Network Working Group
Technical Report 06-1-2
Category: Eclectic

David L. Mills
University of Delaware
January 2006

Network Time Protocol Version 4
Core Protocol Specification

Abstract

This document is a specification for the Network Time Protocol Version 4 (NTPv4), which is used to synchronize the time for Internet hosts, routers and ancillary devices to Coordinated Universal Time (UTC) as disseminated by national standards laboratories. It describes the core state machine, including the state variables, transition functions and processing steps. It also describes the fundamental on-wire protocol used to exchange time values between peers, servers and clients. It describes how to calculate the clock offset, roundtrip delay and various statistics used by the mitigation algorithms to calculate the maximum error and nominal error inherent in these measurements.

This document does not describe the mitigation algorithms themselves, including the filter, selection, clustering and combining algorithms used to develop the best estimates from the available server population, nor does it describe the clock discipline algorithm used to adjust the system clock time and frequency in response to measured time offsets. It does not describe the server discovery schemes, such as the DNS pool scheme or the Manycast scheme, nor does it discuss the Autokey public key authentication scheme. These are discussed in companion documents.

This specification is the basis of the reference implementation available at www.ntp.org. It includes all the procedures described in this document, as well as the mitigation algorithms, clock discipline algorithm and discovery schemes referenced herein.

Keywords: network time synchronization, computer time synchronization, time synchronization protocol

Sponsored by: Naval Surface Weapons Center (Dahlgren) Contract N00178-04-1-9001.

Table of Contents

List of Figures

List of Tables

1. Introduction

This report constitutes a formal specification of the Network Time Protocol Version 4 (NTPv4) core architecture, protocol and algorithms. NTP is used to synchronize the system clocks among a set of distributed time servers and clients. This document defines the core architecture, on-wire protocol, algorithms and data structures used by NTPv4 and is intended primarily for developers. This and related documents collectively updates and obsoletes the Network Time Protocol Version 3 Protocol (NTPv3) Specification RFC-1305 [6] and previous versions of the specification. The core protocol continues to be compatible with all prior versions except the original program (version 0). While certain minor changes have been made in some protocol header fields, these do not affect the interoperation between NTPv4 and previous versions.

The NTP subnet model includes a number of widely accessible primary time servers synchronized by wire or radio to national standards. The purpose of the NTP protocol is to convey timekeeping information from these primary servers to secondary time servers via the Internet. Crafted algorithms mitigate errors that may result from network disruptions, server failures and possible hostile action. Intermediate servers are configured as a forest where time values flow from the primary servers at the root via branching secondary servers toward clients at the leaves of the forest.

The NTPv4 reference implementation available at www.ntp.org complies with this and related specification documents. The new design overcomes significant shortcomings in the NTPv3 design, corrects certain bugs and incorporates new features. In particular, the reference implementation uses floating double data types throughout, except for the first-order timestamp differences required to calculate offset and delay. This results in time resolution better than one nanosecond and frequency resolution better than one nanosecond per second.

Additional improvements include a new clock discipline algorithm which is more responsive to system clock hardware frequency fluctuations and can hold accuracy within a few microseconds with precision time sources and within a few hundred microseconds with poll intervals equal to the maximum NTPv3 poll interval of 1024 seconds. With NTPv4 the poll interval can be extended to 36 hours with only modest loss of accuracy. The new algorithm can calibrate the intrinsic system clock frequency offset accurately within 15 minutes and increase the poll interval more rapidly.

The reference implementation continues to support the IPv4 address family and in addition supports the IPv6 address family and can operate with both families at the same time. It supports over 44 reference clocks, including all known devices in use today plus many older devices that might be found on eBay. It supports new server discovery schemes ant both symmetric key and public key authentication [4].

This document describes only the core structure of NTP; other documents describe additional protocol functions [1], mitigation algorithms [2] and the public key authentication protocol [4]. The NTP service model includes the modes of operation described in Section 2 and the data types described in Sections 5 and 6. The implementation model described in Section 4 is based on a multiple-process operating system architecture, although other architectures could be used

Association Mode	Send Mode	Receive Mode
symmetric active	1	1 or 2
symmetric passive	2	1
client	3	4
server	4	3
broadcast	5	N/A

Table 1: Association and Packet Modes

as well. The on-wire protocol described in Section 7 is based on a returnable-time design which depends only on measured clock offsets, but does not require reliable message delivery. The synchronization subnet is a self-organizing, hierarchical master-slave network with synchronization paths determined by a shortest-path spanning tree and defined metric. While multiple masters (primary servers) may exist, there is no requirement for an election protocol.

This remaining sections of this document define the NTP core protocol, including state variables, protocols and algorithms and is suitable for both a fully featured NTP as well as a subset usually known as Simple Network Time Protocol (SNTP) [5]. SNTP is intended for primary servers equipped with a reference clock, as well as clients with no dependent clients. The fully developed NTP core protocol intended for intermediate servers with multiple upstream servers and multiple downstream clients. Details specific to NTP packet formats used with the IPv4 and IPv6 and User Datagram Protocol (UDP) are presented in Appendix A.

2. NTP Modes of Operation

NTP-capable hosts are described as servers, clients and peers. Servers retain no state after returning the response to a client packet; clients and peers retain state in the form of a data structure called an association. Persistent associations are mobilized when the service starts and are never demobilized. Ephemeral associations are mobilized during operation, such as upon the arrival of a broadcast message, and demobilized by timeout or error. Preemptable associations are mobilized when or after the service starts and demobilized when deemed no longer useful for synchronization. For the purposes of the core specification, only persistent processes are assumed. The reference implementation includes suitable algorithms for ephemeral and preemptable associations, but they are not incorporated in this specification.

There are three NTP protocol variants, client/server, peer and broadcast. Each is associated with an association mode as shown in Table 1. In the client/server variant a client association sends mode 3 (client) packets to a server, which returns mode 4 (server) packets. Servers provide synchronization to one or more clients, but do not accept synchronization from them. A server can also be a reference clock which obtains time directly from a standard source such as a GPS receiver or telephone modem service. We say that clients *pull* synchronization from servers.

In the peer variant a peer can operate with either a symmetric active or symmetric passive association. A symmetric active association sends mode 1 (symmetric active) packets to a symmetric active peer association. Alternatively, a symmetric passive association can be mobilized upon arrival of a mode 1 packet. That association sends mode 2 (symmetric passive) packets and persists until error or timeout. We say that peers both *push* and *pull* synchronization to and from each other. For the purposes of this document, a peer operates like a client, so a reference to server implies peer as well.

In the broadcast variant a broadcast association sends mode 5 (broadcast) packets which are received by multiple broadcast client associations. Ordinarily, clients do not send packets to the servers; however, it is useful to provide an initial volley where the client can exchange a number of packets in order to calibrate the propagation delay and to run the Autokey security protocol, after which the client reverts to listen-only mode. We say that broadcast servers *push* synchronization to willing consumers.

Following conventions established by the telephone industry, the level of each server in the hierarchy is defined by a number called the stratum, with the primary servers assigned stratum one and the secondary servers at each level assigned one greater than the preceding level. As the stratum increases from one, the accuracies achievable degrades somewhat depending on the particular network paths and system clock stabilities. It is useful to assume that mean errors, and thus a metric called the synchronization distance, increase approximately in proportion to the stratum.

Drawing from the experience of the telephone industry, which learned such lessons at considerable cost], the subnet topology should be organized to produce the lowest synchronization distances, but must never be allowed to form a loop. In NTP the subnet topology is determined using a variant of the Bellman-Ford distributed routing algorithm, which computes the shortest distance spanning tree rooted on the primary servers. As a result of this design, the algorithm automatically reorganizes the subnet to produce the most accurate and reliable time, even when one or more primary or secondary servers or the network paths between them fail.

3. Definitions

A number of terms used throughout this document have a precise technical definition. A *timescale* is a frame of reference where time is expressed as the value of a monotonic-increasing binary counter with an indefinite number of bits. It counts in seconds and fraction with the decimal point somewhere in the middle. The Coordinated Universal Time (UTC) timescale represents mean solar time as disseminated by national standards laboratories. The local time is represented by the *system clock* maintained by the operating system kernel. The goal of the NTP algorithms is to minimize both the time difference and frequency difference between UTC and the system clock. When these differences have been reduced below nominal tolerances, the system clock is said to be synchronized to UTC.

The *date* of an event is the UTC time at which it takes place. Dates are ephemeral values which always increase in step with reality and are designated with upper case *T* in this document. It is convenient to define another timescale coincident with the running time of the program, or *daemon*, that provides the synchronization function. This is convenient in order to determine intervals for the various repetitive functions like poll events. Running time or *epoch* is usually designated with lower case *t*.

A *timestamp* $T(t)$ represents either the UTC date or time offset from UTC at epoch t. Which meaning is intended should be clear from context. Let $T(t)$ be the time offset, $R(t)$ the frequency offset, $D(t)$ the ageing rate (first derivative of $R(t)$ with respect to t). Then, if $T(t_0)$ is the UTC time offset determined at $t = t_0$, the UTC time offset after some interval t is

$$T(t + t_0) = T(t_0) + R(t_0)(t + t_0) + \frac{1}{2}D(t_0)(t + t_0)^2 + e, \qquad (1)$$

where e is a stochastic error term discussed later in this document. While the $D(t)$ term is important when characterizing precision oscillators, it is ordinary neglected for computer oscillators.

It is important in computer timekeeping applications to assess the performance of the timekeeping function. The NTP performance model includes four statistics which are updated each time a client makes a measurement with a server. The *offset* θ represents the maximum-likelihood time offset of the server clock relative to the system clock. The *delay* δ represents the roundtrip delay between the client and server. The *dispersion* ε represents the maximum error inherent in the measurement. It increases at a rate equal to the maximum disciplined system clock frequency tolerance Φ, typically 15 PPM. The *jitter* φ, defined as the root-mean-square (RMS) average of the most recent time offset differences, represents the nominal error in θ.

While the θ, δ, ε, and φ statistics represent measurements of the system clock relative to the server clock, the NTP protocol includes mechanisms to accumulate the δ and ε quantities, respectively Δ and E, relative to the primary reference clock. The system offset Θ and system jitter ϑ statistics represent weighted averages of possibly several servers. All of these statistics are available to the dependent applications in order to assess the performance of the synchronization function. The detailed formulations of these quantities are given later in this document.

4. Implementation Model

Figure 1 shows the implementation model for a typical client including an association dedicated to each server and containing data used to mitigate between multiple servers and discipline the system clock. A client sends an NTP packet to one or more servers and processes the replies as received. The server interchanges addresses and ports, overwrites certain fields in the message and returns the message immediately. Information included in the NTP message allows the client to adjust the system clock time and frequency and update the error statistics.

The figure shows two processes for each association, a *peer process* to receive messages from the server or reference clock driver and a *poll process* to transmit messages to the server or reference clock driver. As each NTP message is received, the offset θ between the peer clock and the system clock is computed along with the associated statistics δ, ε and φ.

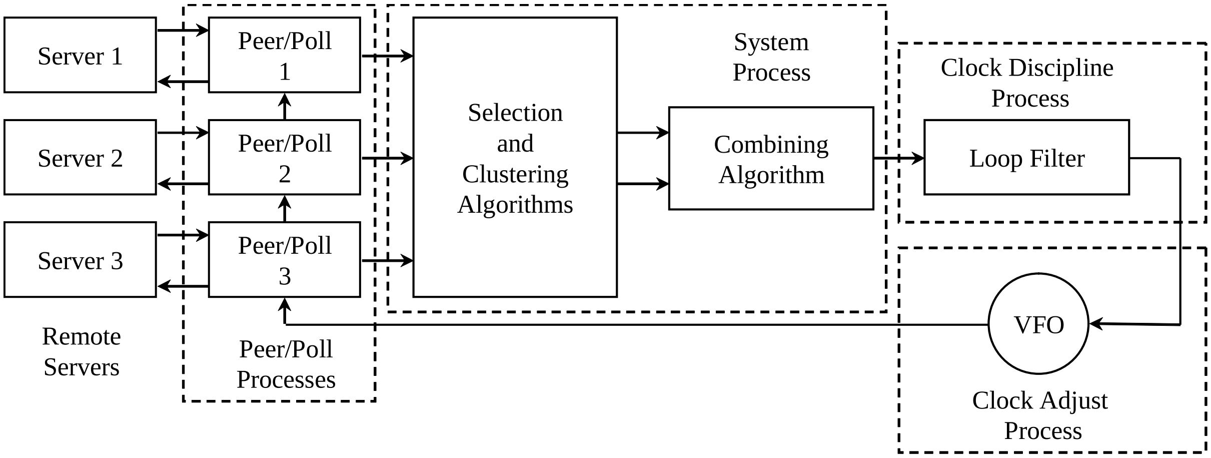


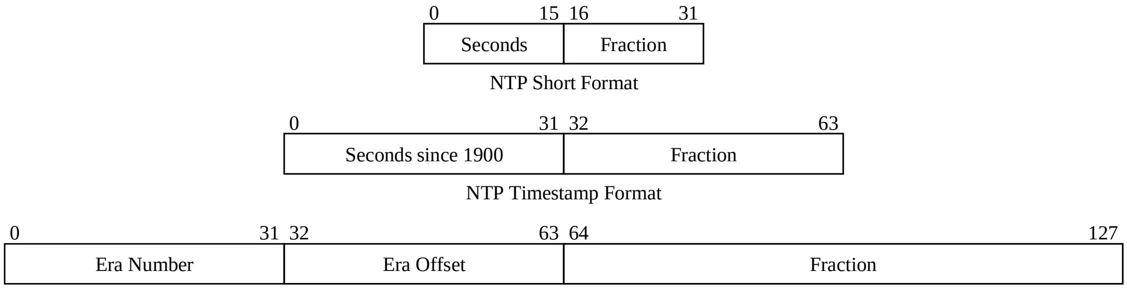
Figure 1. Implementation Model

The *system process* includes the selection and clustering algorithms which mitigate among the various servers and reference clock drivers to determine the most accurate and reliable *survivors* and cull the remainder on a statistical basis. The combining algorithm develops the final clock offset as a statistical average of the survivors. The specific design described in [2] is required for every NTP implementation operating as a client of upstream servers and simultaneously as a server for downstream clients. The reference implementation includes these algorithms, but their description is beyond the scope of this specification.

The *clock discipline process* includes engineered algorithms to control the time and frequency of the system clock, here represented as a variable frequency oscillator (VFO). Timestamps struck from this oscillator close the feedback loop which maintains the system clock time. Associated with the clock discipline process is the *clock adjust process*, which runs once each second. It is used to inject a computed time offset each second in order to maintain constant frequency. The RMS average of past time offset differences represents the nominal error or *system jitter* ϑ. The RMS average of past frequency offset differences represents the oscillator frequency stability or *frequency wander* Ψ.

It is important that the dynamic behavior of the clock discipline algorithms be carefully controlled in order to maintain stability in the NTP subnet at large. The algorithms together are described as an adaptive parameter, hybrid phase/frequency-lock feedback loop. The specific design described in [2]] is required for every NTP implementation operating as a client of upstream servers and simultaneously as a server for downstream clients. The reference implementation includes these algorithms, but their description is beyond the scope of this specification.

While not shown in the figure, the implementation model includes some means to set and adjust the system clock. The operating system is assumed to provide two functions, one to set the time directly, for example the Unix settimeofday() function, and another to adjust the time in small increments advancing or retarding the time by a designated amount, for example the Unix adjtime() function. In the intended design the clock discipline process uses the adjtime() function if the adjustment is less than a designated *step threshold*, and the settimeofday() function if



NTP Short Format

NTP Timestamp Format

NTP Date Format

Figure 2. NTP Time Formats

above the step threshold. The manner in which this is done and the value of the step threshold is not specified in this document.

5. Data Representation

All numeric values in packet headers are represented in twos complement format, with bits numbered in big-endian fashion from zero starting at the left, or high-order, position. There are three NTP formats used to represent time values, a 128-bit *date format*, a 64-bit *timestamp format* and a 32-bit *short format*, as shown in Figure 2. The 128-bit date format is used where sufficient storage and word size is available. It includes a 64-bit signed seconds field, which represents time until the Sun grows cold, and a 64-bit fraction field, which resolves the second within the time a photon takes to cross an atomic nucleus. Dates cannot be produced by NTP directly, nor is there need to do so. When necessary, dates can be derived from external means, such as the filesystem or dedicated hardware.

The 64-bit timestamp format is used in packet headers and other places with limited word size. It includes a 32-bit signed seconds field, which represents the number of seconds since the base date, and a 32 bit fraction field, which resolves the second within 232 picoseconds. The 32-bit short format is used in delay and dispersion header fields where the full resolution and range of the other formats are not necessary. It includes a 16-bit unsigned seconds field and a 16-bit fraction field. The only operations permitted with short format values are addition and multiplication by a constant.

As evident in the figure, the seconds field in the date format includes a 32-bit era field and a 32-bit seconds within the era field. The base date is 0628:16h 7 February 2036 UTC, when all 128 bits are zero. The base era includes signed dates from about 68 years before the base date, 1968, to about 68 years after the base date, 2104. In this era seconds in timestamp format are identical to seconds in date format with the era number extended from the high order (sign) bit of the timestamp. There are means available to disambiguate the era number when converting from timestamp to date format in other eras, but these are beyond the scope of this specification.

Date values are represented in twos complement arithmetic relative to the base date. Values greater than zero represent times after the base date; values less than zero represent times in before it. Dates are signed values and operations on them can produce a result in the same or different eras. Timestamps are signed values and the only operations permitted with them are

Name	Description
`r.`	receive packet header variable
`x.`	transmit packet header variable
`p.`	peer state variable
`s.`	system state variable

Table 2: Name Prefix Conventions

twos complement subtraction, and addition or subtraction by a constant, yielding a 63-bit signed result. In either format a value of zero is a special case representing unknown or unsynchronized time. In either format the clock *resolution* is expressed as the number of significant bits of the second fraction. In order to minimize bias and help make timestamps unpredictable to an intruder, the nonsignificant bits should be set to a random bit string.

It is not the intent of this specification to require any specific number representation in the implementation itself, just the representation in data exchanged between server and client. However, it is critical that operations on dates be in 128-bit integer arithmetic and timestamps be in 128-bit or 64-bit arithmetic and both preserve the full field width for internal storage. However, it is convenient to represent timestamp differences, which are ordinarily small compared to the timestamp magnitudes, and short format values in floating double arithmetic, which is convenient for the sometimes intricate mathematical operations specified in this document.

Some time values are represented in exponent format, including the precision, time constant and poll interval values. These are in 8-bit signed integer format in log2 (log to the base 2) seconds. The only operations permitted on them are increment and decrement. For the purpose of this specification and to simplify the presentation, a reference to one of these state variables by simple name means the exponentiated value, while reference explicate as exponent means the actual value.

6. State Variables

The NTP protocol state machines described in following sections are defined using state variables and transition functions. State variables are separated into classes according to their function in packet headers, peer and poll processes, and the system process. Packet variables represent the NTP header values in transmitted and received packets. Peer and poll variables represent the contents of the association for each server separately. System variables represent the state of the server as seen by its dependent clients. There are other variable classes assigned to the clock discipline and clock adjust processes, but not described in this specification.

6.1 Structure Conventions

In order to disambiguate between different variables of the same name but used in different processes, the following Unix-like structure member naming convention is adopted, as shown in Table 2. Note that named variables are rendered in fixed-width font, while equation variables are in italic sans-serif or Greek font. Each receive packet variable `v` is a member of the packet

Name	Value	Description
PORT	123	NTP port number
VERSION	4	NTP version number
TOLERANCE	15	frequency tolerance (PPM)
MINPOLL	4	minimum poll exponent (16 s)
MAXPOLL	17	maximum poll exponent (36 h)
MAXDISP	16	maximum dispersion (s)
MINDIST	.005	distance increment (s)
MAXDIST	1	distance threshold (s)
MAXSTRAT	16	maximum stratum number

Table 3: Global Parameters

structure `r` with fully qualified name `r.v`. In a similar manner `x.v` is a transmit packet variable and `s.v` is a system process variable. There is a set of peer and poll variables `p.v` for each association.

6.2 Global Parameters

In addition to the variable classes a number of global parameters are defined in this specification, including those shown with values in Table 3. While these are the only parameters needed in this specification, a much larger collection is necessary for the mitigation algorithms, clock discipline process and related implementation dependent functions. Some of these parameter values are cast in stone, like the NTP port number assigned by the IANA and the version number assigned this specification. Others like the frequency tolerance, involve an assumption about the worst case behavior of a host once synchronized and then allowed to drift when its sources have become unreachable. The minimum and maximum parameters define the limits of state variables as described in later sections. While shown with fixed values in this document, some implementations may make them variables adjustable by configuration commands.

6.3 Packet Header Variables

The most important state variables from an external point of view are the packet header variables described below. The NTP packet header follows the UDP and IP headers and the physical header specific to the underlying transport network. It consists of a number of 32-bit (4-octet) words, although some fields use multiple words and others are packed in smaller fields within a word. The NTP packet header shown in Appendix A has 12 words followed by optional extension fields and finally an optional message authentication code (MAC) consisting of the key identifier and message digest fields.

The optional extension fields described in Appendix A are used by the Autokey security protocol [4], which is not described here. The MAC is used by both Autokey and the symmetric key authentication scheme described in Appendix A. As is the convention in other Internet protocols, all fields are in network byte order, commonly called big-endian.

A list of the packet header variables is shown in Table 4 and described in detail below. The packet header fields apply to both transmitted (`x` prefix) and received packets (`r` prefix). The variables are interpreted as follows:

Name	Formula	Description
leap	*leap*	leap indicator (LI)
version	*version*	version number (VN)
mode	*mode*	mode
stratum	*stratum*	stratum
poll	τ	poll interval ($\log_2$ s)
precision	ρ	precision ($\log_2$ s)
rootdelay	Δ	root delay
rootdisp	E	root dispersion
refid	*refid*	reference ID
reftime	*reftime*	reference timestamp
org	T_1	origin timestamp
rec	T_2	receive timestamp
xmt	T_3	transmit timestamp
dst	T_4	destination timestamp
keyid	*keyid*	key ID
digest	*digest*	message digest

Table 4: Packet Header Variables

leap. 2-bit integer warning of an impending leap second to be inserted or deleted in the last minute of the current month, with bit 0 and bit 1, respectively, coded as follows:

00	no warning
01	last minute of the day has 61 seconds
10	last minute of the day has 59 seconds
11	alarm condition (the clock has never been synchronized)

version. 3-bit integer indicating the NTP version number, currently 4.

mode. 3-bit integer indicating the mode, with values defined as follows:

0	reserved
1	symmetric active
2	symmetric passive
3	client
4	server
5	broadcast
6	NTP control message
7	reserved for private use

stratum. 8-bit integer indicating the stratum level, with values defined as follows:

0	unspecified or invalid
1	primary server (e.g., equipped with a GPS receiver)
2-255	secondary server (via NTP)

It is customary to map the value 0 in received packets to MAXSTRAT in the peer variable stratum and to map stratum values of MAXSTRAT or greater to 0 in transmitted packets. This allows reference clocks, which normally appear at stratum 0, to be conveniently mitigated using the same algorithms used for external sources. By convention, stratum values equal to or greater than MAXSTRAT are considered unspecified or invalid.

poll. 8-bit signed integer indicating the maximum interval between successive messages, in log2 seconds. In the reference implementation the values can range from MINPOLL to and including MAXPOLL.

precision. 8-bit signed integer indicating the precision of the system clock, in log2 seconds. For instance a value of −18 corresponds to a precision of about one microsecond. The precision is normally determined when the service first starts up as the minimum of several iterations of the time to read the system clock.

rootdelay. 32-bit unsigned integer indicating the total roundtrip delay to the reference clock, in in NTP short format.

rootdisp. 32-bit unsigned integer indicating the total dispersion to the reference clock, in NTP short format.

refid. 32-bit code identifying the particular server or reference clock. The interpretation depends on the value in the stratum field. For stratum 0 (unspecified or invalid) this is a four-character ASCII string called the kiss code used for debugging and monitoring purposes. For stratum 1 (reference clock) this is a four-octet, left-justified, zero-padded ASCII string assigned to the radio clock. While not enumerated in the NTP specification, the following have been used as ASCII identifiers:

GOES	Geosynchronous Orbit Environment Satellite
GPS	Global Position System
PPS	Generic pulse-per-second
IRIG	Inter-Range Instrumentation Group
WWVB	LF Radio WWVB Ft. Collins, CO 60 kHz
DCF77	LF Radio DCF77 Mainflingen, DE 77.5 kHz
HBG	LF Radio HBG Prangins, HB 75 kHz
MSF	LF Radio MSF Rugby, UK 60 kHz
JJY	LF Radio JJY Fukushima, JP 40 kHz, Saga, JP 60 kHz
LORC	MF Radio LORAN C 100 kHz
TDF	MF Radio Allouis, FR 162 kHz
CHU	HF Radio CHU Ottawa, Ontario
WWV	HF Radio WWV Ft. Collins, CO
WWVH	HF Radio WWVH Kaui, HI
NIST	NIST telephone modem
USNO	USNO telephone modem
PTB, etc.	European telephone modem

Above stratum 1 (secondary servers and clients) this is the reference identifier of the server. If using the IPv4 address family, the identifier is the four-octet IPv4 address. If using the IPv6 address family, it is the first four octets of the MD5 hash of the IPv6 address.

`reftime`. 64-bit signed integer indicating the time when the system clock was last set or corrected, in NTP timestamp format.

`org`. 64-bit signed integer indicating the time at the client when the request departed for the server, in NTP timestamp format.

`rec`. 64-bit signed integer indicating the time at the server when the request arrived from the client, in NTP timestamp format.

`xmt`. 64-bit signed integer indicating the time at the server when the response left for the client, in NTP timestamp format.

`dst`. 64-bit signed integer indicating the time at the client when the reply arrived from the server, in NTP timestamp format. Note: This value is not included in a header field; it is determined upon arrival of the packet and made avaiable in the packet buffer data structure.

`keyid`. 32-bit unsigned integer used by the client and server to designate a secret 128-bit key. Together, the keyid and digest fields collectively are called message authentication code (MAC).

`digest`. 128-bit bitstring computed by the keyed MD5 message digest algorithm described in Appendix A.

7. On-Wire Protocol

The NTP on-wire protocol is the core mechanism to exchange time values between servers, peers and clients. It is inherently resistant to lost or duplicate data packets. Data integrity is provided by the IP and UDP checksums. No flow-control or retransmission facilities are provided or necessary. The protocol uses timestamps, either extracted from packet headers or struck from the system clock upon the arrival or departure of a packet. Timestamps are precision data and should be restruck in case of link level retransmission and corrected for the time to compute a MAC on transmit[1].

The on-wire protocol uses four timestamps numbered T_1 through T_4 and three state variables `org`, `rec` and `xmt`, as shown in Figure 3. This figure shows the most general case where each of two peers, *A* and *B*, independently measure the offset and delay relative to the other. For

[1] The reference implementation strikes a timestamp after running the MD5 algorithm and adds the difference between it and the transmit timestamp to the next transmit timestamp.

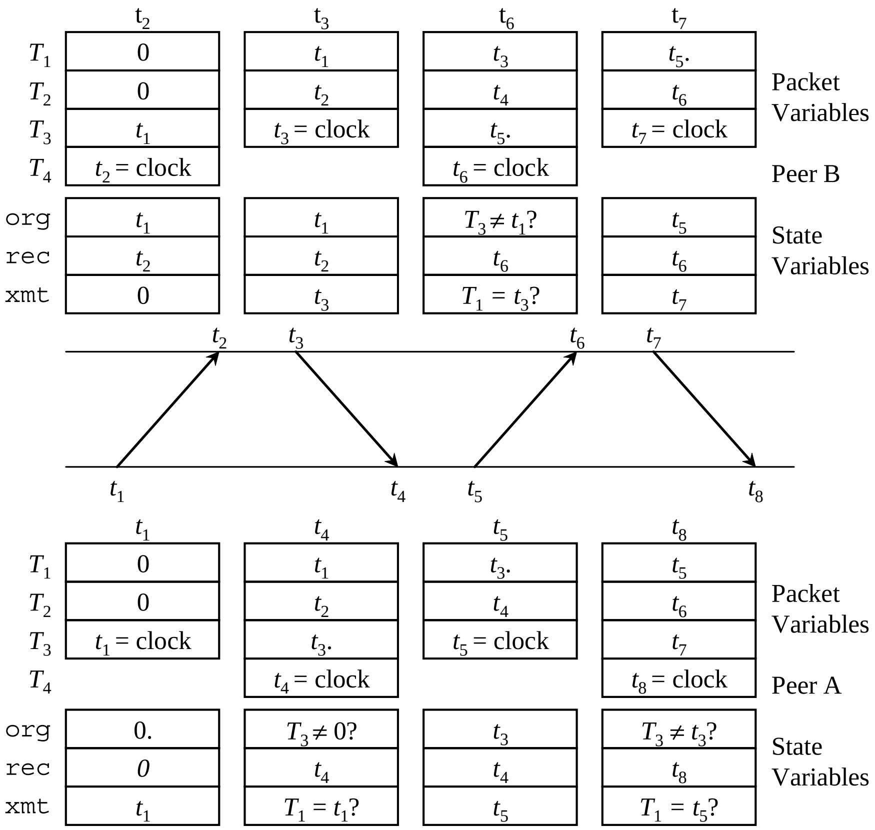


Figure 3. On-Wire Protocol

purposes of illustration the individual timestamp values are shown in lower case with subscripts indicating the order of transmission and reception.

In the figure the first packet transmitted by *A* containing only the transmit timestamp T_3 with value t_1. *B* receives the packet at t_2 and saves the origin timestamp T_1 with value t_1 in state variable org and the destination timestamp T_4 with value t_2 in state variable rec. At this time or some time later *B* sends a packet to *A* containing the org and rec state variables in T_1 and T_2, respectively and in addition the transmit timestamp T_3 with value t_3, which is saved in the xmt state variable. When this packet arrives at *A* the packet header variables T_1, T_2, T_3 and destination timestamp T_4 represent the four timestamps necessary to compute the offset and delay of *B* relative to *A*, as described later.

Before the *A* state variables are updated, two sanity checks are performed in order to protect against duplicate or bogus packets. A packet is a duplicate if the transmit timestamp T_3 in the packet matches the xmt state variable. A packet is bogus if the origin timestamp T_1 in the packet does not match the org state variable. In either of these cases the state variables are updated, but the packet is discarded.

The four most recent timestamps, T_1 through T_4, are used to compute the offset of *B* relative to *A*

$$\theta = T(B) - T(A) = \frac{1}{2}[(T_2 - T_1) + (T_3 - T_4)] \qquad (2)$$

and the roundtrip delay

$$\delta = T(ABA) = (T_4 - T_1) - (T_3 - T_2) . \qquad (3)$$

Note that the quantities within parentheses are computed from 64-bit signed timestamps and result in signed values with 63 significant bits plus sign. These values can represent dates from 68 years in the past to 68 years in the future. However, the offset and delay are computed as the sum and difference of these values, which contain 62 significant bits and two sign bits, so can represent unambiguous values from 34 years in the past to 34 years in the future. In other words, the time of the client must be set within 34 years of the server before the service is started. This is a fundamental limitation with 64-bit integer arithmetic.

In implementations where floating double arithmetic is available, the first-order differences can be converted to floating double and the second-order sums and differences computed in that arithmetic. Since the second-order terms are typically very small relative to the timestamps themselves, there is no loss in significance, yet the unambiguous range is increased from 34 years to 68 years. Additional considerations on these issues, as well as the behavior when moving beyond the prime era, are discussed in online white papers at www.ntp.org but beyond the scope of this specification.

In some scenarios where the frequency offset between the client and server is relatively large and the actual propagation time small, it is possible that the delay computation becomes negative. For instance, if the frequency difference is 100 PPM and the interval $T_4 - T_1$ is 64 s, the apparent delay is –6.4 ms. Since negative values are misleading in subsequent computations, the value of δ should be clamped not less than the system precision defined below.

The discussion above assumes the most general case where two symmetric peers independently measure the offsets and delays between each other. In the case of a stateless server, the protocol can be simplified. A stateless server copies T_3 and T_4 from the client packet to T_1 and T_2 of the server packet and tacks on the transmit timestamp T_3 before sending it to the client.

8. Peer Process

The peer process runs the on-wire protocol to determine the clock offset and roundtrip delay and in addition computes statistics variables used by the system and poll processes. Peer variables are instantiated in the association data structure when the structure is initialized and updated by arriving messages. There is peer process and association for each server.

Name	Formula	Description
Configuration Variables		
`srcaddr`		source address
`srcport`		source port
`dstaddr`		destination address
`dstport`		destination port
`keyid`		key ID
Packet Variables		
`leap`		leap indicator
`version`		version number
`pmode`		packet mode
`stratum`		stratum
`ppoll`		peer poll interval
`rootdelay`	Δ_R	root delay
`rootdisp`	E_R	root dispersion
`refid`		reference ID
`reftime`		reference timestamp
Timestamp Variables		
`t`	t	`epoch`
`org`		origin timestamp
`rec`		receive timestamp
`xmt`		transmit timestamp
Statistics Variables		
`offset`	θ	clock offset
`delay`	δ	roundtrip delay
`disp`	ε	dispersion
`jitter`	φ	jitter

Table 5: Peer Process State Variables

8.1 Peer Process State Variables,

Table 5 summarizes the common names, formula names and a short description of each peer variable, all of which have prefix `p`. The following configuration variables are normally initialized when the association is mobilized, either from a configuration file or upon arrival of the first packet for an ephemeral association.

`p.srcadr`. IP address of the remote server or reference clock. Reference clock addresses are by convention in IPv4 format with prefix 127.127.`t`.`u`, where `t` is the device driver number and `u` the instantiation number. This becomes the destination IP address in packets sent from this association.

`p.srcport`. UDP port number of the server or reference clock. This becomes the destination port number in packets sent from this association. When operating in symmetric modes (1 and 2) this field must contain the NTP port number `PORT` assigned by the IANA. In other modes it can contain any number consistent with local policy.

`p.dstadr`. IP address of the client. This becomes the source IP address in packets sent from this association.

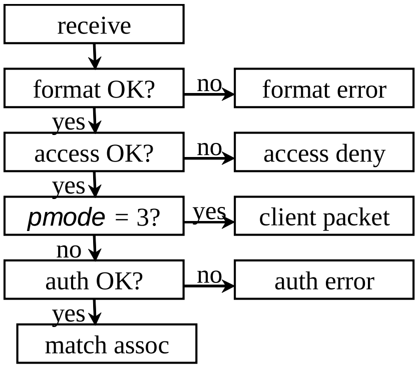


Figure 4. Receive Processing

`p.dstport`. UDP port number of the client, ordinarily the NTP port number `PORT` assigned by the IANA. This becomes the source port number in packets sent from this association.

`p.keyid`. Symmetric key ID This identifies the 128-bit MD5 key used to generate and verify the MAC. The client and server or peer can use different values, but they must map to the same key.

The variables defined below are updatesd from the packet header as each packet is received. They are interpreted in the same way as the as the packet variables of the same names.

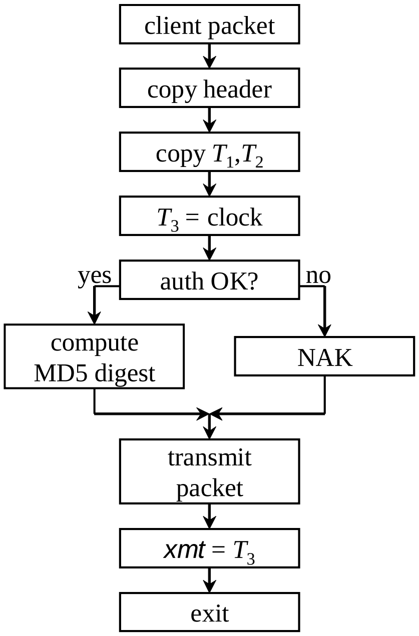
`p.leap`, `p.version`, `p.pmode`, `p.stratum`, `p.ppoll`, `p.rootdelay`, `p.rootdisp`, `p.refid`, `p.reftime`

It is convenient for later processing to convert the NTP short format values for `p.rootdelay` and `p.rootdisp` to floating double in the association itself.

The `p.org`, `p.rec`, `p.xmt` variables represent the timestamps computed by the on-wire protocol described previously. The `p.offset`, `p.delay`, `p.disp`, `p.jitter` variables represent the current time values and statistics produced by the peer process. The offset and delay are computed by the on-wire protocol; the dispersion and jitter are calculated as described below. Strictly speaking, the epoch `p.t` is not a timestamp. It records the system timer when these values are computed and is used to insure strict monotonic ordering for the clock discipline.

8.2 Peer Process Operations

Figure 4 shows the peer process code flow upon the arrival of a packet. There is no specific method required for access control, although it is recommended that implementations include a match-and-mask scheme similar to many others now in widespread use, as well as in the reference implementation. Format checks require correct field length and alignment, acceptable version number (1-4) and correct extension field syntax.



Packet Variable		Variable
x.leap	←	s.leap
x.version	←	r.version
x.mode	←	4
x.stratum	←	s.stratum
x.poll	←	r.poll
x.precision	←	s.precision
x.rootdelay	←	s.rootdelay
x.rootdisp	←	s.rootdisp
x.refid	←	s.refid
x.reftime	←	s.reftime
x.org	←	r.xmt
x.rec	←	r.dst
x.xmt	←	clock
x.keyid	←	r.keyid
x.digest	←	md5 digest

Figure 5. Client Packet Processing

There is no specific requirement for authentication; however, if authentication is implemented, the symmetric key scheme described in Appendix A must be supported. For the most vulnerable applications the Autokey public key scheme described in [4] is recommended. If this is a client (mode 3) packet, the server constructs a server (mode 4) packet and returns it to the client without retaining state. The server packet is constructed as shown in Figure 5. If the p.rootdelay and p.rootdisp system variables are stored in floating double, they must be converted to NTP short format first. Note that, if authentication fails, the server returns a special message called a crypto-NAK. This message includes the normal NTP header data shown in the figure, but with a MAC consisting of four octets of zeros. The client is free to accept or reject the message in this case.

If any other packet mode, symmetric active (1), symmetric passive (2), server (4) or broadcast (5) the association table is searched for matching source address and source port. If the packet matches an association, the code flows as shown in Figure 6. If the packet matches no association and the mode is symmetric active or broadcast, the expected behavior is to mobilize an ephemeral association and proceed as in the figure. The means for doing this are beyond the scope of this specification.

The packet timestamps are carefully checked to avoid invalid, duplicate or bogus packets, as shown in the figure. Note that a crypto-NAK is considered valid only if it survives these tests. Next, the peer variables are copied from the packet header variables as shown in Figure 7. The reference implementation includes a number of data range checks and discards the packet if the ranges are exceeded; however, the header fields are copied even if this occurs, since they are necessary in symmetric modes to construct the subsequent poll message.

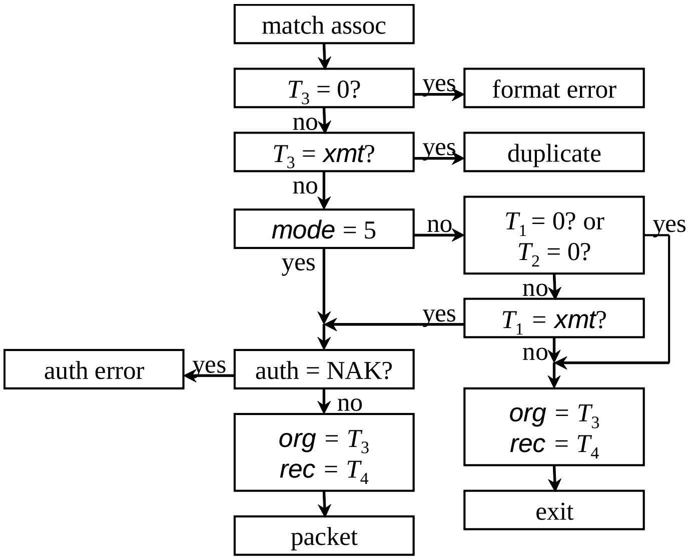
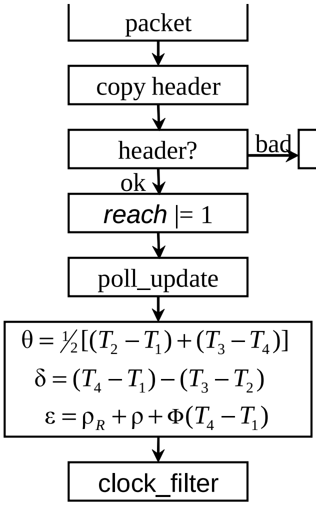


Figure 6. Timestamp Processing


packet
copy header
header?
bad
header error
ok
reach |= 1
poll_update
$\theta = \frac{1}{2}[(T_2 - T_1) + (T_3 - T_4)]$
$\delta = (T_4 - T_1) - (T_3 - T_2)$
$\varepsilon = \rho_R + \rho + \Phi(T_4 - T_1)$
clock_filter

Peer Variables		Packet Variables
`p.leap`	←	`r.leap`
`p.mode`	←	`r.mode`
`p.stratum`	←	`r.stratum`
`p.ppoll`	←	`r.ppoll`
`p.rootdelay`	←	`r.rootdelay`
`p.rootdisp`	←	`r.rootdisp`
`p.refid`	←	`r.refid`
`p.reftime`	←	`r.reftime`

Figure 7. Packet Processing

The 8-bit `p.reach` shift register in the poll process is used to determine whether the server is reachable or not and provide information useful to insure data are fresh and have not become stale. The reach register is shifted left by one bit when a packet is sent. In this process the rightmost bit is set to zero. As valid packets arrive, the rightmost bit is set to one. If the register contains any nonzero bits, the server is considered reachable; otherwise, it is unreachable. Since the peer poll interval might have changed since the last packet, the poll_update routine in the poll process is called to redetermine the host poll interval.

The on-wire protocol calculates the clock offset θ and roundtrip delay δ from the four timestamps in the packet. While it is in principle possible to do all calculations except the first-order timestamp differences in fixed-point arithmetic, it is much easier to do this in floating double arithmetic and this will be assumed in the following description. The dispersion statistic ε(t) represents the maximum error due to the oscillator tolerance and time since the last measurement. It is initialized

$$\varepsilon(t_0) = \rho_R + \rho + \Phi(T_4 - T_1) \qquad (4)$$

when the measurement is made at t_0. Here ρ_R is the peer precision in the packet header `r.precision` and ρ the system precision `s.precision`, both expressed in seconds. These terms are necessary to account for the uncertainty in reading the system clock in both the server and the client. The dispersion then grows at constant rate `TOLERANCE` (Φ) s/s; in other words, at time *t*, $\varepsilon(t) = \varepsilon(t_0) + \Phi(t - t_0)$. With the default value Φ = 15 PPM, this amounts to about 1.3 s per day. With this understanding, the argument *t* will be dropped and the dispersion represented simply as ε.

The remaining statistics are computed by the clock filter algorithm as described in the next section.

8.3 Clock Filter Algorithm

The clock filter algorithm grooms the stream of on-wire data to select the samples most likely to represent the correct time. It is not the purpose of this document to specify the particular algorithms used for this and the other mitigation and discipline functions, but for interoperability purposes it is necessary to specify the statistics available to these algorithms and the manner in which they are developed. There is considerable latitude in the details of the fully implemented algorithm; only the core algorithm is prescribed here.

There are two statistics associated with the on-wire protocol, dispersion ε and jitter φ. These are produced by the clock filter algorithm and used by the mitigation algorithms to determine the best and final offset used to discipline the system clock. They are also used to determine the server health and whether it is suitable for synchronization. The core processing steps of this algorithm are shown in Figure 8.

The clock filter algorithm saves the most recent sample tuples (θ, δ, ε, *t*) in an 8-stage shift register in the order that packets arrive. Here *t* is the system timer at the sample time, not the peer variable of the same name. The following scheme is used to insure sufficient samples are in the register and that old stale data are discarded. Initially, the tuples of all stages are set to the dummy tuple (0, `MAXDISP`, `MAXDISP, 0)`. As valid packets arrive, the (θ, δ, ε, *t*) tuples are shifted into the register causing old samples to be discarded, so eventually only valid samples remain. If the two low order bits of the reach register are zero, indicating two poll intervals have expired with no valid packets received, the poll process calls the clock filter algorithm with the dummy tuple just as if the tuple had arrived from the network. If this persists for eight poll intervals, the register returns to the initial condition.

In the next step, the shift register stages are copied to a temporary list and the list sorted by increasing δ. Let *j* index the stages starting with the lowest δ. If the sample epoch t_0 is not later

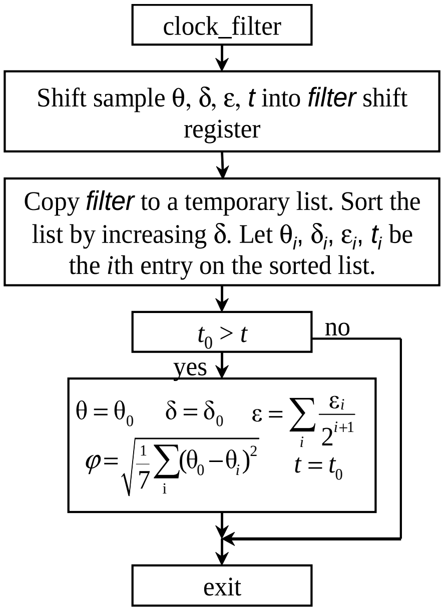

clock_filter
Shift sample θ, δ, ε, *t* into *filter* shift register
Copy *filter* to a temporary list. Sort the list by increasing δ. Let θ_i, δ_i, ε_i, t_i be the *i*th entry on the sorted list.
$t_0 > t$
no
yes
$\theta = \theta_0$ $\delta = \delta_0$ $\varepsilon = \sum_i \frac{\varepsilon_i}{2^{i+1}}$
$\varphi = \sqrt{\frac{1}{7}\sum_i (\theta_0 - \theta_i)^2}$ $t = t_0$
exit

Figure 8. Clock Filter Algorithm

than the last valid sample epoch `p.t`, exit without affecting the current peer variables. Otherwise, let ε_j be the dispersion of the *j*th entry, then

$$\varepsilon = \sum_{j=0}^{7} \frac{\varepsilon_j}{2^{j+1}} \tag{5}$$

is the peer dispersion `p.disp`. Note the overload of ε, whether input to the clock filter or output, the meaning should be clear from context.

The observer should note (a) if all stages contain the dummy tuple with dispersion `MAXDISP` (16 s), the computed dispersion is a little less than 16 s, (b) each time a valid tuple is shifted into the register, the dispersion drops by a little less than half, depending on the valid tuples dispersion, (c) after the fourth shift the dispersion is usually a little less than 1 s, which is the assumed value of the `MAXDIST` parameter. That parameter can be changed to yield more or fewer samples are required in order to discipline the clock.

Let the first stage offset in the sorted list be θ_0; then, for the other stages in any order, the jitter is the RMS average

$$\varphi = \sqrt{\frac{1}{n-1}\sum_{j=1}^{n-1} (\theta_0 - \theta_j)^2}, \tag{6}$$

where *n* is the number of valid tuples in the register. In order to insure consistency and avoid divide exceptions in other computations, the φ is bounded from below by the system precision ρ expressed in seconds. While not in general considered a major factor in ranking server quality, jitter is a valuable indicator of fundamental timekeeping performance and network congestion state.

Of particular importance to the mitigation algorithms is the peer synchronization distance, which is computed from the root delay and root dispersion. The root delay is

$$\delta' = \Delta_R + \delta \tag{7}$$

and the root dispersion is

$$\varepsilon' = E_R + \varepsilon + \varphi . \tag{8}$$

Note that ε and therefore ε' increase at rate Φ. These quantities and the peer synchronization distance

$$\lambda = \frac{\delta'}{2} + \varepsilon' \tag{9}$$

are recalculated as necessary. The peer distance is used by the mitigation algorithms as a metric to evaluate the quality of time available from each server.

9. System Process

The system process implements the mitigation algorithms, including the selection, clustering and combining algorithms. They sort the truechimers from the falsetickers, toss off outliers and combine the survivors to produce the single most cherished offset for the clock discipline process. As the mitigation algorithms are discussed in another report [2], for the purposes of this specification the only function of the system process is to select one of the associations managed by the peer processes as the system peer, and update the system variables from the peer variables. Later, the system variables are used by the poll processes to construct and send packets to the servers and clients.

9.1 System Process State Variables

The system variables associated with the system process are summarized in Table 6, which gives the variable name, formula name and short description. They define the state of the host operating as a server to dependent clients. The `leap`, `stratum`, `reftid` and `reftime` have the same format and interpretation as the peer variables of the same name. The remaining variables with prefix `s` are defined below.

`s.timer`. 32-bit unsigned integer system timer counting the seconds since the service was started.

Name	Formula	Description
`leap`	*leap*	leap indicator
`stratum`	*stratum*	stratum
`precision`	ρ	precision
`offset`	Θ	combined offset
`rootdelay`	Δ	root delay
`rootdisp`	E	root dispersion
`jitter`	ϑ	combined jitter
`refid`	*refid*	reference ID
`reftime`	*reftime*	reference time

Table 6: System Process State Variables

`s.t`. 32-bit unsigned integer value of the system timer at the last update.

`s.tc`. 8-bit signed integer representing the time constant determined by the clock discipline algorithm, in log2 seconds.

`s.precision`. 8-bit signed integer representing the system precision in log2 seconds.

`s.roodelay`. 32-bit NTP short format or floating double representing the roundtrip delayb beteen the client and primary server reference clock.

`s.rootdisp`. 32-bit NTP short format or floating double representing the root dispersion accumulated on the path to the primary server reference clock.

`s.offset`. 64-bit floating double clock offset as computed by the combining algorithm.

`s.jitter`. 64-bit floating double jitter as computed by the combining algorithm.

Initially, all variables are cleared to zero, then the `s.leap` is set to 11 (unsynchronized) and `s.stratum` is set to `MAXSTRAT`. The `s.precision` is determined as log2 of the minimum time in several iterations to read the system clock. The remaining statistics are determined as described below.

9.2 System Process Operations

Each time the clock filter algorithm runs, the selection algorithm in the system process scans all associations and chooses the best or the best combination as the survivors. While the design of these algorithms is not detailed here, it is necessary to establish the criteria for acceptance. Figure 9 summarizes these criteria. Of the survivors one is selected as the system peer, here represented by $(\theta, \delta, \varepsilon, \varphi)$. Following past practice, an update is discarded if its time of arrival `p.update` is not strictly later than the last update used `s.update`. If so, `s.update`, `s.leap`, `s.refid` and `s.reftime` are updated from the corresponding peer variables. Then, `s.stratum` is set to `p.stratum` plus one. The remaining variables are updated as shown in Figure 10. As before, it is desirable to compute these values using floating double arithmetic.

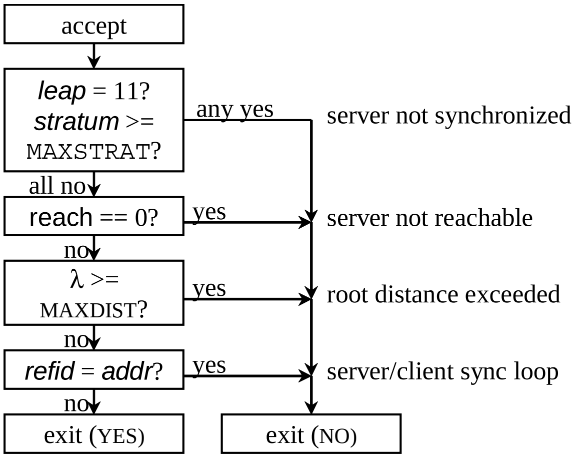


Figure 9. Accept Routine

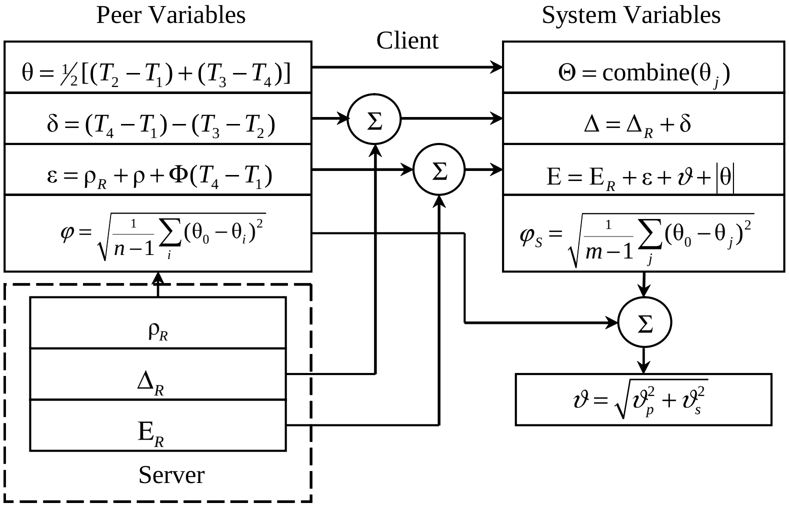

Peer Variables
Client
System Variables
$\theta = \frac{1}{2}[(T_2 - T_1) + (T_3 - T_4)]$
$\Theta = \text{combine}(\theta_j)$
$\delta = (T_4 - T_1) - (T_3 - T_2)$
Σ
$\Delta = \Delta_R + \delta$
$\varepsilon = \rho_R + \rho + \Phi(T_4 - T_1)$
Σ
$E = E_R + \varepsilon + \vartheta + |\theta|$
$\varphi = \sqrt{\frac{1}{n-1}\sum_i (\theta_0 - \theta_i)^2}$
$\varphi_S = \sqrt{\frac{1}{m-1}\sum_j (\theta_0 - \theta_j)^2}$
Σ
ρ_R
Δ_R
E_R
Server
$\vartheta = \sqrt{\vartheta_p^2 + \vartheta_s^2}$

Figure 10. System Variables Processing

The combining algorithm produces the weighted clock offset Θ, which is passed to the clock discipline, and the weighted peer jitter ϑ_p, which is a component of the system jitter. If there is only one survivor, the offset passed to the clock discipline algorithm is $\Theta = \theta$ and the system jitter is $\vartheta = \varphi$. Otherwise, the selection jitter ϑ_S is computed as in (6), where θ_0 represents the offset of the system peer and *j* ranges over the survivors. The system jitter

$$\vartheta = \sqrt{\vartheta_p^2 + \vartheta_s^2}. \qquad (10)$$

is passed to dependent applications programs as the nominal system clock error. The root delay Δ and root dispersion Ε statistics are relative to the primary server reference clock and thus inherited from each server along the path.

The system process includes a timer interrupt facility which drives the clock adjust process and also the peer timers in each association. When suitably configured, the operating system

Name	Formula	Description
hpoll		host poll exponent
next		next poll time
reach		reach register
unreach		unreach counter

Table 7: Poll Process State Variables

interrupts the daemon at one-second intervals. The system timer begins at zero when the service starts and increments at each interrupt. At each interrupt the clock adjust process is called to incorporate the clock discipline time and frequency adjustments, then the associations are scanned to determine if the system timer equals or exceeds the `p.next` state variable defined in the next section. If so, the poll process is called to send a packet and compute the next `p.next` value.

10. Poll Process

Each association supports a poll process that runs at regular intervals to construct and send packets in symmetric, client and broadcast associations. It runs continuously, whether or not servers are reachable.

10.1 Poll Process State Variables

The poll process variables are allocated in the association data structure along with the peer process variables. Table 7 shows the variable names, formula names and short definition. Following is a detailed description of these variables, all of which carry the `p` prefix.

`p.hpoll`. 8-bit signed integer representing the poll exponent, in log2 seconds. The actual poll interval is 1 shifted left by the poll exponent. When the server is reachable, the value follows the clock discipline time constant `s.tc`. When not reachable the value is slowly increased to reduce the network load.

`p.next`. 32-bit unsigned integer. When the poll_update routine is called, this is set as the system timer value for the next packet sent.

`p.reach`. 8-bit integer used as a shift register. Each time a packet is sent, the register is shifted left one bit, with zero entering from the right and overflow bits discarded. If the register is nonzero, the server is reachable; otherwise, the server is unreachable.

`p.unreach`. 32-bit unsigned integer If the server is reachable, the value is set to zero; if not, it increases at each poll interval.

10.2 Poll Process Operations

The poll_update routine shown once on Figure 11 is called when a valid packet is received and immediately after a poll message is sent. First, the routine recomputes the `p.hpoll` variable, as

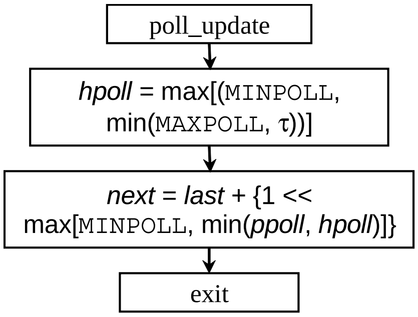

poll_update
hpoll = max[(MINPOLL, min(MAXPOLL, τ))]
next = last + {1 << max[MINPOLL, min(ppoll, hpoll)]}
exit

Figure 11. Poll Update Routine

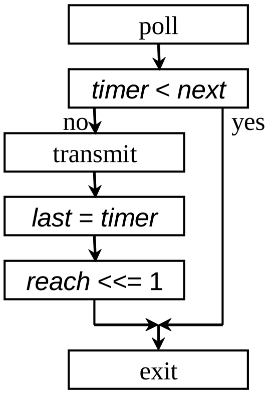


Figure 12. Poll Routine

the clock discipline time constant `s.tc` might have changed. Then it computes the value of the poll timer when the next poll message should be sent. The various clamps are set so that the interval will be the minimum of the host poll interval and peer poll interval, but not less than `MINPOLL`.

Once each second the poll routine shown ni Figure x is called. If the time to the next poll messages is computed and compared to the system timer. If the value is greater than the system timer, the routine exits and returns to the caller. Otherwise, it calls the poll routine to send a packet and sets `p.last` to the value of the system timer. Next, the `p.reach` is variable shifted left by one bit, with zero replacing the rightmost bit.

The poll interval management is particularly relevant in the case of symmetric modes when the clock discipline time constants may be different for each peer. Each peer sends its current host poll exponent in the `x.poll` packet header variable. Then, each peer sets the actual host poll exponent to the minimum of `p.hpoll` and the peer poll exponent `p.ppoll` saved from the last received packet. The time to the next poll is determined from the minimum value. Thus the clock discipline can be oversampled, but not undersampled.

Not shown on the figure is the mechanism to back off the poll interval if the server becomes unreachable. If `p.reach` is nonzero, the server is reachable and `p.unreach` is set to zero; otherwise, `p.unreach` is incremented by one for each poll. The `p.unreach` increases up to a maximum set by the `UNREACH` parameter. Once reaching the maximum, `p.hpoll` is increased by one, which doubles the poll interval. The `p.hpoll` increases up to a maximum set by the

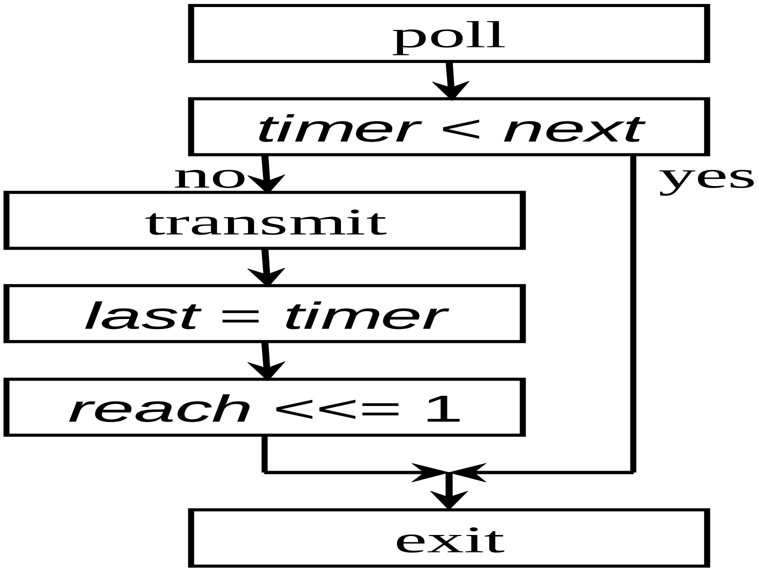


Figure 13. Poll Routine

MAXPOLL parameter. At the first message from the server, p.hpoll is reset to s.poll and operation resumes normally.

When a packet is sent from an association, some header values are copied from the peer variables left by a previous packet and others from the system variables. Figure 13 includes a flow diagram and a table showing which values are copied to each header field. In those implementations using floating double data types for root delay and root dispersion, these must be converted to NTP short format. All other fields are either copied intact from peer and system variables or struck as a timestamp from the system clock.

11. Poll Rate Control and the Kiss-o'-Death

In order to protect the network against unreasonable resource demands, it is necessary to carefully manage the poll interval. Polls should be sent at the longest interval consistent with the required accuracy. As a general rule, the minimum headway between successive client packets must never be less than two seconds and the mean headway never less than 16 seconds. Implementations should include a feature similar to that described in Section 10 that backs off the poll interval after some time when the server has not been heard.

As with any ubiquitous public service, NTP time servers and related networks are vulnerable to overload. NTP servers have a very low resource profile; some servers today service several thousand packets per second and don't even get warm. However, rates like that can consume significant network resources. In practice, the usual scenario has been, in a cast of thousands, only a handful of offenders send at rates in the order of once per a second or more.

In order to deflect such attacks a mechanism is necessary for servers to detect which clients are offending and how to respond to them. The reference implementation maintains a least-recently-used (LRU) cache ordered by time since the last packet received. Each entry in the cache includes the client IP address, headway since the last packet from that address and average headway of past packets. If the headway is less than 2 s or the average headway is less than 15 s, a packet called the *kiss-o'-death* (KoD) is returned to the client. A KoD packet is recognized as a leap value of 11 and stratum 0. The reference ID field in KoD packets contains a four-character string called the *kiss code*. Upon receiving a KoD packet with designated kiss code, the client is expected to cease operation and send a message to the system log and/or operator.

Not all kiss codes are fatal; some provide useful diagnostic information, such as various cryptographic misconfiguration. To resist a clogging attack, KoD messages are throttled at no more than one per second. As experience with the KoD scheme is ongoing and there is merit in exploring the scheme in a wider scope of applications, compliance with the KoD is highly recommended, but not required in this specification. While somewhat naive in concept, effective compliance with this behavior might involve router and operating system intervention and beyond the control of a casual hacker. In particular, routers can intercept a KoD and blacklist the sender. This may be a topic for future standardization.

12. References

1. Burbank, J. (Ed.), J. Martin (Ed.) and D. Mills (Ed.). The Network Time Protocol Version 4 protocol specification, Network Working Group Request for Comments RFC-XXXX), Johns-Hopkins University, TBA.

2. Kasch, W. (Ed.), J. Burbank (Ed.) and D. Mills (Ed.). The Network Time Protocol Version 4 algorithm specification. Network Working Group Request for Comments RFC-XXX, Johns Hopkins University, TBA.

3. Mills, D.L. *Computer Network Time Synchronization - the Network Time Protocol.* CRC Press, March 2006, 290 pp.

4. Mills, D.L. The Autokey security architecture, protocol and algorithms. Electrical and Computer Engineering Technical Report 06-1-1, University of Delaware, January 2006, 59 pp.

5. Mills, D., D. Plonka and J. Montgomery. Simple network time protocol (SNTP) version 4 for IPv4, IPv6 and OSI. Network Working Group Report RFC-4330, University of Delaware, December 2005, 27 pp.

6. Mills, D.L., Network Time Protocol (Version 3) specification, implementation and analysis. Network Working Group Request for Comments RFC-1305, University of Delaware, March 1992.

7. Plonka, D. Requirements for Network Time Protocol Version 4. Network Working Group Request for Comments RFC-XXXX, University of Wisconsin, TBA.

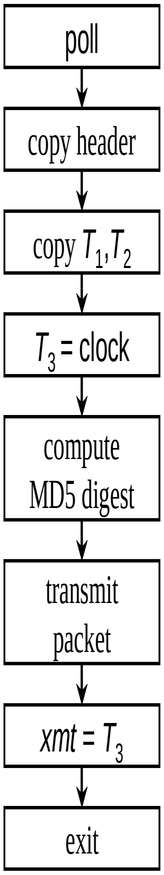
8. Rivest, R. The MD5 message-digest algorithm. Network Working Group Request for Comments RFC-1321. MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992, 21 pp.

Appendix A. NTPv4 Packet Formats

The NTP packet consists of a number of 32-bit (4 octet) words in network byte order. The packet consists of three components, the header itself, one or more optional extension fields and an optional message authentication code (MAC). The header component is identical to the NTPv3 header and previous versions. The optional extension fields are used by the Autokey public key cryptographic algorithms described in [4]. The optional MAC is used by both Autokey and the symmetric key cryptographic algorithms described in the main body of this report.

A.1 NTP Header Field Formats

The NTP header format is shown in Figure 14, where the size of some multiple-word fields is shown in bits if not the default 32 bits. The header extends from the beginning of the packet to the end of the Transmit Timestamp field. The format and interpretation of the header fields is in the main body of this report. When using the IPv4 address family these fields are backwards compatible with NTPv3. When using the IPv6 address family on an NTPv4 server with a NTPv3 client, the Reference Identifier field is a random value and a timing loop might not be detected. The incidence of this, which would be considered a birthday event, will be very rare.



Packet Variable		Variable
x.leap	←	s.leap
x.version	←	p.version
x.mode	←	p.hmode
x.stratum	←	s.stratum
x.poll	←	p.hpoll
x.precision	←	s.prec
x.rootdelay	←	s.rootdelay
x.rootdisp	←	s.rootdisp
x.refid	←	s.refid
x.reftime	←	s.reftime
x.*org*	←	p.org
x.*rec*	←	p.rec
x.*xmt*	←	clock
x.keyid	←	p.keyid
x.digest	←	md5()

Figure 14. NTPv4 Header Format

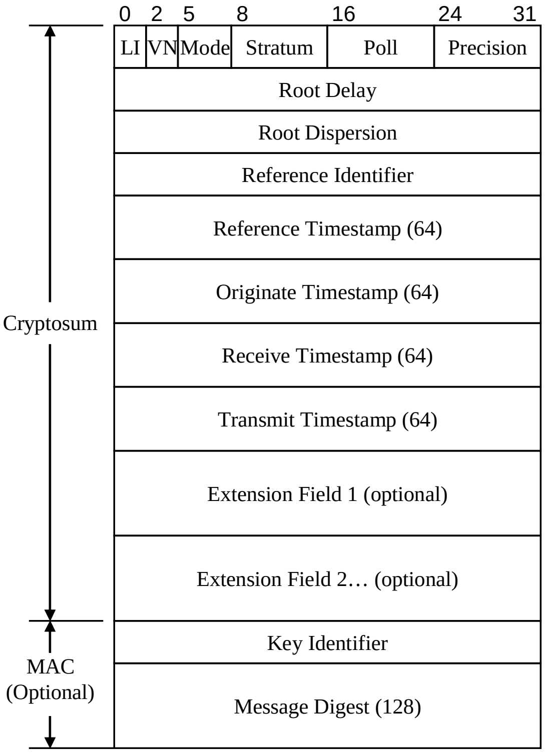


Figure 15. NTPv4 Extension Field Format

The message authentication code (MAC) consists of a 32-bit Key Identifier followed by a 128-bit Message Digest. The message digest, or cryptosum, is calculated as in RFC-1321 [8] over the fields shown in the figure.

A.2 NTPv4 Extension Field Formats

In NTPv4 one or more extension fields can be inserted after the header and before the MAC, which is always present when extension fields are present. The extension fields can occur in any order; however, in some cases there is a preferred order which improves the protocol efficiency. While previous versions of the Autokey protocol used several different extension field formats, in version 2 of the protocol only a single extension field format is used.

An extension field contains a request or response message in the format shown in Figure 15. All extension fields are zero-padded to a word (4 octets) boundary. The Length field covers the entire extension field, including the Length and Padding fields. While the minimum field length is 4 words (16 octets), a maximum field length remains to be established. The reference implementation discards any packet with an extension field length more than 1024 octets.

The presence of the MAC and extension fields in the packet is determined from the length of the remaining area after the header to the end of the packet. The parser initializes a pointer just after the header. If the Length field is not a multiple of 4, a format error has occurred and the packet is discarded. The following cases are possible based on the remaining length in words.

0 The packet is not authenticated.
1 The packet is an error report or crypto-NAK.
2, 3, 4 The packet is discarded with a format error.
5 The remainder of the packet is the MAC.
>5 One or more extension fields are present.

If an extension field is present, the parser examines the Length field. If the length is less than 4 or not a multiple of 4, a format error has occurred and the packet is discarded; otherwise, the parser increments the pointer by this value. The parser now uses the same rules as above to determine whether a MAC is present and/or another extension field. An additional implementation-dependent test is necessary to ensure the pointer does not stray outside the buffer space occupied by the packet.

In the Autokey Version 2 format, the 8-bit Code field specifies the request or response operation, while the 6-bit Version Number (VN) field is 2 for the current protocol version. The R bit is lit for a response and the E bit lit for an error. The Timestamp and Filestamp fields carry the seconds field of an NTP timestamp. The Timestamp field establishes the signature epoch of the data field in the message, while the Filestamp field establishes the generation epoch of the file that ultimately produced the data that is signed. The optional Value field carries the data and the optional Signature field the signature that validates the data. Further details are in [4].