Network Working Group David L. Mills
Request for Comments: 1119 University of Delaware
Obsoletes: RFC-1059, RFC-958 September 1989

Network Time Protocol (Version 2)
Specification and Implementation

Status of this Memo

This document describes the Network Time Protocol (NTP), specifies its formal structure and summarizes information useful for its implementation. NTP provides the mechanisms to synchronize time and coordinate time distribution in a large, diverse internet operating at rates from mundane to lightwave. It uses a returnable-time design 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 reference time via local routing algorithms and time daemons.

This is an Internet Standard Recommended Protocol. Distribution of this memo is unlimited.

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

Table of Contents

List of Figures

List of Tables

1. Introduction

This document constitutes a formal specification of the Network Time Protocol (NTP), which is used to synchronize timekeeping among a set of distributed time servers and clients. It defines the architectures, algorithms, entities and protocols used by NTP and is intended primarily for implementors. A companion document [44] summarizes the requirements, analytical models, algorithmic analysis and performance under typical Internet conditions. NTP was first described in RFC-958 [30], but has evolved in significant ways, culminating in the most recent NTP Version 1 described in RFC-1059 [42]. It is built on the Internet Protocol (IP) [14] and User Datagram Protocol (UDP) [9], which provide a connectionless transport mechanism; however, it is readily adaptable to other protocol suites. NTP is evolved from the Time Protocol [19] and the ICMP Timestamp message [15], but is specifically designed to maintain accuracy and robustness, even when used over typical Internet paths involving multiple gateways, highly dispersive delays and unreliable nets.

The service environment consists of the implementation model, service model and timescale described in Section 2. The implementation model is based on a multiple-process operating system architecture, although other architectures could be used as well. The service model is based on a returnable-time design which depends only on measured clock offsets, but does not require reliable message delivery. The synchronization subnet uses a self-organizing, hierarchical-master-slave configuration, with synchronization paths determined by a minimum-weight spanning tree. While multiple masters (primary servers) may exist, there is no requirement for an election protocol.

NTP itself is described in Section 3. It provides the protocol mechanisms to synchronize time in principle to precisions in the order of nanoseconds while preserving a non-ambiguous date well into the next century. The protocol includes provisions to specify the characteristics and estimate the error of the local clock and the time server to which it may be synchronized. It also includes provisions for operation with a number of mutually suspicious, hierarchically distributed primary reference sources such as radio clocks.

Section 4 describes algorithms useful for deglitching and smoothing clock-offset samples collected on a continuous basis. These algorithms evolved from with suggested in [28], were refined as the results of experiments described in [29] and further evolved under typical operating conditions over the last three years. In addition, as the result of experience in operating multiple-server subnets including radio-synchronized clocks at several sites in the U.S. and with clients in the U.S. and Europe, reliable algorithms for selecting good clocks from a population possibly including broken ones have been developed and are described in Section 4.

The accuracies achievable by NTP depend strongly on the precision of the local-clock hardware and stringent control of device and process latencies. Provisions must be included to adjust the software logical-clock time and frequency in response to corrections produced by NTP. Section 5 describes a local-clock design evolved from the Fuzzball implementation described in [21] and [43]. This design includes offset-slewing, drift-compensation and deglitching mechanisms capable of accuracies in the order of a millisecond, even after extended periods when synchronization to primary reference sources has been lost.

Details specific to NTP packet formats used with the Internet Protocol (IP) and User Datagram Protocol (UDP) are presented in Appendix A, while details of a suggested auxiliary NTP Control Message, which may be used when comprehensive network-monitoring facilities are not available, are presented in Appendix B. Appendix C contains specification and implementation details of an optional authentication mechanism which can be used to control access and prevent unauthorized data modification. Appendix D contains a listing of differences between Version 2 of NTP and previous versions.

1.1. Related Technology

Other mechanisms have been specified in the Internet protocol suite to record and transmit the time at which an event takes place, including the Daytime protocol [18], Time Protocol [19], ICMP Timestamp message [15] and IP Timestamp option [13]. Experimental results on measured times and roundtrip delays in the Internet are discussed in [20], [29], [41] and [42]. Other synchronization algorithms are discussed in [4], [22], [23], [24], [25], [27], [28], [29], [30], [31], [33], [35], [38], [39], [40], [42] and [44], while protocols based on them are described in [11], [12], [21], [26], [30], [35], [42] and [44]. NTP uses techniques evolved from them and both linear-systems and agreement methodologies. Linear methods for digital telephone network synchronization are summarized in [6], while agreement methods for clock synchronization are summarized in [25].

The Fuzzball routing protocol [21], sometimes called Hellospeak, incorporates time synchronization directly into the routing-protocol design. One or more processes synchronize to an external reference source, such as a radio clock or NTP daemon, and the routing algorithm constructs a minimum-weight spanning tree rooted on these processes. The clock offsets are then distributed along the arcs of the spanning tree to all processes in the system and the various process clocks corrected using the procedure described in Section 5 of this document. While it can be seen that the design of Hellospeak strongly influenced the design of NTP, Hellospeak itself is not an Internet protocol and is unsuited for use outside its local-net environment.

The Unix 4.3bsd time daemon *timed* [26] uses a single master-time daemon to measure offsets of a number of slave hosts and send periodic corrections to them. In this model the master is determined using an election algorithm [31] designed to avoid situations where either no master is elected or more than one master is elected. The election process requires a broadcast capability, which is not a ubiquitous feature of the Internet. While this model has been extended to support hierarchical configurations in which a slave on one network serves as a master on the other [35], the model requires handcrafted configuration tables in order to establish the hierarchy and avoid loops. In addition to the burdensome, but presumably infrequent, overheads of the election process, the offset measurement/correction process requires twice as many messages as NTP per update.

A scheme with features similar to NTP is described in [39]. This scheme is intended for multi-server LANs where each of a set of possibly many time servers determines its local-time offset relative to each of the other servers in the set using periodic timestamped messages, then determines the local-clock correction using the Fault-Tolerant Average (FTA) algorithm of [24]. The FTA algorithm, which is useful where up to k servers may be faulty, sorts the offsets, discards the k highest and lowest ones and averages the rest. The scheme, as described in [39], is most suitable to

LAN environments which support broadcast and would result in unacceptable overhead in an internet environment. In addition, for reasons given in Section 4 of this paper, the statistical properties of the FTA algorithm are not likely to be optimal in an internet environment with highly dispersive delays.

A good deal of research has gone into the issue of maintaining accurate time in a community where some clocks cannot be trusted. A *truechimer* is a clock that maintains timekeeping accuracy to a previously published (and trusted) standard, while a *falseticker* is a clock that does not. Determining whether a particular clock is a truechimer or falseticker is an interesting abstract problem which can be attacked using agreement methods summarized in [25] and [38].

A convergence function operates upon the offsets between the clocks in a system to increase the accuracy 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 [23], the CNV algorithm of [24], the majority-subset algorithm of [28], the non-Byzantine algorithm of [40], the egocentric algorithm of [33] and the algorithms in Section 4 of this document.

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 [23] and the optimum algorithm of [38]. However, these algorithms require large numbers 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 document.

In practice it is not possible to determine the truechimers from the falsetickers on other than a statistical basis, especially with hierarchical configurations and a statistically noisy Internet. Thus, the approach taken in this document and its predecessors involves mutually coupled oscillators and maximum-likelihood estimation and selection procedures. From the analytical point of view, the system of distributed NTP peers operates as a set of coupled phase-locked oscillators, with the update algorithm functioning as a phase detector and the local clock as a disciplined oscillator. This similarity is not accidental, since systems like this have been studied extensively [6], [7] and [8].

The particular choice of offset measurement and computation procedure described in Section 3 is a variant of the returnable-time system used in some digital telephone networks [6]. The clock filter and selection algorithms are designed so that the clock synchronization subnet self-organizes into a hierarchical-master-slave configuration [8]. What makes the NTP model unique is the adaptive configuration, polling, filtering and selection functions which tailor the dynamics of the system to fit the ubiquitous Internet environment.

2. System Architecture

The purpose of NTP is to connect a number of primary reference sources, synchronized to national standards by wire or radio, to widely accessible resources such as backbone gateways. These

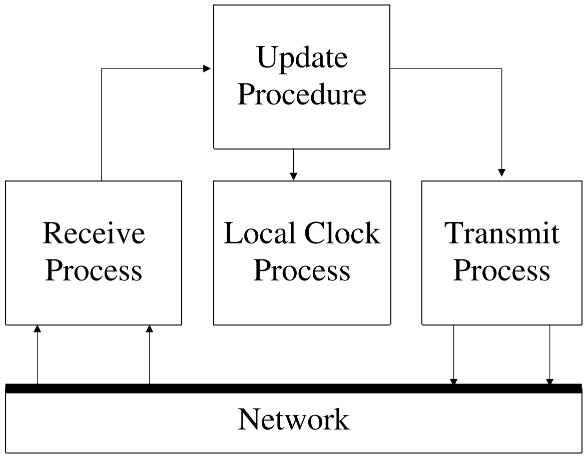

Figure 1. Implementation Model

gateways, acting as primary time servers, use NTP between them to cross-check the clocks and mitigate errors due to equipment or propagation failures. Some number of local-net hosts or gateways, acting as secondary time servers, run NTP with one or more of the primary servers. In order to reduce the protocol overhead, the secondary servers distribute time via NTP to the remaining local-net hosts. In the interest of reliability, selected hosts can be equipped with less accurate but less expensive radio clocks and used for backup in case of failure of the primary and/or secondary servers or communication paths between them.

There is no provision for peer discovery or virtual-circuit management in NTP. Data integrity is provided by the IP and UDP checksums. No circuit-management, duplicate-detection or retransmission facilities are provided or necessary. The service can operate in a symmetric mode, in which servers and clients are indistinguishable, yet maintain a small amount of state information, or in client/server mode, in which servers need maintain no state other than that contained in the client request. A lightweight association-management capability, including dynamic reachability and variable polling-rate mechanisms, is included only to manage the state information and reduce resource requirements. Since only a single NTP message format is used, the protocol is easily implemented and can be used in a variety of solicited or unsolicited polling mechanisms.

It should be recognized that clock synchronization requires by its nature long periods and multiple comparisons in order to maintain accurate timekeeping. While only a few measurements are usually adequate to reliably determine local time to within a second or so, periods of many hours and dozens of measurements are required to resolve oscillator drift and maintain local time to the order of a millisecond. Thus, the accuracy achieved is directly dependent on the time taken to achieve it. Fortunately, the frequency of measurements can be quite low and almost always non-intrusive to normal net operations.

2.1. Implementation Model

In what may be the most common client/server model a client sends an NTP message to one or more servers and processes the replies as received. The server interchanges addresses and ports, 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, as well as select the best from possibly several servers.

While the client/server model may suffice for use on local nets involving a public server and perhaps many 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.

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 an NTP 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 algorithms 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 procedure and adjusts the phase and frequency of the local clock using the mechanisms 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.

2.2. Network Configurations

The 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 [34], the accuracy of each server is defined by a number called its stratum, with the topmost level (primary servers) assigned as one and each level downwards (secondary servers) in the hierarchy assigned as one greater than the preceding level. With current technology and available timecode receivers, single-sample accuracies in the order of a millisecond can be achieved at the network interface of a primary server. Accuracies of this order require special care in the design and implementation of the operating system and the local-clock mechanism, such as described in Section 5.

As the stratum increases from one, the single-sample accuracies achievable will degrade depending on the network paths and local-clock stabilities. In order to avoid the tedious calculations [7] 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 [45], 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 [37] to compute the minimum-weight spanning trees rooted on the primary servers. With the foregoing factors in mind, the distance metric was chosen using the stratum number as the high-order bits and synchronizing distance as the low-order bits.

As a result of this design, the subnet reconfigures automatically in a hierarchical-master-slave configuration to produce the most accurate and reliable time, even when one or more primary or secondary servers or the network paths between them fail. This includes the case where all normal primary servers (e.g., highly accurate WWVB timecode receiver operating at the lowest synchronization distances) on a possibly partitioned subnet fail, but one or more backup primary servers (e.g., less accurate WWV receiver operating at higher synchronization distances) continue operation. However, should all primary servers throughout the subnet fail, the remaining secondary servers will synchronize among themselves while distances ratchet upwards to a preselected maximum infinity due to the well-known properties of the Bellman-Ford algorithm. Upon reaching the maximum on all paths, a server will drop off the subnet and free-run using its last determined time and frequency. Since these computations are expected to be very precise, especially in frequency, even extended outage periods should result in timekeeping errors not greater than a few milliseconds per day.

In the case of multiple primary servers, the spanning-tree computation will usually select the server at minimum synchronization distance. However, when these servers are at approximately the same distance, the computation may result in random selections among them as the result of normal dispersive delays. Ordinarily, this does not degrade accuracy as long as any discrepancy between the primary servers is small compared to the synchronization distance. If not, the filter and selection algorithms will select the best of the available servers and cast out outlyers as intended.

2.3. The NTP Timescale

For many years the most important use of time information was for worldwide navigation and space science, which depend on astronomical observations of the Sun, Moon and stars [32]. Sidereal time is based on the transit of stars across the celestial meridian of an observer. The mean sidereal day is 23 hours, 56 minutes and 4.09 seconds, but is not uniform due to variations in Earth orbit. Ephemeris time is based on tables with which a standard time interval such as the tropical year - one complete revolution of the Earth around the Sun - can be determined through observations of the Sun, Moon and planets. In 1958 the standard second was defined as 1/31,556,925.9747 of the tropical year that began this century. On this scale the tropical year is 365.2421987 days and the lunar month - one complete revolution of the Moon around the Earth - is 29.53059 days; however, the actual tropical year can be determined only to an accuracy of about 50 ms and has been increasing by about 5.3 ms per year.

In order to measure the span of the universe or the decay of the proton, it is necessary to have a standard day numbering plan. Accordingly, the International Astronomical Union has adopted the use of the standard second and Julian Day Number (JDN) to date cosmological events and related phenomena. The standard day consists of 86,400 standard seconds, where time is expressed as a fraction of the whole day, and the standard year consists of 365.25 standard days. In the scheme devised in 1583 by the French scholar Joseph Julius Scaliger and named after his father, Julius Caesar Scaliger, JDN 0.0 corresponds to 12^h (noon) on the first day of the Julian Era, 1 January 4713 BC. The years prior to the Christian Era (BC) are reckoned according to the Julian calendar, while the years of the Christian Era (AD) are reckoned according to the Gregorian calendar (see next section). Since there is no year zero or day zero and 1 BC is a leap year, JDN 1,721,426.0 corresponds to 12^h on the first day of the Christian Era, 1 January 1 AD. The Modified Julian Date (MJD), which is sometimes used to represent dates near our own era in conventional time and with fewer digits, is defined as MJD = JD - 2,400,000.5.

In 1967 the standard second was redefined 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 [1]. Since 1972 the time and frequency standards of the world have been based on International Atomic Time (TAI), which is defined in terms of the standard second and currently maintained using multiple cesium-beam clocks to an accuracy of a few parts in 10^{13}. The Bureau International de l'Heure (BIH) uses astronomical observations provided by the U.S. Naval Observatory and other observatories to determine Coordinated Universal Time (UTC). Starting from apparent mean solar time as observed, the UT0 timescale is determined using corrections for Earth orbit and inclination (the Equation of Time, as used by sundials), the UT1 (navigator's) timescale by adding corrections for polar migration and the UT2 timescale by adding corrections for known periodicity variations. While standard frequencies are based on TAI, conventional civil time is based on UT1, which is presently slowing relative to TAI by a fraction of a second per year. When the magnitude of

June 1972	Dec. 1972	Dec. 1973
Dec. 1974	Dec. 1975	Dec. 1976
Dec. 1977	Dec. 1978	Dec. 1979
June 1981	June 1982	June 1983
June 1985	Dec. 1987	

Table 1. Dates of Leap-Second Insertion

correction approaches 0.7 second, a leap second is inserted or deleted in the UTC timescale on the last day of June or December.

For the most precise coordination and timestamping of events since 1972, it is necessary to know when leap seconds are implemented in UTC and how the seconds are numbered. As specified in CCIR Report 517, which is reproduced in [1], a leap second is inserted following second 23:59:59 on the last day of June or December and becomes second 23:59:60 of that day. A leap second would be deleted by omitting second 23:59:59 on one of these days, although this has never happened. Leap seconds were inserted prior to 1 January 1989 on the occasions listed in Table 1 (courtesy U.S. Naval Observatory). Published BIH corrections consist not only of leap seconds, which result in step discontinuities relative to TAI, but 100-ms UT1 adjustments called DUT1, which provide increased accuracy for navigation and space science.

The NTP timescale is based on UTC. At 0^h 1 January 1972 (MJD 41,318.0) the NTP timescale was set to 2,272,060,800, representing the number of standard seconds since 0^h 1 January 1900 (MJD 15,021.0). The insertion of leap seconds in UTC does not affect the NTP oscillator itself, only the correspondence with conventional civil time. However, since the only institutional memory available to NTP is the UTC broadcast services, the NTP timescale is in effect reset to UTC as each offset estimate is computed. When a leap second is inserted in UTC and subsequently in NTP, knowledge of all previous leap seconds is lost. Thus, if a clock synchronized to NTP in early 1989 was used to establish the time of an event that occurred in early 1972 without correction, it would be fourteen seconds late.

2.4. The NTP Calendar

The calendar systems used in the ancient world reflect the agricultural, political and ritual needs characteristic of the societies in which they flourished. Astronomical observations to establish the winter and summer solstices were in use three to four millennia ago. By the 14th century BC the Shang Chinese had established the solar year as 365.25 days and the lunar month as 29.5 days. The lunisolar calendar, in which the ritual month is based on the Moon and the agricultural year on the Sun, was used throughout the ancient Near East (except Egypt) and Greece from the third millennium BC. Early calendars used either thirteen lunar months of 28 days or twelve alternating lunar months of 29 and 30 days and haphazard means to reconcile the 354/364-day lunar year with the 365-day vague solar year.

The ancient Egyptian lunisolar calendar had twelve 30-day lunar months, but was guided by the seasonal appearance of the star Sirius (Sothis). In order to reconcile this calendar with the solar year, a civil calendar was invented by adding five intercalary days for a total of 365 days. However, in time it was observed that the civil year was about one-fourth day shorter than the actual solar year and thus would precess relative to it over a 1460-year cycle called the Sothic cycle. Along with the Shang Chinese, the ancient Egyptians had thus established the solar year at 365.25 days, or within about 11 minutes of the present measured value. In 432 BC, about a century after the Chinese had done so, the Greek astronomer Meton calculated there were 110 29-day lunar months and 125 30-day lunar months for a total of 235 lunar months in 6940 solar days, or just over 19 years. The 19-year cycle, called the Metonic cycle, established the lunar month at 29.532 solar days, or within about two minutes of the present measured value.

The Roman republican calendar was based on a lunar year and by 50 BC was eight weeks out of step with the solar year. Julius Caesar invited the Alexandrian astronomer Sosigenes to redesign the calendar, which led to the adoption in 46 BC of the Julian calendar. This calendar is based on a year of 365.25 days and has an intercalary day inserted every four years. However, for the first 36 years an intercalary day was mistakenly inserted every three years instead of every four. The result was 12 intercalary days instead of nine, and a series of corrections that was not complete until 8 AD.

The seven-day Sumerian week was introduced only in the fourth century AD by Emperor Constantine I. During the Roman era a 15-year census cycle, called the Indiction cycle, was instituted for taxation purposes. The sequence of day-names for consecutive occurrences of a particular day of the year does not recur for 28 years, called the solar cycle. Thus, the least common multiple of the 28-year solar cycle, 19-year Metonic cycle and 15-year Indiction cycle results in a grand 7980-year supercycle called the Julian Era, which began in 4713 BC. A particular combination of the day of the week, day of the year, phase of the Moon and round of the census will recur beginning in 3268 AD.

By 1545 the discrepancy in the Julian year relative to the solar year had accumulated to ten days. In 1582, following suggestions by the astronomers Christopher Clavius and Luigi Lilio, Pope Gregory XIII issued a papal bull which decreed, among other things, that the solar year would consist of 365.2422 days. In order to more closely approximate the new value, only those centennial years divisible by 400 would be leap years, while the remaining centennial years would not, making the actual value 365.2425, or within about 26 seconds of the current measured value. While the Gregorian calendar is in use throughout most of the world today, some countries did not adopt it until early in the twentieth century.

While it remains a fascinating field for time historians, the above narrative provides conclusive evidence that conjugating calendar dates of significant events and assigning NTP timestamps to them is approximate at best. In principle, reliable dating of such events requires only an accurate count of the days relative to some globally alarming event, such as a comet passage or supernova explosion; however, only historically persistent and politically stable societies, such as the ancient Chinese and Egyptian, and especially the classic Maya, possessed the means and will to do so. Therefore, intercalary dating is considered beyond the scope of this specification and NTP is based solely on the Julian-Day numbering scheme.

2.5. Time and Frequency Dissemination

In order that atomic and civil time can be coordinated throughout the world, national administrations operate primary time and frequency standards and maintain TAI and UTC cooperatively by observing various radio broadcasts and through occasional use of portable atomic clocks. A primary frequency standard is an oscillator that can maintain extremely precise frequency relative to a physical phenomenon, such as a transition in the orbital states of an electron. Presently available atomic oscillators are based on the transitions of the hydrogen, cesium and rubidium atoms and are capable of maintaining reliable agreement to the order of 10^{-13} when operated in multiple ensembles at various national standards laboratories (see Section 5.1).

Most seafaring nations of the world operate some sort of broadcast time service for the purpose of calibrating chronographs, which are used in conjunction with ephemeris data to determine navigational position. In many countries the service is primitive and limited to seconds-pips broadcast by marine communication stations at certain hours. For instance, a chronograph error of one second represents a longitudinal position error of about 0.23 nautical mile at the Equator.

The U.S. National Institute of Standards and Technology (NIST - formerly National Bureau of Standards) operates three radio services for the distribution of primary time and frequency standard information. One of these uses high-frequency (HF or CCIR band 7) transmissions on frequencies of 2.5, 5, 10, 15 and 20 MHz from Fort Collins, CO (WWV), and Kauai, HI (WWVH). Signal propagation is usually by reflection from the upper ionospheric layers, which vary in height and composition throughout the day and season and result in unpredictable delay variations at the receiver. The timecode is transmitted over a 60-second interval at a data rate of 1 bps using a 100-Hz subcarrier on the broadcast signal. While these transmissions and those of Canada (CHU) and other countries can be received over large areas in the western hemisphere, reliable frequency comparisons can be made only to the order of 10^{-7} and time accuracies are limited to the order of a millisecond [1].

A second service operated by NIST is the low-frequency (LF or CCIR band 5) transmissions on 60 kHz from Boulder, CO (WWVB), which can be received over the continental U.S. and adjacent coastal areas. Signal propagation is via the lower ionospheric layers, which are relatively stable and have predictable diurnal variations in height. The timecode is transmitted over a 60-second interval at a rate of 1 pps using periodic reductions in carrier power. With appropriate receiving and averaging techniques and corrections for diurnal and seasonal propagation effects, frequency comparisons to within 10^{-11} are possible and time accuracies of from a few to 50 microseconds can be obtained [1]. However, there is only one station and it operates at modest power levels.

The third service operated by NIST uses ultra-high frequency (UHF or CCIR band 9) transmissions on 468 MHz from the Geosynchronous Orbiting Environmental Satellite (GOES). The timecode is interleaved with messages used to interrogate remote sensors and consists of 60 4-bit binary-coded decimal words transmitted over an interval of 30 seconds. The timecode information includes the UTC time of year, satellite position and UTC correction. There is some speculation on the continued operation of GOES, especially if the LORAN-C [16] and Global Positioning System (GPS) [17] radiopositioning systems operated by other U.S. agencies continue to evolve as expected. While the

OMEGA [3] radionavigation system operated by the U.S. Navy and other countries can in principle provide worldwide frequency and time distribution, this system is unlikely to long survive the operational deployment of GPS.

Note that the current formats used by NIST radio broadcast services [5] do not include provisions for advance notice of leap seconds, so this information must be determined from other sources. NTP includes provisions to distribute advance warnings of leap seconds using the leap-indicator bits described in Section 3. The protocol is designed so that these bits can be set manually at the primary time servers and then automatically distributed throughout the synchronization subnet to all other time servers as described in Section 5.

3. Network Time Protocol

This section consists of a formal definition of the Network Time Protocol, including its data formats, entities, state variables, events and event-processing procedures. The specification is based on the implementation model illustrated in Figure 1, but it is not intended that this model is 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, as well as to serve as a foundation for a more rigorous, comprehensive and verifiable specification.

3.1. Data Formats

All mathematical operations expressed or implied herein are in two's-complement, fixed-point arithmetic. Data are specified as integer or fixed-point quantities, with bits numbered from zero starting at the left, or high-order, position. Since various implementations may scale externally derived quantities for internal use, neither the precision nor decimal-point placement for fixed-point quantities is specified. Unless specified otherwise, all quantities are unsigned and may occupy the full field width with an implied zero preceding bit zero. Hardware and software packages designed to work with signed quantities will thus yield surprising results when the most significant (sign) bit is set. It is suggested that externally derived, unsigned fixed-point quantities such as timestamps be shifted right one bit for internal use, since the precision represented by the full field width is seldom justified.

Since NTP timestamps are cherished data and, in fact, represent the main product of the protocol, a special timestamp format has been established. NTP timestamps are represented as a 64-bit unsigned fixed-point number, in seconds relative to 0^h on 1 January 1900. The integer part is in the first 32 bits and the fraction part in the last 32 bits. This format allows convenient multiple-precision arithmetic and conversion to Time Protocol representation (seconds), but does complicate the conversion to ICMP Timestamp message representation (milliseconds). The precision of this representation is about 200 picoseconds, which should be adequate for even the most exotic requirements.

Timestamps are determined by copying the current value of the local clock to a timestamp when some significant event, such as the arrival of a message, occurs. In order to maintain the highest accuracy, it is important that this be done as close to the hardware or software driver associated with the event as possible. In particular, departure timestamps should be redetermined for each link-level retransmission. In some cases a particular timestamp 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.

Note that since some time in 1968 the most significant bit (bit 0 of the integer part) has been set and that the 64-bit field will overflow some time in 2036. Should NTP be in use in 2036, some external means will be necessary to qualify time relative to 1900 and time relative to 2036 (and other multiples of 136 years). Timestamped data requiring such qualification will be so precious that appropriate means should be readily available. There will exist an 200-picosecond interval, henceforth ignored, every 136 years when the 64-bit field will be zero and thus considered invalid.

3.2. State Variables and Parameters

Following is a summary of the various state variables and parameters used by the protocol. They are separated into classes of system variables, which relate to the operating system environment and local-clock mechanism; peer variables, which represent the state of the protocol machine specific to each peer; packet variables, which represent the contents of the NTP message; and parameters, which represent fixed configuration constants for all implementations of the current version. For each class the description of the variable is followed by its name and the procedure or value which controls it. Note that variables are in lower case, while parameters are in upper case. Additional details on formats and use are presented in later sections and Appendices.

3.2.1. Common Variables

The following variables are common to two or more of the system, peer and packet classes. Additional variables are specific to the optional authentication mechanism as described in Appendix C.

Peer Address (peer.srcadr, pkt.srcadr), Peer Port (peer.srcport, pkt.srcport): These are the 32-bit Internet address and 16-bit port number of the remote host.

Local Address (peer.dstadr, pkt.dstadr), Local Port (peer.dstport, pkt.dstport): These are the 32-bit Internet address and 16-bit port number of the local host. They are included among the state variables to support multi-homing.

Leap Indicator (sys.leap, peer.leap, pkt.leap): This is a two-bit code warning of an impending leap second to be inserted in the NTP timescale. The bits are set before 23:59 on the day of insertion and reset after 00:00 on the following day. This causes the number of seconds (rollover interval) in the day of insertion to be increased or decreased by one. In the case of primary servers the bits are set by operator intervention, while in the case of secondary servers the bits are set by the protocol. The two bits, bit 0 and bit 1, respectively, are coded as follows:

00	no warning
01	last minute has 61 seconds
10	last minute has 59 seconds)
11	alarm condition (clock not synchronized)

In all except the alarm condition (11_2), NTP itself does nothing with these bits, except pass them on to the time-conversion routines that are not part of NTP. The alarm condition occurs when, for whatever reason, the local clock is not synchronized, such as when first coming up or after an extended period when no outside reference source is available.

Mode (peer.hmode, pkt.pmode): This is an integer indicating the association mode, with values coded as follows:

0	unspecified
1	symmetric active
2	symmetric passive
3	client
4	server
5	broadcast
6	reserved for future NTP versions
7	reserved for private use

Stratum (sys.stratum, peer.stratum, pkt.stratum): This is an integer indicating the stratum of the local clock, with values defined as follows:

0	unspecified
1	primary reference (e.g., radio clock)
2-255	secondary reference (via NTP)

For comparison purposes a value of zero is considered greater than any other value. Note that the maximum value of the integer encoded as a packet variable is limited by the parameter NTP.INFIN, currently set to 15.

Peer Poll Interval (peer.ppoll, pkt.ppoll): This is a signed integer indicating the minimum interval between messages sent by the peer, in seconds as a power of two. For instance, a value of six indicates a minimum interval of 64 seconds.

Precision (sys.precision, peer.precision, pkt.precision): This is a signed integer indicating the precision of the local clock, in seconds to the nearest power of two. For instance, a 50-Hz or 60-Hz line-frequency clock would be assigned the value -6, while a 1000-Hz crystal-controlled clock would be assigned the value -10.

Synchronizing Distance (sys.distance, peer.distance, pkt.distance): This is a fixed-point number indicating the estimated roundtrip delay to the primary clock, in seconds.

Synchronizing Dispersion (sys.dispersion, peer.dispersion, pkt.dispersion): This is a fixed-point number indicating the estimated dispersion to the primary clock, in seconds.

Reference Clock Identifier (sys.refid, peer.refid, pkt.refid): This is a 32-bit code identifying the particular reference clock. In the case of stratum 0 (unspecified) or stratum 1 (primary reference),

System Variables	Name	Procedure
Leap Indicator	sys.leap	clock update
Stratum	sys.stratum	clock update
Precision	sys.precision	system
Synchronizing Distance	sys.distance	clock update
Synchronizing Dispersion	sys.dispersion	clock update
Reference Clock Identifier	sys.refid	clock update
Reference Timestamp	sys.reftime	clock update
Logical Clock	sys.clock	clock update
Clock Hold	sys.hold	clock update
Clock Source	sys.peer	selection

Table 2. System Variables

this is a four-octet, left-justified, zero-padded ASCII string, for example (see Appendix A for comprehensive list):

Stratum	Code	Meaning
0	DCN	DCN routing protocol
0	TSP	TSP time protocol
1	WWVB	WWVB LF (band 5) radio
1	GOES	GOES UHF (band 9) satellite
1	WWV	WWV HF (band 7)radio

In the case of type 2 and greater (secondary reference) this is the four-octet Internet address of the reference host.

Reference Timestamp (sys.reftime, peer.reftime, pkt.reftime): This is the local time, in timestamp format, when the local clock was last updated. If the local clock has never been synchronized, the value is zero.

Originate Timestamp (peer.org, pkt.org): This is the local time, in timestamp format, at the peer when its latest NTP message was sent. If the peer becomes unreachable the value is set to zero.

Receive Timestamp (peer.rec, pkt.rec): This is the local time, in timestamp format, when the latest NTP message from the peer arrived. If the peer becomes unreachable the value is set to zero.

Transmit Timestamp (peer.xmt, pkt.xmt): This is the local time, in timestamp format, at which the NTP message departed the sender.

3.2.2. System Variables

Table 2 shows the complete set of system variables. In addition to the common variables described previously, the following variables are used by the operating system in order to synchronize the local clock.

Peer Variables	Name	Procedure
Configured Bit	peer.config	initialization
Authentication Enabled Bit	peer.authenable	initialization
Authentication Bit	peer.authentic	receive
Peer Address	peer.srcadr	receive
Peer Port	peer.srcport	receive
Local Address	peer.dstadr	receive
Local Port	peer.dstport	receive
Leap Indicator	peer.leap	packet
Host Mode	peer.hmode	packet
Stratum	peer.stratum	packet
Peer Poll Interval	peer.ppoll	packet
Host Poll Interval	peer.hpoll	poll update
Precision	peer.precision	packet
Synchronizing Distance	peer.distance	packet
Synchronizing Dispersion	peer.dispersion	packet
Reference Clock Identifier	peer.refid	packet
Reference Timestamp	peer.reftime	packet
Originate Timestamp	peer.org	packet, clear
Receive Timestamp	peer.rec	packet, clear
Transmit Timestamp	peer.xmt	transmit, clear
Reachability Register	peer.reach	packet, transmit, clear
Valid Data Counter	peer.valid	packet, transmit, clear
Peer Timer	peer.timer	receive, transmit, poll update
Filter Register	peer.filter	filter, clear
Delay Estimate	peer.estdelay	filter
Offset Estimate	peer.estoffset	filter

Table 3. Peer Variables

Local Clock (sys.clock): This is the current local time, in timestamp format. Local time is derived from the hardware clock of the particular machine and increments at intervals depending on the design used. An appropriate design, including slewing and drift-compensation mechanisms, is described in Section 5.

Clock Hold (sys.hold): This is a counter used to avoid premature resetting of the local clock after the clock is reset to a new value, rather than being slewed gradually. Once set to a nonzero value, this counter decrements at one-second intervals until reaching zero, then stops.

Packet Variables	Name	Procedure
Peer Address	pkt.srcadr	transmit
Peer Port	pkt.srcport	transmit
Local Address	pkt.dstadr	transmit
Local Port	pkt.dstport	transmit
Leap Indicator	pkt.leap	transmit
Version Number	pkt.version	transmit
Peer Mode	pkt.pmode	transmit
Stratum	pkt.stratum	transmit
Peer Poll Interval	pkt.ppoll	transmit
Precision	pkt.precision	transmit
Synchronizing Distance	pkt.distance	transmit
Synchronizing Dispersion	pkt.dispersion	transmit
Reference Clock Identifier	pkt.refid	transmit
Reference Timestamp	pkt.reftime	transmit
Originate Timestamp	pkt.org	transmit
Receive Timestamp	pkt.rec	transmit

Table 4. Packet Variables

Clock Source (sys.peer): This is a selector identifying the current clock source. Usually this will be a pointer to a structure containing the peer variables.

3.2.3. Peer Variables

Table 3 shows the complete set of peer variables. In addition to the common variables described previously, the following variables are used by the peer management and measurement functions.

Configured Bit (peer.config): This is a bit indicating that the association was created from configuration information and should not be demobilized if the peer becomes unreachable.

Authentication Enabled Bit (peer.authenable): This is a bit indicating that the association is to operate in the authenticated mode. It may be set to one only if the optional authentication mechanism described in Appendix C is implemented.

Authenticated Bit (peer.authentic): This is a bit indicating that the last message received from the peer has been correctly authenticated. It may be set to one only if the optional authentication mechanism described in Appendix C is implemented.

Host Poll Interval (peer.hpoll): This is a signed integer used to indicate the interval between messages transmitted to the peer, in seconds as a power of two. For instance, a value of six indicates a minimum interval of 64 seconds.

Reachability Register (peer.reach): This is a shift register of NTP.WINDOW bits used to determine the reachability status of the peer, with bits entering from the least significant (rightmost) end.

Parameters	Name	Value
Version Number	NTP.VERSION	2
NTP Port	NTP.PORT	123
Max Stratum	NTP.INFIN	15
Max Clock Age	NTP.MAXAGE	86,400 sec
Max Skew	NTP.MAXSKW	.01 sec
Min Distance	NTP.MINDIST	.02 sec
Min Polling Interval	NTP.MINPOLL	6 (64 sec)
Max Polling Interval	NTP.MAXPOLL	10 (1024 sec)
Reachability Register Size	NTP.WINDOW	8
Max Select Weight	NTP.MAXWGT	8
Max Select Size	NTP.MAXLIST	5
Max Select Strata	NTP.MAXSTRA	2
Select Weight	NTP.SELECT	$^3/_4$
Filter Size	PEER.SHIFT	4 or 8

Table 5. Parameters

Valid Data Counter (peer.valid): This is an integer counter used to determine the interval between valid data updates.

Peer Timer (peer.timer): This is an integer counter used to control the interval between transmitted NTP messages.

3.2.4. Packet Variables

Table 4 shows the complete set of packet variables. In addition to the common variables described previously, the following variables are defined.

Version Number (pkt.version): This is an integer indicating the version number of the sender. NTP messages will always be sent with the current version number NTP.VERSION and will always be accepted if the version number matches NTP.VERSION. Exceptions may be advised on a case-by-case basis at times when the version number is changed. Specific guidelines for interoperation between this version and previous versions of NTP are summarized in Appendix D.

3.2.5. Clock Filter Variables

When the filter and selection algorithms suggested in Section 4 are used, the following state variables are defined in addition to the peer variables described previously. There is one set of these variables for every peer operating in an active mode (see below).

Filter Register (peer.filter): This is a shift register of PEER.SHIFT stages, where each stage stores a tuple consisting of the measured delay together with the measured offset associated with a single observation. Delay/offset observations enter from the least significant (rightmost) right and are shifted towards the most significant (leftmost) end and eventually discarded as new observations arrive. The register is cleared to zeros when (a) the peer becomes unreachable or (b) the local clock has just been reset so as to cause a significant discontinuity in local time.

Delay Estimate (peer.estdelay): This is a fixed-point number indicating the latest delay estimate output from the filter, in seconds.

Offset Estimate (peer.estoffset): This is a signed, fixed-point number indicating the latest offset estimate output from the filter, in seconds.

Dispersion Estimate (peer.estdisp): This is a fixed-point number indicating the latest dispersion estimate output from the filter, in seconds.

3.2.6. Parameters

Table 5 shows the parameters assumed for all implementations operating in the Internet system. It is necessary to agree on the values for these parameters in order to avoid unnecessary network overheads and stable peer associations.

Version Number (NTP.VERSION): This is the NTP version number, currently two (2).

NTP Port (NTP.PORT): This is the port number (123) assigned by the Internet Assigned Numbers Authority to NTP.

Maximum Strata (NTP.INFIN): This is the maximum stratum value that can be encoded as a packet variable, also interpreted as "infinity" or unreachable by the routing algorithm and currently set to 15. In some cases it may be desirable to set NTP.INFIN to a lower value in order to avoid long, unstable synchronizing chains.

Maximum Clock Age (NTP.MAXAGE): This is the maximum interval, in seconds, a reference clock will be considered valid after its last update, currently set to 86,400 seconds (one full day).

Maximum Skew (NTP.MAXSKW): This is the maximum allowance for the skew between the local clock and a peer clock over the maximum update interval determined by NTP.MAXPOLL (1024 seconds), currently set to .01 seconds.

Minimum Distance (NTP.MINDIST): This is the minimum synchronization distance between the host and a peer, currently set to .02 seconds.

Minimum Polling Interval (NTP.MINPOLL): This is the minimum polling interval, in seconds to the power of two, allowed by any peer of the Internet system, currently set to 6 (64 seconds).

Maximum Polling Interval (NTP.MAXPOLL): This is the maximum polling interval, in seconds to the power of two, allowed by any peer of the Internet system, currently set to 10 (1024 seconds).

Reachability Register Size (NTP.WINDOW): This is the size of the Reachability Register (peer.reach), currently set to 8.

Maximum Select Weight (NTP.MAXWGT): When the selection algorithm suggested in Section 4 is used, this is the maximum allowable dispersion, currently set to 8.

Maximum Select Size (NTP.MAXLIST): When the selection algorithm suggested in Section 4 is used, this is the maximum size of the selection list, currently set to 5.

Maximum Select Strata (NTP.MAXSTRA): When the selection algorithm suggested in Section 4 is used, this is the maximum number of strata represented in the selection list, currently set to 2.

Select Weight (NTP.SELECT): When the selection algorithm suggested in Section 4 is used, this is the weight used to compute the dispersion, currently set to $^3/_4$.

Filter Size (PEER.SHIFT): When the filter algorithm suggested in Section 4 is used, this is the size of the Clock Filter (peer.filter) shift register. For crystal-stabilized oscillators a value of 8 is suggested, while for mains-frequency oscillators a value of 4 is suggested. Additional considerations are given in Section 5.

Maximum Filter Dispersion (PEER.MAXDISP): When the filter algorithm suggested in Section 4 is used, this is the maximum dispersion, currently set to 64 seconds.

Filter Threshold (PEER.THRESHOLD): When the filter algorithm suggested in Section 4 is used, this is the threshold used to determine whether to increase or decrease the polling interval. While a value of $^1/_2$ is suggested, the value may be changed to suit local conditions on particular peer paths.

Filter Weight (PEER.FILTER): When the filter algorithm suggested in Section 4 is used, this is the weight used to discard noisy data. While a value of $^1/_2$ is suggested, the value may be changed to suit local conditions on particular peer paths.

3.3. 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, called an association. The association can operate in one of five modes as indicated by the host-mode variable (peer.hmode): symmetric active, symmetric passive, client, server and broadcast, which are defined as follows:

Symmetric Active (1): A host operating in this mode sends periodic messages regardless of the reachability state or stratum of its peer. By operating in this mode the host announces its willingness to synchronize and be synchronized by the peer.

Symmetric Passive (2): This type of association is ordinarily created upon arrival of a message from a peer operating in the symmetric active mode and persists only as long as the peer is reachable and operating at a stratum level less than or equal to the host; otherwise, the association is dissolved. However, the association will always persist until at least one message has been sent in reply. By operating in this mode the host announces its willingness to synchronize and be synchronized by the peer.

Client (3): A host operating in this mode sends periodic messages regardless of the reachability state or stratum of its peer. By operating in this mode the host, usually a LAN workstation, announces its willingness to be synchronized by, but not to synchronize the peer.

Server (4): This type of association is ordinarily created upon arrival of a client request message and exists only in order to reply to that request, after which the association is dissolved. By operating in this mode the host, usually a LAN time server, announces its willingness to synchronize, but not to be synchronized by the peer.

Broadcast (5): A host operating in this mode sends periodic messages regardless of the reachability state or stratum of the peers. By operating in this mode the host, usually a LAN time server operating on a high-speed broadcast medium, announces its willingness to synchronize all of the peers, but not to be synchronized by any of them..

The peer mode can be determined explicitly from the packet-mode variable (pkt.pmode) if it is nonzero and implicitly from the source port (pkt.srcport) and destination port (pkt.dstport) variables if it is zero. For the case where pkt.pmode is zero, included for compatibility with previous NTP versions, the peer mode is determined as follows:

pkt.srcport	pkt.dstport	Mode
NTP.PORT	NTP.PORT	symmetric active
NTP.PORT	not NTP.PORT	server
not NTP.PORT	NTP.PORT	client
not NTP.PORT	not NTP.PORT	not possible

Note that it is not possible in this case to distinguish between symmetric active and symmetric passive modes. Use of the pkt.pmode and NTP.PORT variables in this way is not recommended and may not be supported in future versions of the protocol.

A host operating in client mode occasionally sends an NTP message to a host operating in server mode, perhaps right after rebooting and at periodic intervals thereafter. The server responds by simply interchanging addresses and ports, filling in the required information and returning the message to the client. Servers need retain no state information between client requests, while clients are free to manage the intervals between sending NTP messages to suit local conditions. In these modes the protocol machine described in this document can be considerably simplified to a simple remote-procedure-call mechanism without significant loss of accuracy or robustness, especially when operating over high-speed LANs.

In the symmetric modes the client/server distinction (almost) disappears. Symmetric passive mode is intended for use by time servers operating near the root nodes (lowest stratum) of the synchronization subnet and with a relatively large number of peers on an intermittent basis. In this mode the identity of the peer need not be known in advance, since the association with its state variables is created only when an NTP message arrives. Furthermore, the state storage can be reused when the peer becomes unreachable or is operating at a higher stratum level and thus ineligible as a synchronization source.

Symmetric active mode is intended for use by time servers operating near the end nodes (highest stratum) of the synchronization subnet. 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 for every poll.

Normally, one peer operates in an active mode (symmetric active, client or broadcast modes) as configured by a startup file, while the other operates in a passive mode (symmetric passive or server modes), often without prior configuration. However, both peers can be configured to operate in the symmetric active mode. An error condition results when both peers operate in the same mode, but not symmetric active mode. In such cases each peer will ignore messages from the other, so that prior associations, if any, will be demobilized due to reachability failure.

Broadcast mode is intended for operation on high-speed LANs with numerous workstations and where the highest accuracies are not required. In the typical scenario one or more time servers on the LAN send periodic broadcasts to the workstations, which then determine the time on the basis of a preconfigured latency in the order of a few milliseconds. As in the client/server modes the protocol machine can be considerably simplified in this mode; however, a modified form of the clock selection algorithm may prove useful in cases where multiple time servers are used for enhanced reliability.

3.4. Event Processing

The significant events of interest in NTP occur upon expiration of a peer timer (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 section describes the procedures invoked when these events occur.

3.4.1. Transmit Procedure

The transmit procedure is called when the peer timer (peer.timer) decrements to zero, which can occur in all modes except server mode. First, an NTP message is constructed and sent as follows (see Appendix A for format). The IP and UDP packet variables are copied from the peer variables (note the interchange of source and destination addresses and ports):

pkt.srcadr ← peer.dstadr
pkt.srcport ← peer.dstport
pkt.dstadr ← peer.srcadr
pkt.dstport ← peer.srcport

Next, the NTP packet variables are copied (rescaled as necessary) from the system and peer variables:

pkt.leap ← sys.leap
pkt.version ← NTP.VERSION
pkt.pmode ← peer.hmode

pkt.stratum ← sys.stratum
pkt.ppoll ← peer.hpoll
pkt.precision ← sys.precision
pkt.distance ← sys.distance
pkt.dispersion ← sys.dispersion
pkt.refid ← sys.refid
pkt.reftime ← sys.reftime
pkt.org ← peer.org
pkt.rec ← peer.rec
pkt.xmt ← sys.clock
peer.xmt ← pkt.xmt

If message authentication is implemented the encrypt procedure (See Appendix C) is called to generate the authenticator, which follows the NTP message itself.

Note that the transmit timestamp (peer.xmt), which is updated at this time, will be used later in order to validate the reply; thus, implementations must take care to save the value actually transmitted. However, if message authentication is implemented it is likely that the time to compute the authentication information, which involves a crypto-checksum, can seriously affect the overall accuracy. Therefore, implementations should include a system state variable (not mentioned elsewhere in this document) which contains an offset calculated to match the expected time to compute this value and which is added to the transmit timestamp as obtained from the operating system. In addition, the order of copying the timestamps should be designed so that the time to perform the copy operations themselves does not degrade the measurement accuracy, which suggests that the variables be copied in the order shown.

Next, the reachability register (peer.reach) is shifted one position to the left, with zero replacing the vacated bit. If the reachability register (peer.reach) is zero and the association was not configured by the initialization procedure (peer.config bit set to zero), the association is demobilized and the transmit procedure exits. If peer.reach is zero and peer.config is set, the clear procedure is called to purge the clock filter and reselect the clock source, if necessary.

If the valid data counter (peer.valid) is less than two, it is incremented; otherwise, at least two timeout intervals have passed since valid data were shifted into the filter register. If the latter case the clock-filter procedure is called with zeros as the offset and delay arguments. Then, the clock-selection procedure is called to reselect the clock source, if necessary. If this results in a new clock source (sys.peer), the poll-update procedure is called for sys.peer with argument peer.hpoll, since the poll interval for the new clock source must be clamped at NTP.MINPOLL. Note that the zeros (undefined) argument will cause the computed dispersion to increase significantly and subsequently affect the poll interval and clock selection.

Next, the peer.timer is reinitialized with the value

peer.timer ← 1 << max[min(peer.ppoll, peer.hpoll, NTP.MAXPOLL), NTP.MINPOLL]

The host-poll variable (peer.hpoll) is then updated as follows. If the estimated-dispersion variable (peer.estdisp) is greater than the filter-threshold parameter (PEER.THRESHOLD, currently set to $^1/_2$), the poll-update procedure is called with argument peer.hpoll-1 to reduce its value by one; otherwise, that procedure is called with argument peer.hpoll+1 to increase its value by one.

3.4.2. Receive Procedure

The receive procedure is executed upon arrival of an NTP message. If the version number of the message (pkt.version) does not match the current version number (NTP.VERSION), the message is discarded and the procedure exits; however, exceptions may be advised on a case-by-case basis at times when the version number is changed. Next, the source and destination Internet addresses and ports in the IP and UDP headers are matched to the correct peer. If there is a match, processing continues at the next step below. If there is no match a new instantiation of the protocol machine is created and the association mobilized as follows:

peer.srcadr ← pkt.srcadr
peer.srcport ← pkt.srcport
peer.dstadr ← pkt.dstadr
peer.dstport ← pkt.dstport
peer.config ← 0
peer.authenable ← 0
peer.authentic ← (see below)
peer.hmode ← (see below)
peer.reach ← 0
peer.estdelay ← 0 (undefined)
peer.estoffset ← 0 (undefined)

If the optional authentication mechanism described in Appendix C is not implemented, the peer.authentic bit is ordinarily set to one, which allows non-preconfigured peers to become the clock source. If this bit is set to zero, a non-preconfigured peer cannot become the clock source, regardless of stratum or mode. If the mechanism is implemented, additional variables are initialized as described in Appendix C. The values of these variables are obtained using procedures beyond the scope of NTP itself. Ordinarily in this case the peer.authentic bit is set to zero, so that only properly authenticated peers can become the clock source.

If the message is from a peer operating in client mode (3), as determined in Section 3.3, the host mode (peer.hmode) is set to server mode (4); otherwise, it is set to symmetric passive mode (2). This may be modified in case the access-controls suggested in Section 3.5 are implemented. Finally, the clear procedure is called to initialize the remaining peer variables. Finally, the timer mechanism is armed and begins decrementing the peer timer (peer.timer).

If the authentication mechanism is implemented, the decrypt procedure (see Appendix C) is called to verify the authenticator and set the peer.authentic bit. Next, if pkt.pmode is nonzero, this becomes the value of the peer mode used in the following step. If pkt.pmode is zero, the peer is a previous NTP version and the peer mode is determined from the port numbers as described previously. Table

host→ peer↓	sym act 1	sym pas 2	client 3	server 4	bcst 5
sym act	recv	pkt	recv[2]	xmit[2]	xmit[1,2]
sym pas	recv	error	recv[2]	error	error
client	xmit[2]	xmit[2]	error	xmit	xmit[1]
server	recv[2]	error	recv	error	error
bcst	recv[1,2]	error	recv[1]	error	error

Notes:

1. A broadcast server responds directly to the client with pkt.org and pkt.rec containing correct values. At other times the server simply broadcasts the local time with pkt.org and pkt.rec set to zero.
2. Ordinarily, these mode combinations would not be used; however, within the limits of the specification, they would result in correct time.

Table 6. Modes and Actions

6 shows for each combination of peer mode and host mode (peer.hmode) the resulting action, which consists of one of the following steps, following which the receive procedure exits.

error: The packet is discarded. If the association was not configured by the initialization procedure (peer.config bit not set), the association is demobilized.

recv: The packet procedure is called. If any of the sanity checks fail, proceed in the error step. Otherwise, the low-order bit of the reachability register (peer.reach) is set (indicating the peer is reachable) and the packet is discarded.

xmit: The packet procedure is called (to latch the packet variables) and the packet discarded. Then, the poll-update procedure is called with argument peer.ppoll (to insure the reply has the proper value in the pkt.poll field). Finally, the transmit procedure is called (to send the packet and possibly demobilize the association).

pkt: The packet procedure is called. If any of the sanity checks fail, proceed in the xmit step. Otherwise, the low-order bit of the reachability register (peer.reach) is set (indicating the peer is reachable) and the packet is discarded.

3.4.3. Packet Procedure

The packet procedure checks the validity of the data, computes delay/offset samples and calls other procedures to select the peer and update the local clock. First, the following preliminary sanity checks are performed:

1. The transmit timestamp (pkt.xmt) must not match the last one received from the same peer (peer.org); if so, the message might be an old duplicate.

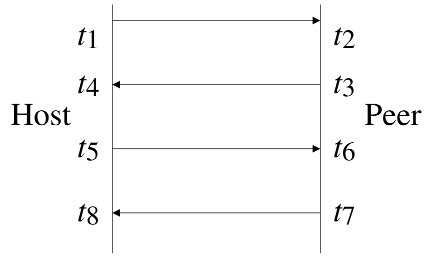

Figure 2. Calculating Delay and Offset

2. The originate timestamp (pkt.org) must match the last one sent to the same peer (peer.xmt); if not, the message might be out of order, bogus or worse.

If either of these checks fail, a sanity flag is set which will be tested later. Before proceeding further, the state variables are updated as follows:

$$
\begin{aligned}
\text{peer.leap} &\leftarrow \text{pkt.leap} \\
\text{peer.stratum} &\leftarrow \text{pkt.stratum} \\
\text{peer.ppoll} &\leftarrow \text{pkt.ppoll} \\
\text{peer.precision} &\leftarrow \text{pkt.precision} \\
\text{peer.distance} &\leftarrow \text{pkt.distance} \\
\text{peer.dispersion} &\leftarrow \text{pkt.dispersion} \\
\text{peer.refid} &\leftarrow \text{pkt.refid} \\
\text{peer.reftime} &\leftarrow \text{pkt.reftime} \\
\text{peer.org} &\leftarrow \text{pkt.xmt} \\
\text{peer.rec} &\leftarrow \text{sys.clock}
\end{aligned}
$$

At this point the poll-update procedure is called with argument peer.hpoll (the peer poll interval (peer.ppoll) may have changed). Then, the final sanity checks are performed:

3. The peer clock must be synchronized (peer.leap not equal to 11_2) and the interval since the peer clock was last updated satisfy

   $$\text{pkt.xmt} - \text{pkt.reftime} < \text{NTP.MAXAGE},$$

   where NTP.MAXAGE is currently set to 86,400 seconds (one full day).

4. The peer.authentic bit must be set to one, either as the result of initial configuration (receive or initialization procedures) or the decrypt procedure.

5. If the peer.config bit is not set, the host must be able to synchronize to the peer; otherwise the association will be demobilized; so, pkt.stratum must be not greater than sys.stratum.

If any of these checks fail, the sanity flag is set. If following all checks the sanity flag is set, the message is discarded and the packet procedure exits. In addition, both pkt.org and pkt.rec timestamps must be nonzero; if either is zero, the association has not synchronized or has lost reachability in one direction. In this case the packet procedure exits, but the sanity flag is not set. Note that, in the case where both pkt.org and pkt.rec are equal to zero and the peer is operating in broadcast mode on a high-speed LAN, the timestamps can optionally be used as-is. If the host happens to synchronize on the peer, the resulting systematic errors may be acceptable with or without further correction.

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 2 and let i be an even integer. Then t_{i-3}, t_{i-2}, t_{i-1} and t_i are the contents of the pkt.org, pkt.rec, pkt.xmt and peer.rec variables respectively. The roundtrip delay d_i and clock offset c_i of the receiving host relative to the sending peer is:

$$d_i = (t_i - t_{i-3}) - (t_{i-1} - t_{i-2}) \ ,$$
$$c_i = \frac{(t_{i-2} - t_{i-3}) + (t_{i-1} - t_i)}{2} \ .$$

This method amounts to a continuously sampled, returnable-time system, which is used in some digital telephone networks [34]. Among the advantages are that the order and timing of the messages is unimportant and that reliable delivery is not required. Obviously, the accuracies achievable depend upon the statistical properties of the outbound and inbound data paths. Further analysis and experimental results bearing on this issue can be found in [44].

In systems involving high-speed LANs, it may happen that, due to small differences in frequency and precision between the host and peer clocks, the delay d_i may appear to be not greater than zero (possibly negative). Let δ_i be the maximum error accumlated over the longest update interval due to the different rates and precisions of the local oscillators involved:

$$\delta_i = (1 \ll \text{sys.precision}) + (1 \ll \text{peer.precision}) + \text{NTP.MAXSKW} \ .$$

Then, $d_i = d_i + \delta_i$. If d_i is still not greater than zero after this adjustment, the sample is discarded and the packet procedure exits. If d_i is greater than zero it is advisable to clamp it to a minimum value to prevent route flaps that can happen with Bellman-Ford algorithms and large delay dispersions. The following adjustment to d_i reduces (but cannot eliminate) this problem:

$$di = \max(d_i, \text{NTP.MINDIST}) \ .$$

Appropriate values for NTP.MAXSKW and NTP.MINDIST are given in Table 5. These values may have to be altered for special circumstances.

If processing reaches this point the received timestamps are considered valid and peer.valid is set to zero. The clock-filter procedure is then called with c_i and the adjusted d_i as arguments to produce the delay estimate (peer.estdelay), offset estimate (peer.estoffset) and dispersion estimate (peer.estdisp) for the peer involved. Specification of the clock filter algorithm is not an integral part of the NTP specification; however, one found to work well in the Internet environment is described in Section 4. Finally, the clock-update procedure is called to reselect the clock source, if necessary, and update the local clock.

3.4.4. Primary-clock procedure

When a primary clock is connected to the host, it is convenient to incorporate its information into the data base as if the clock were represented as an ordinary peer. The clock can be polled once a minute or so and the returned timecheck used to produce a new update for the local clock. The following peer variables can be established upon instantiation of the protocol machine for the clock:

peer.leap ← selected by operator
peer.stratum ← 0
peer.ppoll ← NTP.MAXPOLL
peer.hpoll ← as configured
peer.distance ← (see below)
peer.dispersion ← 0
peer.srcadr ← (see below)
peer.estdelay ← 0
peer.estdisp ← 0

In this case the peer.distance and peer.srcadr can be constants reflecting the accuracy and type of the clock, respectively. By convention, the value for peer.distance, which will become the value of sys.distance when the clock-update procedure is called, is ten times the expected mean error of the clock, for instance, 100 milliseconds for a WWVB clock and 1000 milliseconds for a less accurate WWV clock. Also, the value for peer.srcadr, which will become the value of sys.refid when the clock-update procedure is called, is set to an ASCII string describing the clock type (see Appendix A).

When a valid timecode is received from the clock it is converted to internal timecheck form and the following variables set:

peer.rec ← timecheck
peer.estoffset ← timecheck - sys.clock

Finally, the clock-update procedure is called to reselect the clock source, if necessary, and update the local clock.

Since current broadcast time formats do not include advance notice of leap seconds, it may happen that a leap second is correctly incorporated in the local timescale, but the radio clock may continue at its old timescale until resynchronized. To avoid disruptions when a leap-second event occurs, the clock should be disabled for some interval, depending on the clock type, until it has reliably resynchronized to the broadcast signal.

With a little ingenuity the unused peer variables can be converted to control the clock polling interval, to determine its operating condition and even to use the clock-filter procedure in the usual way to improve its accuracy.

3.4.5. Clock-update procedure

The clock-update procedure is called for a selected peer when a new delay/offset estimate is available. First, the clock-selection procedure is called to determine the best peer on the basis of estimated accuracy and reliability. If this results in a new clock source (sys.peer), the poll-update procedure is called for sys.peer with argument peer.hpoll, since the poll interval for the new clock source must be clamped at NTP.MINPOLL. If sys.hold is nonzero or sys.peer is not the selected peer in the procedure call, the procedure exits. Otherwise, the state variables of the selected peer are used to update the system state variables as follows:

$$\begin{gathered}\text{sys.leap} \leftarrow \text{peer.leap} \\ \text{sys.stratum} \leftarrow \text{peer.stratum} + 1 \\ \text{sys.distance} \leftarrow \text{peer.distance} + \text{peer.estdelay} \\ \text{sys.dispersion} \leftarrow \text{peer.dispersion} + \text{peer.estdisp} \\ \text{sys.refid} \leftarrow \text{peer.srcadr} \\ \text{sys.reftime} \leftarrow \text{peer.rec}\end{gathered}$$

Finally, the local-clock procedure is called with peer.estoffset as argument to update the local clock (sys.clock). It may happen that the local clock may be reset, rather than slewed to its final value. In this case 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. In addition, the system variable sys.hold is set to

$$\text{sys.hold} \leftarrow \text{PEER.SHIFT} * (1 \ll \text{NTP.MINPOLL})$$

and subsequently decrements at one-second intervals to zero. This is necessary so that the local clock will not be updated until all clock filters fill up again and the dispersions settle down.

Specification of the clock selection and local-clock algorithms is not an integral part of the NTP specification. A clock selection algorithm found to work well in the Internet environment is described in Section 4, while a local-clock algorithm is described in Section 5. The clock selection algorithm described in Section 4 usually picks the server at the lowest stratum and minimum delay among all those available, unless that server appears to be a falseticker. The result is that the algorithms all work to build a minimum-weight spanning tree relative to the primary servers and thus a hierarchical-master-slave synchronization subnet.

The basic NTP robustness model is that a host has no other means to verify time other than NTP itself. In some equipment a battery-backed clock/calendar is available for a sanity check. In the common assumption (not always justified) that the clock/calendar is more reliable, but less accurate, than the NTP synchronization subnet, the system clock can be set upon reboot by the clock/calendar. Subsequent corrections can be determined by NTP as available, but only if the adjusted clock remains within a preconfigured range of the clock/calendar. When this is done an operator alarm should be signalled if the adjustment is determined to be out of this range.

3.4.6. Initialization Procedure

Upon reboot the NTP host initializes all system variables as follows:

$$\begin{gathered}\text{sys.leap} \leftarrow 11_2 \text{ (unsynchronized)} \\ \text{sys.stratum} \leftarrow 0 \text{ (undefined)} \\ \text{sys.precision} \leftarrow \text{as required} \\ \text{sys.distance} \leftarrow 0 \text{ (undefined)} \\ \text{sys.dispersion} \leftarrow 0 \text{ (undefined)} \\ \text{sys.refid} \leftarrow 0 \text{ (undefined)}\end{gathered}$$

sys.reftime ← 0 (undefined)
sys.clock ← best available estimate
sys.hold ← 0
sys.peer ← NULL

The local clock (sys.clock) is presumably undefined at reboot; however, in some equipment an estimate is available from the reboot environment, such as a battery-backed clock/calendar. The precision variable (sys.precision) is determined by the intrinsic architecture of the local hardware clock.

Next, an implementation-specific procedure is called repeatedly to mobilize a set of associations as required. The modes and addresses of these peers are determined using information read during the reboot procedure or as the result of operator commands:

peer.srcadr ← peer IP address
peer.srcport ← peer UDP port
peer.dstadr ← host IP address
peer.dstport ← host UDP port (NTP.PORT)
peer.config ← 1
peer.authenable ← (see below)
peer.authentic ← (see below)
peer.hmode ← (as required)

If the optional authentication mechanism described in Appendix C is not implemented, the peer.authenable bit is set to zero and the peer.authentic bit ordinarily set to one, which allows preconfigured peers to become the clock source. If peer.authentic bit is set to zero, a preconfigured peer cannot become the clock source, regardless of stratum or mode. If the mechanism is implemented, additional variables are initialized as described in Appendix C. The values of these variables are obtained using procedures beyond the scope of NTP itself. Ordinarily in this case the peer.authenable bit is set to one and the peer.authentic bit to zero, so that only properly authenticated peers can become the clock source.

FInally, the clear procedure is called to initialize the remaining peer variables and the timer mechanism is armed and begins decrementing the peer timer (peer.timer).

3.4.7. Clear Procedure

The clear procedure is called when some event occurs that results in a significant change in reachability state or potential disruption of the local clock. The peer variables are updated as follows:

peer.hpoll ← NTP.MINPOLL
peer.estdisp ← PEER.MAXDISP
peer.filter ← 0 (all stages undefined)
peer.valid ← 0
peer.org ← 0

$$\text{peer.rec} \leftarrow 0$$
$$\text{peer.xmt} \leftarrow 0$$

Next, the poll-update procedure is called with argument peer.hpoll to reset the peer timer. Finally, the clock-selection procedure is called. If this results in a new clock source (sys.peer), the poll-update procedure is called for sys.peer with argument peer.hpoll, since the poll interval for the new clock source must be clamped at NTP.MINPOLL.

3.4.8. Poll-update procedure

The poll-update procedure is called when a significant event occurs that may result in a change of the host-poll variable (peer.hpoll) or peer timer (peer.timer). The new value requested in the argument is first clamped to the range NTP.MINPOLL and NTP.MAXPOLL. If the peer happens to be the current clock source (sys.peer) or if the local clock is not crystal controlled, the new value is clamped to NTP.MINPOLL in order to avoid instabilities that can occur with larger values. Next, compute the new poll interval in seconds:

$$\text{temp} = 1 \ll \max[\min(\text{peer.ppoll}, \text{peer.hpoll}, \text{NTP.MAXPOLL}), \text{NTP.MINPOLL}].$$

If temp is equal to peer.timer, no change is necessary and the procedure exits. If temp is less than peer.timer, peer.timer must be clamped to that value:

$$\text{peer.timer} \leftarrow \text{temp}.$$

When the poll interval is changed and possibly large numbers of peers are involved, it is important to discourage tendencies to synchronize transmissions between the peers. A prudent preventative is to randomize the first transmission after the polling interval is changed:

$$\text{peer.timer} \leftarrow \text{peer.timer} * \text{ran}(),$$

where ran() is the system function that produces random values over the interval $0 \le x < 1$.

3.5. Access Control Issues

The NTP design is such that accidental or malicious data modification (tampering) or destruction (jamming) at a time server should not in general result in timekeeping errors elsewhere in the synchronization subnet. However, the success of this approach depends on redundant time servers and diverse network paths, together with the assumption that tampering or jamming will not occur at many time servers throughout the synchronization subnet at the same time. In principle, the subnet vulnerability can be engineered through the selection of time servers known to be trusted and allowing only those time servers to become the clock source. The authentication procedures described in Appendix C represent one mechanism to enforce this; however, the encryption algorithms can be quite CPU-intensive and can seriously degrade accuracy, unless precautions such as mentioned in the description of the timeout procedure are taken.

While not a required feature of NTP itself, some implementations may include an access-control feature that prevents unauthorized access and controls which peers are allowed to update the local clock. For this purpose it is useful to distinguish between three categories of access: those that are preauthorized as trusted, preauthorized as friendly and all other (non-preauthorized) accesses. Presumably, preauthorization is accomplished by entries in the configuration file or some kind of ticket-management system such as Kerberos. In this model only trusted accesses can result in the peer becoming the clock source. While friendly accesses cannot result in the peer becoming the clock source, NTP messages and timestamps are returned as specified.

It does not seem useful to maintain a secret clock, as would result from restricting non-preauthorized accesses, unless the intent is to hide the existence of the time server itself. Well-behaved Internet hosts should always return an ICMP error message if the service or resources are unavailable; however, in the case of NTP the resources required are minimal, so there is little need to restrict requests intended only to read the clock. A simple but effective access-control mechanism is then to consider all associations preconfigured in a symmetric mode or client mode (modes 1, 2 and 3) as trusted and all other associations, preconfigured or not, as friendly.

If a more comprehensive trust model is required, the design can be based on an access-control list with each entry consisting of a 32-bit Internet address, 32-bit mask and three-bit mode. If the logical AND of the source address (pkt.srcadr) and the mask in an entry matches the corresponding address in the entry and the mode (pkt.mode) matches the mode in the entry, the access is allowed; otherwise an ICMP error message is returned to the requestor. Through appropriate choice of mask, it is possible to restrict requests by mode to individual addresses, a particular subnet or net addresses, or have no restriction at all. The access-control list would then serve as a filter controlling which peers could create associations.

4. Filtering and Selection Algorithms

A very important factor affecting the accuracy and reliability of time distribution is the complex of algorithms used to deglitch and smooth the offset estimates and to cast out outlyers due to failure of the primary reference sources or propagation media. The algorithms suggested in this section were developed and refined over several years of operation in the Internet under widely varying net configurations and utilizations. While these algorithms are believed the best available at the present time, they are not an integral part of the NTP specification. A comprehensive discussion of the design principles and performance is given in [44].

There are two procedures described in the following, the clock-filter procedure, which is used to select the best offset samples from a given clock, and the clock-selection procedure, which is used to select the best clock among a hierarchical set of clocks.

4.1. Clock-filter procedure

The clock-filter procedure is executed upon arrival of an NTP message or other event that results in new delay/offset sample pairs. The filter register (peer.filter) is initialized with zeros by the clear procedure. New sample pairs are shifted into peer.filter from the left end, causing first zeros then old sample pairs to shift off the right end. The transmit procedure will also shift zeros into peer.filter when two polling intervals elapse without a fresh update. Then those sample pairs in peer.filter with nonzero delay are inserted on a temporary list and sorted in order of increasing delay. The delay estimate (peer.estdelay) and offset estimate (peer.estoffset) are chosen as the delay/offset values corresponding to the minimum-delay sample. In case of ties an arbitrary choice is made.

The dispersion estimate (peer.estdisp) is then computed as the weighted sum of the offsets in the list. Assume the list has n = PEER.SHIFT entries, the first m of which contain valid samples in order of increasing delay. If X_i ($0 \leq i < m$) is the offset of the ith sample, then compute the values

$$d_i = | X_i - X_0 | \text{ if } i < m \text{ and } | X_i - X_0 | < \text{PEER.MAXDISP};$$
$$d_i = \text{PEER.MAXDISP otherwise,}$$

$$\text{peer.estdisp} = \sum_{i=0}^{n-1} d_i \, w^{\,i}$$

where $w < 1$ is a weighting factor (also called the PEER.FILTER parameter), currently set to $^1/_2$, experimentally adjusted to match typical offset distributions. The peer.estdisp variable is intended for use as a quality indicator, with increasing values associated with decreasing quality and given less weight in the clock selection process.

The peer.estdisp variable is used in the transmit procedure to determine whether to increase or decrease the polling interval. If peer.estdisp is greater than the PEER.THRESHOLD parameter, the path quality is deteriorating and the polling interval is decreased; otherwise, the polling interval is increased. The peer.estdisp variable is also used in the clock-selection procedure in conjunction with the peer.dispersion variable as a means to select candidate peers for clock synchronization.

4.2. Clock-selection procedure

The clock-selection procedure uses the values of delay (peer.estdelay), offset (peer.estoffset) and dispersion (peer.estdisp) calculated by the clock-filter procedure for each peer and is called when these values change or when the reachability status changes. It constructs a list of candidate servers, then sorts it in order of estimated robustness and trims it to manageable size. Next, it sorts the list in order of estimated accuracy and repeatedly casts out outlyers on the basis of dispersion until only the most reliable, most accurate candidates are left. The only output from this procedure is the system variable sys.peer, which is set as a pointer to a surviving candidate, if there is one, or to the NULL value if not.

The first subprocedure begins by constructing a list of candidates sorted first by peer.stratum and then by the sum of peer.estdisp, peer.dispersion and δ (see Section 3.4.3). The only criteria for membership on this list is that the peer must pass certain sanity checks:

1. The variable peer.stratum must be greater than zero and less than the maximum that can be encoded as a packet variable (NTP.INFIN).

2. If peer.stratum is greater than one (synchronized by NTP), peer.refid must not match peer.dstadr; otherwise, a synchronization loop would be formed.

3. Both peer.estdelay and peer.estdisp must be less than NTP.MAXWGT (currently 8), which insures that the filter register is at least half full, yet avoids using data from very noisy associations or broken implementations.

If no candidates survive the above sanity checks, the current clock source (sys.peer) is set to NULL and the clock-selection procedure exits.

The list is then pruned from the end to be no longer than the maximum select size parameter (NTP.MAXLIST), currently set to five; however, the current clock source is not pruned from the list, regardless of position. This feature minimizes clock wander (see below) on high-speed, multiple-server networks. Starting from the beginning the list is truncated at the first entry where the number of different strata in the list exceeds the maximum select strata parameter (NTP.MAXSTRA), currently set to two. These rules have 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 next subprocedure is designed to detect falsetickers or other conditions which might result in gross errors. This is done by repeatedly casting out outlyers which exhibit significant dispersions relative to the other members of the list. However, indiscriminately casting out outlyers beyond a limit set by the intrinsic precisions of the local clocks can result in wandering among the servers without producing meaningful improvements in reliability or accuracy, especially on high-speed LANs using several redundant time servers with similar delays, offsets and dispersions. In addition, wandering causes needless network overhead, since the poll interval is clamped at NTP.MINPOLL for each server selected and only slowly increases when the server is no longer selected.

The subprocedure is designed to strike a balance between falseticker discrimination, accuracy optimization and wander avoidance. First, the candidate list is resorted in the order first by peer.stratum and then by peer.estdelay. Let m be the number of candidates remaining in the list and for the ith candidate let X_i be the value of peer.estoffset and δ_i as described in Section 3.4.3. For each i $(0 \le i < m)$ compute the dispersion d_i of the list relative to i:

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

where $v < 1$ is a weighting factor experimentally adjusted for the desired characteristic (see [44]). Note that d_i represents the actual dispersion of the ith candidate, while the skew δ_i represents a dispersion limit below which increased accuracy is doubtful and increased wander is likely. If

$$\max_{i=0}^{m-1}(d_i) > \min_{i=0}^{m-1}(\delta_i) ,$$

then cast out the candidate with maximum d_i or, in case of ties, the maximum i, and repeat the subprocedure; otherwise, stop. If the current clock source (sys.peer) is one of the surviving candidates in the list and there is no other surviving candidate of lower stratum, then simply exit

Oscillator type	Stability (per day)	Drift /Aging (per day)
Hydrogen maser	2 x 10^{-14}	1 x 10^{-12}/yr
Cesium beam	3 x 10^{-13}	3 x 10^{-12}/yr
Rubidium gas cell	5 x 10^{-12}	3 x 10^{-11}/mo
Oven-controlled crystal	1 x 10^{-9} 0-50 deg C	1 x 10^{-10}
Digital-comp crystal	5 x 10^{-8} 0-60 deg C	1 x 10^{-9}
Temp-compensated crystal	5 x 10^{-7} 0-60 deg C	3 x 10^{-9}
Uncompensated crystal	~1 x 10^{-6} per deg C	don't ask

Table 7. Characteristics of Standard Oscillators

the clock-selection procedure without doing anything further. Otherwise, set sys.peer to point to the first candidate in the list and exit the clock-selection procedure.

This subprocedure is designed to favor those candidates near the head of the list, which are at the lowest stratum and lowest delay and presumably can provide the most precise time. With proper selection of weighting factor v (also called NTP.SELECT), currently set to $^3/_4$, entries will be trimmed from the tail of the list, unless a few outlyers disagree significantly with respect to the remaining entries, in which case the outlyers are discarded first. The termination condition is designed to avoid needless switching between clock sources when not statistically justified, yet maintain a bias toward the low-stratum, low-delay peers. In some situations, such as multiple broadcast servers on a high-capacity LAN, it may be useful to fine-tune the termination condition even more, such as always permitting a switch to the first candidate in the list if that peer is already operating at a host-poll interval of NTP.MINPOLL.

5. Local Clocks

In order to implement a precise and accurate local clock, the host must be equipped with a hardware clock consisting of an oscillator and interface and capable of the required precision and stability. A logical clock is then constructed using these components plus software components that adjust the apparent time and frequency in response to periodic corrections computed by NTP or some other time-synchronization protocol such as Hellospeak [21] or the Unix 4.3bsd TSP [26]. This section includes a summary of the characteristics of various standard oscillators, followed by the mathematical model of a method to synchronize an oscillator to a reference standard. The section concludes with a description of the Fuzzball logical-clock model and implementation, which includes provisions for precise time and frequency adjustment and can maintain time to within a millisecond and frequency to within a millisecond per day.

5.1. Standard Oscillators

As mentioned previously, a primary frequency standard is an oscillator that can maintain extremely precise frequency relative to a physical phenomenon, such as a transition in the orbital states of an electron. Presently available atomic oscillators are based on the transitions of the hydrogen, cesium and rubidium atoms. Table 7 shows the characteristics for typical oscillators of these types compared

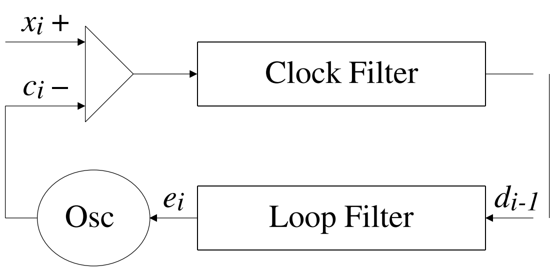

Figure 3. Phase-Lock Loop Model

with those for various types of quartz-crystal oscillators found in electronic equipment. For reasons of cost and robustness cesium oscillators are used worldwide for national primary frequency standards. On the other hand, local clocks used in computing equipment almost always are designed with uncompensated crystal oscillators.

For the three atomic oscillators listed in Table 7 the Drift/Aging column shows the maximum offset per day from nominal standard frequency due to systematic mechanical and electrical characteristics. In the case of crystal oscillators this offset is not constant, which results in a gradual change in frequency with time, called aging. Even if a crystal oscillator is temperature compensated by some means, it must be periodically compared to a primary standard in order to maintain the highest accuracy. For all types of oscillators the Stability column shows the maximum variation in frequency per day due to circuit noise and environmental factors.

As the telephone networks of the world are evolving rapidly to digital technology, consideration should be given to the methods used for frequency synchronization in digital networks. The industry has agreed on a classification of clock oscillators as a function of minimum accuracy, minimum stability and other factors [36]. There are three factors which determine the classification: stability, jitter and wander. Stability refers to the systematic variation of frequency with time and is synonymous with aging, drift, trends, etc. Jitter (also called timing jitter) refers to short-term variations in frequency with components greater than 10 Hz, while wander refers to long-term variations in frequency with components less than 10 Hz. The classification determines the oscillator stratum (not to be confused with the NTP stratum), with the more accurate oscillators assigned the lower strata and less accurate oscillators the higher strata:

Stratum	Min Accuracy (per day)	Min Stability (per day)
1	1×10^{-11}	not specified
2	1.6×10^{-8}	1×10^{-10}
3	4.6×10^{-6}	3.7×10^{-7}
4	3.2×10^{-5}	not specified

The construction, operation and maintenance of stratum-one oscillators is assumed to be consistent with national standards and often includes cesium oscillators or precision crystal oscillators synchronized via LORAN-C to NIST standards. Stratum-two oscillators represent the stability required for interexchange toll switches such as the AT&T 4ESS and interexchange digital cross-connect systems, while stratum-three oscillators represent the stability required for exchange switches such as the AT&T 5ESS and local cross-connect systems. Stratum-four oscillators represent the stability required for digital channel-banks and PBX systems.

5.2. Mathematical Model

The Fuzzball logical-clock model, which is shown in Figure 3, 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 $(i = 0, 1, ...)$ 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

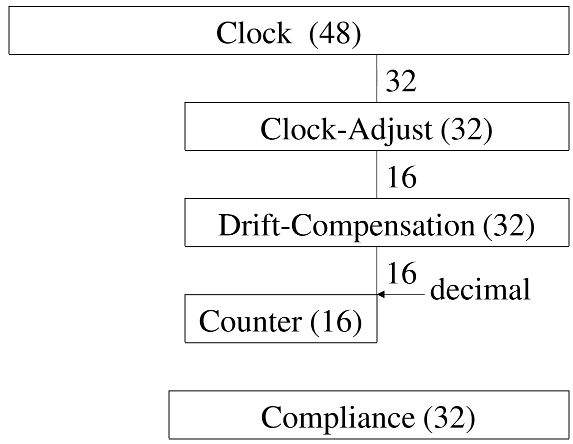

Figure 4. Clock Registers

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 in Section 4 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

Parameter	Name	Crystal	Mains
Update Interval	CLOCK.UPDATE	8	8
Adjustment Interval	CLOCK.ADJ	2	0
Frequency Weight	CLOCK.FREQ	10	10
Phase Weight	CLOCK.PHASE	8	6
Compliance Weight	CLOCK.TRACK	8	not used
Compliance Maximum	CLOCK.COMP	4	not used
Compliance Mask	CLOCK.MASK	37_8	not used
Maximum Aperture	CLOCK.MAX	$\pm$ 0.128	$\pm$ 0.512

Table 8. Clock Parameters

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.

5.3. Fuzzball Implementation

The Fuzzball logical clock is implemented using a 48-bit Clock Register, which increments at 1000-Hz (at the decimal point), a 32-bit Clock-Adjust Register, which is used to slew the Clock Register in response to offset corrections, and a Drift-Compensation Register, which is used to trim the oscillator frequency. In some interface designs such as the DEC KWV11, an additional hardware Counter Register is used as an auxiliary counter. The configuration and decimal point of these registers are shown in Figure 4. The Watchdog Timer and Compliance Register shown in the figure are used to determine validity, compute frequency corrections and adjust the clock tracking characteristics.

The Clock Register, Clock-Adjust Register and Drift-Compensation Register are implemented in memory. In typical clock interface designs such as the DEC KWV11, the Counter Register is implemented as a 16-bit buffered counter driven by a crystal-controlled oscillator at a rate of 1000 Hz. A counter overflow is signalled by an interrupt, which results in an increment of the Clock Register at bit 15. The time of day is determined by reading the Counter Register, which does not disturb the counting process, and adding its value to that of the Clock Register with decimal points aligned. In other interface designs such as the LSI-11 event-line mechanism, each tick of the clock is signalled by an interrupt at intervals of $16\text{-}^2/_3$ ms or 20 ms, depending on interface and mains frequency. When this occurs the appropriate increment in milliseconds, expressed to 32 bits in precision, is added to the Clock Register with decimal points aligned. Monotonicity is insured with the parameters shown in Table 8, as long as the increment is at least 2 ms for crystal-stabilized clocks or 16 ms for mains-frequency clocks.

When the system is initialized all registers, counters and timers are cleared, the leap-indicator bits (sys.leap) are set to 11_2 (unsynchronized) and the Watchdog Timer begins incrementing at intervals of one second. As each update is received the Watchdog Timer is reset and resumes incrementing from zero. If the value of the Watchdog Timer exceeds NTP.MAXAGE (one full day), sys.leap is set to 11_2.

5.4. Uniform Phase Adjustments

Left uncorrected, the logical clock runs at the offset and frequency of its last update. An update is introduced at intervals of $2^{CLOCK.UPDATE}$ seconds as a signed 32-bit integer in milliseconds. When the magnitude of a correction is less than the maximum aperture CLOCK.MAX, bits 16-31 of the update replace bits 0-15 of the Clock-Adjust Register. In order to minimize the effects of truncation and roundoff errors, bits 16-31 are set to zeros if the sign of the update is positive and ones if negative. In addition, the update is also divided by a weighting factor (described later) and added to the Drift-Compensation Register. At adjustment intervals of $2^{CLOCK.ADJ}$ seconds a correction consisting of two components is computed. The first (phase) component consists of the value of the Clock-Adjust Register shifted right CLOCK.PHASE bits, which is then subtracted from the Clock-Adjust Register. The second (frequency) component consists of the value of the Drift-Compensation Register shifted right by the quantity CLOCK.FREQ - CLOCK.UPDATE. The sum of the phase and frequency components is the correction, which is then added to the Clock Register. Operation continues in this way until a new correction is introduced.

For the ultimate stability of about a millisecond per day in the absence of outside updates, it is necessary to reduce the influence of the frequency component once the clock has been running with low offsets for some time. This is done only in the case of crystal oscillators and using the Compliance Register, which contains an exponential average of all prior updates. The average is computed by first shifting the update left eight bits for efficient scaling, then subtracting the contents of the Compliance Register, then shifting the result right CLOCK.TRACK bits and finally adding the result to the Compliance Register.

When the update is added to the Drift Compensation Register, the value in the Compliance Register is copied to a temporary and the low-order bit set to one. Both the update and temporary are shifted left together until the bitwise AND of the temporary and the mask ~CLOCK.MASK become nonzero. The parameters in Table 7 have been selected so that, under good conditions with updates in the order of a few milliseconds, a precision of a millisecond per day (about .01 ppm or 10^{-8}), can be achieved. In the case of mains-frequency clocks the Compliance Register, CLOCK.TRACK, CLOCK.COMP and CLOCK.MASK variables are not used.

When mains-frequency oscillators must be used, the loop parameters must be adapted for the relatively high jitter and wander characteristics of the regional power grid, in which diurnal peak-to-peak phase excursions can exceed four seconds. Simulation of a loop with the parameters of Figure 6 and the clock filter described in Section 4 results in a transient response similar to the crystal-stabilized case, but with somewhat smaller time constants. When presented with actual phase-offset data from the U.S. Eastern, U.S. Western and West German power grids and for typical summer days when the jitter and wander are the largest, the residual errors are in the order of a few tens of milliseconds, but seldom more than 100 ms. With mains-frequency oscillators it is not possible to increase the polling interval above a minute or so without significant increase in loop error or degradation of transient response, so the polling interval peer.hpoll is always clamped at NTP.MINPOLL.

Care is required in the implementation to insure monotonicity of the Clock Register and to preserve the highest precision while minimizing the propagation of roundoff errors. Since all of the multiply/divide operations can be approximated by bitwise-shift operations, it is not necessary to implement a full multiply/divide capability in hardware or software. In the various implementations of NTP for many Unix-based systems it has been the common experience that the single most important factor affecting local-clock stability is the matching of the phase and frequency coefficients to the particular kernel implementation. It is vital that these coefficients be engineered according to the model values, for otherwise the phase-lock loop can fail to track normal oscillator variations and can even become unstable.

5.5. Nonuniform Phase Adjustments

When the magnitude of a correction exceeds the maximum aperture CLOCK.MAX, the possibility exists that the clock is so far out of synchronization with the reference source that the best action is an immediate and wholesale replacement of Clock Register contents, rather than a graduated slewing as described above. If this happens, the Clock-Adjust Register is set to zero, but the other registers remain undisturbed. In addition, as described previously, the clear procedure is called to purge the clock filters and estimation variables for all peers. In practice, the necessity to do this is rare and usually occurs when the local host or reference source is rebooted, for example. This is fortunate, since step changes in the clock can result in the clock apparently running backward, as well as incorrect delay and offset measurements of the synchronization mechanism itself.

Considerable experience with the Internet environment suggests the values of CLOCK.MAX tabulated in Table 7 as appropriate. In practice, these values are 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 case when the maximum is exceeded is when the time-server source is changed and the time indicated by the new and old sources exceeds the maximum due to systematic errors in the primary reference source or large differences in the synchronizing path delays. It is recommended that implementations include provisions to tailor CLOCK.MAX for specific situations. The amount that CLOCK.MAX can be increased without violating the monotonicity requirement depends on the Clock Register increment. For an increment of 10 ms, as used in many workstations, the value shown in Table 7 can be increased by a factor of five.

5.6. Maintaining Date and Time

Conversion from NTP format to the common date and time formats used by application programs is simplified if the internal local-clock format uses separate date and time registers. The time register is designed to roll over at 24 hours, give or take a leap second as determined by the leap-indicator bits, with its overflows (underflows) incrementing (decrementing) the date register. The date and time registers then indicate the number of days and seconds since some previous reference time, but uncorrected for intervening leap seconds.

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. Subsequently, these bits show up at the local host and are passed to the local-clock procedure. 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. While it is possible to turn the bits off automatically, the procedure suggested here insures that all clocks have rolled over and will not be reset incorrectly to the previous day as the result of possible corrections near the instant of rollover.

Lack of a comprehensive mechanism to administer the leap bits in the primary time servers is presently an awkward problem better suited to a comprehensive network-management mechanism yet to be developed. As a practical matter and unless specific provisions have been made otherwise, currently manufactured radio clocks have no provisions for leap seconds, automatic, manual or otherwise. Therefore, the only possible solution is to disable the radio clock immediately following leap insertion/deletion and to wait some hours for the radio clock to regain synchronization before re-enabling it.

6. Acknowledgments

Many people contributed to the contents of this document, which was thoroughly debated by electronic mail and debugged using prototype implementations written by Louis Mamakos and Michael Petry of the University of Maryland for the Unix 4.3bsd operating system and by the author for the Fuzzball operating system [43]. Among the most fervent of the many contributors were Marion Hakanson of Oregon State University and Dennis Ferguson of the University of Toronto, who meticulously tested the several beta-test prototype versions and ruthlessly smoked out the bugs, both in the code and the specification.

7. References

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

2. *Data Encryption Standard*. Federal Information Processing Standards Publication 46. National Bureau of Standards, U.S. Department of Commerce, 1977.

3. Vass, E.R. OMEGA navigation system: present status and plans 1977-1980. *Navigation 25, 1* (Spring 1978).

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

5. Time and Frequency Dissemination Services. NBS Special Publication 432, U.S. Department of Commerce, 1979.

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

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

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

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

10. DES Modes of Operation. Federal Information Processing Standards Publication 81. National Bureau of Standards, U.S. Department of Commerce, December 1980.

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

12. Mills, D.L. DCNET Internet Clock Service. DARPA Network Working Group Report RFC-778, COMSAT Laboratories, April 1981.

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

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

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

16. Frank, R.L. History of LORAN-C. *Navigation 29, 1* (Spring 1982).

17. Beser, J., and B.W. Parkinson. The application of NAVSTAR differential GPS in the civilian community. *Navigation 29, 2* (Summer 1982).

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

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

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

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

22. 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)

23. 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.

24. 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.

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

26. 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.

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

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

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

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

31. 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.

32. Jordan, E.C. (Ed). *Reference Data for Engineers, Seventh Edition.* H.W. Sams & Co., New York, 1985.

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

34. Bell Communications Research. Digital Synchronization Network Plan. Technical Advisory TA-NPL-000436, 1 November 1986.

35. 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.

36. Bell Communications Research. Digital Synchronization Network Plan. Technical Advisory TA-NPL-000436, 1 November 1986.

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

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

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

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

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

42. Mills, D.L. Network Time Protocol (version 1) - specification and implementation. DARPA Network Working Group Report RFC-1059, University of Delaware, July 1988.

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

44. Mills, D.L. Internet time synchronization: the Network Time Protocol. Submitted for publication.

45. Abate, et al. AT&T's new approach to the synchronization of telecommunication networks. *IEEE Communications Magazine* (April 1989), 35-45.

8. Appendix A. NTP Data Format - Version 2

The format of the NTP Message data area, which immediately follows the UDP header, is shown in Figure 5. Following is a description of its fields.

Leap Indicator (LI): This is a two-bit code warning of an impending leap second to be inserted/deleted in the last minute of the current day, with bit 0 and bit 1, respectively, coded as follows:

00	no warning
01	last minute has 61 seconds
10	last minute has 59 seconds)
11	alarm condition (clock not synchronized)

Version Number (VN): This is a three-bit integer indicating the NTP version number, currently two (2).

Mode: This is a three-bit integer indicating the mode, with values defined as follows:

0	reserved
1	symmetric active
2	symmetric passive
3	client
4	server
5	broadcast
6	reserved for NTP control message (see Appendix B)

0			8	16	24 31
LI	VN	Mode	Stratum	Poll	Precision
Synchronizing Distance (32)					
Synchronizing Dispersion (32)					
Reference Identifier (32)					
Reference Timestamp (64)					
Originate Timestamp (64)					
Receive Timestamp (64)					
Transmit Timestamp (64)					
Authenticator (optional) (96)					

Figure 5. NTP Message Header

7	reserved for private use

Stratum: This is a eight-bit integer indicating the stratum level of the local clock, with values defined as follows:

0	unspecified
1	primary reference (e.g., radio clock)
2-255	secondary reference (via NTP)

The values that can appear in this field range from zero to NTP.INFIN inclusive.

Poll Interval: This is an eight-bit signed integer indicating the maximum interval between successive messages, in seconds to the nearest power of two. The values that can appear in this field range from NTP.MINPOLL to NTP.MAXPOLL inclusive.

Precision: This is an eight-bit signed integer indicating the precision of the local clock, in seconds to the nearest power of two.

Synchronizing Distance: This is a 32-bit fixed-point number indicating the estimated roundtrip delay to the primary synchronizing source, in seconds with fraction point between bits 15 and 16.

Synchronizing Dispersion: This is a 32-bit fixed-point number indicating the estimated dispersion to the primary synchronizing source, in seconds with fraction point between bits 15 and 16.

Reference Clock Identifier: This is a 32-bit code identifying the particular reference clock. In the case of stratum 0 (unspecified) or stratum 1 (primary reference), this is a four-octet, left-justified, zero-padded ASCII string. While not ennumerated as part of the NTP specification, the following are suggested ASCII identifiers:

Stratum	Code	Meaning
0	DCN	DCN routing protocol
0	NIST	NIST public modem
0	TSP	TSP time protocol
1	GBR	GBR VLF radio
1	WWVB	WWVB LF radio
1	GOES	GOES UHF satellite
1	GPS	GPS UHF satellite
1	CHU	CHU HF radio
1	MSF	MSF HF radio
1	WWV	WWV HF radio
1	WWVH	WWVH HF radio

In the case of type 2 and greater (secondary reference) this is the four-octet Internet address of the reference host.

Reference Timestamp: This is the local time at which the local clock was last set or corrected, in 64-bit timestamp format.

Originate Timestamp: This is the local time at which the request departed the client host for the service host, in 64-bit timestamp format.

Receive Timestamp: This is the local time at which the request arrived at the service host, in 64-bit timestamp format.

Transmit Timestamp: This is the local time at which the reply departed the service host for the client host, in 64-bit timestamp format.

Authenticator (optional): When the NTP authentication mechanism is implemented, this contains the authenticator information defined in Appendix C.

9. Appendix B. NTP Control Messages

In a comprehensive network-management environment, facilities are presumed available to perform routine NTP control and monitoring functions, such as setting the leap-indicator bits at the primary servers, adjusting the various system parameters and monitoring regular operations. Ordinarily, these functions can be implemented using a network-management protocol such as SNMP and suitable extensions to the MIB database. However, in those cases where such facilities are not available, these functions can be implemented using special NTP control messages described herein. These messages are intended for use only in systems where no other management facilities are available or appropriate, such as in dedicated-function bus peripherals. Support for these messages is not required in order to conform to this specification.

The NTP Control Message has the value 6 specified in the mode field of the first octet of the NTP header and is formatted as shown below. The format of the data field is specific to each command or response; however, in most cases the format is designed to be constructed and viewed by humans and so is coded in free-form ASCII. This facilitates the specification and implementation of simple management tools in the absence of fully evolved network-management facilities. As in ordinary NTP messages, the authenticator field follows the data field. If the authenticator is used the data field is zero-padded to a 32-bit boundary, but the padding bits are not considered part of the data field and are not included in the field count.

IP hosts are not required to reassemble datagrams larger than 576 octets; however, some commands or responses may involve more data than will fit into a single datagram. Accordingly, a simple reassembly feature is included in which each octet of the message data is numbered starting with zero. As each fragment is transmitted the number of its first octet is inserted in the offset field and the number of octets is inserted in the count field. The more-data (M) bit is set in all fragments except the last.

Most control functions involve sending a command and receiving a response, perhaps involving several fragments. The sender chooses a distinct, nonzero sequence number and sets the status field and R and E bits to zero. The responder interprets the opcode and additional information in the data field, updates the status field, sets the R bit to one and returns the three 32-bit words of the header along with additional information in the data field. In case of invalid message format or contents the responder inserts a code in the status field, sets the R and E bits to one and, optionally, inserts a diagnostic message in the data field.

Some commands read or write system variables and peer variables for an association identified in the command. Others read or write variables associated with a radio clock or other device directly connected to a source of primary synchronization information. To identify which type of variable and association a 16-bit association identifier is used. System variables are indicated by the identifier zero. As each association is mobilized a unique, nonzero identifier is created for it. These identifiers are used in a cyclic fashion, so that the chance of using an old identifier which matches a newly created association is remote. A management entity can request a list of current identifiers and subsequently use them to read and write variables for each association. An attempt to use an expired identifier results in an exception response, following which the list can be requested again.

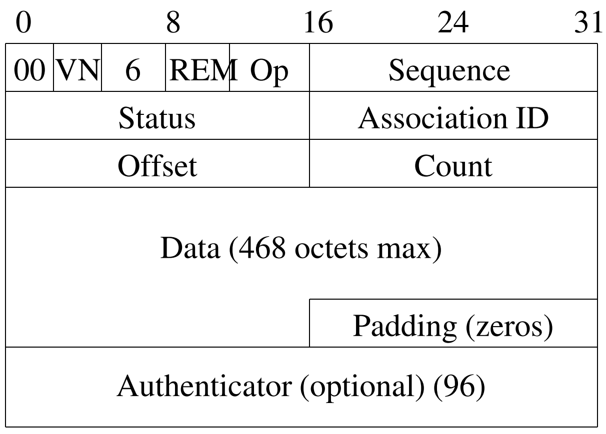

Figure 6. NTP Control Message Header

Some exception events, such as when a peer becomes reachable or unreachable, occur spontaneously and are not necessarily associated with a command. An implementation may elect to save the event information for later retrieval or to send an asynchronous response (called a trap) or both. In case of a trap the IP address and port number is determined by a previous command and the sequence field is set as described below. Current status and summary information for the latest exception event is returned in all normal responses. Bits in the status field indicate whether an exception has occurred since the last response and whether more than one exception has occurred.

Commands need not necessarily be sent by an NTP peer, so ordinary access-control procedures may not apply; however, the optional mask/match mechanism suggested elsewhere in this document provides the capability to control access by mode number, so this could be used to limit access for control messages (mode 6) to selected address ranges.

9.1. NTP Control Message Format

The format of the NTP Control Message header, which immediately follows the UDP header, is shown in Figure 6. Following is a description of its fields. Bit positions marked as zero are reserved and should always be transmitted as zero.

Version Number (VN): This is a three-bit integer indicating the NTP version number, currently two (2).

Mode: This is a three-bit integer indicating the mode. It must have the value 6, indicating an NTP control message.

Response Bit (R): Set to zero for commands, one for responses.

Error Bit (E): Set to zero for normal response, one for error response.

More Bit (M): Set to zero for last fragment, one for all others.

Operation Code (Op): This is a five-bit integer specifying the command function. Values currently defined include the following:

0	2	8	12 15
LI	Clock Source	Event	Event

System Status

0	6	8	12 15
Peer Status	Select	Event	Event

Peer Status Word

0	8 15
Clock Status	Event Code

Radio Status

0	8 15
Error Code	Reserved

Error Status

Figure 7. Status Word Formats

0	reserved
1	read status command/response
2	read variables command/response
3	write variables command/response
4	read clock variables command/response
5	write clock variables command/response
6	set trap address/port command/response
7	trap response
8-31	reserved

Sequence: This is a 16-bit integer indicating the sequence number of the command or response.

Status: This is a 16-bit code indicating the current status of the system, peer or clock, with values coded as described in following sections.

Association ID: This is a 16-bit integer identifying a valid association.

Offset: This is a 16-bit integer indicating the offset, in octets, of the first octet in the data area.

Count: This is a 16-bit integer indicating the length of the data field, in octets.

Data: This contains the message data for the command or response. The maximum number of data octets is 468.

Authenticator (optional): When the NTP authentication mechanism is implemented, this contains the authenticator information defined in Appendix C.

9.2. Status Words

Status words indicate the present status of the system, associations and clock. They are designed to be interpreted by network-monitoring programs and are in one of four 16-bit formats shown in Figure 7 and described in this section. System and peer status words are associated with responses for all commands except the read clock variables, write clock variables and set trap address/port commands. The association identifier zero specifies the system status word, while a nonzero identifier specifies a particular peer association. The status word returned in response to read clock variables and write clock variables commands indicates the state of the clock hardware and decoding software. A special error status word is used to report malformed command fields or invalid values.

9.2.1. System Status Word

The system status word appears in the status field of the response to a read status or read variables command with a zero association identifier. The format of the system status word is as follows:

Leap Indicator (LI): This is a two-bit code warning of an impending leap second to be inserted/deleted in the last minute of the current day, with bit 0 and bit 1, respectively, coded as follows:

00	no warning
01	last minute has 61 seconds
10	last minute has 59 seconds)
11	alarm condition (clock not synchronized)

Clock Source: This is a six-bit integer indicating the current synchronization source, with values coded as follows:

0	unspecified or unknown
1	VLF (band 4) radio (e.g., GBR)
2	LF (band 5) radio (e.g., WWVB)
3	HF (band 7) radio (e.g., CHU, MSF, WWV/H)
4	UHF (band 9) satellite (e.g., GOES, GPS)
5	local net (e.g., DCN, TSP)
6	UDP/NTP
7	UDP/TIME
8	eyeball-and-wristwatch
9	telephone modem (e.g., NIST)
10-63	reserved

System Event Counter: This is a four-bit integer indicating the number of system exception events occuring since the last time the system status word was returned in a response or included in a trap message. The counter is cleared when returned in the status field of a response and freezes when it reaches the value 15.

System Event Code: This is a four-bit integer identifying the latest system exception event, with new values overwriting previous values, and coded as follows:

0	unspecified
1	system restart
2	system or hardware fault
3	system new status word (leap bits or synchronization change)
4	system new clock source or stratum (sys.peer or sys.stratum change)
5	system clock reset (offset correction exceeds CLOCK.MAX)
6	system invalid time or date (see Section 3.4.5)
7	system clock exception (see system clock status word)
8-15	reserved

9.2.2. Peer Status Word

A peer status word is returned in the status field of a response to a read status, read variables or write variables command and appears also in the list of association identifers and status words returned by a read status command with a zero association identifier. The format of a peer status word is as follows:

Peer Status: This is a six-bit code indicating the status of the peer determined by the packet procedure, with bits assigned as follows:

0	configured (peer.config)
1	authentication enabled (peer.authenable)
2	authentication (peer.authentic)
3	reachability okay (peer.reach $\neq$ 0)
4	sanity okay (packet procedure)
5	dispersion okay (peer.dispersion < PEER.THRESHOLD)

Peer Selection (Select): This is a two-bit integer indicating the status of the peer determined by the clock-selection procedure, with values coded as follows:

0	rejected
1	selection candidate (survivor of the pruned and truncated list sorted by stratum/dispersion)
2	synchronization candidate (survivor of the list sorted by delay less outlyer discards)
3	current clock source

Peer Event Counter: This is a four-bit integer indicating the number of peer exception events that occured since the last time the peer status word was returned in a response or included in a trap message. The counter is cleared when returned in the status field of a response and freezes when it reaches the value 15.

Peer Event Code: This is a four-bit integer identifying the latest peer exception event, with new values overwriting previous values, and coded as follows:

0	unspecified
1	peer IP error
2	peer authentication failure (peer.authentic bit was one now zero)
3	peer unreachable (peer.reach was nonzero now zero)
4	peer reachable (peer.reach was zero now nonzero)
5	peer clock exception (see peer clock status word)
6-15	reserved

9.2.3. Clock Status Word

There are two ways a reference clock can be attached to a NTP service host, as an dedicated device managed by the operating system and as a synthetic peer managed by NTP (see Section 3.4.4). As in the read status command, the association identifier is used to identify which one, zero for the system clock and nonzero for a peer clock. Only one system clock is supported by the protocol, although many peer clocks can be supported. A system or peer clock status word appears in the status field of the response to a read clock variables or write clock variables command. This word can be considered an extension of the system status word or the peer status word as appropriate. The format of the clock status word is as follows:

Clock Status: This is an eight-bit integer indicating the current clock status, with values coded as follows:

0	clock operating within nominals
1	reply timeout
2	bad reply format
3	hardware or software fault
4	propagation failure
5	bad date format or value
6	bad time format or value
7-255	reserved

Clock Event Code: This is an eight-bit integer identifying the latest clock exception event, with new values overwriting previous values. When a change to any nonzero value occurs in the radio status field, the radio status field is copied to the clock event code field and a system or peer clock exception event is declared as appropriate.

9.2.4. Error Status Word

An error status word is returned in the status field of an error response as the result of invalid message format or contents. Its presence is indiated when the E (error) bit is set along with the response (R) bit in the response. It consists of an eight-bit integer coded as follows:

0	unspecified

1	authentication failure
2	invalid message length or format
3	invalid opcode
4	unknown association identifier
5	unknown variable name
6	invalid variable value
7	administratively prohibited
8-255	reserved

9.3. Commands

Commands consist of the header and optional data field shown in Figure 6. When present, the data field contains a list of identifiers or assignments in the form

<identifier>[=<value>],<identifier>[=<value>],...

where <identifier> is the ASCII name of a system or peer variable specified in Table 2 or Table 3 and <value> is expressed as a decimal, hexadecimal or string constant in the syntax of the C programming language. Where no ambiguity exists, the "sys." or "peer." prefixes shown in Table 2 or Table 4 can be suppressed. Whitespace (ASCII nonprinting format effectors) can be added to improve readability for simple monitoring programs that do not reformat the data field. Internet addresses are represented as four octets in the form [n.n.n.n], where n is in decimal notation and the brackets are optional. Timestamps, including reference, originate, receive and transmit values, as well as the logical clock, are represented in units of seconds and fractions, preferably in hexadecimal notation, while delay, offset, dispersion and distance values are represented in units of milliseconds and fractions, preferably in decimal notation. All other values are represented as-is, preferably in decimal notation.

Implementations may define variables other than those listed in Table 2 or Table 3. Called extramural variables, these are distinguished by the inclusion of some character type other than alphanumeric or "." in the name. For those commands that return a list of assignments in the response data field, if the command data field is empty, it is expected that all available variables defined in Table 3 or Table 4 will be included in the response. For the read commands, if the command data field is nonempty, an implementation may choose to process this field to individually select which variables are to be returned.

Commands are interpreted as follows:

Read Status (1): The command data field is empty or contains a list of identifiers separated by commas. The command operates in two ways depending on the value of the association identifier. If this identifier is nonzero, the response includes the peer identifier and status word. Optionally, the response data field may contain other information, such as described in the Read Variables command. If the association identifier is zero, the response includes the system identifier (0) and status word, while the data field contains a list of binary-coded pairs

<association identifier> <status word>,

one for each currently defined association.

Read Variables (2): The command data field is empty or contains a list of identifiers separated by commas. If the association identifier is nonzero, the response includes the requested peer identifier and status word, while the data field contains a list of peer variables and values as described above. If the association identifier is zero, the data field contains a list of system variables and values. If a peer has been selected as clock source, the response includes the peer identifier and status word; otherwise, the response includes the system identifier (0) and status word.

Write Variables (3): The command data field contains a list of assignments as described above. The variables are updated as indicated. The response is as described for the Read Variables command.

Read Clock Variables (4): The command data field is empty or contains a list of identifiers separated by commas. The association identifier selects the system clock variables or peer clock variables in the same way as in the Read Variables command. The response includes the requested clock identifier and status word and the data field contains a list of clock variables and values, including the last timecode message received from the clock.

Write Clock Variables (5): The command data field contains a list of assignments as described above. The clock variables are updated as indicated. The response is as described for the Read Clock Variables command.

Set Trap Address/Port (6): The command association identifier, status and data fields are ignored. The address and port number for subsequent trap messages are taken from the source address and port of the control message itself. The initial trap counter for trap response messages is taken from the sequence field of the command. The response association identifier, status and data fields are not significant. Implementations should include sanity timeouts which prevent trap transmissions if the monitoring program does not renew this information after a lengthy interval.

Trap Response (7): This message is sent when a system, peer or clock exception event occurs. The opcode field is 7 and the R bit is set. The trap counter is incremented by one for each trap sent and the sequence field set to that value. The trap message is sent using the IP address and port fields established by the set trap address/port command. If a system trap the association identifier field is set to zero and the status field contains the system status word. If a peer trap the association identifier field is set to that peer and the status field contains the peer status word. Optional ASCII-coded information can be included in the data field.

10. Appendix C. Authentication Issues

NTP robustness requirements are similar to those of other multiple-peer distributed protocols used for network routing, management and file access. These include protection from faulty implementations, improper operation and possibly malicious replay attacks with or without data modification. These requirements are especially stringent with distributed protocols, since damage due to failures can propagate quickly throughout the network, devastating archives, routes and monitoring systems and even bring down major portions of the network in the fashion of the classic Internet Worm.

The access-control mechanism suggested in Section 3.5 responds to these requirements by limiting access to trusted peers. The various sanity checks resist most replay and spoofing attacks by discarding old duplicates and using the originate timestamp as a one-time pad, since it is unlikely that even a synchronized peer can predict future timestamps with the precision required on the basis of past observations alone. In addition, the protocol environment resists jamming attacks by employing redundant time servers and diverse network paths. Resistance to stochastic disruptions, actual or manufactured, are minimized by careful design of the filtering and selection algorithms.

However, it is possible that a determined intruder can disrupt timekeeping operations between peers by subtle modifications of NTP message data, such as falsifying header fields or certain timestamps. In cases where protection from even these types of attacks is required, a specifically engineered message-authentication mechanism based on cryptographic techniques is necessary. Typical mechanisms involve the use of cryptographic certificates, algorithms and key media, together with secure media databases and key-management protocols. Ongoing research efforts in this area are directed toward developing a standard methodology that can be used with many protocols, including NTP. However, while it may eventually be the case that ubiquitous, widely applicable authentication methodology may be adopted by the Internet community and effectively overtake the mechanism described here, it does not appear that specific standards and implementations will happen within the lifetime of this particular version of NTP.

The NTP authentication mechanism described here is intended for interim use until specific standards and implementations operating at the network level or transport level are available. Support for this mechanism is not required in order to conform to the NTP specification itself. The mechanism, which operates at the application level, is designed to protect against unauthorized message-stream modification and misrepresentation of source by insuring that unbroken, authenticated paths exist between a trusted, stratum-one server in a particular synchronization subnet and all other servers in that subnet. It employs a crypto-checksum, computed by the sender and checked by the receiver, together with a set of predistributed algorithms and cryptographic keys indexed by a key identifier included in the message. However, there are no provisions in NTP itself to distribute or maintain the certificates, algorithms or keys. These quantities may occasionally be changed, which may result in inconsistent key information while rekeying is in progress. The nature of NTP itself is quite tolerant to such disruptions, so no particular provisions are included to deal with them.

The intent of the authentication mechanism is to provide a framework that can be used in conjunction with selected mode combinations to build specific plans to manage clockworking communities and implement policy as necessary. It can be selectively enabled or disabled on a per-peer basis (peer.authenable and peer.authentic bits). There is no specific plan proposed to manage the use of such schemes; although several possibilities are immediately obvious. In one scenario a group of time servers peers among themselves using symmetric modes and shares one secret key, say key 1, while another group of servers peers among themselves using symmetric modes and shares another secret key, say key 2. Now, assume by policy it is decided that selected servers in group 1 can provide synchronization to group 2, but not the other way around. The selected servers in group 1 are given key 2, but operated only in server mode, so cannot accept synchronization from group 2; however, group 2 has authenticated access to group-1 servers. Many other scenarios are possible with suitable combinations of modes and keys.

A packet format and crypto-checksum procedure appropriate for NTP is specified in the following sections. The cryptographic information is carried in an authenticator which follows the (unmodified) NTP header fields. The crypto-checksum procedure uses the Data Encryption Standard (DES) [2]; however, only the DES encryption algorithm is used and the decryption algorithm is not necessary. This feature is specifically targeted toward governmental sensitivities on the export of cryptographic technology, since the DES decryption algorithm need not be included in NTP software distributions and thus cannot be extracted and used in other applications to avoid message data disclosure.

10.1. NTP Authentication Mechanism

When it is created and possibly at other times, each association is allocated variables identifying the certificate authority, encryption algorithm, cryptographic key and possibly other data. The specific procedures to allocate and initialize these variables are beyond the scope of this specification, as are the association of the identifiers and keys and the management and distribution of the keys themselves. For example and consistency with the conventions of Section 3.3, a set of appropriate peer and packet variables might include the following:

Key Identifier (sys.keyid, peer.keyid, pkt.keyid): This is an integer identifying the cryptographic key used to generate the message-authentication code as described below. The system variable sys.keyid is used for active associations. The peer.keyid variable is initialized at zero (unspecified) when the association is mobilized. For purposes of authentication an unassigned value is interpreted as zero (unspecified).

Cryptographic Keys (sys.key): These are a set of 64-bit DES keys. Each key is constructed as in the Berkeley Unix distributions, which consists of eight octets, where the seven low-order bits of each octet correspond to the DES bits 1-7 and the high-order bit corresponds to the DES odd-parity bit 8. By convention, the unspecified key 0 (zero), consisting of eight odd-parity zero octets, is used for testing and presumed known throughout the NTP community. The remaining keys are distributed using methods outside the scope of NTP.

Crypto-Checksum (pkt.check): This is a crypto-checksum computed by the encryption procedure.

The authenticator field consists of two subfields, one consisting of the pkt.keyid variable and the other the pkt.check variable computed by the encrypt procedure, which is called by the transmit

0 8 16 24 31
Key Identifier (32)
Crypto-Checksum (64)

Figure 8. Authenticator Format

procedure described in Section 3.4.1, and by the decrypt procedure, which is called by the receive procedure described in Sectio 3.4.2. Its presence is revealed by the fact the total datagram length according to the UDP header is longer than the NTP message length, which includes the header plus the data field, if present. For authentication purposes, the NTP message is zero-padded if necessary to a 64-bit boundary, although the padding bits are not considered part of the NTP message itself. The authenticator format shown in Figure 8 has 96 bits, including a 32-bit key identifier and 64-bit crypto-checksum, and is aligned on a 32-bit boundary for efficient computation. Additional information required in some implementations, such as certificate authority and encryption algorithm, can be inserted between the (padded) NTP message and the key identifier, as long as the alignment conditions are met. Like the authenticator itself, this information is not included in the crupto-checksum. Use of these data are beyond the scope of this specification. These conventions may be changed in future as the result of other standardization activities.

10.2. NTP Authentication Procedures

When authentication is implemented there are two additional procedures added to those described in Section 3.4. One of these (encrypt) constructs the crypto-checksum in transmitted messages, while the other (decrypt) checks this quantity in received messages. The procedures use a variant of the cipher-block chaining method described in [10] as applied to DES. In principal, the procedure is independent of DES and requires only that the encryption algorithm operate on 64-bit blocks. While the NTP authentication mechanism specifies the use of DES, other algorithms could be used by prior arrangement.

For ordinary NTP messages the encryption procedure operates as follows. If authentication is not enabled (peer.authenable set to zero), the procedure simply exits. Otherwise, a 64-bit temporary variable is initialized to zero. For each of the 64-bit NTP header and data words not including the authenticator or additional information and proceeding from the beginning of the header, the header word is XORed with the temporary variable and the variable then encrypted using the DES algorithm. If the association is active (modes 1, 3, 5) the key is determined by the system variable sys.keyid. If the association is passive (modes 2, 4) the key is determined by the peer variable peer.keyid if peer.authentic is set to one and the default key (zero) otherwise. Finally, the authenticator is constructed using the chosen key for pkt.keyid and temporary variable for pkt.check.

For ordinary messages the decryption procedure operates as follows. If the peer is not configured (peer.config bit set to zero) and the message data includes the authenticator, which is placed at the end of the NTP message itself, the peer.authenable bit is set to one; otherwise, it is set to zero. If peer.config is set to one, no change to peer.authenable is made. If peer.authenable is set to zero following this step, the procedure simply exits. Then, if the message data does not include the authenticator fields, the peer.authentic bit is set to zero and the procedure exits. Otherwise, the packet variable pkt.keyid is copied to the peer variable peer.keyid and the crypto-checksum is computed using that variable. The peer.authentic bit is set to one if peer.keyid is nonzero and the checksum matches the pkt.check field following this step; otherwise the bit is set to zero.

For NTP control messages the peer variables are not used. If a command message is received with an authenticator field, the crypto-checksum is computed as in the decrypt procedure and the response message includes the authenticator field as computed by the encrypt procedure. If the received authenticator is correct the key for the response is the same as in the command; otherwise, the default key (zero) is used. Commands causing a change to the peer data base, such as the write variables and set trap address/port commands, must be correctly authenticated; however, the remaining commands are normally not authenticated in order to minimize the encryption overhead.

11. Appendix D. Differences from Previous Versions.

The original NTP, later called NTP Version 0, was described in RFC-958 [30], while the most recent prior NTP Version 1 was described in RFC-1059 [42]. The Version-2 description has been split into two documents, this one defining the architecture and specifying the protocol and algorithms, and another [44] describing the service model, algorithmic analysis and operating experience. In previous versions [30], [42] these two objectives were combined in one document. Differences between NTP Version 2 and previous versions are described in this Appendix. Due to known bugs in very old implementations, continued support for Version-0 implementations is not recommended. It is recommended that new implementations follow the guidelines below when interoperating with Version-1 implementations.

1. Version 1 supports no modes other than symmetric-active and symmetric-passive, which are determined by inspecting the port-number fields of the UDP packet header as described in Section 3.3 above. The low-order three bits of the first octet, specified as zero in Version 1, are used for the mode field in Version 2. Version-2 implementations interoperating with Version-1 implementations should operate in a passive mode only and use the value one in the version number (pkt.version) field and zero in the mode (pkt.mode) field in transmitted messages.

2. Version 1 does not support the NTP control message described in Appendix B. Certain old versions of the Unix NTP daemon *ntpd* use the high-order bits of the stratum field (pkt.stratum) for control and monitoring purposes. While these bits are never set during normal Version-1 or Version-2 operations, new implementations may use the NTP reserved mode 6 described in Appendix B and/or private reserved mode 7 for special purposes, such as remote control and monitoring, and in such cases the format of the packet following the first octet can be arbitrary. While there is no guarantee that different implementations can interoperate using private reserved mode 7, it is recommended that vanilla ASCII format be used whenever possible.

3. Version 1 does not support authentication. The key identifiers, cryptographic keys and procedures described in Appendix C are new to Version 2, along with the corresponding variables, procedures and authenticator fields. In the NTP message described in Appendix A and NTP control message described in Appendix B the format and contents of the header fields are independent of the authentication mechanism and the authenticator itself follows the header fields, so that previous versions will ignore the authenticator.

4. In Version 1 the synchronizing dispersion (pkt.dispersion) field of the NTP header was called the estimated drift rate, but not used in the protocol or timekeeping procedures. Implementations of the Version-1 protocol typically set this field to the current value of the Drift Compensation Register, which is a signed quantity. In a Version 2 implementation apparent large values in this field may affect the order considered in the clock-selection procedure. Version-2 implementations interoperating with older implementations should assume this field is zero, regardless of its actual contents.

5. Version 2 incorporates several sanity checks designed to avoid disruptions due to unsynchronized, duplicate or bogus timestamp information. The leap-indicator bits are set to show

the unsynchronized state if updates are not received from a reference source for a full day or if the reference source has not received updates for a full day. Some Version-1 implementations could claim valid synchronization indefinitely following loss of the reference source.

6. The clock-selection procedure of Version 2 is considerably refined as the result of accumulated experience with the Version-1 implementation. Additional sanity checks are included for authentication, range bounds and to avoid use of very old data. The candidate list is sorted twice, once to select a relatively few robust candidates from a potentially large population of unruly peers and again to order the resulting list by measurement quality. As in Version 1, The final selection procedure repeatedly casts out outlyers on the basis of weighted dispersion.

7. The local-clock procedure of Version 2 is considerably improved over Version 1 as the result of analysis, simulation and experience. Checks have been added to warn that the oscillator has gone too long without update from a reference source. The Compliance Register has been added to improve frequency stability to the order of a millisecond per day. The various parameters were retuned for optimum loop stability using measured data over typical Internet paths and with typical local-clock hardware.

8. Problems in the timekeeping calculations of Version 1 with high-speed LANs were found and corrected. These were caused by jitter due to small differences in clock rates and different precisions between the peers. Subtle bugs in the Version-1 reachability and polling-rate control were found and corrected. The peer.valid and sys.hold variables were added to avoid instabilities when the reference source changes rapidly due to large dispersive delays under conditions of severe network congestion. The peer.config, peer.authenable and peer.authentic bits were added to control special features and simplify configuration.

Security considerations

see Section 3.5 and Section 10 (Appendix C)

Author's address

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