Network Working Group
Request for Comments: 1129

D. L. Mills
University of Delaware

October 1989

Internet Time Synchronization:
the Network Time Protocol

Abstract

This memo describes the Network Time Protocol (NTP) designed to distribute time information in a large, diverse internet system operating at speeds from mundane to lightwave. It uses a returnable-time architecture in which a distributed subnet of time servers operating in a self-organizing, hierarchical, master-slave configuration synchronizes local clocks within the subnet and to national time standards via wire or radio. The servers can also redistribute time information within a network via local routing algorithms and time daemons.

The architectures, algorithms and protocols which have evolved to NTP over several years of implementation and refinement are described in this paper. The synchronization subnet which has been in regular operation in the Internet for the last several years is described along with performance data which shows that timekeeping accuracy throughout most portions of the Internet can be ordinarily maintained to within a few tens of milliseconds, even in cases of failure or disruption of clocks, time servers or networks.

This memo describes the Network Time Protocol, which is specified as an Internet Standard in RFC-1119. Distribution of this memo is unlimited.

Keywords: network clock synchronization, standard time distribution, fault-tolerant architecture, maximum-likelihood estimation, disciplined oscillator, Internet protocol.

Table of Contents

1. Introduction . . . 1
1.1. Performance Requirements . . . 2
1.2. Discussion of Approaches . . . 3
2. Time Standards and Distribution . . . 4
3. Network Time Protocol . . . 5
3.1. Implementation Model . . . 7
3.1.1. Modes of Operation . . . 7
3.1.2. Data Formats . . . 8
3.1.3. State Variables . . . 9
3.2. Procedures . . . 10
3.3. Robustness Issues . . . 11
4. Sample Processing and Selection Operations . . . 13
4.1. Data Filtering . . . 13
4.2. Peer Selection . . . 16
5. Local Clock Design . . . 18
6. NTP in the Internet System . . . 21
6.1. Time Servers . . . 21
6.2. Synchronization Subnet . . . 21
6.3. Performance Analysis . . . 22
7. Future Directions . . . 24
8. References . . . 25

List of Figures

Figure 1. Implementation Model . . . 6
Figure 2. NTP Packet Header . . . 9
Figure 3. Calculating Delay and Offset . . . 11
Figure 4. Offset vs Delay . . . 14
Figure 5. Phase-Lock Loop Model . . . 18
Figure 6. Raw Offsets . . . 22
Figure 7. Filtered Offsets . . . 23
Figure 8. Error Distribution . . . 23

List of Tables

Table 1. Outlyer Selection Procedure . . . 17

1. Introduction

How do hosts and gateways in a large, dispersed networking community know what time it is? How accurate are their clocks? In a recent survey involving 94,260 hosts and gateways of the Internet system [MIL89b], 20,758 provided local time using three time-transfer protocols. About half of the replies had errors greater than two minutes, while ten percent had errors greater than four hours. A few had errors over two weeks. Most local clocks are set by eyeball-and-wristwatch to within a minute or two and rarely checked after that. Many of these are maintained by some sort of battery-backed clock/calender device using a room-temperature crystal oscillator that may drift as much as a second per day and can go for weeks between manual corrections. For many applications, especially distributed internet applications, much greater accuracy and reliability is required.

Accurate, reliable time is necessary for financial and legal transactions, stocks and bonds trading, transportation reservation, dispatch and control, and many other applications involving widely distributed resources. These applications often require multiple, concurrent, real-time access to databases distributed over an internet of many networks. Internet time synchronization is necessary for reliable caching, archiving and transaction sequencing. Even the management functions of the internet and its component networks need accurate time in order to manage control activation, monitoring coordination and significant event timestamping. Coordinated network time is extremely useful when searching for elusive bugs and transients in day-to-day operations.

This paper presents an overview of the architecture, protocol and algorithms of the Network Time Protocol (NTP) used in the Internet system to synchronize clocks and coordinate time distribution. The Internet consists of over 100,000 hosts on about 800 packet-switching networks interconnected by a similar number of gateways. In this paper the capitalized *Internet* refers to this particular system, while the uncapitalized *internet* refers to any generic system of multiple networks interconnected by gateways. While the backbone networks and gateways are carefully engineered for good service, operating speeds and service reliability vary considerably throughout the Internet. This places severe demands on NTP, which must deliver accurate and reliable time in spite of component failures, service disruptions and possibly mis-engineered implementations.

In the remainder of this introductory Section 1, issues in the requirements, approaches and comparisons with previous work are discussed. The architecture of the NTP synchronization subnet, including the primary time references and distribution mechanisms, is described in Section 2. An overview of the NTP protocol is given in Section 3 and further intricacies in a formal specification and implementation guide published elsewhere [MIL89a]. Section 4 describes the algorithms used to improve the accuracy of delay and offset measurements made over statistically noisy internet paths and to select the best clock from among a set of mutually suspicious clocks. Section 5 describes a local clock design based on a first-order, adaptive-parameter phase-lock loop and capable of accuracies to the order of a millisecond, even after extended periods when all synchronization paths to primary reference sources have been lost. The international NTP synchronization subnet of time servers and clients now operating on the Internet is described and its performance assessed in Section 6. Further details on measured performance in the Internet can be found in [MIL89b]. Section 7 discusses further development and issues for future research.

1.1. Performance Requirements

In this paper to synchronize frequency means to adjust the clocks in the network to run at the same frequency, while to synchronize time means to set the clocks so that all agree upon 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. A clock synchronization subnet operates by measuring clock offsets between the various time servers in the subnet and so is vulnerable to statistical delay variations on the various transmission paths. In the Internet the paths involved can have wide variations in delay and reliability. The routing algorithms can select landline or satellite paths, public network or dedicated links or even suspend service without prior notice.

It should be noted that stable frequency synchronization in large subnets requires finely tuned tracking loops and multiple phase comparisons over relatively long periods of time, while reliable time synchronization requires carefully engineered selection algorithms and the use of redundant resources and diverse transmission paths. For instance, while only a few comparisons are usually adequate to determine local time in the Internet to within a few tens of milliseconds, dozens of measurements over many days are required to reliably resolve oscillator drift and stabilize frequency to a few milliseconds per day [MIL89b].

Thus, the performance requirements of an internet-based time synchronization system are particularly demanding. A basic set of requirements must include the following:

1. The primary time reference source(s) must be synchronized to national standards by wire, radio or portable clock. The system of time servers and clients must deliver continuous local time based on UTC, even when leap seconds are inserted in the UTC timescale.

2. The time servers must provide accurate and precise time, even with relatively large stochastic delays on the transmission paths. This requires careful design of the data smoothing and deglitching algorithms, as well as an extremely stable local clock oscillator and synchronization mechanism.

3. The synchronization subnet must be reliable and survivable, even under unstable conditions and where connectivity may be lost for periods up to days. This requires redundant time servers and diverse transmission paths, as well as a dynamically reconfigurable subnet architecture.

4. The synchronization protocol must operate continuously and provide update information at rates sufficient to compensate for the expected wander of the room-temperature crystal oscillators used in ordinary computer systems. It must operate efficiently with large numbers of time servers and clients in continuous-polled and procedure-call modes and in multicast and point-to-point configurations.

5. The system must operate in existing internets including a spectrum of machines ranging from personal workstations to supercomputers, but make minimal demands on the operating system and supporting services. Time-server software and especially client software must be easily installed and configured.

1.2. Discussion of Approaches

There are many ways that hosts distributed throughout a large geographic area can synchronize clocks to UTC. In North America the U.S. and Canada operate broadcast radio services with a UTC timecode modulation which can be decoded by suitable receivers [NBS79]. One approach to time synchronization is to provide timecode receivers at every site where required. However, these receivers are specialized, moderately expensive and subject to occasional gross errors due to propagation and equipment failures.

The U.S. National Institute of Standards and Technology (NIST) (formerly National Bureau of Standards), recently announced a computer time service available to the general public by means of a standard telephone modem [NBS88]. The service is intended for use by personal workstations to set clock-calenders, for example, but would not be suitable for a large population of clients calling on a frequent, regular basis without further redistribution.

In principle, it is possible to use special network facilities designed for time synchronization, such as a dedicated FM or TV subcarrier or timecode rebroadcast by a cable system. For many years AT&T has synchronized digital switching equipment to the Basic Synchronization Reference Frequency (BSRF), which consists of a master oscillator synchronized to UTC and a network of dedicated 2048-kHz links embedded in the transmission plant. AT&T and other carriers are planning to use the Global Positioning System and the LORAN-C radionavigation system to synchronize switches in various areas of the country to UTC. However, neither of these methods would be economically viable for widespread deployment in a large, diverse internet system.

Various mechanisms have been used in the Internet protocol suite to record and transmit the time at which an event takes place, including the Daytime protocol [POS83a], Time protocol [POS83b], ICMP Timestamp message [DAR81a] and IP Timestamp option [SU81]. In the Hellospeak routing protocol [MIL83b] one or more processes synchronize to an external reference source, such as a timecode receiver or time daemon, and the corrections are distributed via routing updates and a minimum-delay spanning tree rooted on these processes. The Unix 4.3bsd time daemon *timed* [GUS85a] uses an elected master host [GUS85b] to measure offsets of a number of slave hosts and send periodic corrections to them.

Experimental results on measured times and delays in the Internet are discussed in [COL88], [MIL83a] and [MIL85b]. Other synchronization algorithms are discussed in [GUS84], [HAL84], [LAM78], [LAM85], [LUN84], [MAR85], [MIL85a], [MIL85b], [MIL89a], [RIC88], [SCH86], [SRI87] and [TRI86] and protocols based on them in [MIL81], [MIL85c], [MIL89a] and [TRI86]. NTP uses techniques evolved from them and both linear systems and Byzantine agreement methodologies. Linear methods for digital telephone network synchronization are summarized in [LIN80], while Byzantine methods for clock synchronization are summarized in [LAM85].

In an internet system involving many networks and gateways a useful approach is to equip a few strategically located time-server hosts (or gateways) with timecode receivers and coordinate time distribution using a suitable internet protocol. However, the success of this approach requires very accurate and reliable mechanisms to process and distribute the time information, since timecode receivers, time servers and the internet itself cannot be considered wholly reliable. While reliable clock synchronization has been studied using agreement algorithms [LAM85], [SRI87], in practice it is not possible to distinguish the *truechimer* clocks, which maintain timekeeping accuracy to a previously published (and trusted) standard, from the *falseticker* clocks, which do not, on other than a statistical basis. In addition, the algorithms and protocols discussed in the literature do not necessarily produce the most accurate time on a statistical basis and may produce unacceptable network overheads and instabilities in a large, diverse internet system.

The above approach was used in the design of the NTP synchronization mechanisms, which were evolved as the result of numerous experiments, analyses and stepwise refinements over an eight-year period. It became evident that accurate and reliable internet time synchronization can be achieved only through a integrated approach to system design including the primary reference sources, time servers, synchronization subnet, protocols and synchronization mechanisms which are at the heart of this paper. From the analytical point of view the distributed system of NTP time servers operates as a set of mutually coupled, phase-locked oscillators with phase comparisons exchanged by means of update messages and a local clock at each time server functioning as a disciplined oscillator. The principal features of this design, described in more detail later in this paper, can be summarized as follows:

1. The synchronization subnet consists of a self-organizing, hierarchical network of time servers configured on the basis of estimated accuracy and reliability.
2. The synchronization protocol operates in connectionless mode in order to minimize latencies, simplify implementations and provide ubiquitous interworking.
3. The synchronization mechanism uses a returnable-time design which tolerates packet loss, duplication and misordering, together with filtering algorithms based on maximum-likelihood principles.
4. The local clock design is based on a first-order, adaptive-parameter phase-lock loop with corrections computed using timestamps exchanged along the arcs of the synchronization subnet.
5. Multiply redundant time servers and multiply diverse transmission paths are used in the synchronization subnet, as well as engineered algorithms which select the most reliable synchronization source and path using a weighted voting procedure.
6. System overhead is reduced through the use of dynamic control of polling rates and association management

2. Time Standards and Distribution

Since 1972 the time and frequency standards of the world have been based on International Atomic Time (TAI), which is currently maintained using multiple cesium-beam clocks to an accuracy of a few parts in 10^{12} [BLA74]. The Bureau International de l'Heure (BIH) uses astronomical observations provided by the U.S. Naval Observatory and other observatories to determine corrections for small changes in the mean solar rotation period of the Earth, which results in Coordinated Universal Time (UTC). UTC is presently decreasing relative to TAI at a fraction of a second per year, so corrections in the form of leap seconds must be inserted from time to time in order to maintain agreement. The U.S. and many other countries operate standard time and frequency broadcast stations covering most areas of the world, although only a few utilize a broadcast timecode suitable for computer use. The U.S. and Canadian timecodes provide UTC time-of-day, day-of-year and related information, but not either the year or advance notice of leap seconds, which must be determined from other sources.

A synchronization subnet is a connected network of primary and secondary time servers, clients and interconnecting transmission paths. A primary time server is directly synchronized to a primary reference source, usually a timecode receiver. A secondary time server derives synchronization, possibly via other secondary servers, from a primary server over network paths possibly shared with other services. Under normal circumstances it is intended that the synchronization subnet of primary and secondary servers assumes a hierarchical master-slave configuration with the primary servers at the root and secondary servers of decreasing accuracy at successive levels toward the leaves.

Following conventions established by the telephone industry, the accuracy of each time server is defined by a number called the stratum, with the root level (primary servers) assigned as one and each succeeding level towards the leaves (secondary servers) assigned as one greater than the preceding level. Using existing stations, available timecode receivers with propagation-delay corrections and allowing for sample accumulations up to a week or more, accuracies in the order of a millisecond can be achieved at the network interface of a primary server [MIL89b]. As the stratum increases from one, the accuracies achievable will degrade depending on the network paths and local clock stabilities. In order to avoid the tedious calculations [BRA80] necessary to estimate errors in each specific configuration, it is useful to assume the measurement errors accumulate approximately in proportion to the total roundtrip path delay to the root of the synchronization subnet, which is called the synchronizing distance.

Again drawing from the experience of the telephone industry, which learned such lessons at considerable cost, the synchronization subnet should be organized to produce the highest accuracy, but must never be allowed to form a loop, regardless of synchronizing distance. An additional factor is that each increment in stratum involves a potentially unreliable time server which introduces additional measurement errors. The selection algorithm used in NTP uses a variant of the Bellman-Ford distributed routing algorithm [BER87] to compute the minimum-weight spanning trees rooted on the primary servers. With the foregoing factors in mind, the distance metric used by the algorithm was chosen using the stratum number as the high-order bits and synchronizing distance as the low-order bits.

3. Network Time Protocol

The Network Time Protocol (NTP) is used to construct and maintain a set of time servers and transmission paths as a synchronization subnet. The protocol was first described in [MIL85c], extensively revised in successive versions and recently established as a formal Internet Standard protocol [MIL89a]. NTP is built on the Internet Protocol (IP) [DAR81b] and User Datagram Protocol (UDP) [POS80], which provide a connectionless transport mechanism; however, it is readily adaptable to other protocol suites. It is evolved from the Time Protocol [POS83b] and the

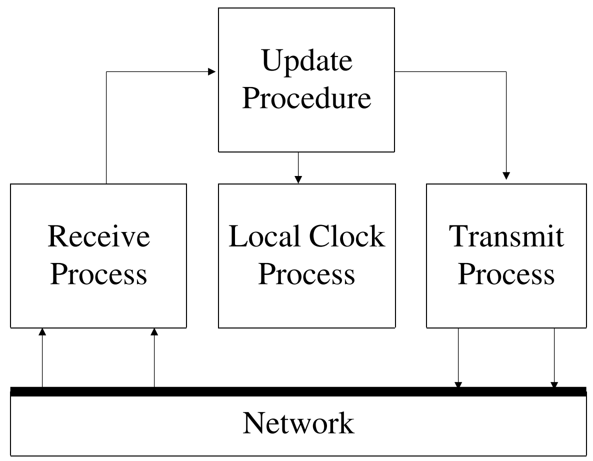

Figure 1. Implementation Model

ICMP Timestamp Message [DAR81a], but is specifically designed to maintain accuracy and reliability, even when used over typical Internet paths involving multiple gateways and unreliable nets.

There is no provision for peer discovery, acquisition, or authentication in NTP itself, although some implementations include these features. Data integrity is provided by the IP and UDP checksums. No circuit-management, duplicate-detection or retransmission facilities are provided or necessary. The protocol can operate in several modes appropriate to different scenarios involving private workstations, public service machines and various network configurations. A lightweight association-management capability, including dynamic reachability and variable polling-rate 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. Since only a single NTP message format is used, the protocol is easily implemented and can be used in a variety of operating-system and networking environments.

The following subsections contain an overview of the data formats, entities, state variables and procedures used in NTP. Further details are contained in the formal specification. The specification is based on the implementation model illustrated below, but it is not intended that this model be the only one upon which a specification can be based. In particular, the specification is intended to illustrate and clarify the intrinsic operations of NTP and serve as a foundation for a more rigorous, comprehensive and verifiable specification.

In what may be the most common client/server modes a client sends an NTP message to one or more time servers, which process the replies as received. A server interchanges addresses, overwrites certain fields in the message, recalculates the checksum and returns the message immediately. Information included in the NTP message allows the client to determine the server time with respect to local time and adjust the local clock accordingly. In addition, the message includes information to calculate the expected timekeeping accuracy and reliability, so that inferior data can be discarded and only the best from possibly several servers can be selected. While the client/server modes may suffice for use on LANs involving a public time server and perhaps many private workstation clients, the full generality of NTP requires distributed participation of a number of client/servers or peers arranged in a dynamically reconfigurable, hierarchically distributed configuration. It also requires sophisticated algorithms for association management, data manipulation and local clock control as described below.

3.1. Implementation Model

Figure 1 shows an implementation model for a time-server host including three processes sharing a partitioned data base, with a partition dedicated to each peer, and interconnected by a message-passing system. The transmit process, driven by independent timers for each peer, collects information in the data base and sends NTP messages to the peers. Each message contains the local timestamp when the message is sent, together with previously received timestamps and other information necessary to determine the hierarchy and manage the association. The message transmission rate is determined by the accuracy required of the local clock, as well as the estimated accuracies of its peers.

The receive process receives NTP messages and perhaps messages in other protocols, as well as information from directly connected timecode receivers. When a message is received the offset between the peer clock and the local clock is computed and incorporated into the data base along with other information useful for error estimation and peer selection. A filtering algorithm described in Section 4 improves the estimates by discarding inferior data.

The update procedure is initiated upon receipt of a message and at other times. It processes the offset data from each peer and selects the best one using the selection algorithm of Section 4. This may involve many observations of a few peers or a few observations of many peers, depending on the accuracies required.

The local-clock process operates upon the offset data produced by the update algorithm and adjusts the phase and frequency of the local clock using the mechanism described in Section 5. This may result in either a step change or a gradual slew adjustment of the local clock to reduce the offset to zero. The local clock provides a stable source of time information to other users of the system and for subsequent reference by NTP itself.

3.1.1. Modes of Operation

An NTP association is formed when two peers exchange messages and one or both of them create and maintain an instantiation of the protocol machine. The machine can operate in one of five modes: symmetric active, symmetric passive, client, server and broadcast. When an association is formed one or both peers must be operating in an active mode (symmetric active, client or broadcast), in which the active peer sends messages to the other peer regardless of the mode, reachability state or stratum of the other peer. If due to crash restart or defective programming both peers are found to be operating in a passive mode (symmetric passive or server), each peer ignores the messages of the other, so that eventually each will find the other unreachable and demobilize the association. In symmetric passive and server modes the identity of the other peer need not be known in advance, since an association with persistent state variables is created only when an NTP message arrives.

Furthermore, the state storage can be reused when the peer becomes unreachable or begins operating at a higher stratum and thus ineligible as a synchronization source.

The symmetric modes are intended for distributed scenarios where either peer can potentially become the reference source for the other. By operating in these modes a host announces its willingness to synchronize to and be synchronized by the peer. Symmetric active mode is designed for use by hosts operating near the leaves (high stratum levels) of the synchronization subnet, while symmetric passive mode is designed for use by hosts operating near the root (low stratum levels) and with a relatively large number of peers on an intermittent basis. Reliable time service can usually be maintained with two peers at the next lower stratum level and one peer at the same stratum level, so the rate of ongoing polls is usually not significant, even when connectivity is lost and error messages are being returned from the network for every poll.

Broadcast mode is intended for operation with high speed LANs and numerous workstations where the highest accuracies are not required. In the typical scenario one or more LAN time-server hosts send periodic NTP messages to the LAN broadcast address. The LAN workstations then determine the time on the basis of a preconfigured latency in the order of a few milliseconds. By operating in this mode the host announces its willingness to provide synchronization to the peers, but not to accept NTP messages from them.

The client and server modes are intended for operation with file servers and workstations requiring the highest accuracies or where broadcast mode is unavailable or inappropriate. A host operating in client mode sends periodic NTP messages to one or more servers. By operating in this mode the host announces its willingness to be synchronized by, but not to provide synchronization to the peer. A host operating in server mode simply interchanges the source and destination addresses/ports, fills in the requested timestamps and returns the message to the client. Hosts operating in server mode need retain no state information between client requests, while clients are free to manage the intervals between messages to suit local conditions. By operating in this mode the host announces its willingness to provide synchronization to, but not to be synchronized by the peer.

3.1.2. Data Formats

All mathematical operations assumed in the protocol are two's-complement arithmetic with integer or fixed-point operands. Since NTP timestamps are cherished data and, in fact, represent the main product of the protocol, a special format has been established. An NTP timestamp is a 64-bit unsigned fixed-point number, with the integer part in the first 32 bits and the fraction part in the last 32 bits and interpreted in standard seconds relative to UTC. When UTC began at zero hours on 1 January 1972 the NTP clock was set to 2,272,060,800.0, representing the number of standard seconds since this time on 1 January 1900 (assuming no prior leap seconds).

This format allows convenient multiple-precision arithmetic and conversion to other formats used by various protocols of the Internet suite. The precision of this representation is about 232 picoseconds, which should be adequate for even the most exotic requirements. Note that since some time in 1968 the most significant bit of the 64-bit field has been set and that the field will overflow some time in 2036. Should NTP be in use in 2036, some external means will be necessary to qualify

<table>
<tr><td>LI</td><td>VN</td><td>Mode</td><td>Stratum</td><td>Poll Int</td><td>Precision</td></tr>
<tr><td colspan="6">Synchronizing Distance</td></tr>
<tr><td colspan="6">Synchronizing Dispersion</td></tr>
<tr><td colspan="6">Reference Timestamp (64 bits)</td></tr>
<tr><td colspan="6">Originate Timestamp (64 bits)</td></tr>
<tr><td colspan="6">Receive Timestamp (64 bits)</td></tr>
<tr><td colspan="6">Reference Timestamp (64 bits)</td></tr>
<tr><td colspan="6">Authenticator (optional) (96 bits)</td></tr>
</table>

Figure 2. NTP Packet Header

time relative to 1900 and subsequent 136-year cycles. Timestamped data requiring such qualification will be so precious that appropriate means should be readily available.

Timestamps are determined by copying the current value of the local clock to a timestamp variable when some significant event occurs, such as the arrival of a message. In some cases a particular variable may not be available, such as when the host is rebooted or the protocol first starts up. In these cases the 64-bit field is set to zero, indicating the value is invalid or undefined. There will exist an 232-picosecond interval, henceforth ignored, every 136 years when the 64-bit field will naturally become zero and thus be considered invalid.

3.1.3. State Variables

Following is a summary description of the important variables and parameters used by the protocol. In appropriate modes a set of state variables is maintained for the host itself along with separate copies for each peer with an active association. Further information on these variables is given later in this paper. A complete description is given in [MIL89a].

Figure 2 shows the NTP packet-header format, which ordinarily follows the IP and UDP headers. Following is a description of the various fields.

Version Number (VN). Identifies the present NTP version.

Leap Indicator (LI). Warns of an impending leap second to be inserted or deleted in the timescale at the end of the current day.

Mode, Stratum, Precision. Indicate the current operating mode, stratum and precision.

Poll Interval. Controls the intervals between NTP messages sent by the host to a peer. The sending host always uses the minimum of its own poll interval and the peer poll interval.

Synchronizing Distance, Synchronizing Dispersion. Indicate the estimated roundtrip delay and estimated sample dispersion, respectively, to the primary reference source.

Reference Clock Identifier, Reference Timestamp. Identify the reference clock and the time of its last update, intended primarily for management functions.

Originate Timestamp. The time when the last received NTP message was originated, copied from its transmit timestamp field upon arrival.

Receive Timestamp. The local time when the latest NTP message was received.

Transmit Timestamp. The local time when the latest NTP message was transmitted.

Authenticator (optional). The key identifier and encrypted checksum of the message contents.

The NTP protocol machine maintains state variables for each of the above quantities and, in addition, other state variables, including the following:

Addresses and Ports. The 32-bit Internet addresses and 16-bit port numbers of the host and its peers, which serve to identify the association.

Peer Timer. A counter used to control the intervals between transmitted NTP messages.

Reachability Register. A shift register used to determine the reachability status of the peer.

Filter Register. A shift register where each stage stores a tuple consisting of the measured delay and offset associated with a single observation.

Delay Estimate, Offset Estimate, Dispersion Estimate. Indicate the current estimated roundtrip delay, clock offset and dispersion produced by the filter procedure.

Clock Source. A selector identifying the current clock source determined by the selection procedure.

Local Clock. The current local time as derived from the host logical clock.

3.2. Procedures

The significant events of interest in NTP occur upon expiration of a peer timer, one of which is dedicated to each peer with an active association, and upon arrival of an NTP message from the various peers. An event can also occur as the result of an operator command or detected system fault, such as a primary clock failure. This subsection describes the procedures invoked when these events occur.

The transmit procedure is called when a peer timer decrements to zero and can occur in all modes except server mode. When this occurs the peer timer is reset and an NTP message is sent including the addresses, variables and timestamps described above. The value used to reset the timer is called the polling interval and is adjusted dynamically to reflect network peer path conditions.

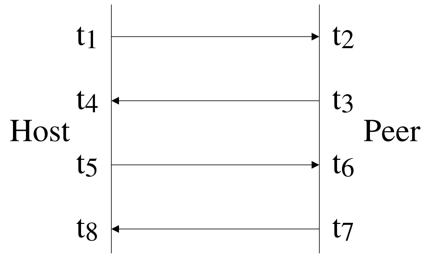

Figure 3. Calculating Delay and Offset

The receive procedure is called upon arrival of an NTP message, which is then matched with the peer association indicated by its addresses and ports. If there is no match a new instantiation of the protocol machine is created and the peer association mobilized. Following a set of sanity checks the roundtrip delay and clock offset relative to the peer are calculated as follows. Number the times of sending and receiving NTP messages as shown in Figure 3 and let i be an even integer. Then t_{i-3}, t_{i-2}, t_{i-1} and t_i are the message arrival and departure timestamps in the order shown and the roundtrip delay d_i and clock offset c_i of the peer relative to the host are computed as follows:

$$d_i = (t_i - t_{i-3}) - (t_{i-1} - t_{i-2}),$$
$$c_i = [(t_{i-2} - t_{i-3}) + (t_{i-1} - t_i)]/2.$$

This method amounts to a continuously sampled, returnable-time system, which is used in some digital telephone networks. Among the advantages are that the order and timing of the messages are unimportant and that reliable delivery is not required.

The offset c_i is then checked for validity relative to the estimated maximum skew and stated precision of the host and peer clocks. The filter procedure is called with c_i and d_i as arguments to produce the estimated delay, offset and dispersion for the peer involved. The minimum-filter algorithm described in Section 4 is used to improve these estimates depending upon the statistical properties of the outbound and inbound transmission paths.

The update procedure is called when a new delay/offset estimate becomes available. A weighted voting procedure described in Section 4 determines the best peer, which may result in a new clock source. If the clock source is the peer involved, the estimated offset is used to update the local clock as described in Section 5. If the local clock is reset, rather than gradually slewed to its final value, the clear procedure is called repeatedly for every active peer to purge the clock filter, reset the polling interval and reselect the clock source, if necessary. A new selection will occur when the filters fill up again and the dispersions settle down.

3.3. Robustness Issues

It has been the experience of the Internet community that something somewhere in the system is always broken at any given time. This caveat applies to timecode receivers, time servers and synchronization paths, any of which can misbehave to produce a bogus timestamp popularly known as a *timewarp*. The very nature of time synchronization requires that it be extremely robust against timewarps and capable of operation even when major breakdowns or attempted breakins occur. This subsection describes some of the measures taken to deal with these problems, including reachability determination, authentication, sequence checking and polling rates.

As shown previously, reliable time synchronization does not require reliable message delivery; however, in order to minimize resource requirements, resist using very old data and manage the number of associations required, a simple reachability protocol is used. Each time an NTP message is sent an eight-bit reachability register is shifted one position to the left and the low-order position set to zero. If an NTP message is received before the next message is sent, the low-order bit is set to one. The peer is considered reachable if the register is nonzero, which will always be the case if at least one message is received during eight consecutive transmit intervals. In the passive modes if a peer becomes unreachable, the association is demobilized and its resources reclaimed for subsequent associations. This shift-register mechanism has also been found most effective in other Internet protocols designed to survive unstable network service.

While not a required feature of NTP itself, some implementations have included an access-control feature that prevents unauthorized access and controls which peers are allowed to update the local clock. This is done with tables of mask-and-match entries which allow only servers with Internet addresses in certain ranges to become peers and to update the local clock. In principle, this feature is not necessarily desirable, since the basic NTP design involving a large degree of diversity is intended to cast out falsetickers whether they are authenticated or not. Nevertheless, the additional authentication feature may help to deflect multiple-peer destructive jamming.

During the course of normal operation various peers may on occasion restart the NTP program itself, either due to loss of reachability, reboot or other trauma, which results in loss of state information. Special sanity checks are built into the receive procedure to avoid disruption in such cases. One check requires the transmit timestamp of a received message to be different than the transmit timestamp of the last message received. If they are the same, the message is a duplicate and must have been circulating in the network for some time after the last message was received, thus containing relatively inferior data. Another check requires the originate timestamp of a received message to be identical to the transmit timestamp of the last message transmitted. If they do not match, the message is either out of order, from a previous association or bogus. Additional checks protect against using very old time information and from associations not completely synchronized.

Where security considerations require the highest level of protection against message modification, replay and other overt attacks, the NTP specification includes optional cryptographic authentication procedures. The procedures are used to insure an unbroken chain of authenticated peers along the synchronization subnet to the primary servers. An authenticator, consisting of a key identifier and encrypted checksum, is computed using the DES encryption algorithm and DES cipher block-chaining method. Present implementations have incorporated special provisions to avoid degradation in timekeeping accuracy due to the delays caused by the encryption computations.

Careful consideration was given during design to factors affecting network overheads. Many of the present Internet primary time servers operate with 100 peers or more and some operate with many more than that. Therefore, it is important that the longest polling intervals consistent with the required accuracy and reliability be used. When reachability is first confirmed and when a peer is currently selected for synchronization it is necessary to use a relatively short polling interval in the order of a minute. In those cases where the association has stabilized, the dispersions are low and the peer is not selected for synchronization, the polling interval can be increased substantially. In the present design the polling interval is increased gradually from about one minute to about 17 minutes as long as the dispersion is below an experimentally determined threshold; otherwise, it is decreased gradually to the initial value. These values may change as the result of further experience.

4. Sample Processing and Selection Operations

At the very heart of the NTP design are the algorithms used to improve the accuracy of the estimated offsets and delays between the host and its various peers, as well as those used to select a particular peer for synchronization. The complexity of these algorithms depends on the statistical properties of the transmission path and the accuracies required. In the case of LANs operating at megabit speeds and above the path delays are usually smaller than the required accuracies, so the raw offsets delivered by the receive procedure can often be used directly to adjust the local clock. However, in the case of paths involving many network hops via regional and backbone networks such as NSFNET, the path delays and delay variances can be much larger than acceptable without further processing.

Since Internet paths often involve multiple hops over networks of widely varying characteristics, it is not possible to design one set of algorithms optimized for a particular path. Therefore, the development of algorithms appropriate for ubiquitous Internet application has involved a process of stepwise refinement, beginning with the designs suggested in [MIL85a], refined as the results of experiments described in [MIL85b] and [MIL89b] and evolved over several years of operation under widely varying conditions of path qualities and reliabilities. In following subsections the issues, approaches and designs of these algorithms are discussed.

4.1. Data Filtering

A number of algorithms for deglitching and filtering time-offset data have been described previously, such as in [MIL85a], [HAL84] and [KOP87]. These fall in two classes: majority-subset algorithms, which attempt to separate good subsets from bad by comparing statistics such as mean and variance, and clustering algorithms, which attempt to improve the estimate by repeatedly casting out outlyers. The former class was suggested as a technique to select the best (i.e., the most reliable) clocks from a population, while the latter class was suggested as a technique to improve the offset estimate for a single clock given a series of observations.

After further development and experimentation using typical Internet paths, a better algorithm was found for casting out outlyers from a continuous stream of offset observations spaced at intervals in the order of minutes. The algorithm can be described as a variant of a median filter, in which a window consisting of the last n sample offsets is continuously updated and the median sample selected as the estimate. However, in the modified algorithm the outlyer (sample furthest from the median) is then discarded and the entire process repeated until only a single sample offset is left, which is then produced as the offset estimate. This algorithm was found to be more resistant to

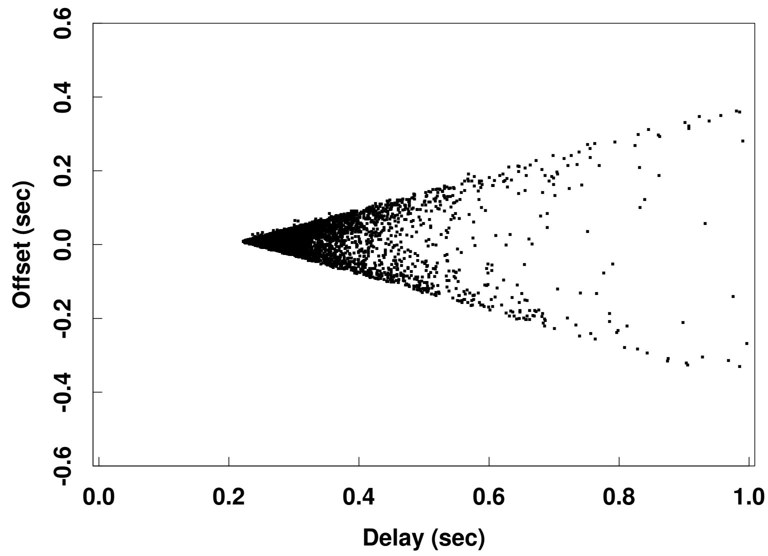

Figure 4. Offset vs Delay

glitches and to provide a more accurate estimate than the unmodified one. It was used in the Fuzzball and Unix implementations for about two years until the end of 1987.

Recent experiments have demonstrated an even better algorithm which provides higher accuracy together with a lower computational burden. The key to the new algorithm became evident through an examination of scatter diagrams plotting clock offset versus roundtrip delay. Recall that the roundtrip delay d_i and clock offset c_i are computed from the four most recently observed timestamps. Without making any assumptions about the delay distributions due to queueing delays for traffic in either direction along the path, but assuming the intrinsic frequency offsets of the host and peer clocks are relatively small, let d_0 and c_0 represent the delay and offset when no other traffic is present on the path and so represents the true values. The problem is to accurately estimate d_0 and c_0 from a sample population of d_i and c_i collected over a relatively long period.

Figure 4 shows a typical scatter diagram in which the points (d_i,c_i) are concentrated near the apex of a wedge defined by lines extending from the apex with slopes +0.5 and -0.5, corresponding to the locus of points as the delay in one direction increases while the delay in the other direction does not. From these data it is obvious that good estimators for (d_0,c_0) would be points near the apex.

Upon reflection, the reason for the particular distribution of points in the scatter diagram can be explained as follows. Packet-switching nets are most often operated well below the knee of the throughput-delay curve, which means the busy periods are relatively short and infrequent. In addition, the routing algorithm most often operates to minimize the number of packet-switch hops and thus the number of queues. Thus, not only is the probability that an arriving NTP packet finds a busy queue in one direction relatively low, but the probability of packets from a single exchange finding busy queues in both directions is even lower.

From the above discussion one would expect that, at low utilizations and hop counts, the points should be concentrated about the apex of the wedge and begin to extend rightward along the extrema lines as the utilizations and hop counts increase. As the utilizations and hop counts continue to increase, the points should begin to fill in the wedge as it expands even further rightward. This behavior is in fact what is observed on typical Internet paths involving ARPANET, NSFNET and regional nets.

These observations cast doubt on the median-filter approach as a good way to cast out offset outlyers and suggests another approach which might be called a minimum filter. From the scatter diagrams it is obvious that the best offset samples occur at the lower delays. Therefore, an appropriate technique would be simply to select from the n most recent samples the sample with lowest delay and use its associated offset as the estimate. Several experiments were designed to test this technique using measurements between selected hosts equipped with timecode receivers, so that delays and offsets could be determined independent of the measurement procedure itself. The experiments were performed over several paths involving ARPANET, NSFNET and local nets and using both median and minimum filters of varying lengths. The results show consistently lower errors for the minimum filter when compared with the median filter of similar length. Perhaps the most dramatic result of the minimum filter is the greatly reduced maximum error at the upper end of the throughput range [MIL89b]. Based on these data the minimum filter was selected as the preferred algorithm. Since its performance did not seem to much improve for values of n above eight, this value was chosen for use in the reference implementation.

In the reference implementation the clock filter procedure is executed upon arrival of an NTP message or other event that results in new delay/offset sample pairs. New sample pairs are shifted into an eight-stage shift register from the left end, causing old entries to shift off the right end. Then those entries in the register with nonzero delay are inserted on a temporary list and sorted in order of increasing delay. In case of ties an arbitrary choice is made. The delay and offset estimates are chosen as the corresponding values of the first entry on the list.

For subsequent processing and use in the clock selection algorithm it is useful to have an estimate of sample dispersion. A good dispersion estimate which weights samples near the apex of the wedge most heavily and is easily computable is simply the weighted sum of the offsets in the temporary list. Assume the shift register has $n > 1$ entries and the list contains m $(0 \le m \le n)$ valid samples in order of increasing delay. If X_i $(0 \le i < n)$ is the offset of the ith sample, then

$$\begin{array}{rl} s_i = |X_i - X_0| & \multirow{2}{*}{$\left\{ \begin{array}{l} \text{if } i < m \text{ and } |X_i - X_0| < D_{max} \\ \text{otherwise} \end{array} \right.$} \\ s_i = D_{max} & \end{array} .$$

The dispersion estimate S is then

$$S = \sum_{i=0}^{n-1} s_i \, v^{\,i} ,$$

where $v \le 1$ is a weighting factor and D_{max} is an experimentally determined threshold adjusted to match typical offset distributions. The reference implementation use the values 0.5 for v and 65.535

for *Dmax*. The dispersion estimate is intended for use as a quality indicator, with increasing values associated with decreasing quality and given less weight in the clock selection algorithm. The sum of the total path dispersions to the root of the synchronization subnet is called the synchronizing dispersion.

In the hope that offset accuracy could be improved through the use of more than just the minimum-delay filter sample, experiments were made involving various types of averages weighted both by zero, first and second-order dispersion. The results using eleven peer paths measured over four weeks demonstrate little improvement over the simple method, which is now used in the reference implementation.

4.2. Peer Selection

The single most important contributing factor in maintaining high reliability with NTP is the peer selection mechanism. When new offset estimates are produced for a peer or are revised as the result of timeout, this mechanism is used to determine which peer should be selected as the clock source. A good deal of research has gone into mechanisms to synchronize clocks in a community where some clocks cannot be trusted. Determining whether a particular clock is a truechimer or falseticker is an interesting abstract problem which can be attacked using methods such as described in [LAM78], [LAM85], [MAR85] and [SCH86].

In methods described in the literature a convergence function operates upon the offsets between the clocks in a system to increase the reliability by reducing or eliminating errors caused by falsetickers. There are two classes of convergence functions, those involving interactive convergence algorithms and those involving interactive consistency algorithms. Interactive convergence algorithms use statistical clustering techniques such as the fault-tolerant average algorithm of [HAL84], the CNV algorithm of [LUN84], the majority-subset algorithm of [MIL85a], the non-Byzantine algorithm of [RIC88], the egocentric algorithm of [SCH86] and the algorithm described below in this paper.

Interactive consistency algorithms are designed to detect faulty clock processes which might indicate grossly inconsistent offsets in successive readings or to different readers. These algorithms use an agreement protocol involving successive rounds of readings, possibly relayed and possibly augmented by digital signatures. Examples include the fireworks algorithm of [HAL84] and the optimum algorithm of [SRI87]. However, these algorithms require an excessive burden of messages, especially when large numbers of clocks are involved, and are designed to detect faults that have rarely been found in the Internet experience. For these reasons they are not considered further in this paper.

The basic principles guiding the design of the NTP clock selection procedure are based on maximum likelihood techniques and the experimental observation that the highest reliability is usually associated with the lowest stratum and synchronizing dispersion, while the highest accuracy is usually associated with the lowest stratum and synchronizing distance. A key design assumption is that truechimers are relatively numerous and represented by random variables narrowly distributed about UTC in the measurement space, while falsetickers are relatively rare and represented by random variables widely distributed throughout the measurement space.

Offset	Dispersion	Result
0,0,0	0,0,0	0,0,-
0,0,1	9,9,28	0,0,-
0,1,0	12,25,12	0,-,0
0,1,1	21,16,16	-,1,1
1,0,0	21,16,16	-,0,0
1,0,1	12,25,12	1,-,1
1,1,0	9,9,28	1,1,-
1,1,1	0,0,0	1,1,-

Table 1. Outlyer Selection Procedure

The NTP clock selection procedure begins by constructing a list of candidate peers sorted first by stratum and then by synchronizing dispersion via the peer to the root of the synchronization subnet. To be included on the candidate list the peer must pass certain sanity checks. One check requires that the clock selection for the peer must not be the host itself; otherwise, a synchronization loop would be formed. Another check requires that the dispersion be bounded by a value which insures that the filter registers are at least half full, which avoids using data from low quality associations or obviously broken implementations. If no candidates pass the sanity checks, the existing clock selection, if any, is cancelled and the local clock free-runs at its intrinsic frequency. The list is then pruned from the end to be no longer than a maximum size, currently set to five. Starting from the beginning, the list is truncated at the first entry where the number of different strata in the list exceeds a maximum, currently set to two. This particular procedure and choice of parameters has been found to produce reliable synchronization candidates over a wide range of system environments while minimizing the "pulling" effect of high-stratum, high-dispersion peers, especially when large numbers of peers are involved.

The final procedure is designed to detect falsetickers or other conditions which might result in gross errors. The pruned candidate list is resorted in the order first by stratum and then by synchronizing distance via the peer to the root of the synchronization subnet. Let $m > 0$ be the number of peers remaining in the list and let X_i be the offset of the ith peer. For each i ($0 \le i < m$) define the clock dispersion q_i relative to peer i:

$$q_i = \sum_{j=0}^{m-1} |X_i - X_j| w^j,$$

where $w \le 1$ is a weighting factor experimentally adjusted for the desired characteristic (see below). Then cast out the entry with maximum q_i or, in case of ties, the maximum i, and repeat the procedure. When only a single peer remains on the list it becomes the clock source.

This procedure is designed to favor those peers near the head of the candidate list, which are at the lowest stratum and lowest delay and presumably can provide the most accurate time. With proper selection of weighting factor w, outlyers will be trimmed from the tail of the list, unless a few of

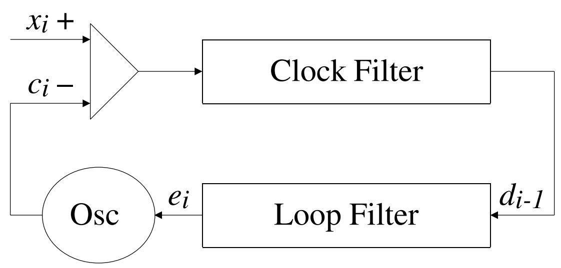

Figure 5. Phase-Lock Loop Model

them disagree significantly with respect to the remaining entries, in which case they are discarded first.

In order to see how this scheme works to cast out outlyers, consider the case of a host and three peers and assume that one or more of the offsets are clustered about zero and others are clustered about one. For w = 0.75 as used in the reference implementation and multiplying by 16 for convenience, the first entry has weight w^0 = 16, the second w^1 = 12 and the third w^2 = 9. Table 1 shows for all combinations of peer offsets the calculated clock dispersion about each of the three entries, along with the outcome. In the four cases where candidate 0 and candidate 1 disagree, the outcome is determined by candidate 2. Similar outcomes occur in the case of four candidates. While these outcomes depend on judicious choice of w, the behavior of the algorithm is substantially the same for values of w between 0.5 and 1.0.

Experience with this procedure shows it is vulnerable to frequent switching between peers, even if they have relatively low offsets and dispersions. Since switching can result in increased oscillator phase noise and increased polling rates, selection of a new peer is suppressed if its offset is less than the sum of the estimated maximum skew and stated precision of the host and peer clocks.

5. Local Clock Design

The full accuracy and reliability made possible by NTP requires careful design of the local-clock hardware and software. In the NTP model the local clock is assumed to use a quartz crystal oscillator without temperature compensation, which is typical of ordinary computing equipment. The fundamental system time reference, or logical clock, increments at some standard rate such as 1000 Hz and can be adjusted to precise time and frequency by means of periodic offset corrections computed by NTP, another time-synchronization protocol or a timecode receiver. A typical logical-clock implementation such as the Fuzzball [MIL89a] can maintain time in room-temperature environments to within a few milliseconds and frequency to within a few milliseconds per day in the absence of corrections. Substantially better performance can be achieved using precision oven-compensated quartz oscillators.

The NTP logical-clock model shown in Figure 5 can be represented as an adaptive-parameter, first-order, phase-lock loop, which continuously adjusts the clock phase and frequency to compensate for its intrinsic jitter, wander and drift. In the figure, x_i represents the reference timestamp and c_i the local timestamp of the ith update. The difference between these timestamps $x_i - c_i$ is the input offset, which is processed by the clock filter. The clock filter previously saved the most recent offsets and selected one of them d_{i-1} as the output offset. The loop filter, represented by the equations given below, produces the oscillator correction e_i, which is used to adjust the oscillator period. During the interval u_i until the next correction the, clock is slewed gradually to the given value e_i. This is done in order to smooth the time indications and insure they are monotone increasing.

The behavior of the phase-lock loop can be described by a set of recurrence equations, which depend upon several variables and constants. The variables used in these equations are (in SI units, unless specified otherwise):

d_i	clock filter output offset
u_i	interval until next update
e_i	oscillator correction
f_i	frequency error
g_i	phase error
h_i	compliance

These variables are set to zero on startup of the protocol. In case the local clock is to be reset, rather than adjusted gradually as described below, the phase error g_i is set to zero, but the other variables remain undisturbed. Various constants determine the stability and transient response of the loop. The constants used in the equations, along with suggested values, are:

U	2^2	adjustment interval
K_f	2^{10}	frequency weight
K_g	2^8	phase weight
K_h	2^8	compliance weight
S	2^4	compliance maximum
T	2^{18}	compliance factor

Let d_i $(i = 0, 1, \ldots)$ be a sequence of updates, with each d_{i+1} occurring u_i seconds after d_i. Let $q = 1 - {}^{1}\!/_{K_g}$ and n_i be the greatest integer in u_i/U ; that is, the number of adjustments that occur in the i th interval. As each update is received, the phase error g_i, frequency error f_i, and compliance h_i are recomputed. Let a_i be a quantity called the frequency gain: $a_i = \max(S - T\,|\,h_i\,|, 1)$. Then, upon receipt of the d_i update:

$$g_{i+1} = d_i \ ,$$

$$f_{i+1} = f_i + \frac{d_i}{a_{i-1}\, u_{i-1}} \ \ (f_0, f_1 = 0;\ i > 0) \ ,$$

$$h_{i+1} = h_i + \frac{d_i - h_i}{K_h} \ \ (h_0 = 0) \ ,$$

At each adjustment interval the quantity $\frac{g_{i+1}}{K_g} + \frac{f_{i+1}}{K_f}$ is added to the local clock and the quantity $\frac{g_{i+1}}{K_g}$ subtracted from g_{i+1}. Thus, at the end of the i th interval just before the d_{i+1} update, the accumulated correction is:

$$e_{i+1} = \frac{d_i}{K_g} \frac{q^{n_i} - 1}{q - 1} + \frac{1}{K_f} \sum_{j=1}^{i} \frac{n_j d_j}{a_{j-1} u_{j-1}} .$$

This can be seen to be the characteristic equation of an adaptive-parameter, first-order, phase-lock loop. Simulation of this loop with the variables and constants specified and the clock filter described previously results in the following characteristics: For a 100-ms phase change the loop reaches zero error in 39 minutes, overshoots 7 ms in 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 minutes, the compliance begins to increase, which causes the frequency gain to decrease, eventually to unity, and the loop to loosen. When the magnitude of correction falls below about 0.1 ppm for a few hours, the compliance begins to decrease, which causes the frequency gain to increase, eventually to 16, and the loop to stiffen. The effect is to provide a broad capture range exceeding four seconds per day, yet the capability to resolve oscillator drift well below a millisecond per day. These characteristics are appropriate for typical crystal-controlled oscillators with or without temperature compensation or oven control.

When the magnitude of a correction exceeds 128 ms, the possibility exists that the logical clock is so far out of synchronization with the reference source that the best action is an immediate and wholesale replacement of the Clock Register contents, rather than a graduated slewing as described above. This particular value was selected by experiment and experience with the current implementations and operating procedures. In practice, this value is exceeded with a single time-server source only under conditions of the most extreme congestion or when multiple failures of nodes or links have occurred. The most common cause is when the time-server source is changed and the difference between the old and new times is too large due to systematic errors in the primary reference source or large differential delays an the synchronization paths.

Conversion to and from the common date and time formats used by application programs is simplified with separate date and time software registers. The time register is designed to roll over at 24 hours, with its overflows (underflows) incrementing (decrementing) the date register. On the day prior to the insertion of a leap second the leap-indicator bits are set at the primary servers, presumably by manual means, and subsequently distributed via NTP throughout the synchronization subnet. This causes the modulus of the time register, which is the length of the current day, to be increased or decreased by one second as appropriate. On the day following insertion the bits are turned off at the primary servers.

6. NTP in the Internet System

The use of NTP in the Internet has steadily increased over the last few years. It is estimated that well over 2000 hosts and gateways presently synchronize their clocks directly to an NTP time server or indirectly via a LAN time server itself synchronized to an NTP time server. In this section an overview of the Fuzzball and Unix NTP time servers is presented along with a description of the NTP synchronization subnet now operating in the Internet.

6.1. Time Servers

The Fuzzball [MIL88] is a software package consisting of a fast, compact operating system, support for the Internet architecture and an array of application programs for network protocol development, testing and evaluation. It usually runs on a LSI-11 personal workstation, to which it also lends its name, and functions as a multi-purpose packet switch, gateway and service host. The Fuzzball is specially designed for applications requiring accuracies in the order of a millisecond, so it represents an ideal platform for the development and testing of time-synchronization architectures, protocols and algorithms, including those described in this paper. NTP and its forebears were developed and tested on the Fuzzball and the present NTP version is the reference implementation for the specification. Fuzzballs are presently installed at 18 locations in the U.S. and Europe, most of which function primarily as time servers for ARPANET, MILNET and NSFNET hosts and gateways. Ten of these, synchronized to UTC via radio or satellite, are part of the NTP primary synchronization subnet, while the remainder are part of the NTP secondary synchronization network.

An implementation of NTP as a Unix system daemon called *ntpd* was built by Michael Petry and Louis Mamakos of the University of Maryland. It includes all of the algorithms described in this paper and adjusts the system time using special Unix kernel primitives to control the local clock phase and frequency. Almost 1000 *ntpd* hosts were found in a recent survey [MIL89b]; however, this survey did not cover all Internet hosts. From other reports and personal communication it is estimated that the total number is well over 2000. Several of these, synchronized to UTC via radio, are part of the NTP primary synchronization subnet, while the remainder are part of the NTP secondary synchronization subnet.

An implementation of NTP as a dedicated processor and control program was built by Dennis Ferguson of the University of Toronto. This device uses a 68000 processor and includes an interface to a radio timecode receiver for the Canadian standard frequency/time station CHU. Other implementations are in progress at Hewlett-Packard Laboratories in Bristol, UK, and at the University of Delaware.

6.2. Synchronization Subnet

The NTP primary synchronization subnet now operating in the Internet consists of 17 primary time servers located in the U.S, Canada and the U.K. There are ten Fuzzball, five Unix and two other servers connected to receivers for WWVB, GOES, WWV/H, CHU (Ottawa) and GBR (Rugby) receivers. Of these, six are gatewayed directly to national backbone networks and are intended for ubiquitous access, while the remainder are connected to regional networks and intended for regional and local access.

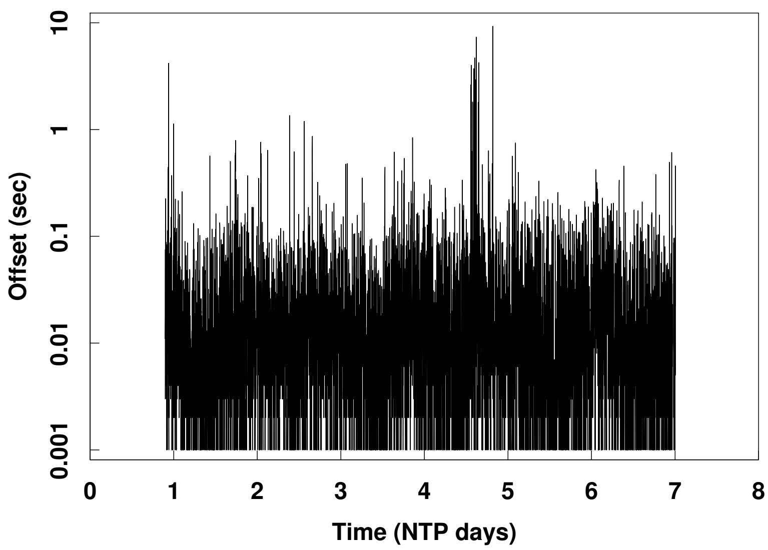

Figure 6. Raw Offsets

Most of the primary servers continuously exchange NTP messages with most of the other primary servers, so the primary synchronization subnet is almost completely connected. This provides an exceptional level of redundancy to protect against component or line failures. For instance, if a timecode receiver fails, the primary server synchronizes via NTP to the neighbor at the lowest available stratum and smallest synchronizing distance and continues operation as a secondary server at the next higher stratum. If a timecode receiver or time server appears to operate correctly but delivers incorrect time (falseticker), discrepancies become apparent to its NTP peers, which then deselect the server as the result of the algorithms described in Section 4.

The NTP secondary synchronization subnet presently includes eight Fuzzball and many more other secondary time servers and clients using some thousands of peer paths on hundreds of networks. A secondary server operating at stratum $n > 1$ ordinarily operates with at least three peers, two at stratum $n - 1$ and one or more at stratum n. In the most robust configurations a set of servers agree to provide backup service for each other, so distribute some of their peers over stratum-($n - 1$) servers and others over stratum-n servers in the same set. In a typical example configuration used at the University of Illinois and the University of Delaware the institution operates three stratum-2 campus servers, each peering with two out of six stratum-1 primary servers and with each other. The three campus servers in turn provide time for several stratum-3 department servers, each peering with all three campus servers. Department servers, many of which also function as file servers, then deliver time to possibly hundreds of stratum-4 workstations in client/server or broadcast modes.

6.3. Performance Analysis

As part of normal operation the Fuzzball time servers monitor delay and offset data from each of their peers. Periodically, these data are collected and analyzed to construct scatter diagrams, time-series diagrams and distribution functions. Scatter diagrams have proven exquisitely sensitive

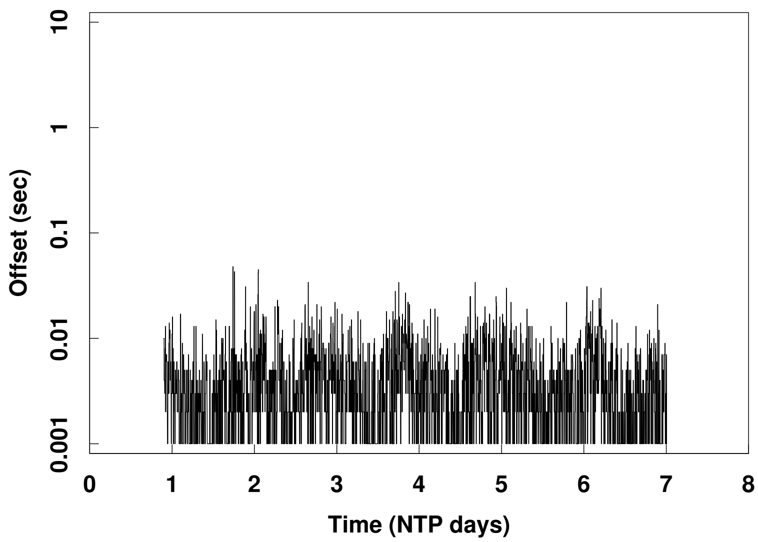

Figure 7. Filtered Offsets

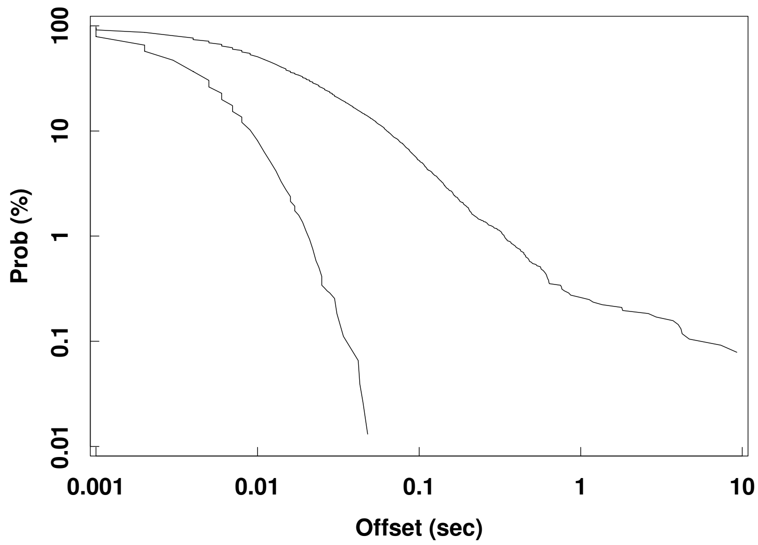

Figure 8. Error Distribution

indicators of system performance and possible malfunctions. A well performing peer path, such as shown in Figure 4, exhibits a distinct wedge-shaped opening to the right and with a sharp apex. As traffic increases on the peer path the wedge fills and extends to the right, showing increased delays and delay dispersions. Asymmetrical paths often show distinct asymmetries of delay dispersions readily detected by eye.

Time-series diagrams are useful for assessing algorithm performance and systematic errors. Figures 6 and 7, constructed from the same data as Figure 4, compare the absolute raw offsets and filtered offsets, respectively, between two primary time servers over an interval of about a week in UTC hours. The servers, each synchronized to WWVB to within about a millisecond, are connected by a path including the ARPANET and an unknown number of LANs and gateways. In typical such cases the reduction in mean offset measurement error is about tenfold. However, the most dramatic reduction with the filter is in maximum error, for which distribution functions with log-log axes are most useful. Figure 8 shows such a function for the above path and the absolute raw offsets (upper curve) and filtered offsets (lower curve), from which it is apparent that the maximum error after the filter is less than about 30 ms for all but about one percent of the samples and less than about 50 ms for all samples.

7. Future Directions

The IRIG-H timecode format established in 1970 and used since then by NBS/NIST radio broadcast services does not include year information, which must be entered manually after reboot at the primary time servers. While the year information, once entered, rolls over at the new year, it has sometimes happened that the year was entered incorrectly after reboot, with surprisingly disruptive effects on file archiving systems. In addition, the current timecode formats do not include advance notice of leap-second insertion. Clearly, a revised timecode format including these data is needed. In fact, the recently introduced NBS telephone time service [NBS88] does include both the year and advance leap-second information.

The current mechanism of time delivery using WWVB, WWV and GOES requires relatively expensive timecode receivers subject to occasional disruption due to propagation path or transmitter failure. Radionavigation systems such as LORAN-C and the Global Positioning System (GPS) are capable of delivering accurate time and frequency information over major portions of the world; however, these systems do not include an unambiguous timecode modulation and appropriate receivers are not yet available. An agenda should be pursued to provide a precise timecode as part of normal service.

As experience accumulates, improvements are being made continuously to the filtering and selection algorithms described in this paper. For example, a possible way to improve accuracy involves the combination of selected peer offsets and computing the ensemble offset using techniques suggested in [BLA74]. Another improvement involves the dynamic activation of selected peer associations when other peer associations become unreachable. The intent of this suggestion is to further reduce the polling overheads when a large number of possible peers are available, but only a few are needed for reliable synchronization.

At present, NTP support is available only for Fuzzball and Unix systems. Support is needed for other systems, including personal workstations of various manufacture. While NTP has been evolved within the Internet protocol suite, there is obvious application to the ISO protocol suite, in particular the protocols of the connectionless (CNLS) branch of that suite. Perhaps the most attractive methodology would be to integrate NTP functions directly into the ES-IS and IS-IS routing functions and network management systems.

8. References

[BER87] Bertsekas, D., and R. Gallager. *Data Networks*. Prentice-Hall, Englewood Cliffs, NJ, 1987.

[BLA74] Blair, B.E. (Ed.). *Time and Frequency Theory and Fundamentals*. National Bureau of Standards Monograph 140, U.S. Department of Commerce, 1974.

[BRA80] Braun, W.B. Short term frequency effects in networks of coupled oscillators. *IEEE Trans. Communications COM-28, 8* (August 1980), 1269-1275.

[COL88] Cole, R., and C. Foxcroft. An experiment in clock synchronisation. *The Computer Journal 31*, 6 (1988), 496-502.

[DAR81a] Defense Advanced Research Projects Agency. Internet Control Message Protocol. DARPA Network Working Group Report RFC-792, U.S.C. Information Sciences Institute, September 1981.

[DAR81b] Defense Advanced Research Projects Agency. Internet Protocol. DARPA Network Working Group Report RFC-791, U.S.C. Information Sciences Institute, September 1981.

[GUS84] Gusella, R., and S. Zatti. TEMPO - A network time controller for a distributed Berkeley UNIX system. *IEEE Distributed Processing Technical Committee Newsletter 6, NoSI-2* (June 1984), 7-15. Also in: *Proc. Summer 1984 USENIX* (Salt Lake City, June 1984).

[GUS85a] Gusella, R., and S. Zatti. The Berkeley UNIX 4.3BSD time synchronization protocol: protocol specification. Technical Report UCB/CSD 85/250, University of California, Berkeley, June 1985.

[GUS85b] Gusella, R., and S. Zatti. An election algorithm for a distributed clock synchronization program. Technical Report UCB/CSD 86/275, University of California, Berkeley, December 1985.

[HAL84] Halpern, J.Y., B. Simons, R. Strong and D. Dolly. Fault-tolerant clock synchronization. *Proc. Third Annual ACM Sympos. on Principles of Distributed Computing* (August 1984), 89-102.

[KOP87] Kopetz, H., and W. Ochsenreiter. Clock synchronization in distributed real-time systems. *IEEE Trans. Computers C-36,* 8 (August 1987), 933-939.

[LAM78] Lamport, L., Time, clocks and the ordering of events in a distributed system. *Comm. ACM 21, 7* (July 1978), 558-565.

[LAM85] Lamport, L., and P.M. Melliar-Smith. Synchronizing clocks in the presence of faults. *JACM 32, 1* (January 1985), 52-78.

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

[LUN84] Lundelius, J., and N.A. Lynch. A new fault-tolerant algorithm for clock synchronization. *Proc. Third Annual ACM Sympos. on Principles of Distributed Computing* (August 1984), 75-88.

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

[MIL81] Mills, D.L. Time Synchronization in DCNET Hosts. DARPA Internet Project Report IEN-173, COMSAT Laboratories, February 1981.

[MIL83a] Mills, D.L. Internet Delay Experiments. DARPA Network Working Group Report RFC-889, M/A-COM Linkabit, December 1983.

[MIL83b] Mills, D.L. DCN local-network protocols. DARPA Network Working Group Report RFC-891, M/A-COM Linkabit, December 1983.

[MIL85a] Mills, D.L. Algorithms for synchronizing network clocks. DARPA Network Working Group Report RFC-956, M/A-COM Linkabit, September 1985.

[MIL85b] Mills, D.L. Experiments in network clock synchronization. DARPA Network Working Group Report RFC-957, M/A-COM Linkabit, September 1985.

[MIL85c] Mills, D.L. Network Time Protocol (NTP). DARPA Network Working Group Report RFC-958, M/A-COM Linkabit, September 1985.

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

[MIL89a] Mills, D.L. Network Time Protocol (Version 2) specification and implementation. DARPA Network Working Group Report RFC-1119, University of Delaware, September 1989.

[MIL89b] Mills, D.L. Measured performance of the Network Time Protocol in the Internet system. DARPA Network Working Group Report RFC-1128, University of Delaware, October 1989.

[MIT80] Mitra, D. Network synchronization: analysis of a hybrid of master-slave and mutual synchronization. *IEEE Trans. Communications COM-28, 8* (August 1980), 1245-1259.

[NBS79] *Time and Frequency Dissemination Services.* NBS Special Publication 432, U.S. Department of Commerce, 1979.

[NBS88] *Automated Computer Time Service (ACTS).* NBS Research Material 8101, U.S. Department of Commerce, 1988.

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

[POS83a] Postel, J. Daytime protocol. DARPA Network Working Group Report RFC-867, USC Information Sciences Institute, May 1983.

[POS83b] Postel, J. Time protocol. DARPA Network Working Group Report RFC-868, USC Information Sciences Institute, May 1983.

[REF85] *Reference Data for Engineers: Radio, Electronics, Computer and Communications (Seventh Edition)*. Howard W. Sams, Indianapolis, IN, 1985.

[RIC88] Rickert, N.W. Non Byzantine clock synchronization - a programming experiment. *ACM Operating Systems Review 22, 1* (January 1988), 73-78.

[SCH86] Schneider, F.B. A paradigm for reliable clock synchronization. Department of Computer Science Technical Report TR 86-735, Cornell University, February 1986.

[SRI87] Srikanth, T.K., and S. Toueg. Optimal clock synchronization. *JACM 34, 3* (July 1987), 626-645.

[SU81] Su, Z. A specification of the Internet protocol (IP) timestamp option. DARPA Network Working Group Report RFC-781. SRI International, May 1981.

[TRI86] Tripathi, S.K., and S.H. Chang. ETempo: a clock synchronization algorithm for hierarchical LANs - implementation and measurements. Systems Research Center Technical Report TR-86-48, University of Maryland, 1986.

Security considerations

See Section 3.3.

Author's address

David L. Mills
Electrical Engineering Department
University of Delaware
Newark, DE 19716
Phone (302) 451-8247
EMail mills@udel.edu