The Network Time Protocol Version 4 Algorithm Specification
draft-ietf-ntp-ntpv4-algorithms-01

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on May 13, 2006.

Copyright Notice

Copyright (C) The Internet Society (2005).

Abstract

The Network Time Protocol (NTP) is widely used to synchronize computer clocks in the Internet. This memorandum describes the algorithms used by Version 4 of the NTP (NTPv4) to calculate time values.
Table of Contents

1. Introduction ................................................. 3
2. Clock Filter Algorithm ....................................... 6
3. Clock Selection Algorithm ..................................... 8
4. Clustering Algorithm .......................................... 9
5. Clock Combining Algorithm .................................... 10
6. Polling Algorithm ............................................. 11
7. Clock Discipline Algorithm .................................... 17
   7.1. Poll Interval Control .................................... 20
   7.2. State Machine ........................................... 20
8. Security Considerations ....................................... 22
9. IANA Considerations ........................................... 22
10. Acknowledgements .............................................. 22
11. References .................................................. 22
   11.1. Normative References .................................. 22
   11.2. Informative References ................................ 23
Authors’ Addresses .............................................. 24
Intellectual Property and Copyright Statements .................... 25
1. Introduction

The Network Time Protocol Version 3 (NTPv3) specified in [1] has been widely used to synchronize computer clocks in the global Internet. It provides comprehensive mechanisms to access national time and frequency dissemination services, organize the NTP subnet of servers and clients and adjust the system clock in each participant. In most places of the Internet of today, NTP provides accuracies of 1-50 ms, depending on the characteristics of the synchronization source and network paths.

NTP is designed for use by clients and servers with a wide range of capabilities and over a wide range of network jitter and clock frequency wander characteristics. Many users of NTP in the Internet of today use a software distribution available from www.ntp.org. The distribution, which includes the full suite of NTP options, mitigation algorithms and security schemes, is a relatively complex, real-time application. While the software has been ported to a wide variety of hardware platforms ranging from personal computers to supercomputers, its sheer size and complexity is not appropriate for many applications. This facilitated the development of the Simple Network Time Protocol Version 4 (SNTPv4) as described in [2].

Since the standardization of NTPv3, there has been significant development which has led to Version 4 of the Network Time Protocol (NTPv4). This document describes NTPv4, which introduces new functionality to NTPv3 as described in RFC 1305, and functionality expanded from that of SNTPv4 as described in RFC 2030 (SNTPv4 is a subset of NTPv4).

When operating with current and previous versions of NTP and SNTP, NTPv4 requires no changes to the protocol or implementations now running or likely to be implemented specifically for future NTP or SNTP versions. The NTP and SNTP packet formats are the same and the arithmetic operations to calculate the client time, clock offset and round trip delay are the same. To a NTP or SNTP server, NTP and SNTP clients are indistinguishable; to a NTP or SNTP client, NTP and SNTP servers are indistinguishable.

NTP usually operates simultaneously with multiple servers and may have multiple clients of its own. NTP employs several algorithms that together allow the calculation of time from messages that come from an NTP or SNTP server. The overall organization of the algorithms is illustrated in Figure 1. For every server there are two processes, a peer process which receives and processes each packet, and a companion poll process which sends packets to the server at programmed intervals. State variables and data measurements are maintained separately for each pair of processes in
a block of memory called the peer variables. The peer and poll processes together with their variables collectively belong to an association. Associations can be either temporary or permanent. Permanent associations are described as persistent, while temporary associations are referred to as preemptable or ephemeral.

Figure 1 NTPv4 Algorithm Interactions

As each NTP packet arrives, the server time is compared to the system clock and an offset specific to that server is determined. The system process refines these offsets using the selection, clustering and combining algorithms and delivers a correction to the clock discipline process, which functions as a lowpass filter to smooth the data and close the feedback loop. The clock adjust process runs at
one-second intervals to amortize the corrections in small adjustments that approximate a continuous, monotonic clock. The output of the combining algorithm represents the best estimate of the system clock offset relative to the server ensemble. The discipline algorithm adjusts the frequency of the variable frequency oscillator (VFO) to minimize this offset. Finally, the timestamps of each server are compared to timestamps derived from the VFO in order to calculate the server offsets and close the feedback loop.

Depending on whether an NTP host is acting as server or client, or whether the host is an SNTP or full NTP host, the subset of algorithms it employs varies. The relationship between host role/type and algorithm employment is summarized in Table 1.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Applicability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Filter</td>
<td>Required by all NTP Servers</td>
</tr>
<tr>
<td>Clock Selection</td>
<td>Applies to NTP hosts utilizing more than one source</td>
</tr>
<tr>
<td>Clustering</td>
<td>Applies to NTP hosts utilizing more than one source</td>
</tr>
<tr>
<td>Clock Combining</td>
<td>Applies to NTP hosts utilizing more than one source</td>
</tr>
<tr>
<td>Polling</td>
<td>Applies to all NTP hosts.</td>
</tr>
<tr>
<td>Clock Discipline</td>
<td>Not required by SNTP clients. Applies to all other NTP hosts.</td>
</tr>
</tbody>
</table>

This document is organized as follows. Section 2 describes the clock filter algorithm. Section 3 describes the clock selection algorithm. Section 4 describes the clustering algorithm. Section 5 describes the clock combining algorithm. Section 6 describes the polling algorithm. Section 7 describes the clock discipline algorithm. Sections 8 and 9 presents Security Considerations and IANA Considerations, respectively. Much of the information contained within this document is based on material from. [3]

NTPv4 is hereafter referred to simply as NTP, unless explicitly noted.

The remainder of this document contains numerous variables and
2. Clock Filter Algorithm

The NTP clock filter algorithm selects the most appropriate sample data while rejecting noise spikes due to packet collisions and network congestion. The clock offset (theta) and roundtrip delay (delta) samples are computed from the four most recent timestamps. Without making any assumptions about the delay distributions, but assuming the frequency difference or skew between the server and peer clocks can be neglected, let (theta, delta) represent the offset and delay when the path is otherwise idle; thus (theta, delta) represents the true offset and delay values. The clock filter algorithm essentially acts as an accurate estimator and produces an estimate of the time, known as (theta_hat, delta_hat), from a sample sequence (theta_i, delta_i), where i denotes a particular sample at some time, collected for the path over an appropriate interval under ambient traffic conditions.

The design of the clock filter algorithm was suggested by the observation that packet switching networks are most often operated well below the knee of the throughput-delay curve, which means that packet queues are mostly small with relatively infrequent bursts. In addition, the routing algorithm most often operates to minimize the number of packet-switch hops and thus the number of queues. Not only is the probability that an NTP packet finds a busy queue in one direction relatively low, but the probability of packets from a single exchange finding busy queues in both directions is even lower. Therefore, the best offset samples should occur with the lowest delays.

Upon arrival of an NTP packet resulting from some poll interval at time t=0, a shift register containing four variables (theta_i, delta_i, e_i, t_i) is populated with the 0th sample, (theta_0, delta_0, e_0, t_0). Here, e is the error (in seconds), which is initially set to precision and grown at a rate r=15 ppm for each epoch. If a packet has not arrived for three successive poll intervals, then the sample (0, 0, 16, t) is shifted into the register, where t is the last current known time. Missing data
samples that force this condition are never used in subsequent filter calculations, but do prevent very old (i.e. stale) samples from being used.

Next, the register contents are copied to a temporary list and sorted by the metric \( \lambda \) designed to avoid missing data and devalued samples older than the compromise Allan intercept \( \sigma_y(x) = 1500 \) s. The Allan intercept is the intersection coordinate \((x, y)\) of the phase and frequency lines. It characterizes each particular timing source and clock oscillator. A useful statistic is the \( x \) value, which specifies the optimum time constant for the particular source and oscillator combination. The \( x \) value ranges from about 500 s to 2000 s. Above this value the performance is limited by oscillator wander, while below this value the performance is limited by system jitter. For comparison, the NTPv4 clock discipline time constant is about 1000 s at a poll interval of 64 s. The \( y \) statistic represents the best stability that can be achieved for the particular source and oscillator, but is not useful for performance optimization. For this reason, the term Allan intercept applies to the \( x \) value at the intercept point.

If \( e_j = \infty \), then \( \lambda_j = \infty \); else, if \( t_j - t > \sigma_y(x) \) then \( \lambda_j = K_d + e_j \); else, \( \lambda_j = \delta_j \), where \( K_d = 1 \) s is the selection threshold. The algorithm essentially sorts the data by exchanging sets; however, an exchange is not made unless to do so would reduce the metric by at least the value of the precision. In other words, it does not make sense to change the order in the list, which might result in the loss of otherwise good samples, unless the metric change is significant. The first entry \((\theta_0, \delta_0, e_0, t_0)\) on the temporary list represents the lowest delay sample, which is used to update the peer offset \( \theta = \theta_0 \) and peer delay \( \delta = \delta_0 \). The peer dispersion \( e \) is calculated from the temporary list:

\[
e = \text{sum from } k=0 \text{ to } k=n-1 \text{ of } \left[e_k/(2^{(k+1)})\right].
\]

Finally, the temporary list is trimmed by discarding all entries where \( \lambda_j = \infty \) and all but the first devalued entry \( \lambda_j \geq K_d \), if one is present, leaving \( 0 \leq m < n \) surviving entries on the list. The peer jitter \( \psi \) is used by the clustering algorithm as a quality metric and in the computation of the expected error:

\[
\psi = \left\{ \frac{1}{m-1} \right\} \ast \left( \text{sum from } k = 1 \text{ to } k = m-1 \text{ of } \left[ (\theta_k - \theta_0)^2 \right] \right)^{1/2}.
\]

A ‘popcorn spike’ is a transient outlier, usually only a single sample, that is typical of congested Internet paths. The popcorn
spike suppressor is designed to detect and remove them. Let theta_prime be the peer offset determined by the previous message and psi the current peer jitter. If |theta - theta_prime| > (K_s * psi), where K_s is a tuning parameter that defaults to 3, the sample is a popcorn spike and is discarded.

Note that the peer jitter will increase to protect a legitimate step change.

As demonstrated by simulation and practical experience, it is prudent to avoid using samples more than once. Let t_p be the epoch the peer variables were last updated and t_0 the epoch of the first sample on the temporary list. If t_0 <= t_p, the new sample is a duplicate or earlier than the last one used. If this is true, the algorithm exits without updating the system clock; otherwise, t_p = t_0 and the offset can be used to update the system clock. The components of the five-tuple (theta, delta, e, psi, t_p) are called the peer variables.

3. Clock Selection Algorithm

In order to provide reliable synchronization, NTP uses multiple redundant servers and multiple disjoint network paths whenever possible. When a number of associations are established, it is not clear beforehand which are truechimers and which are falsetickers. A ‘truechimer’ is a clock that maintains timekeeping accuracy to a previously published (and trusted) standard, while a ‘falseticker’ is a clock that do not maintain that level of timekeeping accuracy. Crucial to the success of this approach is a robust algorithm which finds and discards the falsetickers from the raw server population, since the timekeeping accuracy of a particular server may not be known a priori. The clock selection algorithm determines from among all associations a suitable subset of truechimers capable of providing the most accurate and trustworthy time using principles similar to. [4]

The true offset theta of a correctly operating clock relative to UTC must be contained in a computable range, called the confidence interval, equal to the root distance defined below. Marzullo and Owicki devised an algorithm designed to find the intersection interval containing the correct time given the confidence intervals of m clocks, of which no more than f are considered incorrect. The algorithm finds the smallest intersection interval containing points in at least (m - f) of the given confidence intervals. [5]

The clock selection algorithm operates as follows:
1. For each of m associations, construct a correctness interval
[(theta - rootdist()), (theta + rootdist())].

2. Select the lowpoint, midpoint and highpoint of these
intervals. Sort these values in a list from lowest to highest.
Set the number of falsetickers f = 0.

3. Set the number of midpoints d = 0. Set c = 0. Scan from
lowest endpoint to highest. Add one to c for every lowpoint,
subtract one for every highpoint, add one to d for every midpoint.
If c >= m - f, stop; set l = current lowpoint

4. Set c = 0. Scan from highest endpoint to lowest. Add one to
c for every highpoint, subtract one for every lowpoint, add one to
d for every midpoint. If c >= m - f, stop; set u = current
highpoint.

5. Is d = f and l < u?
if yes, then follow step 5y, else, follow step 5n.

5y. Success: the intersection interval is [l, u].

5n. Add one to f. Is f < (m / 2)? If yes, then go to step 3
again. If no, then go to step 6.

6. Failure; a majority clique could not be found. Stop
algorithm.

4. Clustering Algorithm

NTP configurations usually include several servers in order to
provide sufficient redundancy for the selection algorithm to
determine which are truechimers and which are not. When a sizeable
number of servers are present, the individual clock offsets for each
are not always the same, even if each server is closely synchronized
to UTC by one means or another. Small systematic differences in the
order of a millisecond or two are usually due to interface and
network latencies. Larger differences are due to asymmetric delays
and in the extreme due to asymmetric satellite/landline delays.

The clustering algorithm sifts the truechimers of the selection
algorithm to identify the survivors providing the best accuracy. In
principle, the sift could result in a single survivor and its offset
estimate used to discipline the system clock; however, a better
estimate usually results if the offsets of a number of survivors are
averaged together. So, a balance must be struck between reducing the
The clustering algorithm steps follow:

1. Let \((\theta, \phi, \Lambda)\) represent a candidate peer with offset \(\theta\), jitter \(j\) and a weight factor \(\Lambda = \text{stratum} \times \text{MAXDIST} + \text{rootdist()}\).

2. Sort the candidates by increasing \(\Lambda\). Let \(n\) be the number of candidates and \(NMIN\) the minimum number of survivors.

3. For each candidate compute the selection jitter \(j_{subS}\) (RMS peer offset differences between this and all other candidates).

4. Select \(j_{\max}\) as the candidate with maximum \(j_{\ell S}\).

5. Select \(j_{\min}\) as the candidate with minimum \(j_{\ell S}\).

If yes, go to step 6y. If no, go to step 6n.

6y. Done. The remaining cluster survivors are correct. The survivors are in the \(v\) structure sorted by \(\Lambda\).

6n. Delete the outlyer candidate with \(j_{\max}\); reduce \(n\) by one, and go back to step 3.

5. Clock Combining Algorithm

The selection and clustering algorithms operate to select a single system peer based on stratum and root distance. The result is that the NTP subnet forms a logical tree with the primary servers at the root and other servers at increasing stratum levels toward the leaves. However, since each server on the tree ordinarily runs the NTP protocol with several other servers at equal or lower stratum, these servers can provide diversity paths for backup and cross checking. While these other paths are not ordinarily used directly for synchronization, it is possible that increased accuracy can be obtained by averaging their offsets according to appropriately chosen weights.

The result of the clustering algorithm is a set of survivors (there must be at least one) that represent truechimers, or correct clocks. If only one peer survives or if the prefer peer is among the survivors, that peer becomes the system peer and the combining algorithm is not used. Otherwise, the final clock correction is determined by the combining algorithm.
Let the three-tuple \((\theta_i, \psi_i, \Lambda_i)\) represent the peer offset, peer jitter, and root distance for the \(i\)th survivor. Then the combined peer offset and peer jitter is, respectively:

\[
T = (a \times \text{sum}) \text{ over all } i \text{ of } \frac{\theta_i}{\Lambda_i} \quad \text{and} \quad \psi_r = (a \times \text{sum}) \text{ over all } i \text{ of } \sqrt{\frac{(\psi_i)^2}{\Lambda_i}},
\]

where \(a\) is a normalization constant:

\[
a = \frac{1}{\text{sum over all } i \text{ of } \frac{1}{\Lambda_i}}.
\]

The result \(T\) is the system offset processed by the clock discipline algorithm. Note that the root distance cannot be less than the precision in order to avoid divide exceptions.

Let \(\psi_s\) represent the selection jitter associated with the system peer and \(\psi_r\) as above. Then the system jitter is defined as:

\[
s_j = \sqrt{(\psi_r)^2 + (\psi_s)^2}.
\]

The system jitter represents the best estimate of error in computing the clock offset. It is interpreted as the expected error statistic available to application program.

6. Polling Algorithm

The poll process determines whether and when to send a poll message to the server. Ordinarily, polls are sent at regular intervals determined by the clock discipline time constant. In some cases where justified by network load, performance can be improved and network jitter reduced by sending several messages instead of just one. This can be done when the server is unreachable, when it is reachable or both. The most common cases where this is advisable is when using very large poll intervals in the order of several hours or more.

The poll interval starts out normally at about one minute. If the offset is less than a tuning constant times the system jitter for some number of polls, it is increased, but usually not above 1024 seconds. Otherwise, it is decreased, but usually not below 64 seconds. The limits can be changed to a lower limit of 16 seconds and/or to an upper limit of 36 hours. In order to minimize network traffic, when a server has not been heard for some time, the poll interval is increased in stages to 1024 seconds.

The poll process sends packets to the server at designated intervals \(\tau\) and updates the "reach" register which establishes whether the
server is reachable. Table 2 shows the poll process routines and Table 3 the variables shared by the process routines, including poll(), peer_xmit(), fast_xmit() and poll_update().

**Table 2. Poll Process Routines**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Related routines</th>
</tr>
</thead>
<tbody>
<tr>
<td>poll</td>
<td>poll</td>
<td>*clock_adjust, clock_filtert, peer_xmit, poll_update</td>
</tr>
<tr>
<td>poll_update</td>
<td>poll update</td>
<td>*packet, *poll</td>
</tr>
<tr>
<td>peer_xmit</td>
<td>peer transmit</td>
<td>*poll, md5</td>
</tr>
<tr>
<td>fast_xmit</td>
<td>fast transmit</td>
<td>*receive, md5</td>
</tr>
</tbody>
</table>

**Table 3. Poll Process Variables**

<table>
<thead>
<tr>
<th>Name</th>
<th>Process</th>
<th>Variable Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpoll</td>
<td>poll</td>
<td>host poll interval</td>
</tr>
<tr>
<td>hmode</td>
<td>poll</td>
<td>host mode</td>
</tr>
<tr>
<td>count</td>
<td>poll</td>
<td>burst counter</td>
</tr>
<tr>
<td>reach</td>
<td>poll</td>
<td>&quot;reach&quot; register</td>
</tr>
<tr>
<td>unreach</td>
<td>poll</td>
<td>unreach counter</td>
</tr>
<tr>
<td>t</td>
<td>local clock</td>
<td>current time</td>
</tr>
<tr>
<td>tau</td>
<td>local clock</td>
<td>poll interval</td>
</tr>
<tr>
<td>rho</td>
<td>system</td>
<td>system peer</td>
</tr>
<tr>
<td>M_BCST</td>
<td>parameter</td>
<td>broadcast server</td>
</tr>
<tr>
<td>M-BCLN</td>
<td>parameter</td>
<td>broadcast client</td>
</tr>
<tr>
<td>B_BURST</td>
<td>peer flag</td>
<td>burst enable</td>
</tr>
<tr>
<td>B_IBURST</td>
<td>peer flag</td>
<td>initial burst enable</td>
</tr>
<tr>
<td>B_COUNT</td>
<td>parameter</td>
<td>pkts in a burst</td>
</tr>
</tbody>
</table>

The poll() routine is described in Figure 2. Each time the poll() routine is called, the reach variable is shifted left by one bit. When a packet is accepted by the packet() routine in the peer process the rightmost bit is set to one. As long as reach is nonzero, the server is considered reachable. However, if the rightmost three bits become zero, indicating that packets from the server have not been received for at least three poll intervals, a sample with MAXDIST dispersion is shifted in the clock filter. This causes the server to be devalued in the mitigation process. The unreach counter increments at each poll interval; it is reset to zero if the reach register is nonzero. If the counter exceeds the UNREACH parameter,
the poll exponent is incremented for each succeeding poll. This reduces useless network load in case of server failure. The poll() routine can operate in three modes.

Ordinarily, polls are sent at the interval selected by hpoll and ppoll poll exponents assigned. However, if the iburst feature is enabled and the server is not reachable, a burst of eight polls is sent at two-second intervals. Alternatively or in addition, if the burst feature is enabled and the server is reachable, a burst of eight polls is sent as with iburst. This is especially useful at very large poll intervals of many hours. The remaining routines are straightforward. The poll() routine calls the peer_xmit() routine when an association has been mobilized. The receive() routine calls fast_xmit() when a client mode packet is received. Both cases are shown in Figure 3. These routines copy values from the association (peer_xmit()) or from the arriving packet (fast_xmit()) as shown in the accompanying tables. The poll_update() routine shown in Figure 4 determines the next poll interval or burst interval. Variable names in both routines are referenced in Tables 4 and 5, respectively.

<table>
<thead>
<tr>
<th>poll()</th>
<th>--&gt;</th>
<th>hmode=M_BCST?</th>
</tr>
</thead>
</table>

if hmode=M_BCST == YES:

<table>
<thead>
<tr>
<th>s.rho = NULL?</th>
</tr>
</thead>
</table>

if s.rho=NULL == YES:

| poll_update() |-->| exit() |

if s.rho=NULL == NO:

| hmode=M_BCLN? |

if hmode=M_BCLN == YES:

| poll_update() |-->| exit() |

if hmode=M_BCLN == NO:

| peer_xmit() |-->| poll_update() |-->| exit() |

if hmode=M_BCST == NO:

| burst = 0? |

if burst = 0 == YES:
if reach = 0 == YES:

if unreach > UNREACH == YES:

if unreach > UNREACH == YES:

if unreach > UNREACH == YES:

if unreach > UNREACH == YES:

if unreach > UNREACH == YES:

if B_IBURST == NO:

if B_IBURST == NO:

if reach = 0 == NO:

if reach = 0 == YES:

if reach & 0x7 = 0 == YES:

| clock_filter(0,0,inf,t) |

\|

| hpoll = c,tau |

\|/
if B_BURST == YES:
    -----------------------
    | unreach = 0? |
    -----------------------
    if unreach = 0 == YES:
        -----------------------
        | burst=BCOUNT |-->|hmode=M_BCLN?|
        -----------------------
        | hmode=M_BCLN?|
        -----------------------
    if unreach = 0 == NO:
        -----go to-----
        |hmode=M_BCLN?|
        -----------------------
    if B_BURST == NO:
        -----go to-----
        |hmode=M_BCLN?|
        -----------------------
    if unreach = 0 == NO:
        -----go to-----
        |hpoll=c,tau |
        -----------------------
    if burst = 0 == NO:
        -----------------------
        | burst-- |-->|hmode=M_BCLN?|
        -----------------------
END OF ROUTINE

Figure 2. Poll Routine

Table 4. Peer Fast Transmit Table

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Origin Timestamp in NTP packet field</td>
</tr>
<tr>
<td>T2</td>
<td>Receive Timestamp in NTP packet field</td>
</tr>
<tr>
<td>T3</td>
<td>???</td>
</tr>
<tr>
<td>mac</td>
<td>???</td>
</tr>
</tbody>
</table>
Figure 3. Peer Fast Transmit

Table 5. Poll Process Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Process</th>
<th>Variable Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppoll</td>
<td>peer</td>
<td>peer poll interval</td>
</tr>
<tr>
<td>hpoll</td>
<td>poll</td>
<td>host poll interval</td>
</tr>
<tr>
<td>burst</td>
<td>poll</td>
<td>burst counter</td>
</tr>
<tr>
<td>timer</td>
<td>poll</td>
<td>poll timer</td>
</tr>
<tr>
<td>BTIME</td>
<td>parameter</td>
<td>burst time</td>
</tr>
<tr>
<td>MINPOLL</td>
<td>parameter</td>
<td>minimum poll interval</td>
</tr>
<tr>
<td>MAXPOLL</td>
<td>parameter</td>
<td>maximum poll interval</td>
</tr>
</tbody>
</table>
The clock discipline algorithm synchronizes the computer clock with respect to the best time value from each server and the best combination of servers. This algorithm automatically adapts to changes in operating environment without manual configuration or real-time management functions. The clock discipline algorithm is implemented as the feedback control system shown in Figure 5.
The variable theta_r represents the combined server reference phase and theta_c the control phase of the VFO. Each update received from a server produces a signal V_d representing the instantaneous phase difference theta_r - theta_c. The clock filter for each server functions as a tapped delay line, with the output taken at the tap selected by the clock filter algorithm. The selection, clustering and combining algorithms combine the data from multiple filters to produce the signal V_s. The loop filter, with impulse response F(t), produces the signal V_c which controls the VFO frequency omega_c. Thus, its phase theta_c follows:

\[
theta_c = \text{integral over } t \text{ of } (\omega_c(t) \, dt)
\]

which closes the loop. The V_c signal is generated by an adjustment process which runs at intervals of one second in the NTP daemon or one tick in the kernel. are set to 0,

The NTPv4 discipline includes both PLL and FLL capabilities. The selection of which mode to use, PLL or FLL and in what combination, is made on the basis of the poll exponent tau. In the NTPv4 design, PLL mode is used at smaller values of tau, while FLL mode is used at larger values. In between, a combination of PLL and FLL modes is
used. This improves the clock accuracy and stability, especially for
poll intervals larger than the Allan intercept.

\[ x \leftarrow \text{Phase Correction} \leftarrow y_{\text{FLL}} \]
\[ \quad \text{-(FLL Predict)} \leftarrow V_s \]
\[ \quad \text{\textbackslash|/} \]
\[ \quad y \leftarrow \text{(Sum)} \]
\[ \quad \text{\textbackslash|/} \]
\[ \quad y_{\text{PLL}} \]
\[ \text{-(PLL Predict)} \leftarrow \]

Figure 6. FLL/PLL Prediction Functions

In PLL mode \( y \) is a time integral over all past values of \( V_s \), so the
PLL frequency adjustment required is:

\[ y_{\text{PLL}} = \left( \frac{V_s \times \mu}{(64 \times T_c)^2} \right) \]

where \( T_c \) is the time constant. In FLL mode, \( y \) is an average of past
frequency changes, as computed from \( V_s \) and \( \mu \). The goal of the
algorithm is to reduce \( V_s \) to zero; so, to the extent this has been
successful in the past, previous values can be assumed zero and the
average becomes:

\[ y_{\text{FLL}} = \left( \frac{V_s - x}{8 \times \mu} \right) \]

where \( x \) is the residual phase error computed by the clock adjust
process.

Finally, in both PLL and FLL modes set the phase to \( x = V_s \) and
frequency \( y = [y + y_{\text{PLL}} + y_{\text{FLL}}] \).

Once each second the adjustment process computes a phase increment \( z = \left( \frac{x}{16 \times T_c} \right) \) and new phase adjustment \( x = x - z \). The phase
increment \( z \) is passed to the kernel time adjustment function. This
continues until the next update which recomputes \( x \) and \( y \).

A key factor in the performance of the PLL/FLL hybrid algorithm are
the weight factors for the \( y_{\text{PLL}} \) and \( y_{\text{FLL}} \) adjustments, which depend
on the poll exponent \( \tau \) which in turn determines the time constant
\( T_c = (2^\tau) \), in seconds. PLL contributions should dominate at
the lower values of \( \tau \), while FLL contributions should dominate at
the higher values. The clock discipline algorithm response times to
several PPM deviation examples is presented in . [3]
The NTPv4 algorithm aims to set the averaging time somewhere near the Allan intercept. A key to this strategy is the measured clock jitter and oscillator wander statistics. The clock jitter is estimated from phase differences $\psi_c = \left( <\Delta_x^2> \right)^{1/2}$, where the brackets indicate an exponential average. The oscillator wander is estimated from frequency differences $\psi_f = \left( T_c <\Delta_y^2> \right)^{1/2}$. As the poll exponent $\tau$ increases, it is expected that $\psi_c$ will decrease and $\psi_f$ will increase, depending on the relative contributions of phase noise and frequency noise.

In the NTPv4 algorithm at each update a counter is incremented by one if $x$ is within the bound $|x| < 4 \times \psi_c$, where the constant 4 is determined by experiment, and decremented by one otherwise.

In order to avoid needless hunting, a degree of hysteresis is built into the scheme. If the counter reaches an upper limit 30, $\tau$ is increased by one; if it reaches a lower limit -30, $\tau$ is reduced by two. In either case the counter is reset to zero. Under normal conditions $\tau$ increases in stages from a default lower limit of 6 (64 s) to a default upper limit of 10 (1024 seconds). However, if the wander increases because the oscillator frequency is deviating too fast, $\tau$ is quickly reduced. Once the oscillator wander subsides, $\tau$ is slowly increased again. Under typical operating conditions, $\tau$ hovers close to the maximum; but, on occasions of a heat spike when the oscillator wanders more than about 1 PPM, it quickly drops to lower values until the wander subsides.

### 7.2. State Machine

The clock discipline must operate over an extremely wide range of network jitter and oscillator wander conditions without manual intervention or prior configuration. As determined by past experience and experiment, the data grooming algorithms work well to sift good data from bad, especially under conditions of light to moderate network and server loads. The PLL/FLL hybrid algorithm may perform poorly and even become unstable under heavy network loading. The state machine functions to bypass some discipline functions under conditions of hardware or software failure, severe time or frequency transients and especially when the poll interval is relatively large.

Under normal conditions the NTP discipline algorithm writes the current frequency offset to a file at hourly intervals. Once the file is written and the daemon is restarted after reboot, for example, it initializes the frequency offset from the file, which avoids the training time, possibly several hours, to determine the intrinsic frequency offset when the daemon is started for the first
time. When toll charges accrue for every NTP message, as in a telephone modem service, it is important to determine the presence of a possibly large intrinsic frequency offset, especially if the interval between telephone calls must be 15 minutes or more. For instance, without the state machine it might take many calls spaced at 15 minutes until the frequency offset is determined and the call spacing can be increased. With the state machine it usually takes only two calls to complete the process.

The clock state machine transition function is shown in Table 6. It determines the action and next state when an update with specified offset occurs in a given state shown in the first column. The second column shows what happens if the offset is less than the step threshold, the third when the step threshold is exceeded but not the stepout threshold and the third when both thresholds are exceeded. The state machine responds to the current state and event to cause the action shown.

<table>
<thead>
<tr>
<th>State</th>
<th>abs(T) &lt; STEP</th>
<th>abs(T) &gt; STEP</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSET</td>
<td>&gt; FREQ; adjust time</td>
<td>&gt; FREQ; step time</td>
<td>no frequency file</td>
</tr>
<tr>
<td>FSET</td>
<td>&gt; SYNC; adjust time</td>
<td>&gt; SYNC; step time</td>
<td>frequency file</td>
</tr>
<tr>
<td>SPIK</td>
<td>&gt; SYNC; adjust freq, adjust time</td>
<td>if (&lt;900 s)&gt;SPIK freq, step time</td>
<td>outlier detected</td>
</tr>
<tr>
<td>FREQ</td>
<td>if (&lt;900 s)&gt; FREQ else &gt;SYNC; step freq, adjust time</td>
<td>if (&lt;900 s)&gt;FREQ else &gt;SYNC; step freq, adjust time</td>
<td>initial frequency</td>
</tr>
<tr>
<td>SYNC</td>
<td>&gt;SYNC; adjust freq, adjust time</td>
<td>if (&lt;900 s)&gt;SPIK freq, step time</td>
<td>normal operation</td>
</tr>
</tbody>
</table>

The actions listed in the state diagram include adjust-frequency, step-frequency, adjust-time and step-time actions. The normal action in the SYNC state is to adjust both frequency and time. The step-time action is to set the system clock, while the step-frequency action is to calculate the frequency offset directly. This has to be done carefully to avoid contamination of the frequency estimate by the phase adjustment since the last update.
The machine can be initialized in two states, FSET if the frequency file is present or NSET if it has not yet been created. If the file is not present, this may be the first time the discipline has ever been activated, so it may have to quickly determine the oscillator intrinsic frequency offset. It is important to realize that a number of NTP messages may be exchanged before the mitigation algorithms determine a reliable time offset and call the clock discipline algorithm.

When the first valid offset arrives in the NSET state, (1) the time is stepped to that offset, if necessary, (2) the watchdog counter is started and (3) the machine exits to the FREQ state. Subsequently, updates will be ignored until the stepout threshold has been reached, at which time the frequency is stepped, the time is stepped if necessary, and the machine exits to SYNC state. When the first valid offset arrives in the FSET state, the frequency has already been initialized, so the machine does the same things as in the NSET state, but exits to the SYNC state.

In the SYNC state the machine watches for outliers above the step threshold. If one is found, the machine exits to SPIK state and starts the watchdog timer. If another offset less than the step threshold is found, the counter is stopped and the machine exits to the SYNC state. If the watchdog timer reaches the stepout threshold, the time and frequency are both stepped as required and the machine exits to the SYNC state.

8. Security Considerations

There are no security considerations.

9. IANA Considerations

There are no IANA considerations.

10. Acknowledgements

11. References

11.1. Normative References
11.2. Informative References


Authors’ Addresses

William Kasch (editor)
The Johns Hopkins University Applied Physics Lab
11100 Johns Hopkins Road
Laurel, Maryland  20723-6099
United States

Phone: +1 443 778 7463
Email: william.kasch@jhuapl.edu

Jack Burbank (editor)
The Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Road
Laurel, Maryland  20723-6099
United States

Phone: +1 443 778 7127
Email: jack.burbank@jhuapl.edu

Dr. David L. Mills
University of Delaware
Newark, Delaware  19716
United States

Phone: +1 302 831 8247
Email: mills@udel.edu
Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2005). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.