Network Working Group
Request for Comments: 1059

D. Mills
University of Delaware
July 1988

Network Time Protocol (Version 1)
Specification and Implementation

Status of this Memo

This memo 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 logical 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.

The NTP architectures, algorithms and protocols which have evolved over several years of implementation and refinement are described in this document. The prototype system, which has been in regular operation in the Internet for the last two years, is described in an Appendix along with performance data which shows that timekeeping accuracy throughout most portions of the Internet can be ordinarily maintained to within a few tens of milliseconds, even in cases of failure or disruption of clocks, time servers or nets. This is a Draft Standard for an Elective protocol. Distribution of this memo is unlimited.

Table of Contents

1. Introduction 3
1.1. Related Technology 4
2. System Architecture 6
2.1. Implementation Model 7
2.2. Network Configurations 9
2.3. Time Scales 10
3. Network Time Protocol 12
3.1. Data Formats 12
3.2. State Variables and Parameters 13
3.2.1. Common Variables 15
3.2.2. System Variables 17
3.2.3. Peer Variables 18
3.2.4. Packet Variables 19
3.2.5. Clock Filter Variables 19
3.2.6. Parameters 20

3.3. Modes of Operation 21
3.4. Event Processing 22
3.4.1. Timeout Procedure 23
3.4.2. Receive Procedure 24
3.4.3. Update Procedure 27
3.4.4. Initialization Procedures 29
4. Filtering and Selection Algorithms 29
4.1. Clock Filter Algorithm 29
4.2 Clock Selection Algorithm 30
4.3. Variable-Rate Polling 32
5. Logical Clocks 33
5.1. Uniform Phase Adjustments 35
5.2. Nonuniform Phase Adjustments 36
5.3. Maintaining Date and Time 37
5.4. Calculating Estimates 37
6. References 40

Appendices
Appendix A. UDP Header Format 43
Appendix B. NTP Data Format 44
Appendix C. Timeteller Experiments 47
Appendix D. Evaluation of Filtering Algorithms 49
Appendix E. NTP Synchronization Networks 56

List of Figures
Figure 2.1. Implementation Model 8
Figure 3.1. Calculating Delay and Offset 26
Figure 5.1. Clock Registers 34
Figure D.1. Calculating Delay and Offset 50
Figure E.1. Primary Service Network 57

List of Tables
Table 2.1. Dates of Leap-Second Insertion 11
Table 3.1. System Variables 14
Table 3.2. Peer Variables 14
Table 3.3. Packet Variables 15
Table 3.4. Parameters 15
Table 4.1. Outlyer Selection Procedure 32
Table 5.1. Clock Parameters 35
Table C.1. Distribution Functions 47
Table D.1. Delay and Offset Measurements (UMD) 52
Table D.2.a Delay and Offset Measurements (UDEL) 52
Table D.2.b Offset Measurements (UDEL) 53
Table D.3. Minimum Filter (UMD - NCAR) 54
Table D.4. Median Filter (UMD - NCAR) 54
Table D.5. Minimum Filter (UDEL - NCAR) 55
Table E.1. Primary Servers 56

1. Introduction

This document describes the Network Time Protocol (NTP), including the architectures, algorithms and protocols to synchronize local clocks in a set of distributed clients and servers. The protocol was first described in RFC-958 [24], but has evolved in significant ways since publication of that document. NTP is built on the Internet Protocol (IP) [10] and User Datagram Protocol (UDP) [6], which provide a connectionless transport mechanism; however, it is readily adaptable to other protocol suites. It is evolved from the Time Protocol [13] and the ICMP Timestamp message [11], but is specifically designed to maintain accuracy and robustness, even when used over typical Internet paths involving multiple gateways and unreliable nets.

The service environment consists of the implementation model, service model and time scale 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 offsets, or skews, but does not require reliable message delivery. The subnet is 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 began with those suggested in [22], were refined as the results of experiments described in [23] and further evolved under typical operating conditions over the last two years. In addition, as the result of experience in operating multiple-server nets including radio-synchronized clocks at several sites in the US and with clients in the US 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 logical clock design evolved from the Fuzzball implementation described in [15]. This design includes offset-slewing, drift-compensation and deglitching mechanisms capable of accuracies in order of a millisecond, even after extended periods when synchronization to primary reference sources has been lost.

The UDP and NTP packet formats are shown in Appendices A and B. Appendix C presents the results of a survey of about 5500 Internet hosts showing how their clocks compare with primary reference sources using three different time protocols, including NTP. Appendix D presents experimental results using several different deglitching and smoothing algorithms. Appendix E describes the prototype NTP primary service net, as well as proposed rules of engagement for its use.

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 [12], Time Protocol [13], ICMP Timestamp message [11] and IP Timestamp option [9]. Experimental results on measured times and roundtrip delays in the Internet are discussed in [14], [23] and [31]. Other synchronization protocols are discussed in [7], [17], [20] and [28]. NTP uses techniques evolved from both linear and nonlinear synchronization methodology. Linear methods used for digital telephone network synchronization are summarized in [3], while nonlinear methods used for process synchronization are summarized in [27].

The Fuzzball routing protocol [15], 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 model [20] 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 [25] 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 [28], 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 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 methods summarized in [19] and [27].

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 [17], the CNV algorithm of [19], the majority-subset algorithm of [22], the egocentric algorithm of [27] 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 [17] and the optimum algorithm of [30]. 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 logical clock as a voltage-controlled oscillator. This similarity is not accidental, since systems like this have been studied extensively [3], [4] and [5].

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 [3]. The clock filter and selection algorithms are designed so that the clock synchronization subnet self-organizes into a hierarchical master-slave configuration [5]. 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 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, acquisition, or authentication 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 logical clock accordingly. In addition, the message includes information to calculate the expected timekeeping accuracy and reliability, thus 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 logical clock control. Figure 2.1 shows a possible implementation model including four processes sharing a partitioned data base, with a partition dedicated to each peer and interconnected by a message-passing system.

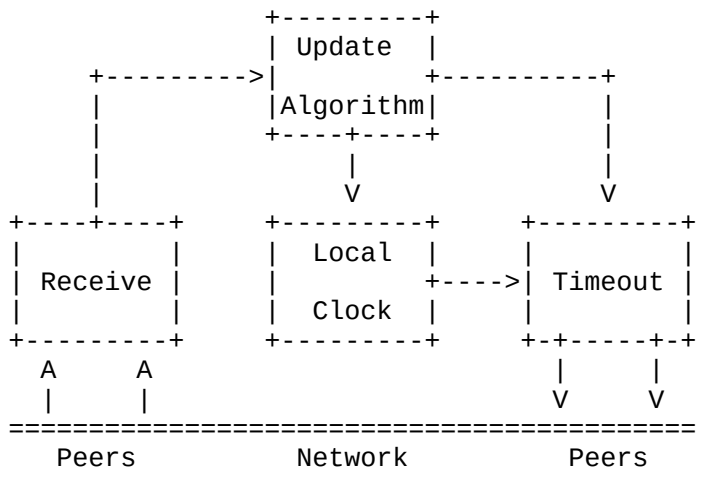


Figure 2.1. Implementation Model

The timeout process, driven by independent timers for each peer, collects information in the data base and sends NTP messages to other peers in the net. Each message contains the local time the message is sent, together with previously received information and other information necessary to compute the estimated error and manage the association. The message transmission rate is determined by the accuracy expected of the local system, as well as its peers.

The receive process receives NTP messages and perhaps messages in other protocols as well, including ICMP, other UDP or TCP time protocols, local-net protocols and directly connected radio clocks. When an NTP message is received the offset between the sender clock and the local clock is computed and incorporated into the data base along with other information useful for error estimation and clock selection.

The update algorithm is initiated upon receipt of a message and at other times. It processes the offset data from each peer and selects the best peer using algorithms such as those described in Section 4. This may involve many observations of a few clocks or a few observations of many clocks, depending on the accuracies required.

The local clock process operates upon the offset data produced by the update algorithm and adjusts the phase and frequency of the logical clock using mechanisms such as described in Section 5. This may result in either a step change or a gradual slew adjustment of the logical clock to reduce the offset to zero. The logical 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

A primary time server is connected to a primary reference source, usually a radio clock synchronized to national standard time. A secondary time server derives time synchronization, possibly via other secondary servers, from a primary server. Under normal circumstances it is intended that a subnet of primary and secondary servers assumes a hierarchical master-slave configuration with the more accurate servers near the top and the less accurate below.

Following conventions established by the telephone industry, the accuracy of each server is defined by a number called its stratum, with the stratum of a primary server assigned as one and each level downwards in the hierarchy assigned as one greater than the preceding level. With current technology and available receiving equipment, single-sample accuracies in the order of a millisecond can be achieved at the radio clock interface and in the order of a few milliseconds at the packet interface to the net. Accuracies of this order require special care in the design and implementation of the operating system, such as described in [15], and the logical clock mechanism, such as described in Section 5.

As the stratum increases from one, the single-sample accuracies achievable will degrade depending on the communication paths and local clock stabilities. In order to avoid the tedious calculations [4] necessary to estimate errors in each specific configuration, it is useful to assume the errors accumulate approximately in proportion to the minimum total roundtrip path delay between each server and the primary reference source to which it is synchronized. This is called the synchronization distance.

Again drawing from the experience of the telephone industry, who learned such lessons at considerable cost, the synchronization paths should be organized to produce the highest accuracies, but must never be allowed to form a loop. The clock filter and selection algorithms used in NTP accomplish this by using a variant of the Bellman-Ford distributed routing algorithm [29] to compute the minimum-weight spanning trees rooted on the primary servers. This results in each server operating at the lowest stratum and, in case of multiple peers at the same stratum, at the lowest synchronization distance.

As a result of the above design, the subnet reconfigures automatically in a hierarchical master-slave configuration to produce the most accurate time, even when one or more primary or secondary servers or the communication paths between them fail. This includes the case where all normal primary servers (e.g., backbone WWVB clocks) on a possibly partitioned subnet fail, but one or more backup primary servers (e.g., local WWV clocks) continue operation.

However, should all primary servers throughout the subnet fail, the remaining secondary servers will synchronize among themselves for some time and then gradually drop off the subnet and coast using their last offset and frequency computations. Since these computations are expected to be very precise, especially in frequency, even extend outage periods of a day or more should result in timekeeping errors of not over a few tens of milliseconds.

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. Time Scales

Since 1972 the various national time scales have been based on International Atomic Time (TA), which is currently maintained using multiple cesium-beam clocks to an accuracy of a few parts in 10^12. The Bureau International de l'Heure (BIH) uses astronomical observations provided by the US Naval Observatory and other observatories to determine corrections for small changes in the mean rotation period of the Earth. This results in Universal Coordinated Time (UTC), which is presently decreasing from TA at a fraction of a second per year. When the magnitude of the correction approaches 0.7 second, a leap second is inserted or deleted in the UTC time scale on the last day of June or December. Further information on time scales can be found in [26].

For the most precise coordination and timestamping of events since 1972 it is necessary to know when leap seconds were inserted or deleted in UTC and how the seconds are numbered. 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 on the following fourteen occasions prior to January 1988 (courtesy US Naval Observatory):

1	June 1972	8	December 1978
2	December 1972	9	December 1979
3	December 1973	10	June 1981
4	December 1974	11	June 1982
5	December 1975	12	June 1983
6	December 1976	13	June 1985
7	December 1977	14	December 1987

Table 2.1. Dates of Leap-Second Insertion

Like UTC, NTP operates with an abstract oscillator synchronized in frequency to the TA time scale. At 0000 hours on 1 January 1972 the NTP time scale was set to 2,272,060,800, representing the number of TA seconds since 0000 hours on 1 January 1900. The insertion of leap seconds in UTC does not affect the oscillator itself, only the translation between TA and UTC, or conventional civil time. However, since the only institutional memory assumed by NTP is the UTC radio broadcast service, the NTP time scale 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 1988 was used to establish the time of an event that occured in early 1972, it would be fourteen seconds early.

When NTP is used to measure intervals between events that straddle a leap second, special considerations apply. When it is necessary to determine the elapsed time between events, such as the half life of a proton, NTP timestamps of these events can be used directly. When it is necessary to establish the order of events relative to UTC, such as the order of funds transfers, NTP timestamps can also be used directly; however, if it is necessary to establish the elapsed time between events relative to UTC, such as the intervals between payments on a mortgage, NTP timestamps must be converted to UTC using the above table and its successors.

The current formats used by NBS radio broadcast services [2] 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 clocks and then automatically distributed throughout the system for delivery to all logical clocks and then effected as described in Section 5.

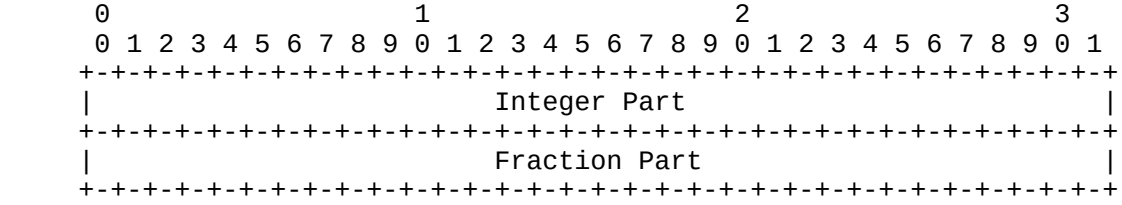
3. Network Time Protocol

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 model is based on the implementation model illustrated in Figure 2.1, 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 and 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 arithmetic. Data are specified as integer or fixed-point quantities. Since various implementations would be expected to 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, if designated, with an implied zero preceding the most significant (leftmost) bit. 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 0000 UT on 1 January 1900. The integer part is in the first 32 bits and the fraction part in the last 32 bits, as shown in the following diagram.



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 0.2

nanosecond, which should be adequate for even the most exotic requirements.

Timestamps are determined by copying the current value of the logical clock to a timestamp variable 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 0.2-nanosecond 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 tabular 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 logical clock mechanism; peer variables, which are specific to each peer operating in symmetric mode or client mode; packet variables, which represent the contents of the NTP message; and parameters, which are fixed in 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.

System Variables	Name	Control
Logical Clock	sys.clock	update
Clock Source	sys.peer	selection algorithm
Leap Indicator	sys.leap	update
Stratum	sys.stratum	update
Precision	sys.precision	system
Synchronizing Distance	sys.distance	update
Estimated Drift Rate	sys.drift	system
Reference Clock Identifier	sys.refid	update
Reference Timestamp	sys.reftime	update

Table 3.1. System Variables

Peer Variables	Name	Control
Peer Address	peer.srcadr	system
Peer Port	peer.srcport	system
Local Address	peer.dstadr	system
Local Port	peer.dstport	system
Peer State	peer.state	receive, transmit
Reachability Register	peer.reach	receive, transmit
Peer Timer	peer.timer	system
Timer Threshold	peer.threshold	system
Leap Indicator	peer.leap	receive
Stratum	peer.stratum	receive
Peer Poll Interval	peer.ppoll	receive
Host Poll Interval	peer.hpoll	receive, transmit
Precision	peer.precision	receive
Synchronizing Distance	peer.distance	receive
Estimated Drift Rate	peer.drift	receive
Reference Clock Identifier	peer.refid	receive
Reference Timestamp	peer.reftime	receive
Originate Timestamp	peer.org	receive
Receive Timestamp	peer.rec	receive
Filter Register	peer.filter	filter algorithm
Delay Estimate	peer.delay	filter algorithm
Offset Estimate	peer.offset	filter algorithm
Dispersion Estimate	peer.dispersion	filter

Table 3.2. Peer Variables

Packet Variables	Name	Control
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
Stratum	pkt.stratum	transmit
Poll	pkt.poll	transmit
Precision	pkt.precision	transmit
Synchronizing Distance	pkt.distance	transmit
Estimated Drift Rate	pkt.drift	transmit
Reference Clock Identifier	pkt.refid	transmit
Reference Timestamp	pkt.reftime	transmit
Originate Timestamp	pkt.org	transmit
Receive Timestamp	pkt.rec	transmit
Transmit Timestamp	pkt.xmt	transmit

Table 3.3. Packet Variables

Parameters	Name	Value
NTP Version	NTP.VERSION	1
NTP Port	NTP.PORT	123
Minimum Polling Interval	NTP.MINPOLL	6 (64 sec)
Maximum Polling Interval	NTP.MAXPOLL	10 (1024 sec)
Maximum Dispersion	NTP.MAXDISP	65535 ms
Reachability Register Size	PEER.WINDOW	8
Shift Register Size	PEER.SHIFT	4/8
Dispersion Threshold	PEER.THRESHOLD	500 ms
Filter Weight	PEER.FILTER	.5
Select Weight	PEER.SELECT	.75

Table 3.4. Parameters

Following is a description of the various variables used in the protocol. Additional details on formats and use are presented in later sections and appendices.

3.2.1. Common Variables

The following variables are common to the system, peer and packet classes.

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 time scale. The bits are set before 23:59 on the day of insertion and reset after 00:01 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 are coded as follows:

00	no warning (day has 86400 seconds)
01	+1 second (day has 86401 seconds) seconds)
10	-1 second (day has 86399 seconds) seconds)
11	alarm condition (clock not synchronized)

In all except the alarm condition (11) 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 logical clock is not synchronized, such as when first coming up or after an extended period when no outside reference source is available.

Stratum (sys.stratum, peer.stratum, pkt.stratum)

This is an integer indicating the stratum of the logical clock. A value of zero is interpreted as unspecified, one as a primary clock (synchronized by outside means) and remaining values as the stratum level (synchronized by NTP). For comparison purposes a value of zero is considered greater than any other value.

Peer Poll Interval (peer.ppoll, pkt.poll)

This is a signed integer used only in symmetric mode and indicating the minimum interval between messages sent to the peer, in seconds as a power of two. For instance, a value of six

indicates a minimum interval of 64 seconds. The value of this variable must not be less than NTP.MINPOLL and must not be greater than NTP.MAXPOLL.

Precision (sys.precision, peer.precision, pkt.precision)

This is a signed integer indicating the precision of the logical clock, in seconds to the nearest power of two. For instance, a 60-Hz line-frequency clock would be assigned the value -6, while a 1000-Hz crystal-derived 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.

Estimated Drift Rate (sys.drift, peer.drift, pkt.drift)

This is a fixed-point number indicating the estimated drift rate of the local clock, in dimensionless units.

Reference Clock Identifier (sys.refid, peer.refid, pkt.refid)

This is a code identifying the particular reference clock or server. The interpretation of the value depends on the stratum. For stratum values of zero (unspecified) or one (primary clock), the value is an ASCII string identifying the reason or clock, respectively. For stratum values greater than one (synchronized by NTP), the value is the 32-bit Internet address of the reference server.

Reference Timestamp (sys.reftime, peer.reftime, pkt.reftime)

This is the local time, in timestamp format, when the logical clock was last updated. If the logical clock has never been synchronized, the value is zero.

3.2.2. System Variables

The following variables are used by the operating system in order to synchronize the logical clock.

Logical 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 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

Following is a list of state variables used by the peer management and measurement functions. There is one set of these variables for every peer operating in client mode or symmetric mode.

Peer State (peer.state)

This is a bit-encoded quantity used for various control functions.

Host Poll Interval (peer.hpoll)

This is a signed integer used only in symmetric mode and indicating the minimum interval between messages expected from the peer, in seconds as a power of two. For instance, a value of six indicates a minimum interval of 64 seconds. The value of this variable must not be less than NTP.MINPOLL and must not be greater than NTP.MAXPOLL.

Reachability Register (peer.reach)

This is a code used to determine the reachability status of the peer. It is used as a shift register, with bits entering from the least significant (rightmost) end. The size of this register is specified as PEER.SHIFT bits.

Peer Timer (peer.timer)

This is an integer counter used to control the interval between transmitted NTP messages.

Timer Threshold (peer.threshold)

This is the timer value which, when reached, causes the timeout procedure to be executed.

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.

3.2.4. Packet Variables

Following is a list of variables used in NTP messages in addition to the common variables above.

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.

Transmit Timestamp (pkt.xmt)

This is the local time, in timestamp format, at which the NTP message departed the sender.

3.2.5. Clock Filter Variables

When the filter and selection algorithms suggested in Section 4 are used, the following state variables are defined. There is one set of these variables for every peer operating in client mode or symmetric mode.

Filter Register (peer.filter)

This is a shift register of PEER.WINDOW bits, where each stage is a tuple consisting of the measured delay concatenated 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 logical clock has just been reset so as to cause a significant discontinuity in local time.

Delay Estimate (peer.delay)

This is a signed, fixed-point number indicating the latest delay estimate output from the filter, in seconds. While the number is signed, only those values greater than zero represent valid delay estimates.

Offset Estimate (peer.offset)

This is a signed, fixed-point number indicating the latest offset estimate output from the filter, in seconds.

Dispersion Estimate (peer.dispersion)

This is a fixed-point number indicating the latest dispersion estimate output from the filter, in scrambled units.

3.2.6. Parameters

Following is a list of 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 one (1).

NTP Port (NTP.PORT)

This is the port number (123) assigned by the Internet Number Czar to NTP.

Minimum Polling Interval (NTP.MINPOLL)

This is the minimum polling interval 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 allowed by any peer of the Internet system, currently set to 10 (1024 seconds).

Maximum Dispersion (NTP.MAXDISP)

This is the maximum dispersion assumed by the filter algorithms, currently set to 65535 milliseconds.

Reachability Register Size (PEER.WINDOW)

This is the size of the Reachability Register (peer.reach), currently set to eight (8) bits.

Shift Register Size (PEER.SHIFT)

When the filter and selection algorithms suggested in Section 4 are used, this is the size of the Clock Filter (peer.filter) shift register, in bits. For crystal-stabilized oscillators a value of eight (8) is suggested, while for mains-frequency oscillators a value of four (4) is suggested. Additional considerations are given in Section 5.

Dispersion Threshold (PEER.THRESHOLD)

When the filter and selection algorithms suggested in Section 4 are used, this is the threshold used to discard noisy data. While a value of 500 milliseconds 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 filter weight used to discard noisy data. While a value of 0.5 is suggested, the value may be changed to suit local conditions on particular peer paths.

Select Weight (PEER.SELECT)

When the selection algorithm suggested in Section 4 is used, this is the select weight used to discard outlyers. data. While a value of 0.75 is suggested, the value may be changed to suit local conditions on particular peer paths.

3.3. Modes of Operation

An NTP host can operate in three modes: client, server and symmetric. The mode of operation is determined by whether the source port (peer.srcport) or destination port (peer.dstport) peer variables contain the assigned NTP service port number NTP.PORT (123) as shown in the following table.

peer.srcport	peer.dstport	Mode
not NTP.PORT	not NTP.PORT	not possible
not NTP.PORT	NTP.PORT	server
NTP.PORT	not NTP.PORT	client
NTP.PORT	NTP.PORT	symmetric

A host operating in client mode occasionally sends an NTP message to a host operating in server mode. The server responds by simply interchanging addresses and ports, filling in the required information and returning the message to the client. Servers then need retain no state information between client requests. Clients are free to manage the intervals between sending NTP messages to suit local conditions.

In symmetric mode the client/server distinction disappears. Each host maintains a table with as many entries as active peers. Each entry includes a code uniquely identifying the peer (e.g., Internet address and port), together with status information and a copy of the timestamps last received. A host operating in symmetric mode periodically sends NTP messages to each peer including the latest copy of the timestamps. The intervals between sending NTP messages are managed jointly by the host and each peer using the polling variables peer.ppoll and peer.hpoll.

When a pair of peers operating in symmetric mode exchange NTP messages and each determines that the other is reachable, an association is formed. One or both peers must be in active state; that is, sending messages to the other regardless of reachability status. A peer not in active state is in passive state. If a peer operating in passive state discovers that the other peer is no longer reachable, it ceases sending messages and reclaims the storage and timer resources used by the association. A peer operating in client mode is always in active state, while a peer operating in server mode is always in passive state.

3.4. Event Processing

The significant events of interest in NTP occur upon expiration of the peer timer, one of which is dedicated to each peer operating in symmetric or client modes, 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. Timeout Procedure

The timeout procedure is called in client and symmetric modes when the peer timer (peer.timer) reaches the value of the timer threshold (peer.threshold) variable. First, the reachability register (peer.reach) is shifted one position to the left and a zero replaces the vacated bit. Then an NTP message is constructed and sent to the peer. If operating in active state or in passive state and peer.reach is nonzero (reachable), the peer.timer is reinitialized (resumes counting from zero) and the value of peer.threshold is set to:

```
peer.threshold <- max( min( peer.ppoll, peer.hpoll,
                  NTP.MAXPOLL), NTP.MINPOLL) .
```

If operating in active state and peer.reach is zero (unreachable), the peer variables are updated as follows:

```
peer.hpoll <- NTP.MINPOLL
peer.disp <- NTP.MAXDISP
peer.filter <- 0 (cleared)
peer.org <- 0
peer.rec <- 0
```

Then the clock selection algorithm is called, which may result in a new clock source (sys.peer). In other cases the protocol ceases operation and the storage and timer resources are reclaimed for subsequent use.

An NTP message is constructed as follows (see Appendices A and B for formats). First, 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.distance <- sys.distance
pkt.version <- NTP.VERSION      pkt.drift <- sys.drift
pkt.stratum <- sys.stratum      pkt.refid <- sys.refid
pkt.poll <- peer.hpoll          pkt.reftime <- sys.reftime
pkt.precision <- sys.precision
```

Finally, the NTP packet timestamp variables are copied, depending on whether the peer is operating in symmetric mode and reachable, in

symmetric mode and not reachable (but active) or in client mode:

Symmetric Reachable	Symmetric Active	Client
pkt.org <- peer.org	pkt.org <- 0	pkt.org <- sys.clock
pkt.rec <- peer.rec	pkt.rec <- 0	pkt.rec <- sys.clock
pkt.xmt <- sys.clock	pkt.xmt <- sys.clock	pkt.xmt <- sys.clock

Note that the order of copying should be designed so that the time to perform the copy operations themselves does not degrade the measurement accuracy, which implies that the sys.clock values should be copied last. The reason for the choice of zeros to fill the pkt.org and pkt.rec packet variables in the symmetric unreachable case is to avoid the use of old data after a possibly extensive period of unreachability. The reason for the choice of sys.clock to fill these variables in the client case is that, if for some reason the NTP message is returned by the recipient unaltered, as when testing with an Internet-echo server, this convention still allows at least the roundtrip time to be accurately determined without special handling.

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; however, exceptions may be advised on a case-by-case basis at times when the version number is changed.

If the clock of the sender is unsynchronized (pkt.leap = 11), or the receiver is in server mode or the receiver is in symmetric mode and the stratum of the sender is greater than the stratum of the receiver (pkt.stratum > sys.stratum), the message is simply returned to the sender along with the timestamps. In this case the addresses and ports are interchanged in the IP and UDP headers:

```
pkt.srcadr <-> pkt.dstadr       pkt.srcport <-> pkt.dstport
```

The following packet variables are updated from the system variables:

```
pkt.leap <- sys.leap            pkt.distance <- sys.distance
pkt.version <- NTP.VERSION      pkt.drift <- sys.drift
pkt.stratum <- sys.stratum      pkt.refid <- sys.refid
pkt.precision <- sys.precision  pkt.reftime <- sys.reftime
```

Note that the pkt.poll packet variable is unchanged. The timestamps are updated in the order shown:

```
pkt.org <- pkt.xmt
pkt.rec <- sys.clock
pkt.xmt <- sys.clock
```

Finally, the message is forwarded to the sender and the server receive procedure terminated at this point.

If the above is not the case, 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 and symmetric mode is not indicated (either pkt.srcport or pkt.dstport not equal to NTP.PORT), the message must be a reply to a previously sent message from a client which is no longer in operation. In this case the message is dropped and the receive procedure terminated at this point.

If there is no match and symmetric mode is indicated, (both pkt.srcport and pkt.dstport equal to NTP.PORT), an implementation-specific instantiation procedure is called to create and initialize a new set of peer variables and start the peer timer. The following peer variables are set from the IP and UDP headers:

```
peer.srcadr <- pkt.srcadr       peer.srcport <- pkt.srcport
peer.dstadr <- pkt.dstadr       peer.dstport <- pkt.dstport
```

The following peer variables are initialized:

```
peer.state <- symmetric (passive)
peer.timer <- 0 (enabled)
peer.hpoll <- NTP.MINPOLL
peer.disp <- NTP.MAXDISP
```

The remaining peer variables are undefined and set to zero.

Assuming that instantiation is complete and that match occurs, the least significant bit of the reachability register (peer.reach) is set, indicating the peer is now reachable. The following peer variables are copied (rescaled as necessary) from the NTP packet variables and system variables:

```
peer.leap <- pkt.leap           peer.distance <- pkt.distance
peer.stratum <- pkt.stratum     peer.drift <- pkt.drift
peer.ppoll <- pkt.poll          peer.refid <- pkt.refid
peer.precision <- pkt.precision peer.reftime <- pkt.reftime
peer.org <- pkt.xmt             peer.rec <- sys.clock
peer.threshold <- max( min( peer.ppoll, peer.hpoll,
                NTP.MAXPOLL), NTP.MINPOLL)
```

If either or both the pkt.org or pkt.rec packet variables are zero, the sender did not have reliable values for them, so the receive procedure is terminated at this point. If both of these variables are nonzero, the roundtrip delay and clock offset relative to the peer are calculated as follows. Number the times of sending and receiving NTP messages as shown in Figure 3.1 and let i be an even integer. Then t(i-3), t(i-2) and t(i-1) and t(i) are the contents of the pkt.org, pkt.rec, pkt.xmt and peer.rec variables respectively.

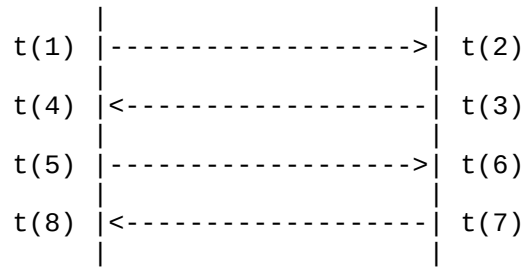


Figure 3.1. Calculating Delay and Offset

The roundtrip delay d and clock offset c of the receiving peer relative to the sending peer is:

$$d = (t(i) - t(i-3)) - (t(i-1) - t(i-2))$$
$$c = [(t(i-2) - t(i-3)) + (t(i-1) - t(i))]/2 .$$

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

The c and d values are then input to the clock filter algorithm to produce the delay estimate (peer.delay) and offset estimate (peer.offset) for the peer involved. If d becomes nonpositive due to low delays, long polling intervals and high drift rates, it should be

considered invalid; however, even under these conditions it may still be useful to update the local clock and reduce the drift rate to the point that d becomes positive again. 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.

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 by an ordinary peer. The clocks are usually polled once or twice a minute and the returned timecheck used to produce a new update for the logical clock. The update procedure is then called with the following assumed peer variables:

```
peer.offset <- timecheck - sys.clock
peer.delay <- as determined
peer.dispersion <- 0
peer.leap <- selected by operator, ordinarily 00
peer.stratum <- 0
peer.distance <- 0
peer.refid <- ASCII identifier
peer.reftime <- timecheck
```

In this case the peer.delay and peer.refid can be constants reflecting the type and accuracy of the clock. By convention, the value for peer.delay is ten times the expected mean error of the clock, for instance, 10 milliseconds for a WWVB clock and 1000 milliseconds for a less accurate WWV clock, but with a floor of 100 milliseconds. Other peer variables such as the peer timer and reachability register can be used to control the polling interval and to confirm the clock is operating correctly. In this way the clock filter and selection algorithms operate in the usual way and can be used to mitigate the clock itself, should it appear to be operating correctly, yet deliver bogus time.

3.4.3. Update Procedure

The update procedure is called when a new delay/offset estimate is available. First, the clock selection algorithm determines the best peer on the basis of estimated accuracy and reliability, which may result in a new clock source (sys.peer). If sys.peer points to the peer data structure with the just-updated estimates, the state variables of that peer are used to update the system state variables

as follows:

```
sys.leap <- peer.leap
sys.stratum <- peer.stratum + 1
sys.distance <- peer.distance + peer.delay
sys.refid <- peer.srcadr
sys.reftime <- peer.rec
```

Finally, the logical clock procedure is called with peer.offset as argument to update the logical clock (sys.clock) and recompute the estimated drift rate (sys.drift). It may happen that the logical clock may be reset, rather than slewed to its final value. In this case the peer variables of all reachable peers are are updated as follows:

```
peer.hpoll <- NTP.MINPOLL
peer.disp <- NTP.MAXDISP
peer.filter <- 0 (cleared)
peer.org <- 0
peer.rec <- 0
```

and the clock selection algorithm is called again, which results in a null clock source (sys.peer = 0). A new selection will occur when the filters fill up again and the dispersion settles down.

Specification of the clock selection algorithm and logical clock procedure 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 logical clock procedure is described in Section 5. The clock selection algorithm described in Section 4 usually picks the server at the highest 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 system similar to those used by some digital telephone networks.

3.4.4. Initialization Procedures

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

```
sys.clock <- best available estimate
sys.leap <- 11 (unsynchronized)
sys.stratum <- 0 (undefined)
sys.precision <- as required
sys.distance <- 0 (undefined)
sys.drift <- as determined
sys.refid <- 0 (undefined)
sys.reftime <- 0 (undefined)
```

The logical clock sys.clock is presumably undefined at reboot; however, in some designs such as the Fuzzball an estimate is available from the reboot environment. The sys.precision variable is determined by the intrinsic architecture of the local hardware clock. The sys.drift variable is determined as a side effect of subsequent logical clock updates, from whatever source.

Next, an implementation-specific instantiation procedure is called repeatedly to establish the set of client peers or symmetric (active) peers which will actively probe other servers during regular operation. The mode and addresses of these peers is determined using information read during the reboot procedure or as the result of operator commands.

4. Filtering 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.

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

4.1. Clock Filter Algorithm

The clock filter algorithm is executed upon arrival of each NTP message that results in new delay/offset sample pairs. New sample

pairs are shifted into the filter register (peer.filter) from the left end, causing first zeros then old sample pairs to shift off the right end. 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.delay) and offset estimate (peer.offset) 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.dispersion) is then computed as the weighted sum of the offsets in the list. Assume the list has PEER.SHIFT entries, the first m of which contain valid samples in order of increasing delay. If X(i) (0 =< i < PEER.SHIFT) is the offset of the ith sample, then,

```
        d(i) = |X(i) - X(0)|    if i < m and |X(i) - X(0)| < 2^15
        d(i) = 2^15 - 1         otherwise

                peer.dispersion = Sum(d(i)*w^i) ,
                        (0 =< i < PEER.SHIFT)
```

where w < 1 is a weighting factor experimentally adjusted to match typical offset distributions. The peer.dispersion variable is intended for use as a quality indicator, with increasing values associated with decreasing quality. The intent is that samples with a peer.dispersion exceeding a configuration threshold will not be used in subsequent processing. The prototype implementation uses a weighting factor w = 0.5, also called PEER.FILTER, and a threshold PEER.THRESHOLD of 500 ms, which insures that all stages of peer.filter are filled and contain offsets within a few seconds of each other.

4.2. Clock Selection Algorithm

The clock selection algorithm uses the values of peer.delay, peer.offset and peer.dispersion calculated by the clock filter algorithm and is called when these values change or when the reachability status changes. It constructs a list of candidate estimates according to a set of criteria designed to maximize accuracy and reliability, then sorts the list in order of estimated precision. Finally, it repeatedly casts out outlyers on the basis of dispersion until only a single candidate is left.

The selection process operates on each peer in turn and inspects the various data captured from the last received NTP message header, as well as the latest clock filter estimates. It selects only those peers for which the following criteria are satisfied:

1. The peer must be reachable and operating in client or symmetric modes.

2. The peer logical clock must be synchronized, as indicated by the Leap Indicator bits being other than 11.

3. If the peer is operating at stratum two or greater, it must not be synchronized to this host, which means its reference clock identifier (peer.refid) must not match the Internet address of this host. This is analogous to the split-horizon rule used in some variants of the Bellman-Ford routing algorithm.

4. The sum of the peer synchronizing distance (peer.distance) plus peer.delay must be less than 2^13 (8192) milliseconds. Also, the peer stratum (peer.stratum) must be less than eight and peer.dispersion must be less than a configured threshold PEER.THRESHOLD (currently 500 ms). These range checks were established through experience with the prototype implementation, but may be changed in future.

For each peer which satisfies the above criteria, a sixteen-bit keyword is constructed, with the low-order thirteen bits the sum of peer.distance plus peer.delay and the high-order three bits the peer.stratum reduced by one and truncated to three bits (thus mapping zero to seven). The keyword together with a pointer to the peer data structure are inserted according to increasing keyword values and truncated at a maximum of eight entries. The resulting list represents the order in which peers should be chosen according to the estimated precision of measurement. If no keywords are found, the clock source variable (sys.peer) is set to zero and the algorithm terminates.

The final procedure is designed to detect falsetickers or other conditions which might result in gross errors. Let m be the number of samples remaining in the list. For each i (0 =< i < m) compute the dispersion d(i) of the list relative to i:

$$d(i) = \sum_{(0 \le j < m)} |X(j) - X(i)| * w^j ,$$

where w < 1 is a weighting factor experimentally adjusted for the desired characteristic (see below). Then cast out the entry with maximum d(i) or, in case of ties, the maximum i, and repeat the procedure. When only a single entry remains in the list, sys.peer is set as its peer data structure pointer and the peer.hpoll variable in that structure is set to NTP.MINPOLL as required by the logical clock mechanism described in Section 5.

This procedure is designed to favor those peers near the head of the list, which are at the highest stratum and lowest delay and presumably can provide the most precise time. With proper selection of weighting factor w, also called PEER.SELECT, 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.

In order to see how this procedure works to select outlyers, consider the case of three entries and assume that one or more of the offsets are clustered about zero and others are clustered about one. For w = 0.75 as used in the prototype implementations and multiplying by 16 for convenience, the first entry has weight w^0 = 16, the second w^1 = 12 and the third w^2 = 9. Table X shows for all combinations of peer offsets the calculated dispersion about each of the three entries, along with the results of the procedure.

Peer	0	1	2	Dispersion			Cast	Result
Weight	16	12	9	0	1	2	Out	
	0	0	0	0	0	0	2	0	0
	0	0	1	9	9	28	2	0	0
	0	1	0	12	25	12	1	0	0
	0	1	1	21	16	16	0	1	1
	1	0	0	21	16	16	0	0	0
	1	0	1	12	25	12	1	1	1
	1	1	0	9	9	28	2	1	1
	1	1	1	0	0	0	2	1	1

Table 4.1. Outlyer Selection Procedure

In the four cases where peer 0 and peer 1 disagree, the outcome is determined by peer 2. Similar outcomes occur in the case of four peers. While these outcomes depend on judicious choice of w, the behavior of the algorithm is substantially the same for values of w between 0.5 and 1.0.

4.3. Variable-Rate Polling

As NTP service matures in the Internet, the resulting network traffic can become burdensome, especially in the primary service net. In this expectation, it is useful to explore variable-rate polling, in which the intervals between NTP messages can be adjusted to fit prevailing network conditions of delay dispersion and loss rate. The prototype NTP implementation uses this technique to reduce the network overheads to one-sixteenth the maximum rate, depending on observed dispersion and loss.

The prototype implementation adjusts the polling interval peer.hpoll in response to the reachability register (peer.reach) variable along with the dispersion (peer.dispersion) variable. So long as the clock source variable (sys.peer) does not point to the peer data structure, peer.reach is nonzero (reachable) and peer.dispersion is less than the PEER.THRESHOLD parameter, the value of peer.hpoll is increased by one for each call on the update procedure, subject to a maximum of NTP.MAXPOLL. Following the timeout procedure, if peer.reach indicates messages have not been received for the preceding two polling intervals (low-order two bits are zero), the value of peer.hpoll is decreased by one, subject to a minimum of NTP.MINPOLL. If peer.reach becomes zero (unreachable), the value of peer.hpoll is set to NTP.MINPOLL.

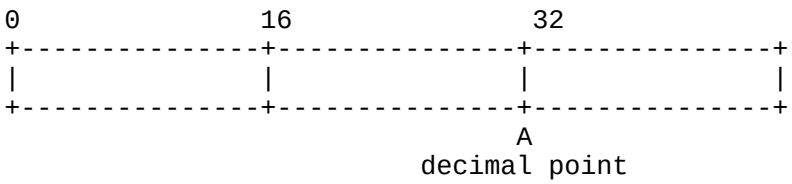
The result of the above mechanism is that the polling intervals for peers not selected for synchronization and in symmetric mode creep upwards once the filter register (peer.filter) has filled and the peer.dispersion has settled down, but decrease again in case peer.dispersion increases or the loss rate increases or the peer becomes unreachable.

5. Logical Clocks

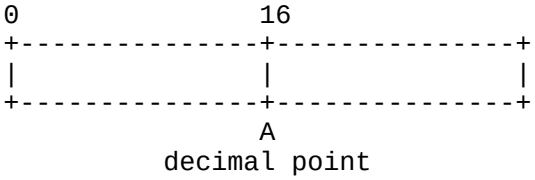
In order to implement a logical clock, the host must be equipped with a hardware clock consisting of an oscillator and interface and capable of the required precision and stability. The logical clock is adjusted by means of periodic offset corrections computed by NTP or some other time-synchronization protocol such as Hellospeak [15] or the Unix 4.3bsd TSP [20]. Following is a description of the Fuzzball logical clock, which includes provisions for precise time and frequency adjustment and can maintain time to within a millisecond and frequency to within a day per millisecond.

The 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 register, the Counter Register, is used as an auxiliary counter. The configuration and decimal point of these registers are shown in Figure 5.1.

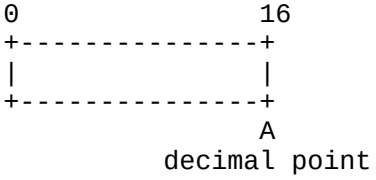
Clock Register



Clock-Adjust Register



Drift-Compensation Register



Counter Register

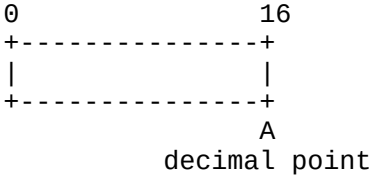


Figure 5.1. Clock Registers

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 buffered counter driven by a crystal oscillator. A counter overflow is signalled by an interrupt, which results in an increment of the Clock Register at bit 15 and the propagation of carries as required. 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 16-2/3 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.

5.1. Uniform Phase Adjustments

Left uncorrected, the logical clock runs at the rate of its intrinsic oscillator. A correction is introduced as a signed 32-bit integer in milliseconds, which is added to the Drift-Compensation Register and also replaces bits 0-15 of the Clock-Adjust Register, with bits 16-31 set to zero. At adjustment intervals of CLOCK.ADJ a correction consisting of two components is computed. The first (phase) component consists 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 Drift-Compensation Register shifted right CLOCK.FREQ bits. 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.

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. This can be done by buffering the corrections and adding them to the increment at the time the Clock Register is next updated. Monotonicity is insured with the parameters shown in Table 5.1, as long as the increment is at least 2 ms. This table shows the above parameters and others discussed below for both a crystal-stabilized oscillator and a mains-frequency oscillator.

Parameter	Name	Crystal	Mains
Update Interval	CLOCK.ADJ	4 sec	1 sec
Phase Shift	CLOCK.PHASE	-8	-9
Frequency Shift	CLOCK.FREQ	-16	-16
Maximum Aperture	CLOCK.MAX	+-128 ms	+-256 ms
Shift Register Size	PEER.SHIFT	8	4
Host Poll Interval	peer.hpoll	NTP.MINPOLL (64 sec)	NTP.MINPOLL (64 sec)

Table 5.1. Clock Parameters

The above design constitutes a second-order phase-lock loop which adjusts the logical clock phase and frequency to compensate for the intrinsic oscillator jitter, wander and drift. Simulation of a loop

with parameters chosen from Table 5.1 for a crystal-stabilized oscillator and the clock filter described in Section 4 results in the following transient response: For a phase correction of 100 ms the loop reaches zero error in 34 minutes, overshoots 7 ms in 76 minutes and settles to less than 1 ms in about four hours. The maximum frequency error is about 6 ppm at 40 minutes and returns to less than 1 ppm in about eight hours. For a frequency correction of 10 ppm the loop settles to within 1 ppm in about nine hours and to within 0.1 ppm in about a day. These characteristics are appropriate for typical computing equipment using board-mounted crystals without oven temperature control.

In those cases where mains-frequency oscillators must be used, the loop parameters must be adapted for the relatively high jitter and wander characteristics of the national power grid, in which diurnal peak-to-peak phase excursions can exceed four seconds. Simulation of a loop with parameters chosen from Table 5.1 for a mains-frequency oscillator and the clock filter described in Section 4 results in a transient response similar to the crystal-stabilized case, but with time constants only one-fourth those in that case. When presented with actual phase-offset data for typical Summer days when the jitter and wander are the largest, the loop errors are in the order of a few tens of milliseconds, but not greater than 150 ms.

The above simulations assume the clock filter algorithm operates to select the oldest sample in the shift register at each step; that is, the filter operates as a delay line with delay equal to the polling interval times the number of stages. This is a worst-case scenario, since the larger the overall delay the harder it is to maintain low loop errors together with good transient response. The parameters in Table 5.1 were experimentally determined with this scenario and the constraint that the polling interval could not be reduced below 64 seconds. With these parameters it is not possible to increase the polling interval above 64 seconds without significant increase in loop error or degradation of transient response. Thus, when a clock is selected according to the algorithms of Section 4, the polling interval peer.hpoll is always set at NTP.MINPOLL.

5.2. Nonuniform Phase Adjustments

When the magnitude of a correction exceeds a 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. In practice the necessity to do this is rare and 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 5.1 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 occured. 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.

5.3. 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 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 logical 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.

5.4. Estimating Errors

After an NTP message is received and until the next one is received, the accuracy of the local clock can be expected to degrade somewhat. The magnitude of this degradation depends on the error at the last update time together with the drift of the local oscillator with respect to time. It is possible to estimate both the error and drift rate from data collected during regular operation. These data can be used to determine the rate at which NTP neighbors should exchange NTP messages and thus control net overheads.

NTP messages include the local-clock precision of the sender, as well

as the reference time, estimated drift and a quantity called the synchronizing distance. The precision of the local clock, together with its peer clocks, establishes the short-term jitter characteristics of the offset estimates. The reference time and estimated drift of the sender provide an error estimate at the time the latest update was received. The synchronizing distance provides an estimate of error relative to the primary reference source and is used by the filtering algorithms to improve the quality and reliability of the offset estimates.

Estimates of error and drift rate are not essential for the correct functioning of the clock algorithms, but do improve the accuracy and adjustment with respect to net overheads. The estimated error allows the recipient to compute the rate at which independent samples are required in order to maintain a specified estimated error. The estimated drift rate allows the recipient to estimate the optimum polling interval.

It is possible to compute the estimated drift rate of the local clock to a high degree of precision by simply adding the n offsets received during an interval T to an accumulator. If X1 and X2 are the values of the accumulator at the beginning and end of T, then the estimated drift rate r is:

$$r = \frac{X2 - X1}{n} \cdot \frac{n}{T} .$$

The intrinsic (uncorrected) drift rate of typical crystal oscillators under room-temperature conditions is in the order of from a few parts per million (ppm) to as much as 100 ppm, or up to a few seconds per day. For most purposes the drift of a particular crystal oscillator is constant to within perhaps one ppm. Assuming T can be estimated to within 100 ms, for example, it would take about a day of accumulation to estimate r to an uncertainty in the order of one ppm.

Some idea of the estimated error of the local clock can be derived from the variance of the offsets about the mean per unit time. This can be computed by adding the n offset squares received during T to an accumulator. If Y1 and Y2 are the values of the accumulator at the beginning and end of T, then the estimated error s is:

$$s = \left(\frac{Y2 - Y1}{n} - \frac{(X2 - X1)^2}{n * n} \right) \frac{n}{T} .$$

The quantities r and s have direct utility to the peer as noted above. However, they also have indirect utility to the recipient of

an NTP message sent by that peer, since they can be used as weights in such algorithms as described in [22], as well as to improve the estimates during periods when offsets are not available. It is most useful if the latest estimate of these quantities are available in each NTP message sent; however, considerable latitude remains in the details of computation and storage.

The above formulae for r and s imply equal weighting for offsets received throughout the accumulation interval T. One way to do this is using a software shift register implemented as a circular buffer. A single pointer points to the active entry in the buffer and advances around one entry as each new offset is stored. There are two accumulators, one for the offset and the other for its squares. When a new offset arrives, a quantity equal to the new offset minus the old (active) entry is added to the first accumulator and the square of this quantity is added to the second. Finally, the offset is stored in the circular buffer.

The size of the circular buffer depends on the accumulation interval T and the rate offsets are produced. In many reachability and routing algorithms, such as GGP, EGP and local-net control algorithms, peers exchange messages on the order of once or twice a minute. If NTP peers exchanged messages at a rate of one per minute and if T were one day, the circular buffer would have to be 1440 words long; however, a less costly design might aggregate the data in something like half-hour segments, which would reduce the length of the buffer to 48 words while not significantly affecting the quality of the data.

6. References

1. Lamport, L., "Time, Clocks and the Ordering of Events in a Distributed System", Communications of the ACM, Vol. 21, No. 7, pgs. 558-565, July 1978.

2. "Time and Frequency Dissemination Services", NBS Special Publication No. 432, US Department of Commerce, 1979.

3. Lindsay, W., and A. Kantak, "Network Synchronization of Random Signals", IEEE Trans. Comm., COM-28, No. 8, pgs. 1260-1266, August 1980.

4. Braun, W., "Short Term Frequency Effects in Networks of Coupled Oscillators", IEEE Trans. Comm., COM-28, No. 8, pgs. 1269-1275, August 1980.

5. Mitra, D., "Network Synchronization: Analysis of a Hybrid of Master-Slave and Mutual Synchronization", IEEE Trans. Comm. COM-28, No. 8, pgs. 1245-1259, August 1980.

6. Postel, J., "User Datagram Protocol", RFC-768, USC/Information Sciences Institute, August 1980.

7. Mills, D., "Time Synchronization in DCNET Hosts", IEN-173, COMSAT Laboratories, February 1981.

8. Mills, D., "DCNET Internet Clock Service", RFC-778, COMSAT Laboratories, April 1981.

9. Su, Z., "A Specification of the Internet Protocol (IP) Timestamp Option", RFC-781, SRI International, May 1981.

10. Defense Advanced Research Projects Agency, "Internet Protocol", RFC-791, USC/Information Sciences Institute, September 1981.

11. Defense Advanced Research Projects Agency, "Internet Control Message Protocol", RFC-792, USC/Information Sciences Institute, September 1981.

12. Postel, J., "Daytime Protocol", RFC-867, USC/Information Sciences Institute, May 1983.

13. Postel, J., "Time Protocol", RFC-868, USC/Information Sciences Institute, May 1983.

14. Mills, D., "Internet Delay Experiments", RFC-889, M/A-COM Linkabit, December 1983.

15. Mills, D., "DCN Local-Network Protocols", RFC-891, M/A-COM Linkabit, December 1983.

16. Gusella, R., and S. Zatti, "TEMPO - A Network Time Controller for a Distributed Berkeley UNIX System", IEEE Distributed Processing Technical Committee Newsletter 6, No. SI-2, pgs. 7-15, June 1984. Also in: Proc. Summer 1984 USENIX, Salt Lake City, June 1984.

17. Halpern, J., Simons, B., Strong, R., and D. Dolly, "Fault-Tolerant Clock Synchronization", Proc. Third Annual ACM Symposium on Principles of Distributed Computing, pgs. 89-102, August 1984.

18. Lundelius, J., and N. Lynch, "A New Fault-Tolerant Algorithm for Clock Synchronization:, Proc. Third Annual ACM Symposium on Principles of Distributed Computing, pgs. 75-88, August 1984.

19. Lamport, L., and P. Melliar-Smith "Synchronizing Clocks in the Presence of Faults", JACM 32, No. 1, pgs. 52-78, January 1985.

20. 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.

21. Marzullo, K., and S. Owicki, "Maintaining the Time in a Distributed System", ACM Operating Systems Review 19, No. 3, pgs. 44-54, July 1985.

22. Mills, D., "Algorithms for Synchronizing Network Clocks", RFC-956, M/A-COM Linkabit, September 1985.

23. Mills, D., "Experiments in Network Clock Synchronization", RFC-957, M/A-COM Linkabit, September 1985.

24. Mills, D., "Network Time Protocol (NTP)", RFC-958, M/A-COM Linkabit, September 1985.

25. 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.

26. Sams, H., "Reference Data for Engineers: Radio, Electronics, Computer and Communications (Seventh Edition)", Indianapolis, 1985.

27. Schneider, F., "A Paradigm for Reliable Clock Synchronization", Technical Report TR 86-735, Cornell University, February 1986.

28. Tripathi, S., and S. Chang, "ETempo: A Clock Synchronization Algorithm for Hierarchical LANs - Implementation and Measurements", Systems Research Center Technical Report TR-86-48, University of Maryland, 1986.

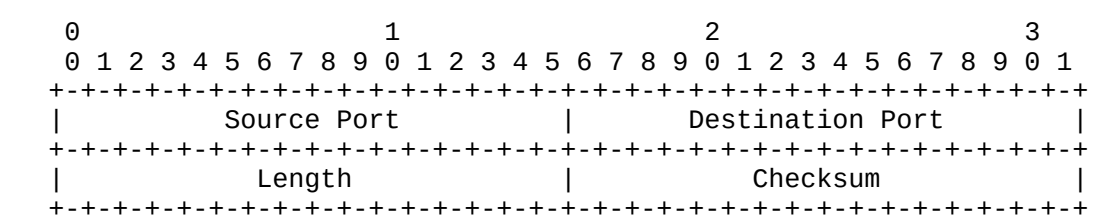
29. Bertsekas, D., and R. Gallager, "Data Networks", Prentice-Hall, Englewood Cliffs, NJ, 1987.

30. Srikanth, T., and S. Toueg. "Optimal Clock Synchronization", JACM 34, No. 3, pgs. 626-645, July 1987.

31. Rickert, N., "Non Byzantine Clock Synchronization - A Programming Experiment", ACM Operating Systems Review 22, No. 1, pgs. 73-78, January 1988.

Appendix A. UDP Header Format

An NTP packet consists of the UDP header followed by the NTP data portion. The format of the UDP header and the interpretation of its fields are described in [6] and are not part of the NTP specification. They are shown below for completeness.



Source Port

UDP source port number. In the case of a client request this field is assigned by the client host, while for a server reply it is copied from the Destination Port field of the client request. In the case of symmetric mode, both the Source Port and Destination Port fields are assigned the NTP service-port number 123.

Destination Port

UDP destination port number. In the case of a client request this field is assigned the NTP service-port number 123, while for a server reply it is copied from the Source Port field of the client request. In the case of symmetric mode, both the Source Port and Destination Port fields are assigned the NTP service-port number 123.

Length

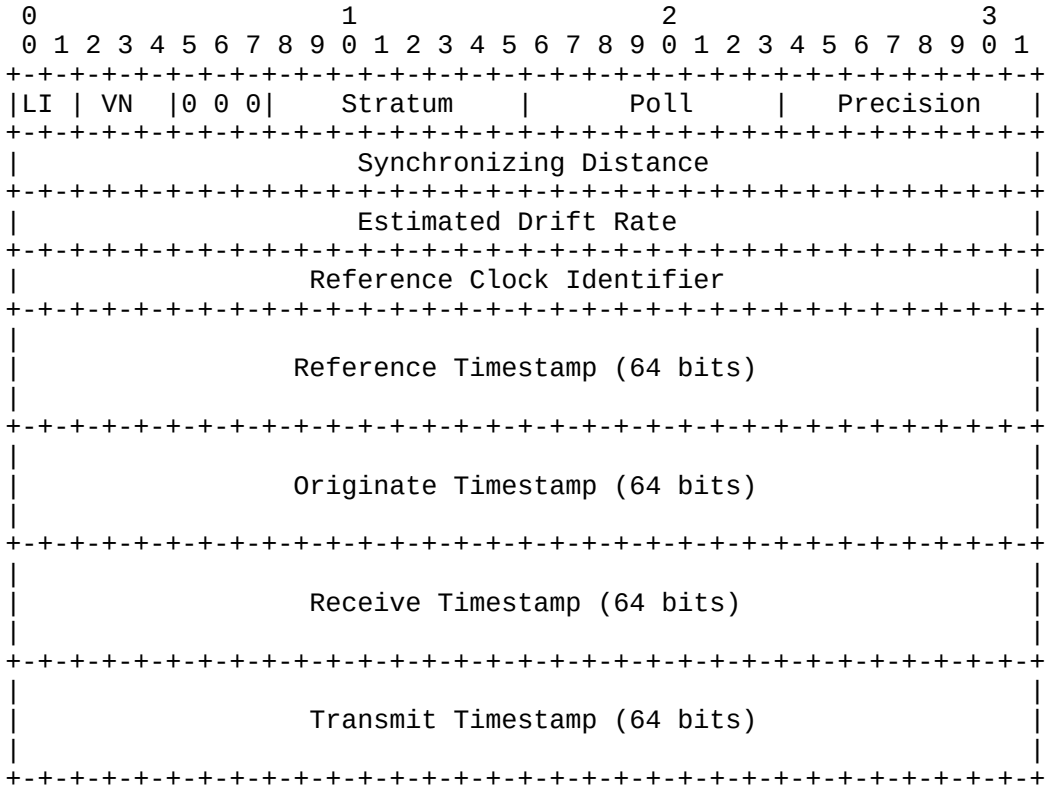
Length of the request or reply, including UDP header, in octets

Checksum

Standard UDP checksum

Appendix B. NTP Data Format - Version 1

The format of the NTP data portion, which immediately follows the UDP header, is shown below along with a description of its fields.



Leap Indicator (LI)

Two-bit code warning of impending leap-second to be inserted at the end of the last day of the current month. Bits are coded as follows:

00	no warning
01	+1 second (following minute has 61 seconds)
10	-1 second (following minute has 59 seconds)
11	alarm condition (clock not synchronized)

Version Number (VN)

Three-bit code indicating the version number, currently one (1).

Reserved

Three-bit field consisting of all zeros and reserved for future use.

Stratum

Integer identifying stratum level of local clock. Values are defined as follows:

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

Poll

Signed integer indicating the maximum interval between successive messages, in seconds to the nearest power of two.

Precision

Signed integer indicating the precision of the local clock, in seconds to the nearest power of two.

Synchronizing Distance

Fixed-point number indicating the estimated roundtrip delay to the primary synchronizing source, in seconds with fraction point between bits 15 and 16.

Estimated Drift Rate

Fixed-point number indicating the estimated drift rate of the local clock, in dimensionless units with fraction point to the left of the most significant bit.

Reference Clock Identifier

Code identifying the particular reference clock. In the case of type 0 (unspecified) or type 1 (primary reference), this is a left-justified, zero-filled ASCII string, for example:

Type	Code	Meaning
0	DCN	Determined by DCN routing algorithm
1	WWVB	WWVB radio clock (60 kHz)
1	GOES	GOES satellite clock (468 MHz)
1	WWV	WWV radio clock (5/10/15 MHz)
(and others as necessary)		

In the case of type 2 and greater (secondary reference), this is the 32-bit Internet address of the reference host.

Reference Timestamp

Local time at which the local clock was last set or corrected.

Originate Timestamp

Local time at which the request departed the client host for the service host.

Receive Timestamp

Local time at which the request arrived at the service host.

Transmit Timestamp

Local time at which the reply departed the service host for the client host.

Appendix C. Timeteller Experiments

In order to update data collected in June 1985 and reported in RFC-957, a glorious three-day experiment was carried out in January 1988 with all the hosts and gateways listed in the NIC data base. Four packets were sent at five-second intervals to each host and gateway using UDP/NTP, UDP/TIME and ICMP/TIMESTAMP protocols and the clock offsets (in milliseconds) for each protocol averaged with respect to local time, which is synchronized via NTP to a radio-clock host. While the ICMP/TIMESTAMP protocol has much finer granularity (milliseconds) than UDP/TIME (seconds), it has no provisions for the date, so is not suitable as a time-synchronization protocol; however, it was included in the experiments both as a sanity check and in order to assess the precision of measurement.

In the latest survey of 5498 hosts and 224 gateways, 46 responded to UDP/NTP requests, 1158 to UDP/TIME and 1963 to ICMP/TIMESTAMP. By contrast, in the 1985 survey of 1775 hosts and 110 gateways, 163 responded to UDP/TIME requests and 504 to ICMP/TIMESTAMP. At that time there were no UDP/NTP implementations. There are many more hosts and gateways listed in the rapidly growing domain-name system, but not listed in the NIC data base, and therefore not surveyed. The results of the survey are given in Table C.1, which shows for each of the three protocols the error X for which the distribution function P[x =< X] has the value shown.

P[x=<X]	UDP/NTP	UDP/TIME	ICMP/TIMESTAMP
.1	11	4632	5698
.2	37	18238	27965
.3	66	38842	68596
.4	177	68213	127367
.5	364	126232	201908
.6	567	195950	285092
.7	3466	267119	525509
.8	20149	422129	2.91426E+06
.9	434634	807135	5.02336E+07
1	1.17971E+09	1.59524E+09	2.11591E+09

Table C.1. Distribution Functions

It can be seen that ten percent of the UDP/NTP responses show errors of 11 milliseconds or less and that ten percent of the UDP/TIME responses show errors greater than 807135 milliseconds (about 13 minutes). Fifty percent of the UDP/NTP timetellers are within 364 milliseconds, while fifty percent of the UDP/TIME tellers are within 126232 milliseconds (just over two minutes). Surprisingly, ICMP/TIMESTAMP responses show errors even larger than UDP/TIME.

However, the maximum error shown in all three protocols exceeded the range that could be recorded, in this case about 12 days. Clearly, there are good timetellers and bad.

Appendix D. Evaluation of Filtering Algorithms

A number of algorithms for deglitching and filtering time-offset data were described in RFC-956. These fall in two classes: majority-subset algorithms, which attempt to separate good subsets from bad by comparing their means, and clustering algorithms, which attempt to improve the estimate by repeatedly casting out outlyers. The former class was suggested as a technique to select the best (i.e. the most reliable) clocks from a population, while the latter class was suggested as a technique to improve the offset estimate for a single clock given a series of observations.

Following publication of RFC-956 and after further development and experimentation using typical Internet paths, a better algorithm was found for casting out outlyers from a continuous stream of offset observations spaced at intervals in the order of minutes. The algorithm is described as a variant of a median filter, in which a window consisting of the last n sample offsets is continuously updated and the median sample selected as the estimate. However, in the modified algorithm the outlyer (sample furthest from the median) is then discarded and the entire process repeated until only a single sample offset is left, which is then selected as the estimate.

The modified algorithm was found to be more resistant to glitches and to provide a more accurate estimate than the unmodified one. It has been implemented in the NTP daemons developed for the Fuzzball and Unix operating systems and been in regular operation for about two years. However, recent experiments have shown there is an even better one which provides comparable accuracy together with a much lower computational burden. The key to the new algorithm became evident through an examination of scatter diagrams plotting sample offset versus roundtrip delay.

To see how a scatter diagram is constructed, it will be useful to consider how offsets and delays are computed. Number the times of sending and receiving NTP messages as shown in Figure D.1 and let i be an even integer. Then the timestamps t(i-3), t(i-2) and t(i-1) and t(i) are sufficient to calculate the offset and delay of each peer relative to the other.

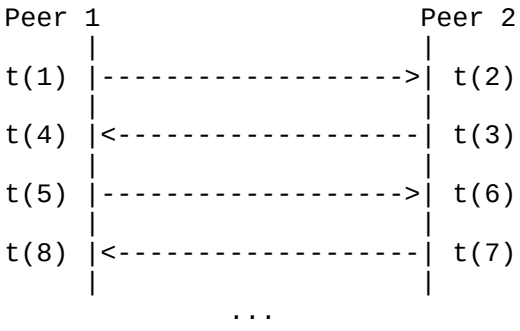


Figure D.1. Calculating Delay and Offset

The roundtrip delay d and clock offset c of the receiving peer relative to the sending peer are:

$$d = (t(i) - t(i-3)) - (t(i-1) - t(i-2))$$
$$c = [(t(i-2) - t(i-3)) + (t(i-1) - t(i))]/2 .$$

Two implicit assumptions in the above are that the delay distribution is independent of direction and that the intrinsic drift rates of the client and server clocks are small and close to the same value. If this is the case the scatter diagram would show the samples concentrated about a horizontal line extending from the point (d,c) to the right. However, this is not generally the case. The typical diagram shows the samples dispersed in a wedge with apex (d,c) and opening to the right. The limits of the wedge are determined by lines extending from (d,c) with slopes +0.5 and -0.5, which correspond to the locus of points as the delay in one direction increases while the delay in the other direction does not. In some cases the points are concentrated along these two extrema lines, with relatively few points remaining within the opening of the wedge, which would correspond to increased delays on both directions.

Upon reflection, the reason for the particular dispersion shown in the scatter diagram is obvious. Packet-switching nets are most often operated with relatively small mean queue lengths in the order of one, which means the queues are often idle for relatively long periods. In addition, the routing algorithm most often operates to minimize the number of packet-switch hops and thus the number of queues. Thus, not only is the probability that an arriving NTP packet finds a busy queue in one direction reasonably low, but the probability of it finding a busy queue in both directions is even lower.

From the above discussion one would expect that, at low utilizations

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

These observations cast doubt on the median-filter approach as a good way to cast out offset outlyers and suggests another approach which might be called a minimum filter. From the scatter diagrams it is obvious that the best offset samples occur at the lower delays. Therefore, an appropriate technique would be simply to select from the n most recent samples the sample with lowest delay and use its associated offset as the estimate. An experiment was designed to test this technique using measurements between selected hosts equipped with radio clocks, so that delays and offsets could be determined independent of the measurement procedure itself.

The raw delays and offsets were measured by NTP from hosts at U Maryland (UMD) and U Delaware (UDEL) via net paths to each other and other hosts at Ford Research (FORD), Information Sciences Institute (ISI) and National Center for Atmospheric Research (NCAR). For the purposes here, all hosts can be assumed synchronized to within a few milliseconds to NBS time, so that the delays and offsets reflect only the net paths themselves.

The results of the measurements are given in Table D.1 (UMD) and Table D.2 (UDEL), which show for each of the paths the error X for which the distribution function P[x =< X] has the value shown. Note that the values of the distribution function are shown by intervals of decreasing size as the function increases, so that its behavior in the interesting regime of low error probability can be more accurately determined.

UMD Delay	FORD 1525	ISI 2174	NCAR 1423	UMD Offset	FORD 1525	ISI 2174	NCAR 1423
.1	493	688	176	.1	2	17	1
.2	494	748	179	.2	4	33	2
.3	495	815	187	.3	9	62	3
.4	495	931	205	.4	18	96	8
.5	497	1013	224	.5	183	127	13
.6	503	1098	243	.6	4.88E+8	151	20
.7	551	1259	265	.7	4.88E+8	195	26
.8	725	1658	293	.8	4.88E+8	347	35
.9	968	2523	335	.9	4.88E+8	775	53
.99	1409	6983	472	.99	4.88E+8	2785	114
.999	14800	11464	22731	.999	4.88E+8	5188	11279
1	18395	15892	25647	1	4.88E+8	6111	12733

Table D.1. Delay and Offset Measurements (UMD)

UDEL Delay	FORD 2986	UMD 3442	ISI 3215	NCAR 2756
.1	650	222	411	476
.2	666	231	436	512
.3	692	242	471	554
.4	736	256	529	594
.5	787	272	618	648
.6	873	298	681	710
.7	1013	355	735	815
.8	1216	532	845	1011
.9	1836	1455	1019	1992
.99	4690	3920	1562	4334
.999	15371	6132	2387	11234
1	21984	8942	4483	21427

Table D.2.a Delay Measurements (UDEL)

UDEL Offset	FORD 2986	UMD 3442	ISI 3215	NCAR 2756
.1	83	2	16	12
.2	96	5	27	24
.3	108	9	36	36
.4	133	13	48	51
.5	173	20	67	69
.6	254	30	93	93
.7	429	51	130	133
.8	1824	133	165	215
.9	4.88E+8	582	221	589
.99	4.88E+8	1757	539	1640
.999	4.88E+8	2945	929	5278
1	5.63E+8	4374	1263	10425

Table D.2.b Offset Measurements (UDEL)

The results suggest that accuracies less than a few seconds can usually be achieved for all but one percent of the measurements, but that accuracies degrade drastically when the remaining measurements are included. Note that in the case of the UMD measurements to FORD almost half the measurements showed gross errors, which was due to equipment failure at that site. These data were intentionally left in the sample set to see how well the algorithms dealt with the problem.

The next two tables compare the results of minimum filters (Table D.3) and median filters (Table D.4) for various n when presented with the UMD - - NCAR raw sample data. The results show consistently lower errors for the minimum filter when compared with the median filter of nearest value of n. Perhaps the most dramatic result of both filters is the greatly reduced error at the upper end of the range. In fact, using either filter with n at least three results in no errors greater than 100 milliseconds.

	Filter Samples				
	1	2	4	8	16
P[x=<X]	1423	1422	1422	1420	1416
.1	1	1	1	0	0
.2	2	1	1	1	1
.3	3	2	1	1	1
.4	8	2	2	1	1
.5	13	5	2	2	1
.6	20	10	3	2	2
.7	26	15	6	2	2
.8	35	23	11	4	2
.9	53	33	20	9	3
.99	114	62	43	28	23
.999	11279	82	57	37	23
1	12733	108	59	37	23

Table D.3. Minimum Filter
(UMD - NCAR)

	Filter Samples		
	3	7	15
P[x=<X]	1423	1423	1423
.1	2	2	2
.2	2	4	5
.3	5	8	8
.4	10	11	11
.5	13	14	14
.6	18	17	16
.7	23	21	19
.8	28	25	23
.9	36	30	27
.99	64	46	35
.999	82	53	44
1	82	60	44

Table D.4. Median Filter
(UMD - NCAR)

While the UMD - NCAR data above represented a path across the NSFNET Backbone, which normally involves only a few hops via Ethernets and 56-Kbps links, the UDEL - NCAR path involves additional ARPANET hops, which can contribute substantial additional delay dispersion. The following Table D.5. shows the results of a minimum filter for various n when presented with the UDEL - NCAR raw sample data. The range of error is markedly greater than the UMD - NCAR path above, especially near the upper end of the distribution function.

	Filter Samples				
	1	2	4	8	16
P[x=<X]	2756	2755	2755	2753	2749
.1	12	9	8	7	6
.2	24	19	16	14	14
.3	36	27	22	20	19
.4	51	36	29	25	23
.5	69	47	36	30	27
.6	93	61	44	35	32
.7	133	80	56	43	35
.8	215	112	75	53	43
.9	589	199	111	76	63
.99	1640	1002	604	729	315
.999	5278	1524	884	815	815
1	10425	5325	991	835	815

Table D.5. Minimum Filter (UDEL - NCAR)

Based on these data, the minimum filter was selected as the standard algorithm. Since its performance did not seem to much improve for values of n above eight, this value was chosen as the standard.
Network Time Protocol (Version 1): Specification and Implementation.

Appendix E. NTP Synchronization Networks

This section discusses net configuration issues for implementing a ubiquitous NTP service in the Internet system. Section E.1 describes the NTP primary service net now in operation, including an analysis of failure scenarios. Section E.2 suggests how secondary service nets, which obtain wholesale time from the primary service net, can be configured to deliver accurate and reliable retail time to the general host population.

E.1. Primary Service Network

The primary service net consists of five primary servers, each of which is synchronized via radio or satellite to a national time standard and thus operates at stratum one. Each server consists of an LSI-11 Fuzzball, a WWVB or GOES radio clock and one or more net interfaces. Some servers provide switching and gateway services as well. Table E.1 shows the name, Internet address, type of clock, operating institution and identifying code.

Name	Address	Clock	Operating Institution and (Code)
DCN5.ARPA	128.4.0.5	WWVB	U Delaware, Newark, DE (UDEL)
FORD1.ARPA	128.5.0.1	GOES	Ford Research, Dearborn, MI (FORD)
NCAR.NSF.NET	128.116.64.3	WWVB	National Center for Atmospheric Research, Boulder, CO (NCAR)
UMD1.UMD.EDU	128.8.10.1	WWVB	U Maryland, College Park, MD (UMD)
WWVB.ISI.EDU	128.9.2.129	WWVB	USC Information Sciences Institute, Marina del Rey, CA (ISI)

Table E.1. Primary Servers

Figure E.1 shows how the five primary servers are interconnected as NTP peers. Note that each server actively probes two other servers (along the direction of the arrows), which means these probes will continue even if one or both of the two probed servers are down. On the other hand, each server is probed by two other servers, so that the result, assuming all servers are up, is that every server peers with every other server.

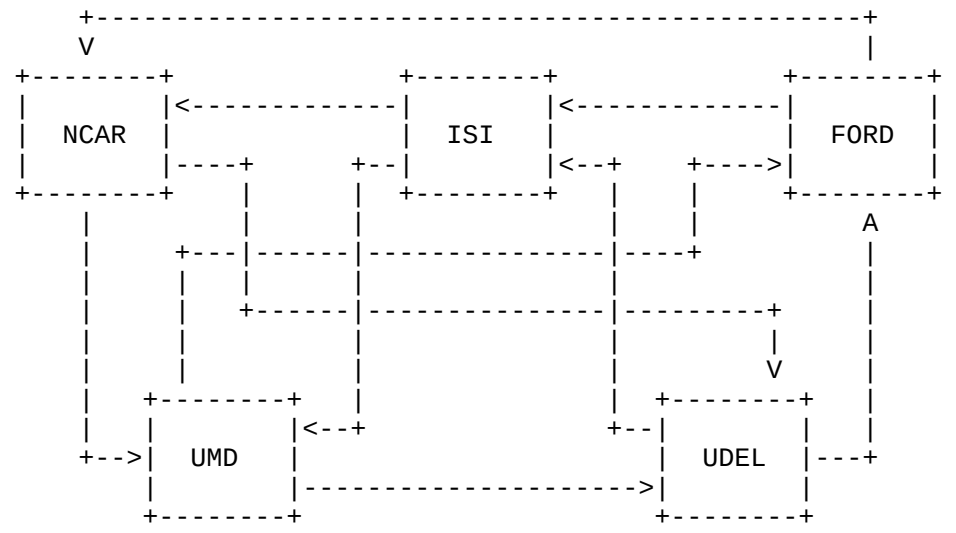


Figure E.1. Primary Service Network

All of the five primary servers shown are directly connected to a radio clock and thus normally operate at stratum one. However, if the radio clock itself becomes disabled or the propagation path to its synchronizing source fails, then the server drops to stratum two and synchronizes via NTP with its neighbor at the smallest synchronizing distance. If a radio clock appears to operate correctly but delivers incorrect time (falseticker), the server may remain synchronized to the clock. However, gross discrepancies will become apparent via the NTP peer paths, which will ordinarily result in an operator alarm.

Assume that, if a radio clock appears up, it is a truechimer; otherwise, the clock appears down. Then the above configuration will continue to provide correct time at all primary servers as long as at least one radio clock is up, all servers are up and the servers remain connected to each other through the net. The fact that the graph and all of its subgraphs are completely connected lends an incredible resilience to the configuration.

If some radio clocks appear up but are in fact falsetickers, the primary servers connected to those clocks will not provide correct time. However, as the consequents of the voting procedure and complete connectivity of the graph and its subgraphs, any combination of two falsetickers or of one falseticker and one down server will be detected by their truechimer neighbors.

E.2. Secondary Service Networks

A secondary server operating at stratum n > 1 ordinarily obtains synchronization using at least three peer paths, two with servers at stratum n-1 and one or more with servers at stratum n. In the most robust configurations a set of servers agree to provide backup service for each other, so distribute some of their peer paths over stratum-(n-1) servers and others over stratum-n servers in the same set. For instance, in the case of a stratum-2 service net with two secondary servers and the primary service net of Figure E.1, there are five possible configurations where each stratum-1 path ends on a different primary server. Such configurations can survive the loss of three out of the four stratum-1 servers or net paths and will reject a single falseticker on one of the two stratum-1 paths for each server.

Ordinary hosts can obtain retail time from primary or secondary service net using NTP in client/server mode, which does not require dedicated server resources as does symmetric mode. It is anticipated that ordinary hosts will be quite close to a secondary server, perhaps on the same cable or local net, so that the frequency of NTP request messages need only be high enough, perhaps one per hour or two, to trim the drift from the local clock.