not continuous and depend on the precision of the counter.

A *calendar* is a mapping from epoches in some timescale to the year-day *dates* used in everyday life. Since multiple calendars are in use today and sometimes disagree on the dating of the same epoches in the past, the metrology of past and present epoches is an art practiced by historians. However, the ultimate timescale for our world is based on cosmic oscillators, such as the Sun, Moon and other galactic orbiters. Since the frequencies of these oscillators are relatively unstable and not known exactly, the ultimate reference standard oscillator has been chosen by international agreement as a synthesis of many observations of an atomic transition of exquisite stability. The frequency $R(t)$ of each heavenly and Earthbound oscillator defines a distinctive timescale, not necessarily always continuous, relative to that of the standard oscillator.

The International Standard (SI) definition of *time interval* is in terms of the standard second: "the duration of 9,192,631,770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the cesium-133 atom." Let u represent the standard unit of time interval so defined and its reciprocal $v = u^{-1}$ be the standard unit of frequency. The *standard epoch*, denoted by t, is defined as the reading of a counter that runs at frequency v and began counting at some agreed initial epoch t_0, which defines the *standard* or *absolute timescale*. For the purposes of this report, the standard epoch, as well as the time indicated by a clock will be considered continuous. In practice, time is discrete and determined relative to a clock constructed from an atomic oscillator and system of counter/dividers, which defines a timescale associated with that particular oscillator. Standard time and frequency are then determined from an ensemble of such timescales and algorithms designed to combine them to produce a composite timescale approximating the standard timescale.

In this report the *stability* of a clock is how well it can maintain a constant frequency, the *accuracy* is how well its time compares with national standards and the *precision* is to what degree time can be resolved in a particular timekeeping system. These terms will be given precise definitions when necessary. The *time offset* of clock i relative to clock j is the time difference between them $T_{ij}(t) \equiv T_i(t) - T_j(t)$, while the *frequency offset* of clock i relative to clock j is the frequency difference between them $R_{ij}(t) \equiv R_i(t) - R_j(t)$. Note that $T_{ij} = -T_{ji}$, $R_{ij} = -R_{ji}$ and $T_{ii} = R_{ii} = 0$. In this report reference to simply "offset" means time offset, unless indicated otherwise. The *reliability* of a timekeeping system is the fraction of the time it can be kept connected to the network and operating correctly relative to stated accuracy and stability tolerances.

In order to synchronize clocks, there must be some way to directly or indirectly compare them in time and frequency. In network architectures such as DECnet and Internet local clocks are synchronized to designated *time servers*, which are timekeeping systems belonging to a *synchronization subnet*, in which each server measures the offsets between its local clock and the clocks of the peers in the subnet. In this report to *synchronize frequency* means to adjust the subnet clocks to run at the same frequency, to *synchronize time* means to set them to agree at a particular epoch with respect to Coordinated Universal Time (UTC), as provided by national standards, and to *synchronize clocks* means to synchronize them in both frequency and time.

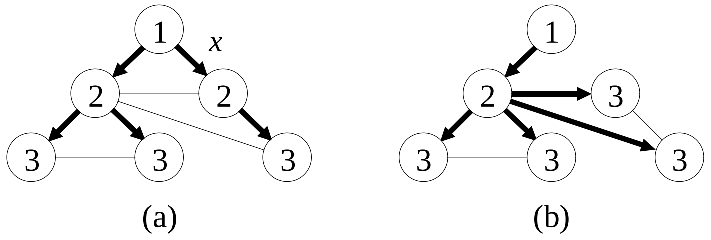

Figure 1. Subnet Synchronization Topologies

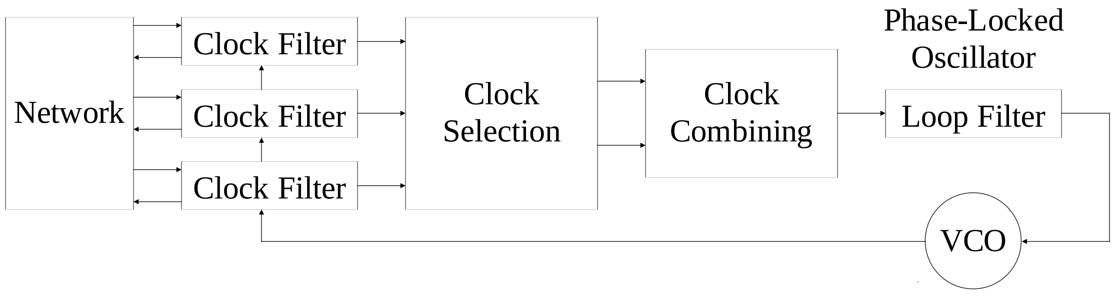

Figure 2. Network Time Protocol

3. Network Time Protocol

The Network Time Protocol (NTP) is used by Internet time servers and their peers to synchronize clocks, as 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 formal description and error analysis of the protocol is contained in [MIL92]. A detailed description of the NTP architecture and protocols is contained in [MIL91a], while a summary of operational experience and performance is contained in [MIL90] and a detailed discussion on timescales is contained in [MIL91b].

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 NTP Version 3 adopted as a Proposed Standard [MIL92]. 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 root (stratum one), 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

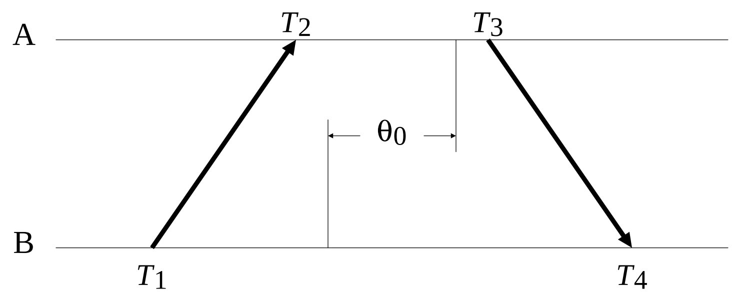

Figure 3. Measuring Delay and Offset

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 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 [MIL92], this algorithm selects from among the last several samples the one with minimum delay and presents the associated offset as the output.

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 $T_3 > T_2$. Let

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

If the subnet delays from A to B and from B to A are similar, the delay δ and offset θ of *B* relative to *A* at time T_i are close to

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

The errors in these quantities, both systematic and random, will be discussed later in this report.

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 reliable message delivery is not required (see [MILL92] for an extended discussion of these issues).

The clock-selection algorithm determines from among all peers a suitable subset of peers 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 maximum-likelihood principles to improve accuracy [MIL91a]. The resulting offsets of this subset are first combined on a weighted-average

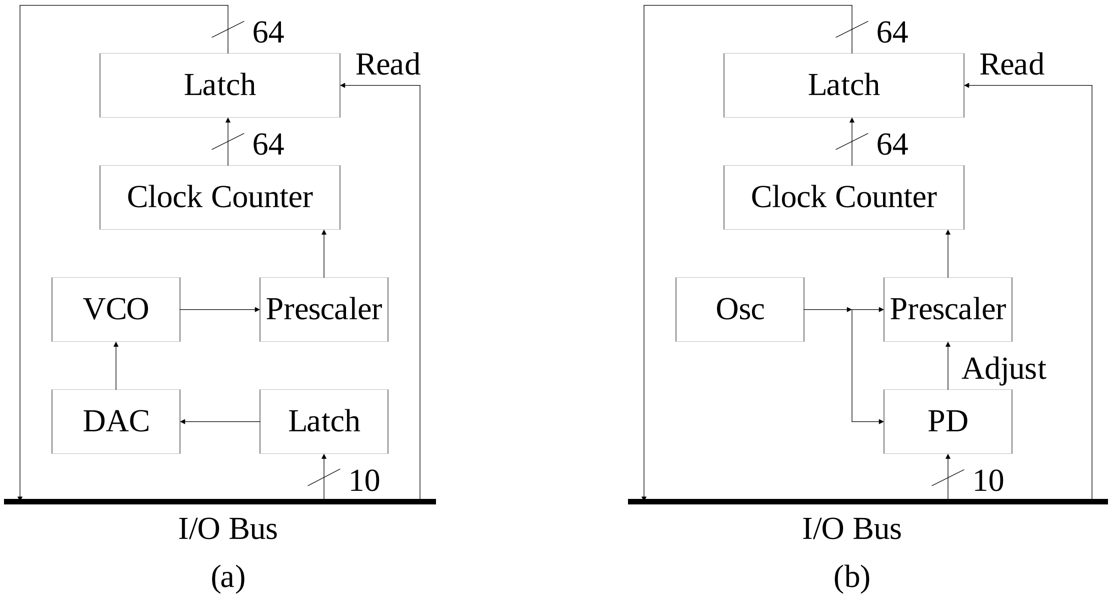

Figure 4. Hardware Clock Models

basis using an algorithm such as described later in this report and then processed by a phase-lock loop (PLL). In the PLL the combined 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 voltage-controlled oscillator (VCO) frequency. The VCO furnishes the phase (timing) reference to produce the timestamps used in all timing calculations.

4. Computer Clock Models

A computer clock includes some kind of reference oscillator stabilized by a quartz crystal or some other means, such as the power grid. The oscillator frequency is usually divided by a prescaler to a convenient frequency, such as 1 MHz or 100 Hz. This is followed by a clock counter, implemented in hardware, software or some combination of the two, which can be read by the processor. For systems intended to be synchronized to an external source of standard time, there must be some means to correct the time and frequency by occasional vernier adjustments produced by the timekeeping protocol. Special care is necessary in all timekeeping system designs to insure that the clock indications are always monotonic increasing; that is, system time never “runs backwards.” This is called the *monotonic requirement*.

The simplest computer clock consists of a hardware latch which is set by prescaler overflow. This causes a processor interrupt or *tick*. The latch is reset when acknowledged by the processor, which then increments the value of a software clock counter. The clock time is adjusted by adding corrections to the counter as necessary. The clock frequency is adjusted by changing the value of the increment, in order to make the counter run faster or slower. The precision of this simple clock model, which is a software emulation of the *phase accumulation method* described in [WIL90], is limited to the value of the increment, usually about 10 ms.

This software clock model requires a processor interrupt on every tick, which can cause significant overhead if the increment is much smaller than 1 ms with the newer RISC processors. Thus, in order to achieve timekeeping precisions less than 1 ms, some kind of hardware assist is usually required. A straightforward design consists of a voltage-controlled oscillator (VCO), in which the frequency is controlled by a buffered, digital/analog converter (DAC). Such a design constructed entirely of hardware logic components is shown in Figure 4a. The clock is read by first pulsing the read signal, which latches the current value of the clock counter, then adding its contents and a 64-bit clock-offset software variable to produce the timestamp. The clock time is set by loading the clock-offset variable, while the clock frequency is adjusted by loading the DAC latch. In principle, this clock model can be adapted to any precision by changing the width of the prescaler or clock counter or changing the VCO frequency. However, it does not seem useful to reduce precision much below the minimum interrupt latency, which is in the low tens of microseconds for a modern RISC processor.

If it is not possible to vary the oscillator frequency, which might be the case if the oscillator is an external frequency standard, a design such as shown in Figure 5b may be used. This approach is similar to the *periodic phase modification* method described in [WIL90]. It includes an oscillator operating at a fixed frequency f_c and a prescaler which divides the oscillator frequency to the working frequency of the clock. The prescaler includes a state machine to advance or retard the phase of the oscillator by one cycle. A programmable divider (PD) is loaded with a value N and produces a pulse train at a frequency $\frac{f_c}{N}$. For each pulse the state machine inserts (stuffs) or deletes (swallows) one oscillator cycle.

The pulse train produced by the prescaler is controlled over a given range by the value of N. If programmed with a high value or zero, relatively few pulses are stuffed or swallowed per second and the frequency counted is near the center of its range; while, if programmed with a low value, relatively many pulses are stuffed or swallowed and the frequency counted is near the upper or lower limits. Assuming some degree of freedom in the choice of oscillator frequency and prescaler ratio, this design can compensate for a wide range of oscillator tolerances.

In all the above designs it is necessary to limit the amount of adjustment incorporated in any step in order to avoid violating the monotonic requirement. With the software clock model this is assured as long as the increment is always positive. If not, the adjustment must be spread over multiple tick intervals. This strategy amounts to a deliberate frequency offset sustained for an interval equal to the total number of ticks required and, in fact, is a feature of the clock models discussed below.

In the hardware clock models the same considerations apply; however, in these designs the tick interval amounts to a single pulse at the prescaler output. In order to avoid decreasing the indicated time when a negative time correction occurs, it is necessary to avoid modifying the clock-offset variable in processor memory and to confine all adjustments to the VCO or prescaler. Thus, all time adjustments must be performed by means of programmed frequency adjustments in much the same way as with the software clock model described above.

It is interesting to conjecture on the design of a processor assist that could provide all of the above functions in a compact, general-purpose hardware interface. In a design similar to that described in [WIL91], the interface might consist of a multifunction timer chip such as the AMD 9513A, which includes five 16-bit counters, each with programmable load and hold registers, plus an onboard crystal oscillator, prescaler and interface circuitry. Four of the 16-bit counters would be used for a 64-bit hardware clock counter and the fifth for the programmable divider. With the addition of a programmable-array logic device and architecture-specific host interface, this compact design could provide all the functions necessary for a comprehensive timekeeping system.

In any clock implementation which requires multiple accesses to read all the clock bits, a protocol is necessary to avoid inconsistent data which could result when one word overflows while another is being read. The usual protocol is to read all the words, then read them all again and compare all except the low-order words. If any word from the first read disagrees with the second, then read all the words again. While this is a simple and effective method, much more efficient and elegant methods are available [LAM90].

4.1. The Fuzzball Clock Model

The Fuzzball is an operating system for the PDP11 family of computers [MIL88]. It supports the Internet protocol suite and includes a number of hardware and software algorithms useful for precision timekeeping in the wide-area Internet. The Fuzzball was instrumental in the design and testing of the Network Time Protocol and the related algorithms described in this report.

The Fuzzball clock model uses a combination of hardware and software to provide precision timing with a minimum of processor overhead. The model includes an oscillator, prescaler and hardware clock counter; however, the oscillator frequency remains constant and the hardware counter produces only a fraction of the total number of bits required by the full clock counter. In the model implementation the hardware counter counts in milliseconds and the software counter is represented in integer milliseconds and fraction, although only the integer portion is available to client applications. A hardware-counter overflow causes the processor to increment the software counter at the bit corresponding to the frequency $1000 \mathrm{x} 2^N$, where N is the width in bits of the hardware counter. The processor reads the clock by first generating a read pulse, which latches the hardware counter, and then adding its contents, suitably aligned, to the software counter. In order to insure atomicity, interrupts are disabled while the counter is latched. If a counter overflow is raised after its contents have been latched, the software counter is incremented and the hardware counter read again.

The Fuzzball clock can be corrected in time by adding a (signed) adjustment to the software clock counter. In practice, this is done only when the actual time is substantially different from the clock time, such as when the system is first started, and may violate the monotonic requirement. Vernier time adjustments which occur in normal system operation are performed at intervals of 4 s, called the *adjustment interval*, and are limited in magnitude to ±500 μs to avoid affecting the visible portion of the counter by more than one tick. This interval provides a maximum frequency adjustment range of ±125 ppm. The adjustment opportunities are created using the interval-timer facility, which is a feature of most operating systems and independent of the time-of-day clock.

This design is similar to the *hidden offset* method described in [WIL90]; however, that method avoids violating the monotonic requirement by using special hardware which is not commonly available in real-time clock interfaces. With the fuzzball design it is necessary to provide a variable used to remember the last clock reading. At the next clock reading the value returned is the maximum of the previous and current clock readings. In fault-tolerant computer architectures using replicated finite-state machines [SCH90] it is necessary that every distinct event be assigned a unique timestamp. In the Fuzzball model this is assured as long as the minimum interval between successive clock readings is greater than 1 ms (one tick). If not, it is necessary to artificially increment each succeeding reading in the same tick, which of course decreases the accuracy relative to network time. However, under the assumption that the average rate of timestamp requests is well below 1000 per second, the probability of more than one request in the same tick is small and the accuracy degradation is slight.

In some applications involving the Fuzzball model, an external pulse-per-second (pps) signal is available from a reference source such as a cesium clock or Global Positioning System (GPS) timing receiver. Such a signal generally provides much higher accuracy than the serial character string produced by a radio clock and is typically in the low nanoseconds. The pps signal can be processed by an interface which produces a hardware interrupt coincident with the arrival of the pps pulse. The processor then determines the current time and computes the residue modulo 1 s. Assuming the seconds numbering of the clock counter has been determined by a reliable source, such as an ordinary radio clock or even NTP itself, the final offset within the second can be determined from the residue.

The above technique has an inherent error equal to the latency of the interrupt system, which in modern RISC processors is in the low tens of microseconds. In the fuzzball this is avoided by latching the hardware clock counter directly by the pps pulse and then reading the counter in the same way as usual. The additional circuitry required to prioritize the pps signal and latch the counter is a feature of some clock interfaces available for the PDP11.

4.2. The Unix Clock Model

The Unix 4.3bsd clock model is based on two system calls, *settimeofday* and *adjtime*, together with two intrinsic variables *tick* and *tickadj*. The *settimeofday* call unceremoniously resets the kernel clock to the value given, while the *adjtime* call slews the kernel clock to a new value numerically equal to the sum of the present time of day and the (signed) argument given in the *adjtime* call. In order to understand the behavior of the Unix clock as incorporated into precision timing systems, it is helpful to explore the operations of *adjtime* in more detail.

The Unix clock model assumes an interrupt produced by an onboard frequency source in the 100-Hz range, such as the oscillator and prescaler described previously. Each interrupt causes an increment called *tick* to be added to a software clock counter. 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 microseconds.

The Unix clock can actually run at three different rates, one at the intrinsic oscillator frequency $\frac{1}{tick}$, another at a slightly higher frequency and a third at a slightly lower frequency. Setting the *amortization rate* $R = \frac{tickadj}{tick}$ for convenience, these three rates correspond to frequency offsets of zero, $+R$ and $-R$ respectively. Normally the zero offset is used; but, if *adjtime* is called, the argument δ given is used to calculate an interval $\Delta t = \frac{\delta}{R}$, called the *amortization interval* (in ticks), during which either the $+R$ or $-R$ rate is used, depending on the sign of δ. The effect is to slew the clock to a new value at a small, constant rate, rather than incorporate the adjustment all at once, which could violate the monotonic requirement.

In most Unix systems the values of *tick* and *tickadj* are expressed as integer microseconds. With common values of $tick = 10{,}000$ μs and $tickadj = 4$ μs, the amortization rate is ±400 ppm. The effective clock frequency can be adjusted by changing the value of *tick*, thus adjusting the frequency

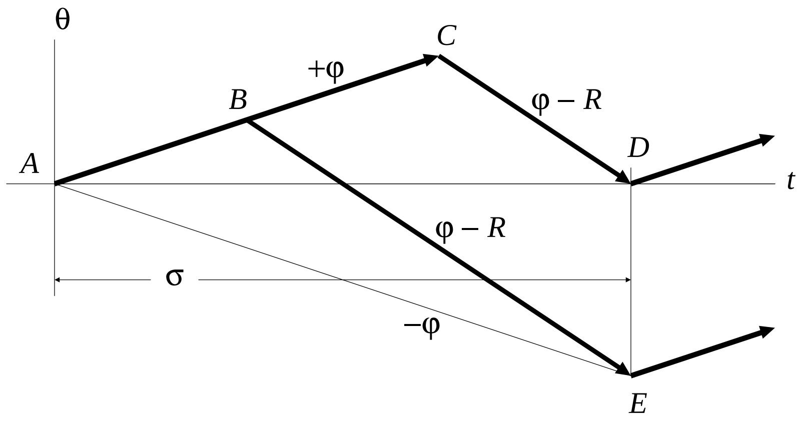

Figure 5. Clock Adjustment Process

in steps of approximately 100 ppm. The amortization rate can be adjusted by changing the value of *tickadj*. The ways in which these quantities affect the accuracy and stability of the local clock are discussed in the next section.

4.3. A Precision Clock Model

In the basic Unix model the precision of the local clock is no better than the value of *tick*, usually about 10 ms; while, in the Fuzzball model the precision is no better than 1 ms. As mentioned previously, it is highly desirable that distinct events be assigned unique timestamps. This is facilitated if the precision of the local clock is of the order of the clock-reading routine execution time, or some few microseconds in a modern RISC workstation. In order to achieve this level of precision, it is necessary to use a high-frequency hardware clock, such as the microsecond clock which is a feature of some Unix platforms.

In order to provide time accurate to much less than 1 ms using message update intervals appropriate for a wide area network, it is necessary to calibrate the intrinsic oscillator frequency error and compensate for it using periodic adjustments. This is accomplished by calling *adjtime* at regular *adjustment intervals*. In newer RISC systems equipped with microsecond clocks, it is possible to obtain accuracies to the order of 100 μs. Using phase-lock techniques as described in the following section, it is possible to sustain this accuracy with update intervals extending to well over ten minutes. The following analysis can be used to establish the parameters of the model.

Figure 5 shows the offset θ as a function of time t for the Unix clock model. The oscillator has an assumed tolerance of $\pm\varphi$ ppm, a fixed amortization rate $\pm R$ ppm and an adjustment interval σ. In the diagram the heavy lines show the offsets for two cases, each labelled with the effective frequency offset. If no corrections are applied, the time offset grows at an assumed maximum positive rate φ as shown by the line *AC* in the diagram. The path *ACE* shows the offset for the case where the clock is steered at zero nominal rate and the line *ABC* for the case where it is steered at an assumed maximum negative rate $-\varphi$. For the first case the amortization interval Δt extends from *C* to *D*, when the clock resumes its intrinsic rate; while, in the second case it extends from *B* to *E*.

If θ is the true offset required at the end of the amortization interval, then

$$\theta = \varphi(\sigma - \Delta t) + \Delta t(\varphi - R) \ .$$

If ε is the maximum absolute error allowed, the amortization interval Δt must satisfy the inequality

$$\Delta t \le \frac{\varepsilon}{R - \varphi} .$$

In order to steer the clock to a nominal rate of zero, $\theta = 0$. The above expressions yield the inequality

$$\sigma \le \frac{R}{\varphi} \frac{\varepsilon}{R - \varphi} .$$

In order to steer the clock to a nominal rate of $-\varphi$, $\theta = -\varphi\sigma$. According to the above, this requires $R \ge 2\varphi$ in order that the adjustments complete before the end of the adjustment interval.

For example, let the maximum absolute error be $\varepsilon = 100$ µs, oscillator tolerance $\varphi = 200$ ppm and amortization rate $R = 400$ ppm. Substituting in the above expressions results in the inequality $\sigma \le 1$ s. While considerably less than the Fuzzball value, an adjustment interval of 1 s is not likely to burden a modern processor.

As a practical matter, and especially with Sun Microsystems SPARCstations, the frequency error can be up to several hundred ppm. SPARCstations have two clocks, a microsecond clock used for ordinary system time and a low resolution calendar clock used to maintain the date. The microsecond clock is periodically compared with the calendar clock and, if off by more than one tick, is adjusted as described previously. In addition, each clock reading is guaranteed to be monotonic increasing using methods similar to the Fuzzball.

While it is possible to compensate for large frequency errors using the above model with appropriate parameters, the adjustment interval would become quite small. A better approach is to adjust the value of *tick* to reduce the frequency error first. For instance, reducing the value of *tick* by one increases the clock frequency by approximately 100 ppm. In this way the frequency error can be adjusted to well within the ±200 ppm bracket assumed in the above analysis.

5. Mathematical Model of the Generic Local Clock

The local clock is the source of all timing information used by the operating system and its clients, as well as the timestamps used by the time-synchronization protocol. In fault-resistant distributed systems, the goal of the timekeeping system is to minimize the error of all timestamps relative to the minimum transmission delay over the network path between any two nodes. As shown later, this is not possible under the assumption that the contributing oscillators are not frequency stabilized. However, it is possible if they are designed as illustrated in this section.

In a previous section a precision clock model is developed which can maintain a given maximum error by introducing corrections at given adjustment intervals. In the model illustrated the resulting time offsets appear as triangles with a peak of 100 µs, but requires an adjustment interval not greater than 1 s. In wide area networks the time-synchronization protocol can deliver measurement updates only at intervals much greater than this, depending on protocol design and the accuracy required. Therefore, it is necessary to provide some kind of “flywheel” having a relatively long time constant in the order of many minutes or even hours.

The models discussed in the previous section include provisions to adjust the clock in both time and frequency. However, a significant improvement in accuracy and stability is possible by modelling the local clock and its adjustment mechanisms as a *disciplined oscillator*. In a disciplined oscillator the frequency is stabilized by a feedback loop with a relatively long time constant, so the frequency is “learned” over some minutes or hours of integration. Besides improving accuracy, a disciplined

Variable	Description
v_d	phase detector output
v_s	clock filter output
v_c	loop filter output
θ_r	reference phase
θ_o	VCO phase
ω_c	PLL crossover frequency
ω_z	PLL corner frequency

Table 1. Phase-Lock Loop Variables

Parameter	Value	Description
α	2^0	VCO gain
σ	2^0	adjustment interval
τ	1	PLL time constant
T	2^9	clock-filter delay
K_f	2^{24}	frequency weight
K_g	2^{10}	phase weight
K_h	2^{13}	compliance weight
K_s	2^4	compliance maximum
K_t	2^{14}	compliance multiplier
K_p	2^6	poll-interval multiplier

Table 2. Phase-Lock Loop Parameters

oscillator can stabilize the frequency to a degree consonant with the intrinsic stability of the clock oscillator, which is usually much less than its tolerance. This allows the update interval to be increased substantially without loss of accuracy.

A disciplined oscillator can be implemented as the phase-lock loop (PLL) shown in Figure 2. The PLL provides the means to adjust the local-clock time and frequency in response to corrections delivered by the time-synchronization protocol. Its behavior can be described using an extensive body of mathematics developed for the purpose, such as given in [SMI86] and elaborated in this section. In addition, a design example is given for implementation guidance in operating-systems environments such as Unix and Fuzzball. Table 1 summarizes the quantities ordinarily treated as variables in the model. By convention, v is used for internal loop variables, θ for phase, ω for frequency and τ for time. Table 2 summarizes those quantities ordinarily fixed as constants in the model. Note that these are all expressed as a power of two, so that multiplies and divides can be accomplished by shifts.

As shown in Figure 6, the variable θ_r represents the phase of the reference signal and θ_o the phase of the voltage-controlled oscillator (VCO). The phase detector (PD) produces a voltage v_d representing the phase difference $\theta_r - \theta_o$. The clock filter, if used, functions as a tapped delay line, with the output v_s taken at the tap selected by the clock-filter algorithm. The loop filter, represented by the equations given below, produces a VCO correction voltage v_c, which controls the oscillator frequency and thus the phase θ_o.

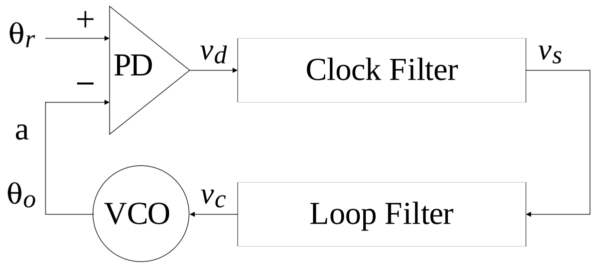

Figure 6. Phase-Lock Loop (PLL) Model

The open-loop transfer function $G(s)$ is constructed by breaking the loop at point a on Figure 6 and computing the ratio of the output phase $\theta_o(s)$ to the reference phase $\theta_r(s)$. This function is the product of the individual transfer functions for the phase detector, clock filter, loop filter and VCO. With appropriate scaling the phase detector delivers a voltage $v_d(t) = \theta_r(t)$ V/rad, so its transfer function is simply $F_d(s) = 1$. The VCO delivers a frequency change $\Delta\omega = \frac{d\,\theta_o(t)}{dt} = \alpha v_c(t)$, where α is the VCO gain in rad/V-s and $\theta_o(t) = \alpha \int v_c(t)\,dt$. Its transfer function is the Laplace transform of the integral, $F_o(s) = \frac{\alpha}{s}$. The clock filter contributes a statistical delay due to the clock-filter algorithm; but, for present purposes, this delay will be assumed a constant T, so its transfer function is the Laplace transform of the delay, $F_s(s) = e^{-Ts}$. The open-loop gain is the product of these four transfer functions

$$G(s) \equiv \frac{\theta_o(s)}{\theta_r(s)} = F_d(s)F_s(s)F(s)F_o(s) = \frac{\alpha}{s}e^{-Ts}F(s)\ .$$

In many cases the effects of the clock-filter delay T can be neglected, so that $e^{-Ts} = 1$. Then, the above equation can be written

$$G(s) = \frac{\alpha}{s}F(s)\ . \tag{1}$$

The PLL behavior is then completely determined by the loop gain α and transfer function $F(s)$. In the simplest case where $F(s)$ is a constant, the loop will exhibit the classic behavior of a single-time-constant (STC) system; however, in order to reduce incidental timing jitter, it is usually desirable to include a low-pass characteristic in $F(s)$. In this case the loop becomes a *type-I PLL*. Since such loops do not provide a way to reduce the residual time error to zero, an additional integration stage can be introduced, in which case the loop becomes a *type-II PLL*. The mathematical analysis is different for each type of PLL, as shown in the following sections.

In either type of PLL the closed-loop gain is determined from classic feedback analysis

$$H(s) = \frac{G(s)}{1 + G(s)}\ . \tag{2}$$

Assuming the output is taken at v_s, the closed-loop input/output transfer function is the ratio of the forward gain to the open-loop gain

$$H_o(s) \equiv \frac{v_s(s)}{\theta_r(s)} = \frac{F_d(s)e^{-Ts}}{1 + G(s)} \ . \tag{3}$$

The acquisition and tracking behavior of a PLL are determined by the loop gain and time constants established by the loop filter. The *capture range* of a PLL is the maximum frequency range over which the PLL will achieve phase-lock, having previously been not in lock. The *tracking range* is the maximum frequency range over which the PLL will remain locked, once phase-lock has been achieved. The capture and tracking ranges of a PLL must be at least as large as the VCO tolerance. In the design discussed here, the capture range is equal to the tracking range and both are greater than the oscillator tolerance since, as shown previously, the adjustment rate R is greater than the tolerance φ by a factor of two.

5.1. Type-I Phase-Lock Loop

A type-I PLL is characterized by a single low-pass filter with transfer function

$$F(s) = \left(1 + \frac{s}{\omega_L}\right)^{-1} ,$$

where ω_L is the corner frequency. Substituting in Equations (1) and (2) and rearranging yields the closed-loop gain

$$H(s) = \left(\frac{s^2}{\omega_n^2} + 2\frac{\zeta}{\omega_n} + 1\right)^{-1} ,$$

where ω_n and ζ are related to the loop gain α and corner frequency ω_L:

$$\omega_n^2 = \alpha\omega_L \quad \text{and} \quad \zeta = \frac{1}{2}\left(\frac{\omega_L}{\alpha}\right)^{½} .$$

For a first-order filter function $\omega_L = \frac{1}{\tau}$, where τ is the time constant of integration. For a critically damped (Butterworth) system ζ should be equal to $2^{-½}$, which implies that the product $2\alpha\tau = 1$. This means that, as the time constant of integration is increased, the loop gain must be decreased proportionally in order to maintain the same dynamic characteristics.

A type-I PLL can be implemented as a sampled-data system using an exponential-average algorithm for the loop filter. Let v_s be the current sample from the clock filter and $\hat{v}_s$ be the current average. Then the new average is

$$\hat{v}_s = \hat{v}_s + \frac{\hat{v}_s - v_s}{\tau} ,$$

where $\tau = \frac{1}{\omega_L}$ is the time constant. The control voltage is then

$$v_c = \alpha\hat{v}_s(s) \ .$$

Up until this point the effects of the clock-filter delay have been neglected. To be effective, the clock filter should contain a number of offset updates, with eight assumed a working number. Then, for a nominal update interval of one minute, the loop delay T is about eight minutes. In order to preserve stability, the time constant τ should be long compared to the loop delay, with one hour assumed a working number, or about $\tau = 2^{-12}$ s. For a critically damped system, this requires the loop gain to be very small $\alpha = 2^{-13}$. However, in a type-I PLL the residual time-offset error is proportional to the oscillator frequency error and inversely proportional to the loop gain. Therefore, at long averaging times and large frequency errors the offset error can become substantive. When time constants in the order of an hour or more are required, a better choice is a type-II PLL, as described in the next section.

5.2. Type-II Phase-Lock Loop

The type-I PLL does not have the capability to compensate for the intrinsic oscillator frequency error, which by previous assumption can be as large as 100 ppm. With a frequency error of this magnitude, it is necessary to provide frequent updates computed by the time-synchronization protocol. For instance, if these adjustments were discontinued for a day, the time error would be accumulate to over 8 s.

In order to minimize the residual time and frequency errors, yet allow relatively long update intervals and averaging times, it is necessary to add another stage of integration, which amounts to the addition of another pole at zero frequency. However, a PLL with two poles at zero frequency is unstable, since the phase characteristic of the Bode plot approaches 180 degrees as the amplitude characteristic passes through unity gain. Instability can be avoided through the addition of a zero in the transfer function, as shown in the following

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

where ω_c is the crossover frequency, ω_z is the corner frequency (required for loop stability) and τ determines the PLL time constant and thus the bandwidth. While this is a first-order function and some improvement in phase noise might be gained from a higher-order function, in practice the improvement is lost due to the effects of the clock-filter delay. Making the following substitutions,

$$\omega_c^2 = \frac{1}{K_f} \quad \text{and} \quad \omega_z = \frac{K_g}{K_f}$$

and rearranging yields

$$G(s) = \frac{\alpha}{s}F(s) = \alpha\left(\frac{1}{K_g\,\tau} + \frac{1}{K_f\,\tau^2 s}\right),$$

which corresponds to a constant term plus an integrating term scaled by the loop gain α and time constant τ. This form is convenient for implementation as a sampled-data system, as described later.

With the parameter values given in Table 2, the Bode plot of the open-loop transfer function $G(s)$ consists of a –12 dB/octave line which intersects the 0-dB baseline at $\omega_c = 2^{-12}$ rad/s, together with a +6 dB/octave line at the corner frequency $\omega_z = 2^{-14}$ rad/s. The damping factor $\zeta = \frac{\omega_c}{2\omega_z} = 2$ suggests the PLL will be stable and have a large phase margin together with a low overshoot.

Variable	Value	Description
μ		update interval
ρ		poll interval
f		frequency error
g		phase error
h		compliance

Table 3. Variables Used in the NTP Local-Clock Model

However, if the clock-filter delay T is not small compared to the loop delay, which is approximately equal to the reciprocal of the crossover frequency, the above analysis becomes unreliable and the loop can become unstable. With the values determined as above, T is ordinarily small enough to be neglected.

Assuming the output is taken at v_S, the closed-loop transfer function can be obtained from Equation (3). If only the relative response is needed and the clock-filter delay can be neglected, this can be written

$$H_O(s) = \frac{1}{1 + G(s)} = s^2 \left(s^2 + \frac{\omega_C^2}{\omega_Z \tau} s + \frac{\omega_C^2}{\tau^2} \right)^{-1} .$$

For some input function $I(s)$ the output function $I(s)H(s)$ can be inverted to find the time response. Using a unit-step input $I(s) = \frac{1}{s}$ and the values determined as above, This yields a PLL risetime of about 52 minutes, a maximum overshoot of about 4.8 percent in about 1.7 hours and a settling time to within one percent of the initial offset in about 8.7 hours.

5.3. The NTP Local-Clock Model

The type-II PLL behavior can also be described as a sampled-data system using a set of recurrence equations. The variables and parameters used in these equations are shown in Tables 2 and 3. Note the use of powers of two, which facilitates implementation using arithmetic shifts and avoids the requirement for a multiply/divide capability.

A capsule overview of the design 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 the values of the frequency error f and phase error g. These variables are computed from the timestamps in messages received at nominal update intervals μ, which are variable from about one to over 17 minutes. As part of update processing, the compliance h is computed and used to adjust τ. Finally, the poll interval ρ for transmitted NTP messages is determined as a fixed multiple of τ.

Updates are numbered from zero, with those in the neighborhood of the ith update shown in Figure 7. All variables are initialized at $i = 0$ to zero, except the time constant $\tau(0) = \tau$, update interval $\mu(0) = K_P$ and compliance $h(0) = K_S$ (from Table 2). After an interval $\mu(i)$ $(i > 0)$ from the previous update the ith update arrives at time $t(i)$ including the time offset $v_S(i)$. Then, after an interval $\mu(i + 1)$ the i+1th update arrives at time $t(i + 1)$ including the time offset $v_S(i + 1)$. When the update $v_S(i)$ is received, the frequency error $f(i + 1)$ and phase error $g(i + 1)$ are computed:

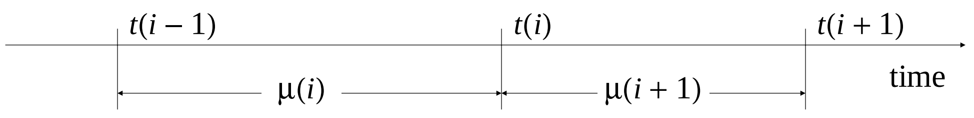

Figure 7. Timing Intervals

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

Note that these computations depend on the value of the time constant $\tau(i)$ and update interval $\mu(i)$ previously computed from the i–1th update. Then, the time constant for the next interval is computed from the current value of the compliance $h(i)$

$$\tau(i+1) = \max[K_s - |\, h(i)|, 1].$$

Next, using the new value of τ, called τ' to avoid confusion, the poll interval is computed

$$\rho(i+1) = K_p \, \tau'.$$

Finally, the compliance $h(i+1)$ is recomputed for use in the i+1th update:

$$h(i+1) = h(i) + \frac{K_t \, \tau' v_s(i) - h(i)}{K_h}.$$

The factor τ' in the above has the effect of adjusting the bandwidth of the PLL as a function of compliance. When the compliance 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 frequency estimate. On the other hand, if the compliance becomes high due to greatly increased jitter or a systematic frequency offset, τ' is decreased and the bandwidth is increased. In this mode the PLL is most adaptive to transients which can occur due to reboot of the system or a major timing error. In order to maintain optimum stability, the poll interval ρ is varied directly with τ.

A model suitable for simulation and parameter refinement can be constructed from the above recurrence relations. It is convenient to set the temporary variable $a = g(i+1)$. At each adjustment interval σ the quantity $\frac{a}{K_g} + \frac{f(i+1)}{K_f}$ is added to the local-clock time and the quantity $\frac{a}{K_g}$ is subtracted from a. For convenience, let n be the greatest integer in $\frac{\mu(i)}{\sigma}$; that is, the number of adjustments that occur in the ith interval. Thus, at the end of the ith interval just before the i+1th update, the VCO control voltage is:

$$v_c(i+1) = v_c(i) + [1 - (1 - \frac{1}{K_g})^n] \, g(i+1) + \frac{n}{K_f} f(i+1).$$

Detailed simulation of the NTP PLL with the values specified in Tables 2 and 3 and the clock filter described in the NTP specification results in the following characteristics: For a 100-ms time change the loop reaches zero error in 39 minutes, overshoots 7 ms at 54 minutes and settles to less than 1 ms in about six hours. For a 50-ppm frequency change the loop reaches 1 ppm in about 16 hours and 0.1 ppm in about 26 hours. When the magnitude of correction exceeds a few milliseconds or a few ppm for more than a few updates, the compliance begins to increase, which causes the loop time constant and update interval to decrease. When the magnitude of correction falls below about 0.1 ppm for a few hours, the compliance begins to decrease, which causes the loop time constant and update interval to increase. The effect is to provide a broad capture range exceeding 8 s per day together with the capability to resolve oscillator frequency well below 1 ms per day. These characteristics are appropriate for typical crystal-controlled oscillators with or without temperature compensation or oven control.

5.4. Bandwidth Management

A key feature of the type-II PLL design is its capability to compensate for the intrinsic frequency error of the local oscillator. This requires an initial period of adaptation in order to refine the frequency estimate. The τ parameter determines the PLL time constant and thus the loop bandwidth, which is inversely proportional to τ. It also determines the poll interval, which is a fixed multiple of τ. When operated with a relatively large bandwidth (small τ), as in the analysis above, the PLL adapts quickly to changes in the input reference signal, but has poor long term stability. Thus, it is possible to accumulate substantial errors if the system is deprived of its reference signal for an extended period. When operated with a relatively small bandwidth (large τ), the PLL adapts slowly to changes in the input reference signal. Assuming the frequency estimate has stabilized, it is possible for the PLL to coast for an extended period without external corrections and without accumulating significant error.

There are several tradeoffs in providing an adaptive-bandwidth capability. Long averaging times (low bandwidth) reduces timing jitter, improves frequency stability and allows infrequent update messages. However, the low bandwidth PLL converges very slowly to changes in the operating environment. Short averaging times (high bandwidth) speeds convergence time, but requires frequent update messages. An often overlooked advantage of the adaptive-bandwidth model is the freedom from intricate configuration management to match the PLL parameters to the particular ambient oscillator stability and statistical network delay variations. The methods described in the previous sections do this automatically.

With the adaptive-bandwidth approach it is necessary to compute each value of τ based on the measured values of offset, delay and dispersion, as produced by the time-synchronization protocol itself. The traditional way of doing this in precision timekeeping systems based on cesium clocks is to relate τ to the Allan variance, which is defined as the mean of the first-order differences of sequential samples measured during a specified interval τ. In the NTP local-clock model this is called compliance h and is approximated on a continuous basis by exponentially averaging the first-order differences of the offset samples using an empirically determined averaging constant. Using somewhat ad-hoc mapping functions determined from simulation and experience, the compliance is manipulated to produce the loop time constant τ and poll interval ρ.

Bandwidth management is done using the α and τ parameters shown in Table 2. The α parameter is determined by the tolerance of the local oscillator and the maximum jitter requirements of the timekeeping system. This parameter is usually an architecture constant and fixed during system operation. In the implementation model described previously, the reciprocal of α, called the adjustment interval σ, determines the time between adjustments of the local clock, and thus the value of α. The value of σ can be determined using the methods of Section 4.3.

The local-clock model as described is appropriate for a convergence interval of about one hour. In some applications much more rapid convergence times are required. In such cases the value of τ can be materially reduced; however, in order to maintain stability, the adjustment interval σ must be small compared to the update interval μ. In the design above where the convergence time is about one hour, with $\mu = 64$ s, a value of $\sigma = 1$ s is conservative. However, if the value of σ is changed, the parameters K_f and K_g must be adjusted in order to retain the same transient behavior; in particular, the same ω_c and ω_z. Since α varies as the reciprocal of σ, if σ is changed to something other than 1, as in Table 2, it is necessary to divide both K_f and K_g by σ..

6. Clock-Combining Algorithms

A common problem in synchronization subnets is systematic time offsets resulting from asymmetric transmission paths, where the transmission delay in one direction is substantially different from the delay in the other. These errors can range from microseconds on high speed ring networks to large fractions of a second on satellite/landline paths. It has been found experimentally that these errors can be considerably reduced by combining the apparent offsets of a number of peer servers to produce a more accurate timescale than any of its contributors. Following is a description of a method similar to that used by national standards laboratories to determine a synthetic timescale from an ensemble of cesium clocks [ALL74b].combining method applicable to the general problem.

Consider the time offsets of a number of real clocks connected by real networks. A histogram of offsets relative to the standard timescale will appear as a system of bell-shaped curves, but with some further away from nominal zero than others. The bells will normally be scattered over the offset space, more or less close to each other, with some overlapping and some not. In addition, due to intrinsic frequency offsets, successive histograms will reveal that the bells precess slowly with time relative to the standard timescale. The problem is to estimate the true offset relative to the standard timescale using information collected routinely between the clocks.

The notation $< x >$ in the following designates the (infinite) time average of x, which is usually approximated by an exponential time average, while the notation $\hat{x}$ designates an estimator for x and $|x|$ designates its absolute value. A composite timescale can be determined from a sequence of offsets measured between the n clocks of an ensemble at nominal intervals τ. Let $R_i(t_0)$ be the frequency and $T_i(t)$ be the time of the ith clock at epoch t_0 relative to the standard timescale. Recall that the time difference or offset between clock i and clock j is $T_{ij}(t) \equiv T_i(t) - T_j(t)$. Then, an estimator for T_i computed at epoch t_0 for epoch $t_0 + \tau$ is

$$\hat{T}_i(t_0 + \tau) = \hat{R}_i(t_0)\tau + T_i(t_0) \ ,$$

neglecting second-order terms. Consider a set of n independent offset measurements made between the clocks at epoch $t_0 + \tau$. The offset of clock i relative to clock j at that epoch is

$$T_{ij}(t_0 + \tau) = T_i(t_0 + \tau) - T_j(t_0 + \tau) \ .$$

Let $w_i(\tau)$ be a previously determined weight factor associated with the ith clock for the nominal interval τ chosen such that

$$\sum_{i=1}^{n} w_i(\tau) = 1 \ .$$

The basis for new estimates at epoch $t_0 + \tau$ is then

$$T_j(t_0 + \tau) = \sum_{i=1}^{n} w_i(\tau)[\hat{T}_i(t_0 + \tau) + T_{ji}(t_0 + \tau)] .$$

That is, the apparent time indicated by the jth clock is a weighted average of the estimated time of each clock at epoch $t_0 + \tau$ plus the offset measured between the jth clock and that clock at epoch $t_0 + \tau$.

An intuitive grasp of the behavior of this algorithm can be gained with the aid of a few examples. For instance, if $w_i(\tau)$ is unity for the ith clock and zero for all others, the apparent time for each of the other clocks is simply the estimated time $\hat{T}_i(t_0 + \tau)$. If $w_i(\tau)$ is zero for the ith clock, that clock can never affect any other clock and its apparent time is determined entirely from the other clocks. If $w_i(\tau) = 1/n$ for all i, the apparent time of the ith clock is equal to the average of the time estimates computed at t_0 plus the average of the time offsets measured to all other clocks. Finally, in a system with two clocks and $w_i(\tau) = ½$ for each, and if the estimated time at epoch $t_0 + \tau$ is fast by 1 s for one clock and slow by 1 s for the other, the apparent time for both clocks will coincide with the standard timescale.

In order to establish a basis for the next interval τ, it is necessary to update the frequency estimate $\hat{R}_i(t_0 + \tau)$ and weight factor $w_i(\tau)$. The average frequency assumed for the ith clock during the previous interval τ is simply the difference between the times at the beginning and end of the interval divided by τ. A good estimator for $R_i(t_0 + \tau)$ has been found to be the exponential average of these differences, which is given by [ALL74a]

$$\hat{R}_i(t_0 + \tau) = \hat{R}_i(t_0) + \alpha_i[\hat{R}_i(t_0) - \frac{T_i(t_0 + \tau) - T_i(t_0)}{\tau}] ,$$

where α_i is an experimentally determined weight factor which depends on the estimated stability of the ith clock. In order to calculate the weight factor $w_i(\tau)$, it is necessary to determine the expected error $\varepsilon_i(\tau)$ for each clock. An estimate of the magnitude of the unbiased error of the ith clock accumulated over the nominal interval τ is [ALL74a]

$$\varepsilon_i(\tau) = |\hat{T}_i(t_0 + \tau) - T_i(t_0 + \tau)| + \frac{0.8 < \varepsilon_e^2(\tau) >}{< \varepsilon_i^2(\tau) >^{1/2}} ,$$

where $\varepsilon_i(\tau)$ and $\varepsilon_e(\tau)$ are the accumulated error of the ith clock and entire clock ensemble, respectively. The accumulated error of the entire ensemble is

$$< \varepsilon_e^2(\tau) > = \left[\sum_{i=1}^{n} \frac{1}{< \varepsilon_i^2(\tau) >} \right]^{-1} .$$

Finally, the weight factor for the ith clock is calculated as

$$w_i(\tau) = \frac{< \varepsilon_e^2(\tau) >}{< \varepsilon_i^2(\tau) >} .$$

When all estimators and weight factors have been updated, the origin of the estimation interval is shifted and the new value of t_0 becomes the old value of $t_0 + \tau$.

While not entering directly into the above calculations, it is useful to estimate the frequency error, since the ensemble clocks can be located some distance from each other and become isolated for some time due to subnet failures. The frequency-offset error in R_i is equivalent to the fractional frequency y_i,

$$y_i = \frac{\nu_i - \nu_I}{\nu_I} ,$$

where the frequency ν_i is measured on the ith timescale and the frequency ν_I is measured on the standard or ensemble timescale. Temporarily dropping the subscript i for clarity, consider a sequence of N independent frequency-offset samples $y(j)$ ($j = 1, 2, \ldots, N$) where the interval between samples is uniform and equal to T. Let τ be the nominal interval over which these samples are averaged. The Allan variance $\sigma_y^2(N, T, \tau)$ [ALL74a] is defined as

$$<\sigma_y^2(N, T, \tau)> = < \frac{1}{N-1}\left[\sum_{j=1}^{N} y(j)^2 - \frac{1}{N}\left(\sum_{j=1}^{N} y(j)\right)^2\right]> ,$$

A particularly useful formulation is $N = 2$ and $T = \tau$:

$$<\sigma_y^2(N = 2, T = \tau, \tau)> \equiv \sigma_y^2(\tau) = < \frac{[y(j+1) - y(j)]^2}{2} > ,$$

so that

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

The Allan variance is particularly useful when comparing the intrinsic stability of the oscillators used in different local clocks and timecode receivers. The oscillators are uncoupled from all synchronization sources and left to flywheel for periods up to a month. At intervals of a minute or so the oscillator frequency is measured relative to a precision standard, such as a cesium oscillator or timing receiver. Figure 8 is a plot on log-log scales of the Allan variance for a Spectracom 8170 WWVB timecode receiver measured at the 1-pps output compared to a Global Positioning System timing receiver. The WWVB receiver uses an uncompensated crystal oscillator disciplined to the WWVB signal at 60 kHz. The 1-pps output is extracted from the timecode modulation using a counter with a resolution of 100 µs. The resulting short-term stability, averaged over periods up to about two hours is constant at about 5 parts in 10^{-9} (ppb). The effects of the oscillator discipline is clearly evident in the fall of the curve at larger averaging periods. The flattened portion of the curve clearly shows the improvement in short-term frequency stabilization provided by the PLL and low-pass filter.

7. Analysis of Errors

This section contains an in-depth analysis of time and frequency errors due to systematic and random phenomena. Errors originate in the local clocks and transmission paths in the synchronization subnet

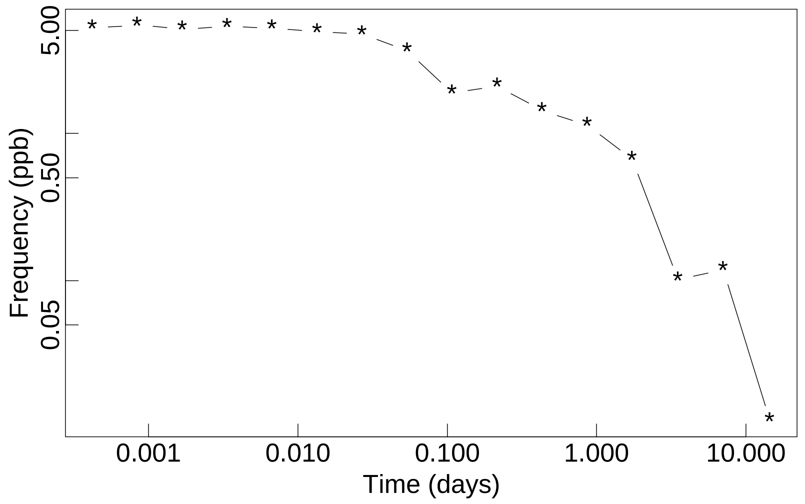

Figure 8. Allan Variance of WWVB Receiver

and in the service used to disseminate standard time (UTC) to the primary servers. Errors originating in the local clock are due to the clock-counter resolution and oscillator frequency uncertainty. In the models considered in this report, the time-synchronization protocol operates to distribute timing information over a hierarchically structured subnet spanning all participants and rooted at the primary (stratum-1) time server. Therefore, errors accumulate at each level in the hierarchy due to local-clock errors and statistical transmission delays. In this report radio propagation phenomena and receiver inaccuracies at the primary server will be neglected. See [MIL90] for further discussion on these issues.

It is necessary at the outset to identify two types of error bounds. *Absolute error bounds* establish the worst-case errors in a subnet where all time servers are operating correctly and synchronized to UTC. As proved in this section, absolute bounds establish the correctness of a clock reading, since UTC time is always contained within the interval defined by the absolute bounds and any reading outside these bounds must be due to a faulty clock. *Statistical error bounds* establish the maximum expected errors as a function of measured statistical variations. They include systematic quantities due to the local-clock precision and tolerance, as well as random quantities due to sample variance. Knowledge of statistical bounds is necessary in order to calibrate the performance in actual operation and mitigate among the several synchronization sources usually available at each level in the subnet hierarchy.

The notation $< x >$ in the following designates the (infinite) time average of x, which is usually approximated by an exponential time average, while the notation $\hat{x}$ designates an estimator for x and $|x|$ designates its absolute value. The notation $w = [u, v]$ describes the closed interval in which u is the lower limit and v the upper limit. Thus, $u = \min(w) \le v = \max(w)$, and for scalar a, $a + w = [a + u, a + v]$ and $aw = [au, av]$. Table 4 shows a summary of other notation used in the analysis. The lower-case Greek letters θ, δ and ε are used to designate measurement data for the local clock relative to a peer server, while the upper-case Greek letters Θ, Δ and E are used to designate measurement data for the local clock relative to the primary server. Exceptions will be noted as they arise.

Variable	Description
r	reading error
ρ	max reading error
f	frequency error
φ	max frequency error
θ, Θ	clock offset
δ, Δ	roundtrip delay
ε, E	error/dispersion
t	time
τ	time interval
T	NTP timestamp
s	clock divider increment
f_c	clock oscillator frequency

Table 4. Notation Used in Error Analysis

7.1. Clock Modelling

The standard second (1 s) is defined as “9,192,631,770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the cesium-133 atom” [ALL74b], which implies a granularity of about $1.1\text{x}10^{-10}$ s. Other intervals can be determined as rational multiples of 1 s. Ordinary computer clocks have resolutions much larger than $1.1\text{x}10^{-10}$ s, so the inherent error in resolving time relative to the standard second can be neglected.

Let $T(t)$ be the time displayed by a clock at epoch t relative to the standard timescale:

$$T(t) = \tfrac{1}{2}D(t_0)[t - t_0]^2 + R(t_0)[t - t_0] + T(t_0) + x(t) ,$$

where $D(t_0)$ is the drift (first derivative of frequency) per unit time, $R(t_0)$ the frequency and $T(t_0)$ the time at some previous epoch t_0. In the usual stationary model these quantities are assumed constant or changing slowly with epoch. The random nature of the clock is characterized by $x(t)$, which represents the error relative to the standard timescale.

In the usual analysis the second-order term $D(t_0)$ is ignored and the noise term $x(t)$ modelled as a normal distribution with predictable spectral density or autocorrelation function. The probability density function (pdf) of $x(t)$ usually appears as a bell-shaped curve centered near zero. The width and general shape of the curve are determined by the oscillator jitter and wander characteristics, as well as the measurement system and its transmission paths. Beginning at epoch t_0 the bell creeps either to the left or right, depending on the value of $R(t_0)$ and accelerates depending on the value of $D(t_0)$.

In this analysis the local clock is represented by a counter/divider which increments at intervals of s seconds and is driven by an oscillator which operates at frequency $f_c = \frac{n}{s}$ for some integer n. A timestamp $T(t)$ is determined by reading the clock at an arbitrary time t (the argument t will be usually omitted for conciseness). Reading the clock truncates some number of low-order bits, which introduces a reading error represented by the random variable r bounded by the interval $[-\rho, 0]$, where ρ depends on the *precision*, or interval between clock ticks. Since the intervals between reading the same clock are almost always independent of and much larger than s, successive readings can be considered independent and identically distributed. The frequency error of the clock oscillator is represented by the random variable f bounded by the interval $[-\varphi, \varphi]$, where φ represents the tolerance of the oscillator throughout its service life. While f for a particular clock is a random variable with respect to the population of all clocks, for any one clock it ordinarily changes only slowly with time and can usually be assumed a constant for that clock. Thus, the error in a timestamp can be represented by the random variable

$$\varepsilon(\tau) \equiv t - T(t) = r + f\tau ,$$

where t represents a clock reading, τ represents the time interval since this reading and minor approximations inherent in the measurement of τ are neglected.

In order to assess the nature and expected magnitude of timestamp errors and the calculations based on them, it is useful to examine the characteristics of the pdfs $p_r(x)$ and $p_f(x)$ for r and f respectively. Assuming the clock reading and counting processes are independent, $p_r(x)$ is uniform over the interval $[-\rho, 0]$ and zero elsewhere. With conventional manufacturing processes and temperature variations $p_f(x)$ can be approximated by a truncated, zero-mean Gaussian distribution with standard deviation σ over the interval $[-\varphi, \varphi]$ and zero elsewhere. In conventional manufacturing processes σ is maneuvered so that the fraction of samples rejected outside the interval $[-\varphi, \varphi]$ is acceptable. The pdf for the total timestamp error is thus the sum of the r and f contributions, computed as the convolution

$$p_{\varepsilon(\tau)}(x) = \tau \int_{-\infty}^{\infty} p_r(u) p_f(x-u)\,du ,$$

which appears as a bell-shaped curve, expanding with τ, symmetric about $-\frac{\rho}{2}$ and bounded by the interval

$$[\min(r) + \min(f\tau), \max(r) + \max(f\tau)] = [-\rho - \varphi\tau, \varphi\tau] .$$

Since f changes only slowly over time for any single clock, the error is bounded by the interval

$$\varepsilon \equiv [\min(r) + f\tau, \max(r) + f\tau] = [-\rho, 0] + f\tau ,$$

which should not be confused with the random variable $\varepsilon(\tau)$, which represents the error itself. In the following development subscripts will be used on various quantities to indicate to which entity or timestamp the quantity applies. Occasionally, ε will be used to designate an absolute maximum error, rather than the interval, but the distinction will be clear from context.

7.2. Measurement Errors

The roundtrip delay and clock offset between two peers A and B are determined by a procedure in which timestamps are exchanged via the subnet paths between them. The procedure involves the four most recent timestamps numbered as shown in Figure 3, where the θ_0 represents the true offset of peer B relative to peer A. The T_1 and T_4 timestamps are determined relative to the A clock, while the T_2 and T_3 timestamps are determined relative to the B clock. The measured delay δ and clock θ of B relative to A are given by

$$\delta = (T_4 - T_1) - (T_3 - T_2) \quad \text{and} \quad \theta = \frac{(T_2 - T_1) + (T_3 - T_4)}{2} . \tag{4}$$

The errors inherent in determining the timestamps T_1, T_2, T_3 and T_4 are, respectively,

$$\varepsilon_1 = [-\rho_A, 0],\ \varepsilon_2 = [-\rho_B, 0],\ \varepsilon_3 = [-\rho_B, 0] + f_B(T_3 - T_2),\ \varepsilon_4 = [-\rho_A, 0] + f_A(T_4 - T_1) .$$

For specific peers A and B, where f_A and f_B can be considered constants, the interval containing the maximum error inherent in determining δ is given by

$$[\min(\varepsilon_4) - \max(\varepsilon_1) - \max(\varepsilon_3) + \min(\varepsilon_2), \max(\varepsilon_4) - \min(\varepsilon_1) - \min(\varepsilon_3) + \max(\varepsilon_2)]$$
$$= [-\rho_A - \rho_B, \rho_A + \rho_B] + f_A(T_4 - T_1) - f_B(T_3 - T_2) .$$

In a precision local clock the residual frequency errors f_A and f_B are minimized through the use of a type-II phase-lock loop (PLL). Under most conditions these errors will be small and can be ignored. The pdf for the remaining errors is symmetric, so that $\hat{\delta} = \langle\delta\rangle$ is an unbiased maximum-likelihood estimator for the true delay, independent of the particular values of ρ_A and ρ_B.

However, in order to reliably bound the errors under all conditions of component variation and operational regimes, the design of the PLL and its oscillator tolerance must be controlled so that it is not possible under any circumstances for f_A or f_B to exceed the bounds $[-\varphi_A, \varphi_A]$ or $[-\varphi_B, \varphi_B]$, respectively. Setting $\rho = \max(\rho_A, \rho_B)$ for convenience, the absolute maximum error ε_δ inherent in determining delay δ is given by

$$\varepsilon_\delta \equiv \rho + \varphi_A(T_4 - T_1) + \varphi_B(T_3 - T_2) ,$$

neglecting residuals.

As in the case for δ, where f_A and f_B can be considered constants, the interval containing the maximum error inherent in determining θ is given by

$$\frac{[\min(\varepsilon_2) - \max(\varepsilon_1) + \min(\varepsilon_3) - \max(\varepsilon_4), \max(\varepsilon_2) - \min(\varepsilon_1) + \max(\varepsilon_3) - \min(\varepsilon_4)]}{2}$$
$$= [-\rho_B, \rho_A] + \frac{f_B(T_3 - T_2) - f_A(T_4 - T_1)}{2} .$$

Under most conditions the errors due to f_A and f_B will be small and can be ignored. If $\rho_A = \rho_B = \rho$; that is, if both the A and B clocks have the same resolution, the pdf for the remaining errors is symmetric, so that $\hat{\theta} = \langle\theta\rangle$ is an unbiased maximum-likelihood estimator for the true offset θ_0, independent of the particular value of ρ. If $\rho_A \neq \rho_B$, $\langle\theta\rangle$ is not an unbiased estimator; however, the bias error is in the order of

$$\frac{\rho_A - \rho_B}{2} .$$

and can usually be neglected.

Again setting $\rho = \max(\rho_A, \rho_B)$ for convenience, the absolute maximum error ε_θ inherent in determining offset θ is given by

$$\varepsilon_\theta \equiv \frac{\rho + \varphi_A(T_4 - T_1) + \varphi_B(T_3 - T_2)}{2} .$$

7.3. Network Errors

In practice, errors due to statistical network delays usually dominate the error budget. In general, it is not possible to characterize network delays as a stationary random process, since network queues can grow and shrink in chaotic fashion and packet arrivals are frequently bursty. However, it is a simple exercise to calculate absolute error bounds as a function of measured delay. Let $T_2 - T_1 = a$ and $T_3 - T_4 = b$. Then, from (4)

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

The true offset of B relative to A is called θ_0 in Figure 3. Let x denote the actual delay between the departure of a message from A and its arrival at B. Therefore, $x + \theta_0 = T_2 - T_1 \equiv a$. Since x must be positive in our universe, $x = a - \theta_0 \geq 0$, which requires $\theta_0 \leq a$. A similar argument requires that $b \leq \theta_0$, so surely $b \leq \theta_0 \leq a$. This inequality can also be expressed

$$b = \frac{a + b}{2} - \frac{a - b}{2} \leq \theta_0 \leq \frac{a + b}{2} + \frac{a - b}{2} = a ,$$

which is equivalent to

$$\theta - \frac{\delta}{2} \leq \theta_0 \leq \theta + \frac{\delta}{2} .$$

In the previous section bounds on delay and offset errors were determined. This inequality can also be written

$$\theta - \varepsilon_\theta - \frac{\delta + \varepsilon_\delta}{2} \leq \theta_0 \leq \theta + \varepsilon_\theta + \frac{\delta + \varepsilon_\delta}{2} ,$$

where ε_θ is the maximum offset error and ε_δ is the maximum delay error derived previously. The quantity

$$\varepsilon = \varepsilon_\theta + \frac{\varepsilon_\delta}{2} = \rho + \varphi_A(T_4 - T_1) + \varphi_B(T_3 - T_2) ,$$

called the peer dispersion, defines the maximum statistical error in the inequality. Thus, the maximum absolute error must be bounded on the interval

$$[\theta - \frac{\delta}{2} - \varepsilon, \theta + \frac{\delta}{2} + \varepsilon] . \tag{5}$$

By construction, if the peer server is correctly synchronized to UTC with zero error; then, with respect to the local clock, the true value of UTC must lie somewhere in this interval.

7.4. Inherited Errors

Since the synchronization subnet is structured as a tree rooted at the primary server, some time servers are farther from the primary server than others. Errors accumulate at each subnet server and on the network transmission paths between them. The following discussion establishes how errors inherent in the time-transfer process accumulate within the subnet and contribute to the error budget at each level.

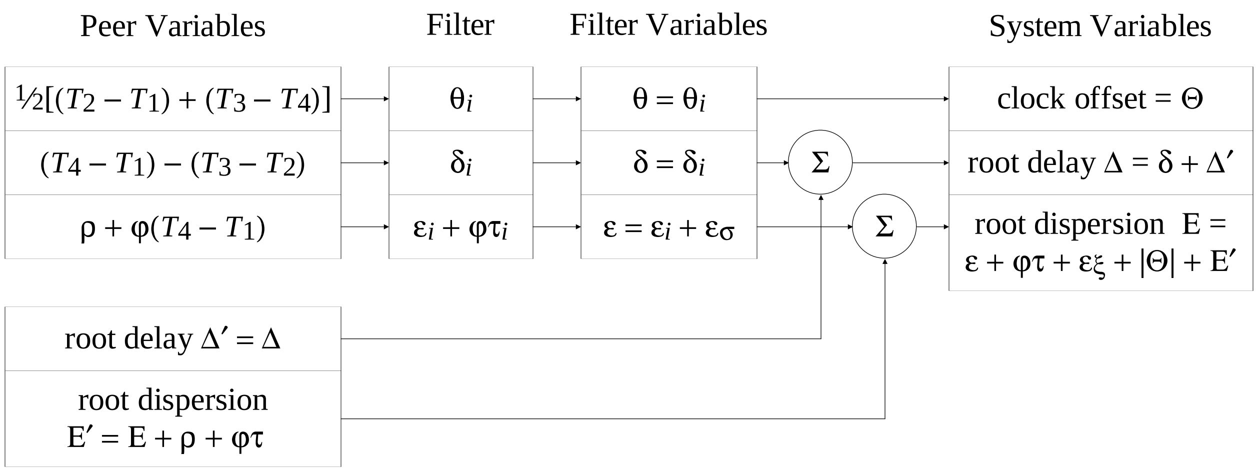

Figure 9. Error Accumulations

The time servers at each level calculate the local clock offset Θ, root delay Δ and root dispersion E relative to the primary server. As described previously, the update message includes timestamps from which the offset θ, delay δ and dispersion ε of the local clock relative to a peer server are computed. In addition, the precision ρ, root delay Δ and root dispersion E of the peer server relative to the primary server are included in the message. The dispersion ε includes the following contributions:

1. Each time the local clock is read a reading error is incurred due to the finite granularity or precision of the implementation. This is called the measurement dispersion ρ.
2. Once a time offset is determined, an error due to frequency offset accumulates with time. This is called the skew dispersion φτ, where φ is the maximum frequency error and τ is the interval since the dispersion was last updated.
3 When a series of offsets are determined at regular intervals and accumulated in a window of samples, as in the NTP clock-filter algorithm, the (estimated) additional error due to the offset sample variance is called the filter dispersion ε_σ.
4. When a number of peers are considered for synchronization and two or more are determined to be correctly synchronized to a primary server, as in the NTP clock-selection algorithm, the (estimated) additional error due to the combined offset sample variance is called the selection dispersion ε_ξ.

Figure 9 shows how these errors accumulate in the ordinary course of processing. The peer variables include the timestamps and most recent root delay and root dispersion computed by the peer. From the four most recent timestamps T_1, T_2, T_3 and T_4 the offset and delay of the local clock relative to the peer server are calculated. Included in the message are the root delay Δ′ and root dispersion E′ of the peer itself; however, before sending, the peer adds the measurement dispersion ρ and skew dispersion φτ, where these quantities are determined by the peer and τ is the interval determined by the peer since its clock was last updated.

The clock-filter procedure saves the several most recent samples θ_i and δ_i in a shift register. The quantities ρ and φ characterize the local clock maximum reading error and maximum frequency error, respectively. Upon arrival each sample is assigned a dispersion $\varepsilon_i = \rho + \varphi(T_4 - T_1)$, which represents the dispersion accumulated since reading the T_1 timestamp. When a new sample arrives, all samples in the filter are updated with the skew dispersion $\varphi\tau_i$, where τ_i is the interval since the last sample arrived. The clock-filter algorithm determines the selected offset θ, together with the associated delay δ and filter dispersion ε_σ. It adds ε_σ to the associated sample dispersion ε_i to form the peer dispersion ε.

The clock-selection procedure selects a single peer to become the synchronization source. The operation of the algorithm determines the final offset Θ, delay Δ and dispersion E relative to the primary server, as shown in Figure 9. Note the inclusion of the selected peer dispersion and skew accumulation since that dispersion was last updated, as well as the select dispersion ε_ξ computed by the clock-select algorithm itself. In order to preserve overall synchronization subnet stability, the final offset Θ is in fact determined from the offset of the local clock relative to the peer server, rather than the primary server. Finally, note that the variables Δ' and E' are in fact determined from the latest message received, not at the precise time the offset selected by the clock-filter algorithm was determined. Minor errors arising due to these simplifications are ignored. Thus, the total dispersion accumulation relative to the primary server is

$$\mathrm{E} = \varepsilon + \varphi\tau + \varepsilon_\xi + |\Theta| + \mathrm{E}' ,$$

where τ is the time since the local variables were last updated and $|\Theta|$ is the absolute error in setting the local clock.

The synchronization subnet topology is that of a tree rooted at the primary server. Ordinarily, the offset Θ_0, root delay Δ_0 and root dispersion E_0 at the primary server are all zero, although E_0 can be manipulated to reflect the quality of the radio clock or other source of standard time. These three values are all additive; that is, if Θ_i, Δ_i and E_i represent the values at peer i relative to the primary server, the values

$$\Theta_j(t) \equiv \Theta_i + \theta_j(t) , \quad \Delta_j(t) \equiv \Delta_i + \delta_j , \quad \mathrm{E}_j(t) \equiv \mathrm{E}_i + \varepsilon_i + \varepsilon_j(t) ,$$

represent the offset, delay and dispersion of peer j at time t. The time dependence of $\theta_j(t)$ and $\varepsilon_j(t)$ represents the local-clock correction and dispersion accumulated since the last update was received from peer i, while the term ε_i represents the dispersion accumulated by peer i from the time its clock was last set until the latest update was sent to peer j.

Since there is an unbroken path from every time server to the primary server, the statistical error bound is the sum of the maximum absolute errors and estimated statistical errors accumulated from the primary server. According to the above development the statistical error bound is in fact the root dispersion E. From (5) and the above development, the absolute error bound, also called the *synchronization distance,* is

$$\Lambda \equiv \mathrm{E} + \frac{\Delta}{2} .$$

The synchronization distance is the metric used by the NTP clock-selection algorithm to organize the synchronization subnet, which is in fact a minimum-distance spanning tree rooted at the primary server. In this manner the algorithm operates to minimize the overall absolute error at each level in the subnet.

8. Summary

This report has presented an in-depth analysis of issues important to achieve accurate, stable and reliable time synchronization in a computer network. These issues include the design of the synchronization protocol, the local clock, and the algorithms used to filter, select and combine the reading of possibly many peer servers. The various design parameters are established using an analysis of the local clock modelled as a linear feedback loop. The loop includes a disciplined oscillator, which is realized as an adaptive-parameter, phase-locked loop. The behavior of this loop can be controlled automatically to adjust to oscillators of varying stability and network paths of widely varying delay.

In order to reliably quantify the correctness and accuracy of a local clock, it is necessary to establish bounds on the maximum error of the clock, as well as the errors expected in ordinary system operation. The error analysis in this report constructs an exhaustive error budget starting from the basic process of reading the clock, continuing through the process of exchanging timestamps between the peers of the hierarchical synchronization subnet and concluding with the process of combining the information from possibly many peers and establishing error bounds for the next hierarchical level.

9. References

[ALL74a] 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.

[ALL74b] Allan, D.W., J.E. Gray and H.E. Machlan. The National Bureau of Standards atomic time scale: generation, stability, accuracy and accessibility. In: Blair, B.E. (Ed.). *Time and Frequency Theory and Fundamentals*. National Bureau of Standards Monograph 140, U.S. Department of Commerce, 1974, 205-231.

[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.

[JON83] Jones, R.H., and P.V. Tryon. Estimating time from atomic clocks. *J. Research of the National Bureau of Standards 88, 1* (January-February 1983), 17-24.

[LAM90] Lamport, L. Concurrent reading and writing of clocks. *ACM Trans. Computing Systems 8,* 4 (November 1990), 305-310.

[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.

[MIL88] Mills, D.L. The Fuzzball. *Proc. ACM SIGCOMM 88 Symposium* (Palo Alto, CA, August 1988), 115-122.

[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.

[MIL92] 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.

[NIS90] *NIST Time and Frequency Dissemination Services.* NBS Special Publication 432 (Revised 1990), National Institute of Science and Technology, U.S. Department of Commerce, 1990.

[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.

[SCH90] Schneider, F.B. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Computing Surveys 22, 4* (December 1990), 299-320.

[SMI86] Smith, J. *Modern Communications Circuits.* McGraw-Hill, New York, NY, 1986.

[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.

[WIL90] Wilcox, D.R. Backplane bus distributed realtime clock synchronization. Technical Report 1400, Naval Ocean Systems Center, December 1990, 52 pp.

[WIL91] Wilcox, D.R. Local area network distributed realtime clock synchronization. Technical Report 1466, Naval Ocean Systems Center, November 1991, 77 pp.