

Table of Contents

Part I Overview 5

1 What is Presentense Server	5
2 Network Time Synchronization	7
3 Time Protocols & Accuracy	8
4 Which Time Protocol is Right for Me ?	8
5 GPS Time Synchronization	9
6 Free Run Support	10
7 Standard Deviation Statistical Filter	13
8 System Requirements	13
9 FAQ	13

Part II Configuration 14

1 Admin Tab	14
2 Settings Tab	15
3 Internet Time Server Tab	16
4 Handheld GPS Tab	17
5 NTP4+GPS Tab	18
6 NMEA Tab	22
7 Performance Tab	23
8 Log Tab	24
9 NTP4 Key Generator	24
10 NTP4 Plug-In Drivers	25
11 Undisciplined Local Clock	26
12 NTP4 Plug-In Driver Calibration	26
13 NTP4 Plug-In Driver Troubleshooting	28

Part III Hardware 29

1 TripNav TN-200	29
2 Generic NMEA GPS	30
3 Hand Held GPS Receivers	30
4 GPS Parts List	31
5 Arbiter	32
6 Spectracom Clock Formats	34
7 Spectracom	35
8 Hopf DCF77/GPS PCI-Bus Receiver	38

9	Hopf 6870 & Compatibles	40
10	Ultralink	42
Part IV	RFC 868	43
1	RFC 868	43
Part V	RFC 2030	44
1	RFC 2030	44
Part VI	RFC 1305	56
1	RFC 1305	56
2	1. Introduction	62
3	1.1 Related Technology	63
4	2. System Architecture	66
5	2.1 Implementation Model	67
6	2.2 Network Configurations	68
7	3. Network Time Protocol	70
8	3.1 Data Formats	70
9	3.2 State Variables and Parameters	71
10	3.2.1 Common Variables	71
11	3.2.2 System Variables	74
12	3.2.3 Peer Variables	75
13	3.2.4 Packet Variables	78
14	3.2.5 Clock-Filter Variables	79
15	3.2.6 Authentication Variables	79
16	3.2.7 Parameters	80
17	3.3 Modes of Operation	82
18	3.4 Event Processing	83
19	3.4.1 Notation Conventions	83
20	3.4.2 Transmit Procedure	85
21	3.4.3 Receive Procedure	87
22	3.4.4 Packet Procedure	89
23	3.4.5 Clock-Update Procedure	91
24	3.4.6 Primary-Clock Procedure	93
25	3.4.7 Initialization Procedures	93
26	3.4.7.1 Initialization Procedure	93
27	3.4.7.2 Initialization-Instantiation Procedure	94
28	3.4.7.3 Receive-Instantiation Procedure	95
29	3.4.7.4 Primary Clock-Instantiation Procedure	95
30	3.4.8 Clear Procedure	96

31	3.4.9 Poll-Update Procedure	97
32	3.5 Synchronization Distance Procedure	97
33	3.6 Access Control Issues	98
34	4. Filtering and Selection Algorithms	99
35	4.1 Clock-Filter Procedure	100
36	4.2 Clock-Selection Procedure	101
37	4.2.1 Intersection Algorithm	101
38	4.2.2. Clustering Algorithm	103
39	5. Local Clocks	105
40	5.1 Fuzzball Implementation	106
41	5.2 Gradual Phase Adjustments	107
42	5.3 Step Phase Adjustments	109
43	5.4 Implementation Issues	110
44	6. Acknowledgments	111
45	7. References	111
46	Appendix A	117
47	Appendix B	120
48	Appendix C	129
49	Appendix D	134
50	Appendix E	137
51	Appendix F	147
52	Appendix G	153
53	Appendix H	166
54	Appendix I	176

Part VII Time Tools 179

1	PresenTense NT/2000/XP Client	179
2	PresenTense 98/95/ME Client	181
3	NTP Server Scanner	182
4	NMEA Recorder	183
5	NTP4 Query Command Line Tool	184

Part VIII NTP Auditor 185

1	PresenTense NTP Auditor	185
2	NASD OATS Regulatory Compliance	187

Part IX About 188

1	About PresenTense Server	188
2	About Bytefusion Ltd.	188

Index**0**

1 Overview

1.1 What is PresenTense Server



PresenTense Time Server - Advantages at a glance

- **Accurate and reliable** (microsecond accuracy, 24/7 availability)
- **Easy to install and maintain** (most users up and running in less than a minute)
- **Versatile** (able to use almost any reference time source including GPS and internet time servers)
- **Low total cost of ownership** (support included meaning no yearly support fees)

PresenTense Server is a multi protocol time server which synchronizes itself with any true time source and then acts as a server for all network clients. It is easy to set up but has powerful configuration settings for networks which require a high degree of accuracy. This means you can use PresenTense Server to synchronize your PC to a primary time source such as an atomic clock on the internet and offer time services to clients on your local area network. PresenTense server also allows synchronization to GPS receivers via RS-232 and the NMEA 0183 protocol. PresenTense Time Server ships with drivers for many reference clocks including Spectracom and Hopf. If you need an accurate, inexpensive and reliable time server, read on!

PresenTense Server operates three server components concurrently on the ports shown below :

1. Port 123 UDP ([SNTP - RFC 2030](#))
2. Port 37 UDP ([TIME - RFC 868](#))
3. Port 37 TCP ([TIME - RFC 868](#))
4. Port 123 UDP ([NTP3 / NTP4 - RFC 1305](#)) **Free Plug-In included in product and demo**

This feature enables you to operate time clients on your network using different time protocols simultaneously depending on your requirements for accuracy. To obtain the latest version of our NTP4 plug-in for PresenTense Server free of charge at any time, kindly email us at support@bytefusion.com.

PresenTense Server features include :

- **Multi Protocol Support**

When operating as a secondary time server, PresenTense Server can synchronize to primary, network time servers of the following protocol families :

- [1. NTP/SNTP](#)
- [2. Network Time UDP](#)
- [3. Network Time TCP](#)

- **GPS Receiver Support**

When operating as a primary time server, PresenTense Server can synchronize to GPS receivers via the PC's serial port and the NMEA 0183 protocol. In order to be compatible with PresenTense Server, GPS receivers must emit the GPRMC NMEA sentence (recommended minimum coordinates) at one or two second intervals. Special support is included for the Garmin GPS 2 Plus and the Garmin 38 hand held GPS receivers. You may use either PresenTense Server's native GPS support or the GPS support furnished by the [NTP4 plug-in's reference clock support](#).

- **Free Run Support**

PresenTense Server automatically compensates for local system clock drift when operating in NTP or SNTP mode. This ensures maximum clock accuracy when free-running in the event of loss of connectivity to the reference time source.

The [National Emergency Number Association \(NENA\)](#) identifies time keeping requirements for **Public Safety Answering Points (PSAP)** such as 911 call centers. These requirements include continuous accuracy of 100 milliseconds to UTC when locked to the reference time source and a maximum clock drift or error accumulation of one second per day when the system must "free run". See also [PresenTense NTP Auditor](#).

- **Redundancy**

An automatic fail-over mechanism switches to a redundant network time source if the primary time source becomes unavailable (only available when synchronizing to a primary network time server).

- **Email Alarm Action**

PresenTense Server can notify you by email if the time source becomes unavailable or critical errors are encountered during operation (SMTP only) .

- **NT Service**

PresenTense Server operates as an NT service - invisibly in the background. Configuration changes are made via the PresenTense Server applet in the system control panel.

- **Highly Customizable**

PresenTense Server features advanced configuration options, including the ability to evaluate the accuracy of its time source to within milliseconds.

- **Security**

Initial setup and later configuration via the control panel require administrator privileges. This minimizes the risk of tampering with system parameters by unauthorized users.

- **Remote Monitoring and Administration**

PresenTense Server may be administered fully by remote and monitored over the network to ensure accurate time keeping. Please check www.bytefusion.com for availability of monitoring software.

- **Advanced Statistical Filters**

Advanced statistical filters ensure that erroneous measurements are discarded, thus avoiding erratic system clock behavior.

- **Time Zone Specific Reference Clock Selection**

PresenTense Server automatically selects the most suitable primary reference clock server, based on your time zone configuration and geographic location.

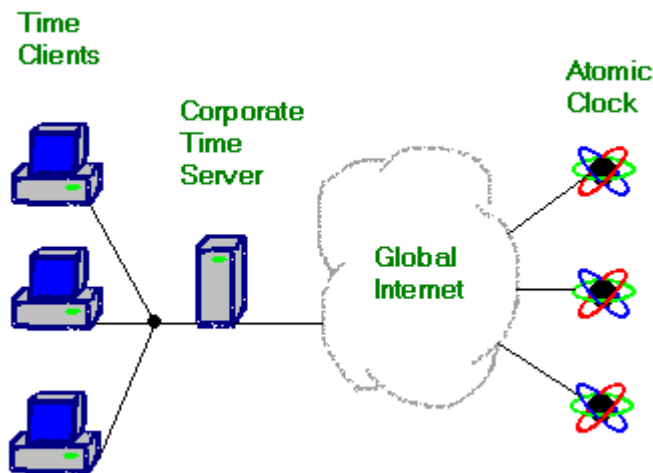
Please note that while PresenTense Server should work with other time clients that comply with the relevant RFCs, the software has only been tested with PresenTense time clients, and no support is available for the operation of PresenTense Server with non PresenTense time clients.

Trademarks

Microsoft Windows and Windows NT, Windows 2000 and Windows XP are either registered trademarks or trademarks of Microsoft Corporation. No associations are implied. Garmin 38 and Garmin GPS 2 Plus are registered trademarks or trademarks of Garmin Corporation.

1.2 Network Time Synchronization

A typical network time synchronization scenario is depicted in the diagram shown below.



In this scenario, each workstation synchronizes to a central time server on the corporate LAN. The server operates PresenTense Server software to synchronize its own clock to a primary and / or redundant time source on the global internet and each work station in turn operates PresenTense time client software to synchronize to the corporate server.

In this manner, time stamps are propagated from a single network host with access to an accurate time source to clients on the network without such access. This guarantees a coherent propagation of time to all clients.

1.3 Time Protocols & Accuracy

A number of time protocols are widely used to synchronize computer clocks with a network time server. These are :

- **NTP (Network Time Protocol)**

"Network Time Protocol". NTP has the ability to synchronize computers across multiple time servers while compensating for network jitter and other statistical errors. Because of this ability, it is generally suitable for the synchronization of a corporate time server to an atomic clock on the global internet to within tens of milliseconds. This protocol is defined in the Internet standard [RFC 1305](#).

- **SNTP (Simple Network Time Protocol)**

"Simple Network Time Protocol". SNTP is a subset of the Network Time Protocol. SNTP is able to compensate for network round trip delay and take account of nominal dispersion to the root time source, but normally depends on a single time server at any given time. Because of this, it is generally suitable for synchronization of clients to a known time source on a corporate intranet.

Accurate synchronization between client and server to the order of microseconds is attainable on local area networks using SNTP. This protocol is defined in the internet standard [RFC 2030](#).

- **TIME, UDP**

The UDP/TIME protocol permits synchronization of clients approximately to the nearest second. It provides no mechanisms to compensate for network round trip delay or similar sources of error. This protocol is defined in the internet standard [RFC 868](#).

- **TIME, TCP**

The TCP/TIME protocol is a variant of UDP/TIME which utilizes the Transmission Control Protocol (TCP) for delivery of data between client and server. While the use of TCP as a communications protocol provides no general enhancements to the synchronization between client and server, it is generally held to be more reliable than UDP. This protocol is defined in the internet standard [RFC 868](#).

See also [Which Time Protocol is Right for Me?](#)

1.4 Which Time Protocol is Right for Me ?

PresenTense Server is a multi protocol time server. This means PresenTense Server can acquire accurate time from various different time sources using a number of protocols and in turn distribute accurate time to clients using a range of network time protocols. Because time protocols differ in the degree of accuracy they facilitate, they each serve different purposes.

Many UNIX servers traditionally offer time on port 37 via UDP and TCP, but the accuracy of the time protocol ([RFC 868](#)) is limited to the nearest second. See also [Time Protocols & Accuracy](#). PresenTense Server offers time according to ([RFC 868](#)) as a matter of course and in order to complement its primary time services. However, the time protocol according to [RFC 868](#) is insufficient for all but the most rudimentary local area network time synchronization needs as it makes no provision to account for network delays or other network errors to facilitate its limited one second accuracy. For this reason, we will focus on newer, more sophisticated network time protocols: the Simple Network Time Protocol (SNTP) & the Network Time Protocol (NTP).

SNTP and NTP both use port 123 and share a common packet format. Because of this, SNTP hosts may interoperate with NTP hosts. Both NTP and SNTP compensate for network delays and other network errors. Using SNTP a theoretical precision of microseconds can be achieved. However, NTP distinguishes itself from SNTP in several important ways. NTP effects synchronization of a computer to an accurate time source in a more elegant fashion. Rather than simply correcting the local system time to bring it into agreement with the time source in an "*ad hoc*" fashion, NTP aims to gradually bring the local clock into step with the accurate time source via small, incremental adjustments to the system time or by tiny adjustments to the system clock frequency. Further, NTP employs a range of sophisticated error mitigation methods and performs statistical averaging of time samples it collects. Because of this, NTP facilitates a much greater degree of accuracy than SNTP and thus minimizes clock hopping and other erratic clock behavior.

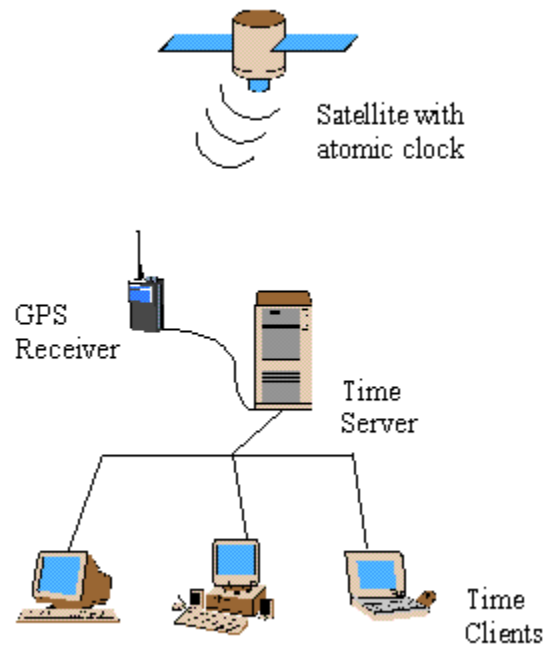
Until version 3.4, Presentense Server shipped with its own, built-in error mitigation and statistical averaging facilities. Starting with version 3.5, Presentense Server also includes a plug-in, distributed as a DLL, which employs the code base of UNIX XNTPD and supports NTP version 3 & 4 according to [RFC 1305](#). This plug-in is distributed with Presentense Server free of charge. If you require the newest, industry standard version of NTP, the use of the [NTP version 4 plug-in](#) is recommended.

1.5 GPS Time Synchronization

The global positioning system (GPS) is composed of a network of 24 satellites that orbit the earth at an altitude of about 11,625 miles. Each satellite is equipped with an atomic clock which measures time by counting oscillations of a certain type of atom.

This timing information is obtained by GPS receivers which require precise time to compute their distance to each satellite and ultimately their position on earth. GPS clocks utilize the same technology to determine accurate time. Contrary to ordinary GPS receivers which focus on making positioning information available to the user, GPS clocks focus on making timing information available at precise intervals.

Presentense Server allows you to synchronize to GPS clocks and ordinary GPS receivers via the industry standard NMEA 0183 protocol, turning your PC into an accurate time server on your local area network. A typical network time synchronization scenario using GPS technology is depicted in the diagram shown below.



In order to be compatible with PresenTense Server, GPS receivers must emit the GPRMC NMEA sentence at one or two second intervals. Special support is included for the Garmin GPS 2 Plus and the Garmin 38 hand held GPS receivers. You may use either PresenTense Server's native GPS support or the GPS support furnished by the [NTP4 plug-in's reference clock support](#).

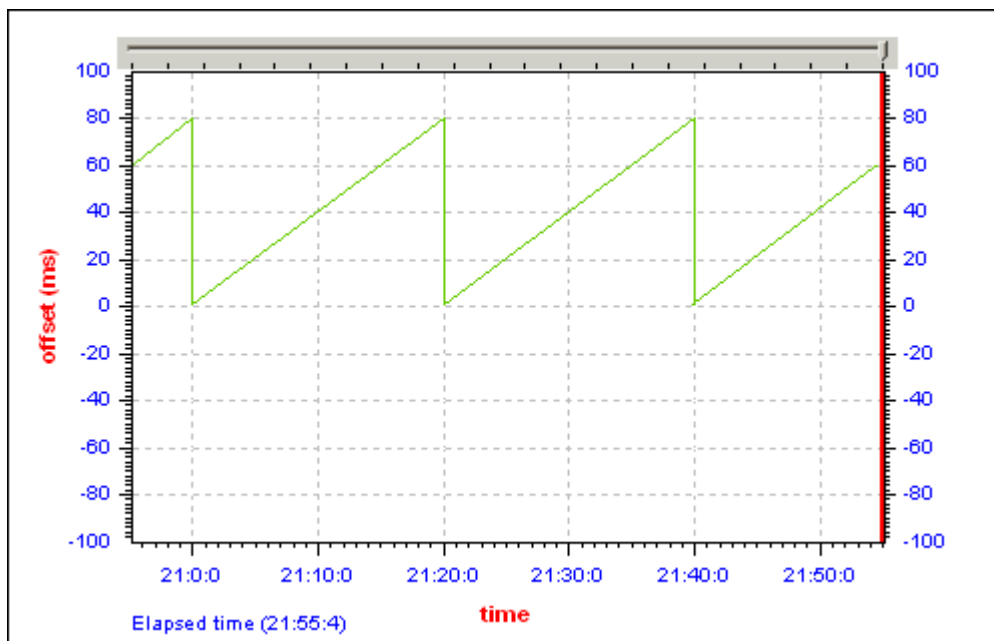
See also [Hand Held GPS Receivers](#)

1.6 Free Run Support

Most clocks lose or gain time so within a few days they no longer display the correct time. In the days of old grandfather would wind up the family clock once every couple of days and, where appropriate, adjust the hands of the clock to reflect the correct time. If grandfather found that the clock was too slow or too fast by a constant number of minutes every day, he would adjust the clock pendulum so as to correct the clock speed.



Most network time synchronization software restricts itself to the equivalent of "adjusting the hands" at certain intervals. Thus the longer the time between adjustments, the greater the cumulative error, or offset, will be. This effect is illustrated below. The graph shows the clock offset accumulating as the clock slowly drifts away from the correct time. Then an adjustment is made, thereby correcting the offset and the cycle begins anew.



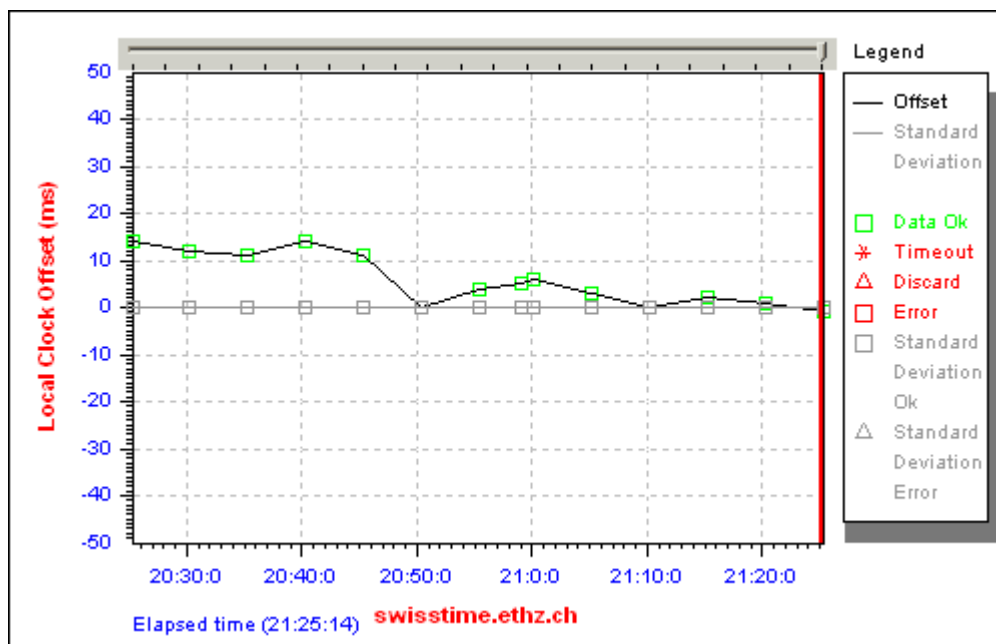
Depending on your time keeping requirements, the accumulated offset between adjustments may exceed the maximum acceptable value. Public Safety Answering Points (PSAP) such as 911 call centers, for example, are required by their governing body [NENA](#) to maintain a continuous accuracy of 100 milliseconds when synchronizing to the reference time source.

Thus, in order to effect continuous accuracy, it is not sufficient to adjust the system time at certain intervals, it is also necessary to correct the system clock speed. Once your computer system clock speed itself is correct, it is only necessary to verify correct time at certain intervals and make minimal corrections where required. Your computer may now "free run" without access to its reference time

source and without accumulating a large error between time checks.

PresenTense features free run support when synchronizing to network time sources both in [SNTP mode](#) as well as [NTP mode](#). PresenTense achieves this by first measuring the average offset or clock drift between adjustment intervals and then adjusting the operating system clock speed accordingly. Both the client and server editions of PresenTense support free run operation on the following operating systems: Windows NT, Windows 2000, Windows XP and Windows 2003. These operating systems allow micro management of the system clock via access to the clock interrupt which typically occurs every ten milliseconds. At every clock interrupt, time may be added or subtracted with 100 nanosecond precision, thus correcting the overall system clock speed.

The benefits of free run support are illustrated below. The image shows an actual [PresenTense NTP Auditor](#) graph on a 50 millisecond scale as PresenTense progressively refines the system clock speed on a typical desktop system.



Free run support is enabled by default when operating in the following modes :

1)

[Settings Tab](#): Click **Network** in the **Reference Time Source** box.

[Network Tab](#): Click **NTP/SNTP** in the **Time Source Type** box. Enter reference time servers as appropriate.

2)

[Settings Tab](#): Click **NTP V.4 Plug-In** in the **Reference Time Source** box.

[NTP4 Tab](#): Enter reference time servers as appropriate.

Trademarks:

Windows NT, Windows 2000, Windows XP and Windows 2003 are either registered trademarks or trademarks of Microsoft Corporation. No associations are implied.

1.7 Standard Deviation Statistical Filter

Standard Deviation is a statistical measure of variance in a sample set. It is calculated from the square root of the variance of the sample set as shown below :

$$S = \sqrt{\frac{1}{n-1} \sum_{i=0}^{n-1} (X_i - \bar{X})^2}$$

where **n** is the number of data samples
and $\bar{X}$ denotes the arithmetic mean of the sample set.

The standard deviation of time measurements is important to PresenTense because it reflects the quality of the network connection to the legal source of Coordinated Universal Time and therefore reflects the quality of the time measurements themselves. As the quality of service on the internet is subject to variation, the standard deviation is computed for every set of measurements. If the standard deviation is excessively large or varies substantially from previous measurements, the affected measurements are marked as discarded.

1.8 System Requirements

PresenTense Server Requires :

- A Personal Computer with a 486/33 MHz CPU or higher
- Microsoft Windows NT 4.0, Window 2000 or Windows XP
- Mouse or compatible pointing device
- 800 KB of hard-disk space

Also Required:

- TCP/IP networking must be installed and configured.
- Administrator privileges are required for the installer on each workstation on which PresenTense is to be installed.

1.9 FAQ

- **Are version upgrades free ?**

Upgrades within the major version number are free. This includes bug fixes and patches. Example: If you purchased product version 3.4 (major version number three), then upgrades to version 3.5 through 3.9 are free. Such updates are available to licensed users upon request.

- **Will PresenTense keep my PCs synchronized without user intervention ?**

Yes, PresenTense Server and PresenTense Client for Windows NT operate as system services. PresenTense98 operates as a simple service under Windows 95 and Windows 98. All are designed to start at boot time and operate before user log on and after the user has logged off, thus keeping your PCs synchronized at all times.

- **Why operate separate server and client versions of PresenTense ?**

Theoretically, you could install the server software on all PC's on your network and synchronize to a single time source as PresenTense server has both server and client components. However, this would enable individual users on your network to offer time services on your network

- **What GPS clocks are presently supported by PresenTense server ?**

At the time of writing, any generic GPS receiver that communicates via a standard RS-232 serial connection using the NMEA 0183 protocol and emits the GPRMC sentence is supported. Due to the low cost and popularity of hand held GPS systems, special support for the Garmin II Plus, Garmin III Plus, Garmin V, and Garmin 38 hand held GPS receivers is included also. Using the free NTP4 Plug-In on PresenTense Server, a wider range of hardware clocks is supported. See NTP4 Plug-In Drivers (PresenTense Server Only).

- **Where does PresenTense Server obtain accurate time ?**

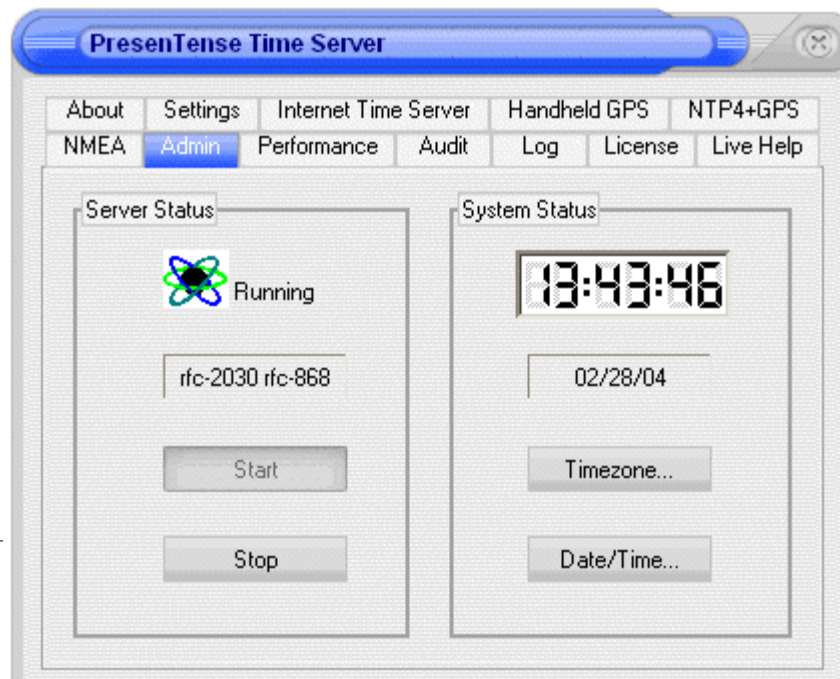
As with any other clock, the BIOS clock of your PC is subject to a gradual drift. This means PresenTense server cannot rely on the PC's system clock for accurate timing information. Instead it must be able to query an accurate time source for precise timing information. Such accurate time source is typically an atomic clock on the internet or a locally operated Global Positioning System (GPS) receiver.

- **Does PresenTense server operate through a firewall ?**

Yes. Please ask your system administrator to open ports 123 and 37 depending on the protocol you have selected.

2 Configuration

2.1 Admin Tab



- **Service Status**

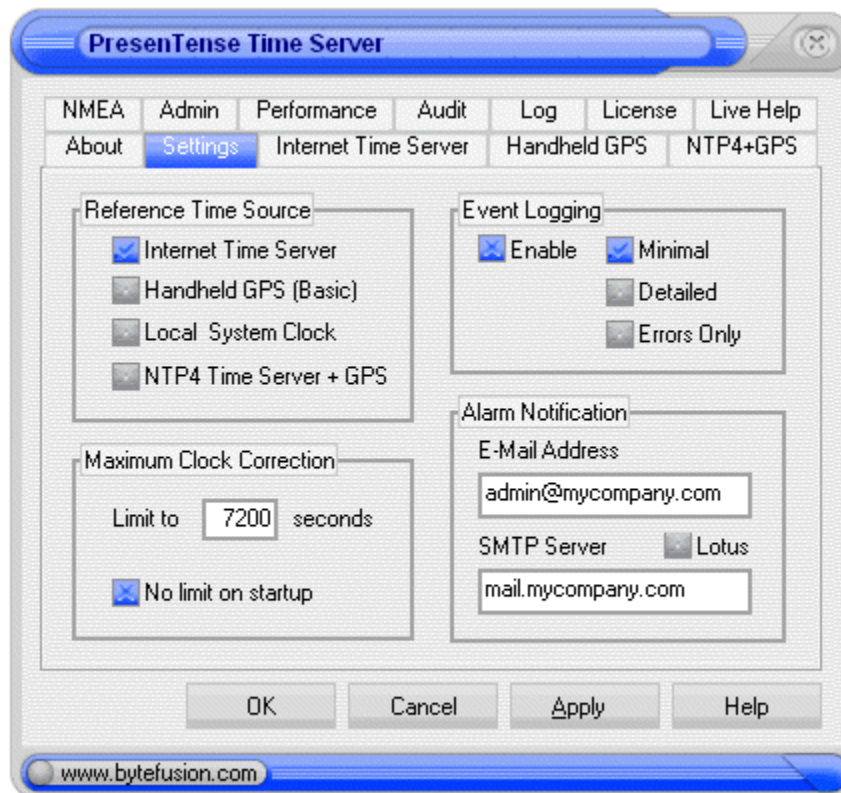
This panel indicates if PresenTense is running or stopped. Use the buttons to start and/or stop the service.

- **System Status**

This panel indicates the system time in 24-hour format. Use the buttons to conveniently access other control panel date/time functions.

NOTE : Making configuration changes through the control panel applet requires administrator privileges.

2.2 Settings Tab



- **Reference Time Source**

This option allows you to specify how PresenTense Server obtains accurate time. Possible options are :

a) Internet Time Server : Obtain accurate time from another time server on your local area network or the public internet from servers supporting common [time protocols](#). Selecting this option will enable the [Internet Time Server tab](#).

b) Handheld GPS (Basic) : Obtain accurate time from a Global Positioning System receiver. Support is included for GPS units featuring the NMEA 0183 protocol data via an RS-232 link (serial port) and emits the GPRMC sentence. Special algorithms permit obtaining accurate time from low cost, hand held GPS receivers. Selecting this option will enable the [Handheld GPS tab](#) and the [NMEA tab](#).

c) Local System Clock : Assume that the local system clock reflects accurate time. Most computer clocks are subject to a drift and thus do not remain synchronized to accurate time. However, some GPS clocks provide their own software to synchronize the local system to their GPS hardware. If you have one of these units and wish to use PresenTense server to distribute time to clients on your network, use the Local BIOS clock option.

d) NTP4 Time Server + GPS : Use free NTP4 plug-in DLL. The NTP4 plug-in is shipped with both the evaluation and the retail edition of PresenTense Server and supports common GPS receivers and NTP network time servers. See [NTP4 Plug-In](#).

- **Alarm Notification**

If you wish to be notified via email if PresenTense Server is unable to contact the time source, specify your email address and the mail server PresenTense should use for sending of messages. At this time, only the SMTP (Simple Mail Transfer Protocol) is supported. Check the **Lotus** box if you are running a Lotus Domino mail server. This feature enables a Lotus Domino work-around.

- **Event Logging**

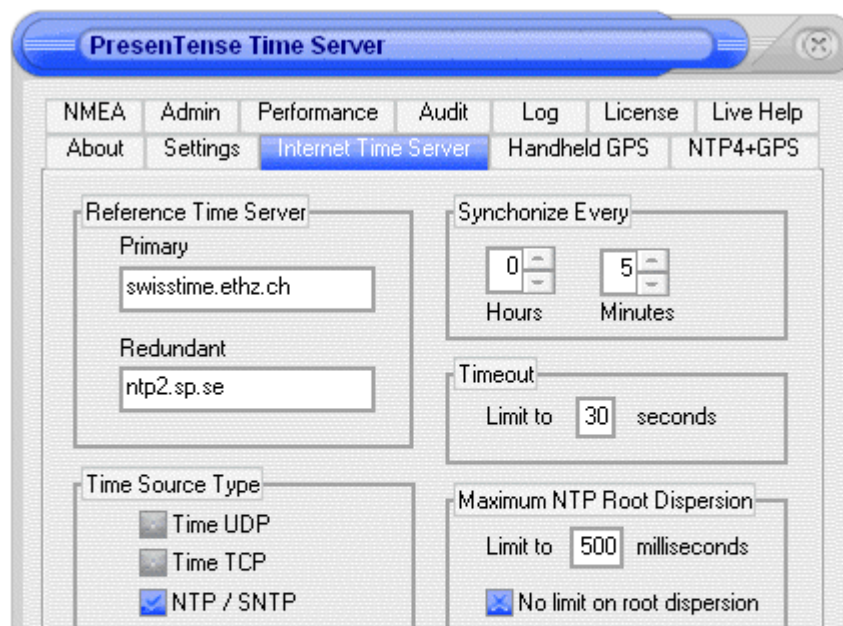
Use this option to enable/disable event logging the Windows NT application event log and to specify what message types qualify for logging

- **Maximum Clock Correction**

This option limits the number of seconds by which PresenTense Server will adjust its own system clock. This prevents serious time server errors from propagating across the network. It is recommended to override this option for initial clock adjustment to allow PresenTense to make adjustments as necessary on system startup.

NOTE : Making configuration changes through the control panel applet requires administrator privileges.

2.3 Internet Time Server Tab



- **Reference Time Server**

This option allows you to specify one primary and one redundant time server from which PresenTense Server derives its own time. You may specify either an IP address, hostname, or DNS name. If you elect to specify a hostname or DNS name, please ensure that name resolution is properly configured on your TCP/IP network.

Automatic fail-over from the primary to the redundant time source occurs if the name for the primary source cannot be resolved, or PresenTense Server times out waiting for a reply from the primary time source.

If you decide to synchronize to a time server on the internet rather than a corporate time server, ensure also that firewall rules permit the network traffic for the protocol you select.

- **Time Source Type**

This option allows you to specify the type of time source you wish to use. Your choice here depends mainly on the availability of services offered and your time keeping requirements. The recommended and by far most accurate option is NTP/SNTP.

- **Synchronize Every**

This option allows you to set the polling interval. How often PresenTense Server needs to poll its time source depends primarily on the clock drift of the local BIOS clock and your time keeping requirements. The default value is 15 minutes.

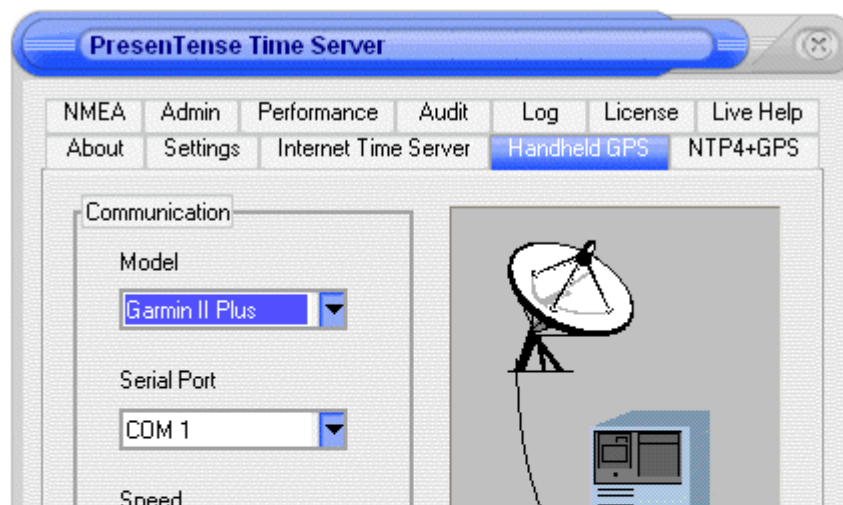
- **Timeout**

This option determines how long PresenTense Server waits for a reply from its time source. This value should not exceed the polling interval.

- **Maximum NTP Root Dispersion**

This option can be used to validate time stamps received from NTP time sources. Use it to reject time stamps which exceed a certain nominal error.

2.4 Handheld GPS Tab



- **Model**

Use this option to specify the model of your GPS receiver. Supported units at this time are:

- a) **Generic** : Support for generic NMEA GPS receivers that conform to the NMEA 0183 standard and emit the GPRMC or GPGGA sentence at regular, precise intervals of one or two seconds.
- b) **Garmin 38** : Hand held GPS receiver from Garmin Corporation.
- c) **Garmin II Plus** : Hand held GPS receiver from Garmin Corporation with support for external antenna connection. This option includes the **Garmin III Plus** and the **Garmin V**.
- d) **TripNav TN-200** : USB port powered GPS receiver from Rayming Corporation. Please note that long USB cables and operating system overheads in servicing the virtual COM port may result in signal offsets that are specific to your site. If you require a high degree of accuracy, we recommend the use of the [NTP4 plugin TripNav TN-200](#) driver and suggest you calibrate your GPS installation as illustrated in the section [NTP4 Plug-In Driver Calibration](#).

- **Serial Port**

Use this option to specify the serial port on your PC to which the GPS receiver is connected.

- **Speed**

At present, NMEA 0183 only supports 4800 Baud.

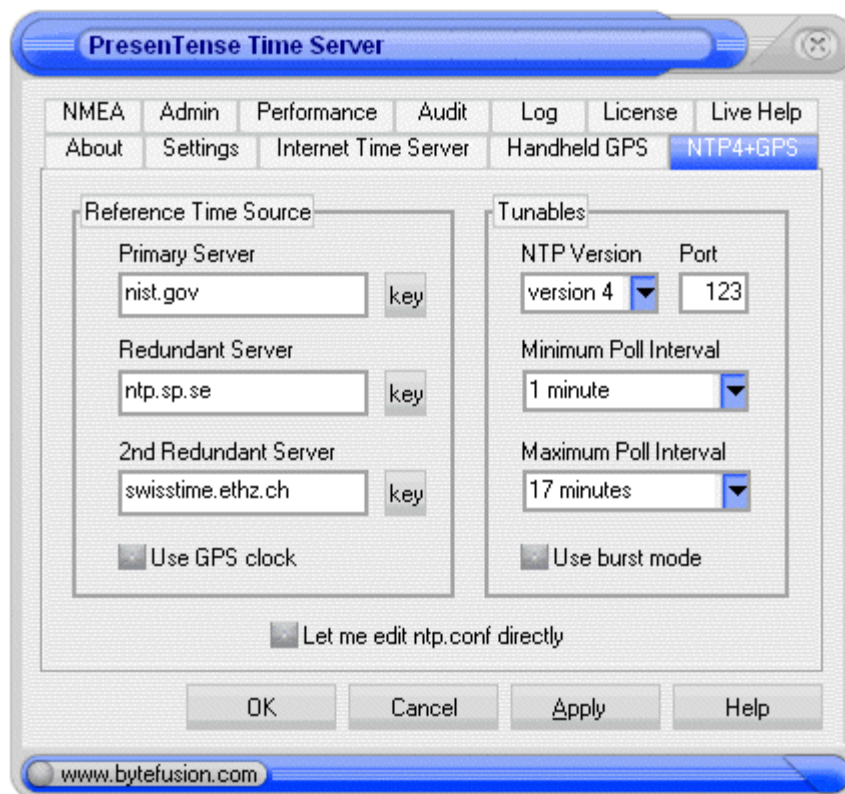
Trademarks

Microsoft Windows and Windows NT are either registered trademarks or trademarks of Microsoft Corporation. No associations are implied.

2.5 NTP4+GPS Tab

Starting with version 3.5, Presentense Server includes a plug-in which supports NTP version 3 & 4 according to [RFC 1305](#). This plug-in is distributed with Presentense Server free of charge. If you require the newest, industry standard version of NTP, the use of the NTP version 4 plug-in is recommended. To obtain the latest version of our NTP4 plug-in for Presentense Server free of charge at any time, kindly email us at support@bytefusion.com.

Presentense Server generates the configuration file NTP.CONF for the NTP4 plug-In automatically from the settings you specify or you can edit this configuration file directly. The latter is only recommended for advanced users. Please refer to the parameter list shown below for a detailed explanation of each parameter / setting.



- **Accurate time Source**

This option allows you to specify one primary and two redundant time servers from which the NTP4 plug-in derives its own time. You may specify either an IP address, hostname, or DNS name for each server. If you elect to specify a hostname or DNS name, please ensure that name resolution is properly configured on your TCP/IP network. If you decide to synchronize to a time server on the internet rather than a corporate time server, ensure also that firewall rules permit the network traffic for the protocol you select. Time servers specified here should support version 3 or 4 of the NTP protocol. Click the key icon to select an encryption key to be used to authenticate each server if you require secure NTP. Authentication uses the MD5 secure hash algorithm. See also [NTP4 Key Generator](#).

- **NTP Version**

This option allows you to specify the version number to be used for outgoing NTP packets. Possible values are **version 3** and **version 4**. Regardless of the value chosen here, the NTP plug-in will service clients of both version 3 and 4 when it itself is queried by other NTP hosts.

- **NTP Port**

This option allows you to specify an alternate NTP port. By convention, the NTP protocol operates on port 123 and in all but the most extraordinary circumstances you will want to use port 123 in order to retain compatibility with other NTP systems. Some circumstances, however, warrant changing the default NTP port. If you operate PresenTense Server on a local area network behind a firewall which blocks incoming traffic to privileged ports (<1024) and are unable or not permitted to reconfigure your firewall to suit, you might wish to reconfigure the NTP4 plug-in to operate on an alternate port. Be

advised that you will need to adjust any time client software to use the new port settings also.

- **Minimum Poll Interval**

This option allows you to specify the minimum poll interval for NTP messages. Typically the NTP4 plug-in will poll time servers at the rate specified as the minimum poll interval until the algorithm has achieved synchronization of the local system with its accurate time source(s).

- **Maximum Poll Interval**

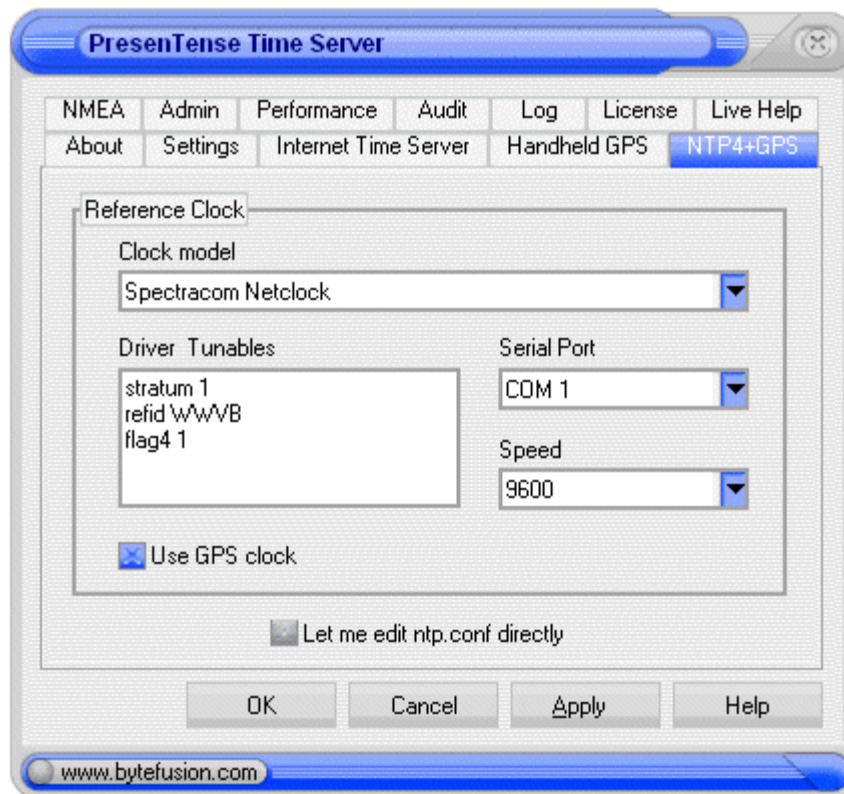
This option allows you to specify the maximum poll interval for NTP messages. After synchronization with its accurate time source(s) has been achieved, the NTP4 Plug-In may grow the interval between queries to its accurate time source(s) up to the maximum poll interval or as required to maintain synchronization.

- **Use Burst Mode**

This option tells the NTP4 plug-in to use eight packets instead of the usual one packet when polling a time server. When enabled, this option considerably increases timekeeping accuracy.

- **Use Hardware Clock**

This option allows you to switch to NTP4 reference clock support. Use this option if instead of obtaining accurate time over the network from another time server, you wish to obtain accurate time from a hardware device attached to the serial port (RS-232 interface) of your computer, such as a radio clock or GPS clock. Please note that this functionality duplicates some of PresenTense Server's built-in capabilities. For example, PresenTense Server has native support for NMEA enabled GPS clocks - the NMEA driver in the NTP4 plug-in duplicates this functionality. You are free to chose whichever you prefer.



- **Clock Model**

Use this option to specify the make and model of your hardware clock. Supported options also include "*Undisciplined local clock*" and "*Generic NMEA GPS receiver*". The "*Undisciplined local clock*" option advises the NTP4 plug-in to trust the time on your computer as accurate time - useful when clocks provide native software to synchronize the local system. The "*Generic NMEA GPS receiver*" option is useful if your GPS system or radio clock is not specifically listed here but supports the NMEA protocol. Note that the "*Undisciplined local clock*" option corresponds to PresenTense Server's native "*Local BIOS clock*" option on the [settings tab](#). Equally, the "*Generic NMEA GPS receiver*" NTP4 plug-in option corresponds to PresenTense Server's native "Generic" NMEA support option on the [GPS tab](#). Please refer to [NTP4 Plug-In Drivers](#) for a comprehensive list of hardware devices supported by the NTP4 plug-in.

- **Serial Port**

Use this option to specify the serial port on your PC to which the GPS or radio clock receiver is connected. Supported options are *COM1* through *COM4*.

- **Speed**

Use this option to specify the Baud rate of your GPS or radio clock. Supported options are 1200 Baud, 2400 Baud, 4800 Baud and 9600 Baud.

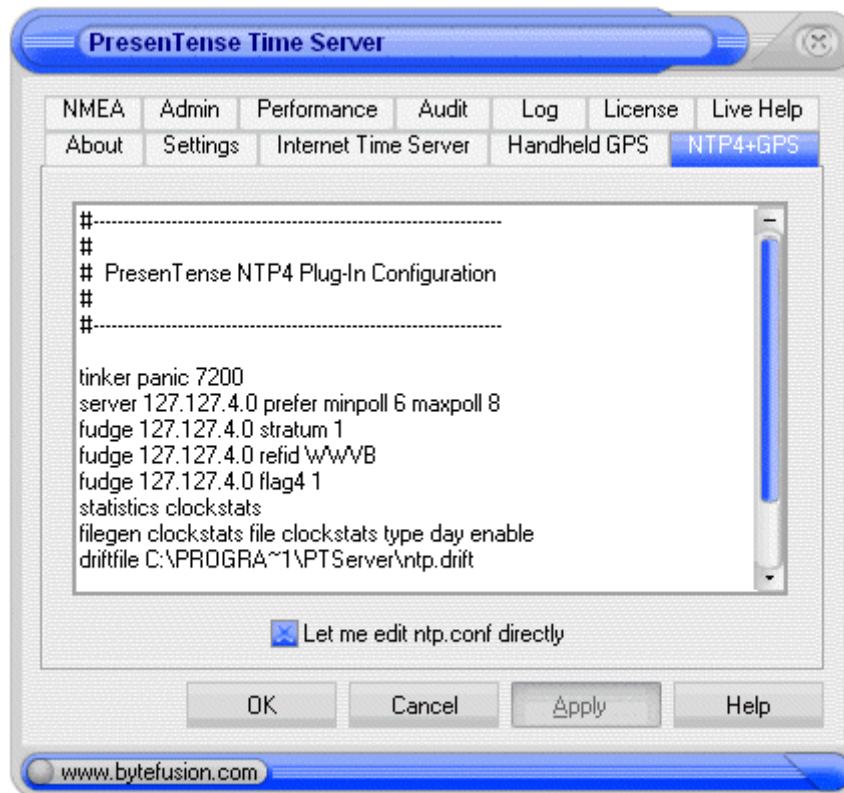
- **Driver Tunables**

Use this option to specify additional parameters to be passed directly to the hardware clock driver

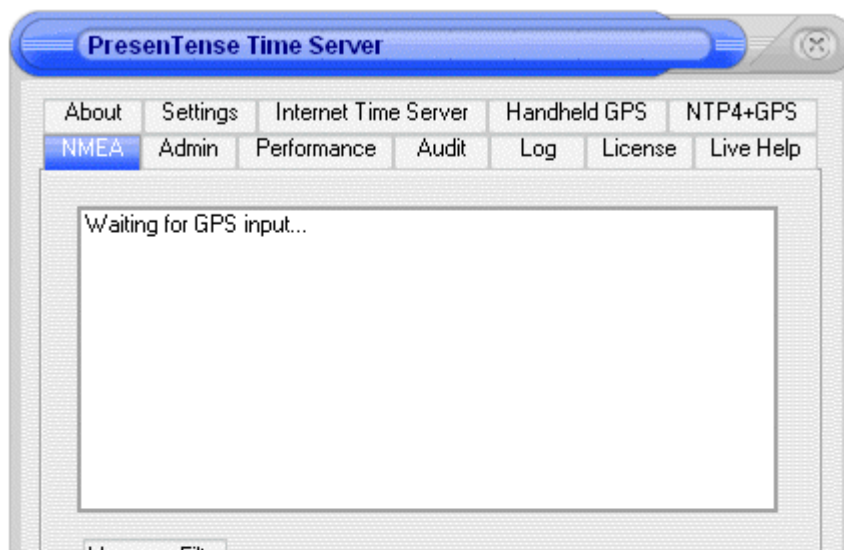
specified under "Clock Model". This option is recommended for advanced users only.

- **Let me edit ntp.conf directly**

This option allows you to switch to direct edit mode. In direct edit mode, you are free to compose your own NTP.CONF file. This option is recommended for advanced users only.

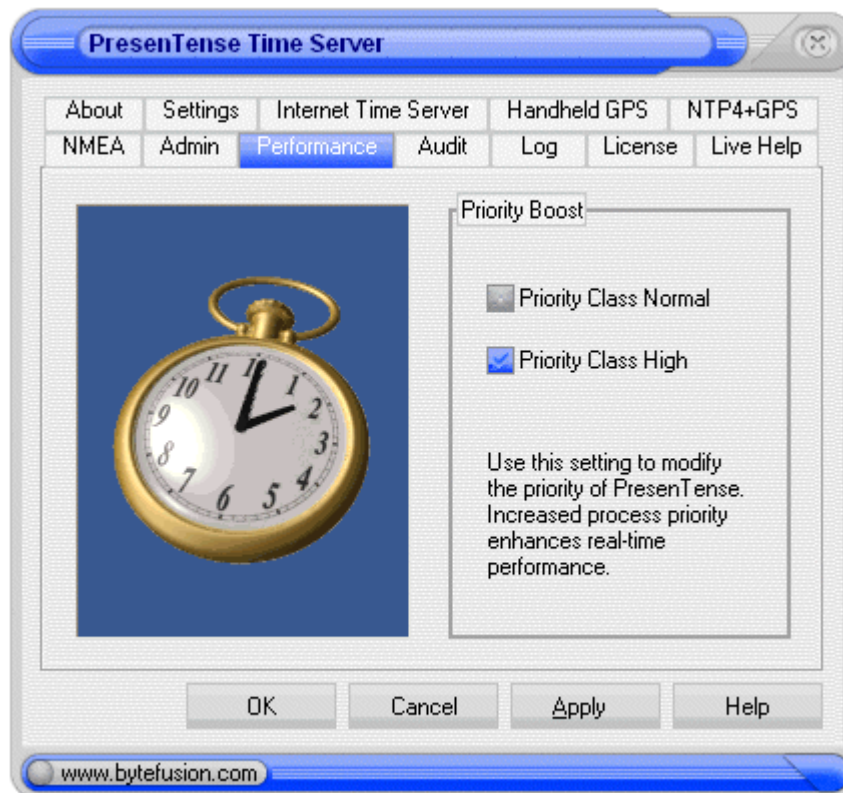


2.6 NMEA Tab



This window will capture NMEA output from your GPS receiver. Use it to verify connectivity to your GPS as well as COM port and Baud settings. If you cannot see any output from your GPS in this window, PresenTense Server will not be able to synchronize to your GPS.

2.7 Performance Tab



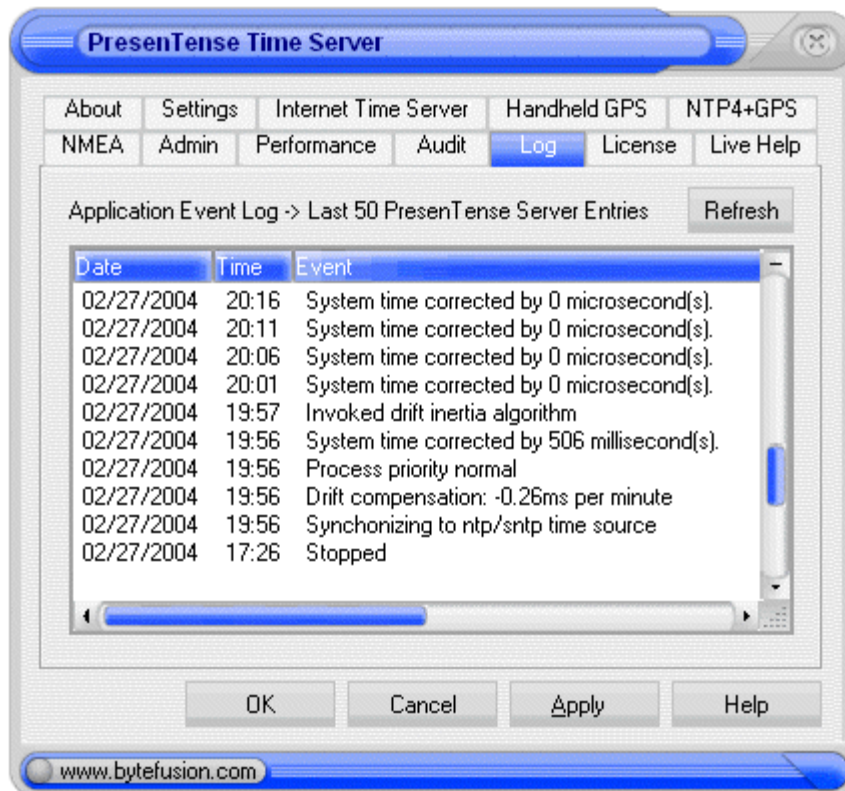
The **Priority Boost** setting allows you to increase the real-time priority of PresenTense in the context of the Microsoft Windows operating system. Tasks with a higher priority receive a greater share of CPU attention. If your system runs many CPU intensive applications, use the "Priority Class High" setting to enhance the accuracy of PresenTense Server.

Trademarks

Microsoft Windows is a registered trademark of Microsoft Corporation.

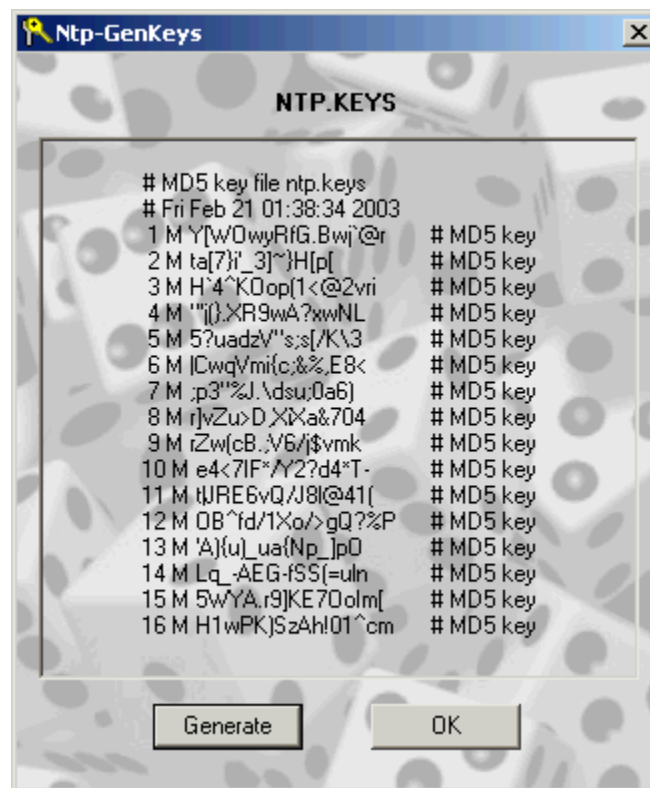
2.8 Log Tab

The Log tab gives you access to the operating system event log and always displays the last ten entries PresenTense Server has committed to the system event log. You need to have logging enabled on the [Settings tab](#) in order for this feature to work. The log tab entries are refreshed from the operating system event log whenever you switch to the log tab from any of the other tabs or when the **Refresh** button is clicked.



2.9 NTP4 Key Generator

The NTP4 key generator, **ntp-genkeys**, creates encryption keys for your server. These keys may be used by PresenTense client software to authenticate your server's identity. The keys are maintained in the file **NTP.KEYS** in the PresenTense Server application folder. You will need to distribute this file securely to workstations running PresenTense client and install the file in the PresenTense application folder. Newly generated keys will be read as soon the PresenTense server or PresenTense client service is restarted. **ntp-genkeys** is a companion program of the [NTP4 Plug-In](#). The newest version of ntp-genkeys may be obtained free of charge by writing to support@bytefusion.com.



2.10 NTP4 Plug-In Drivers

Besides synchronizing to other NTP servers via a local area or wide area network, the [NTP4 plug-in](#) also allows you to synchronize your computer to a number of hardware clocks, including [GPS satellite receivers](#) and radio clocks. A number of physics laboratories in various countries operate low frequency radio transmitters with ranges spanning continents. These include :

- [WWV / WWVB](#), National Institute of Standards and Technology, Ft. Collins, Colorado, USA. The signal transmitted from Ft. Collins can be received throughout most of the continental United States as well as throughout most of Canada.
- [DCF77](#), National Institute of Natural and Engineering Sciences, Germany. The DCF77 signal can be received throughout most of western Europe, the British Isles, and is reported to be received at night as far as India, Israel and Yemen.
- [JJY](#), Communication Research Laboratory (CRL), Japan. CRL broadcasts [Japan Standard Time](#) via long wave radio.

The PresenTense [NTP4 plug-in](#) contains drivers for these and more. A comprehensive list of drivers supported by the NTP4 plug-in is shown in the table below.

Driver	Type	Coverage
Undisciplined local clock	Local BIOS	n.a.
Generic NMEA GPS receiver	GPS	global
Arbiter	GPS	global
Ultralink	WWVB	US / Canada
NeoClock	DCF77	Germany / Europe
Spectracom	WWVB	US / Canada
Hopf 6870 & Compatibles	DCF77 / GPS	Germany / global
Hopf DCF77/GPS PCI-Bus Receiver	DCF77 / GPS	Germany / global

See also [The "GPS" Tab](#) for a list of low cost, hand held GPS receivers supported by the native Presentense Server drivers.

2.11 Undisciplined Local Clock

The *undisciplined local clock* driver assumes that the local system clock reflects accurate time. This is useful when, in the absence of an external time source, you simply wish for all hosts on a network to agree on time. In this case, you may elect one host to act as server using the *undisciplined local clock* driver and synchronize all other hosts to this server. Also, some GPS clocks provide their own software to synchronize the local system to their GPS hardware. If you have one of these units and wish to use Presentense server to distribute time to clients on your network, you can use the *undisciplined local clock* option. This NTP4 plug-in option corresponds to Presentense Server's native Local BIOS Clock option under [Settings](#).

2.12 NTP4 Plug-In Driver Calibration

GPS and radio clocks communicating with your server via a serial port connection are subject to serial port delays and other operating system dependent propagation errors. In some cases long antenna cables may introduce further signal propagation delays. For this reason it is important to calibrate your clock for use with your time server.

NTP driver calibration is easiest achieved using the [NTP4 Query Command Line Tool](#).

```

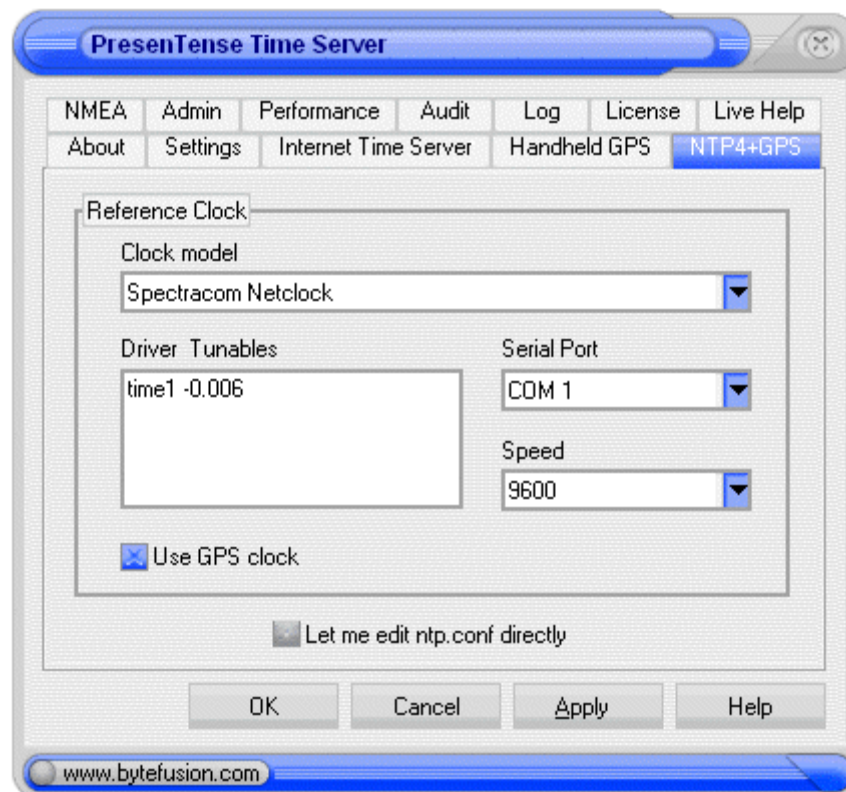
C:\Program Files\PTServer>ntpq -p
      remote           refid       st t when poll reach  delay  offset  jitter
=====
SPECTRACOM<0> .WWUB.           0 -  42  64   7   0.000  -4.409 996.182
192.168.1.19   .GPS.             1 -  40  64   7   2.921   3.516  4.607
time1.stupi.se .PPS.             1 u  58  64   7  340.338  19.392 16.967
hora.cs.tu-berl .PPS.             1 u  61  64   7  390.191  50.770 19.164

C:\Program Files\PTServer>ntpq -p
      remote           refid       st t when poll reach  delay  offset  jitter
=====
*SPECTRACOM<0> .WWUB.           0 -  17  64  77   0.000   6.312  3.631
+192.168.1.19   .GPS.             1 -  12  64  77   2.035  13.182  0.547
+time1.stupi.se .PPS.             1 u  33  64  77  330.321  38.231  7.344
-hora.cs.tu-berl .PPS.             1 u  38  64  77  340.068  49.315  8.496

C:\Program Files\PTServer>

```

The screenshot above shows a test system running a [Spectracom 8189 clock](#) as primary time source and a local area network NTP server 192.168.1.19 as secondary time source. Two public NTP servers on the internet facilitate external verification of test results. The two servers on the internet are subject to round trip delays on the order of 300ms as well as substantial network jitter and therefore exhibit greater offsets with respect to the local system clock than the local area network NTP server.



You can use the **time1** driver parameter to compensate for signal delays and other propagation errors. The example shown above illustrates how to compensate for a nominal 6 millisecond error. The final screenshot shown below shows both time sources in agreement to within 2.5 milliseconds. Depending on your requirements you might wish to calibrate to within closer specifications.

```

Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\>cd program files
C:\Program Files>cd ptserver
C:\Program Files\PTServer>ntpq -p
      remote          refid      st t when poll reach  delay  offset  jitter
=====
*SPECTRACOM(0)      .WWVB.          0 - 61  64  377   0.000  -1.839   0.008
+192.168.1.19      .GPS.           1 u 22  64  377   1.975   4.194   1.473

C:\Program Files\PTServer>ntpq -p
      remote          refid      st t when poll reach  delay  offset  jitter
=====
*SPECTRACOM(0)      .WWVB.          0 - 46  64  377   0.000  -1.548   0.067
+192.168.1.19      .GPS.           1 u 56  64  377   2.008   1.080   4.882

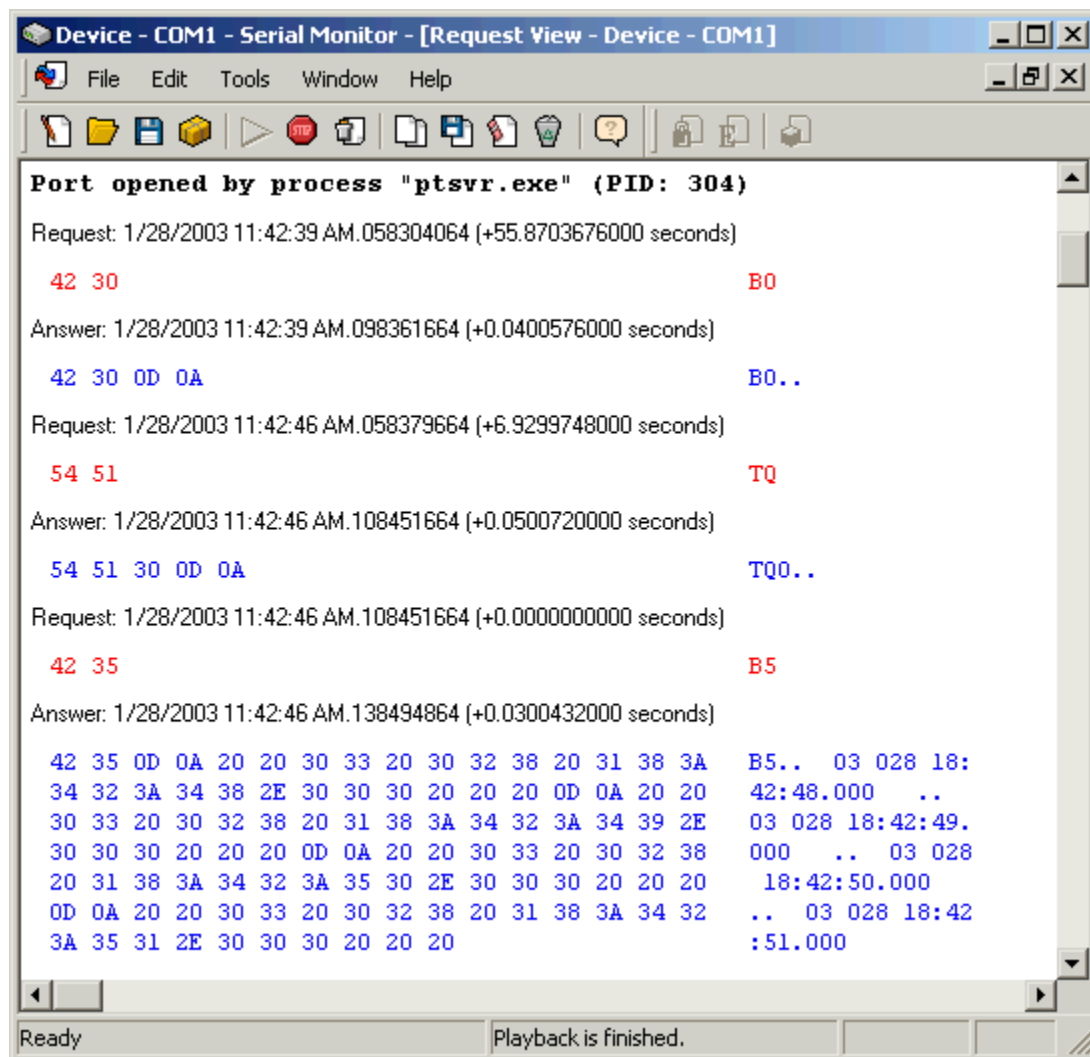
C:\Program Files\PTServer>

```

2.13 NTP4 Plug-In Driver Troubleshooting

If your GPS or radio clock is listed as supported or you have a clock model which you feel is compatible with a clock listed under [NTP4 Plug-In Drivers](#) and your clock is unable to communicate with Presentense Server, kindly let us know. Please submit a detailed problem report by email to support@bytefusion.com listing details of both the operating system and service pack information of your computer as well as the exact model number and description of your GPS or radio clock.

Additionally, we request that you submit a serial port capture, showing the interaction between Presentense Server and your clock. This will confirm to us that your PC hardware is operating correctly and that your GPS or radio clock is properly connected to your PC. The serial port capture will also help us troubleshoot the interaction between your clock and Presentense Server. We request that you submit your serial port capture as a ".lgs" file as produced by the [HHD Serial Port Monitor](#). The HHD Serial Port Monitor is shareware and available for download from [HHD Software](#). A sample serial port capture showing the interaction between Presentense server and an [Arbiter GPS receiver](#) is displayed below.



3 Hardware

3.1 TripNav TN-200

With PresenTense Server and the [TripNav TN-200](#) you can build your own network enabled, low cost, GPS master clock to synchronize all your computers to atomic time via the global positioning satellite system. The [TripNav TN-200](#) is easy to use and requires only a standard USB port, no external power and features a water resistant housing for installation inside or outside of buildings. Simply add any off-the-shelf USB extension cable to suit your own antenna cable length requirements.

The TripNav TN-200 features a built in 12 channel GPS antenna, is WAAS capable for enhanced precision, and achieves an accuracy of 1 microsecond to UTC. Find out more about the TripNav TN-200 at [Rayming Corporation's](#) home page www.rayming.com.

Installation Notes : The TripNav TN-200 software installs a virtual COM port, typically on COM 5, which relays the NMEA output from the TN-200 GPS unit to PresenTense Server. Please note that long USB cables and operating system overheads in servicing the virtual COM port may result in signal offsets that are specific to your site. You may need to calibrate your GPS installation via the offset calibration factor as illustrated in the section [NTP4 Plug-In Driver Calibration](#). See also [PresenTense NTP Auditor](#)

for additional information on NTP calibration.

Driver tunables :

Tunable	Values/Units	Description
time1	seconds	offset calibration factor
stratum	0-15	server stratum, default 0
refid	one to four letters	server identifier, default GPS

Note: TripNav TN-200 specifications are subject to change. Please visit [Raymarine Corporation](http://www.raymarine.com) for more information.

3.2 Generic NMEA GPS

The "Generic NMEA GPS receiver" option is useful if your GPS system or radio clock is not specifically listed here but supports the NMEA protocol. In order to operate NMEA GPS clocks with the PresenTense Server NTP4 plug-in, your GPS must conform to the NMEA 0183 standard and emit the GPRMC or GPGLA sentence once per second. The format of the GPRMC sentence can be seen on the [NMEA tab](#). Please note that the "Generic NMEA GPS receiver" NTP4 plug-in option corresponds to PresenTense Server's native "Generic" NMEA support option on the [GPS tab](#).

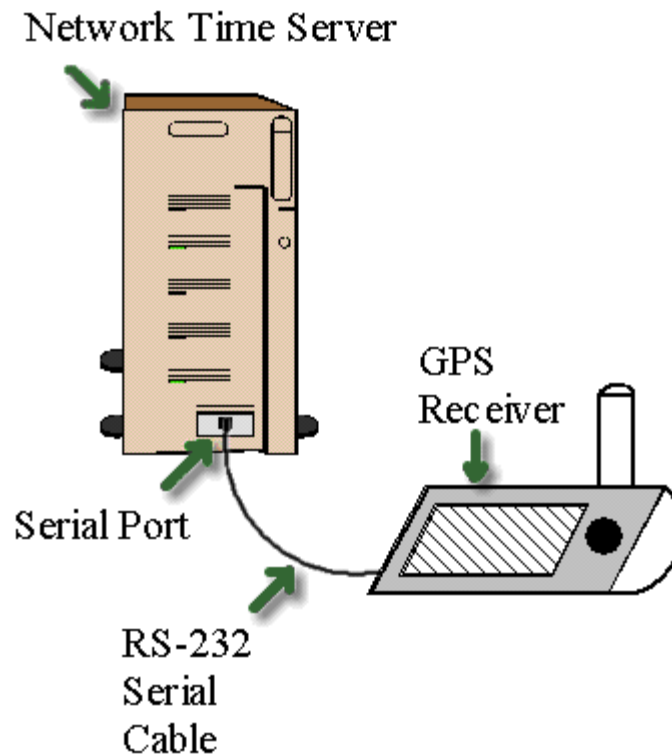
Driver tunables :

Tunable	Values/Units	Description
time1	seconds	offset calibration factor
stratum	0-15	server stratum, default 0
refid	one to four letters	server identifier, default GPS

3.3 Hand Held GPS Receivers

Many hand held GPS receivers deliver time information accurate to within 3 to 70 milliseconds via a serial cable which can be connected to your PC's RS-232 serial port. Many GPS clocks advertise accuracy to within the nanosecond range. PresenTense Server supports both hand held GPS receivers as well as GPS clocks that communicate via the NMEA 0183 protocol.

The benefits of hand held GPS receivers are cost effectiveness and availability - they may be purchased at many outdoor outfitters and office supply stores in every major city. Depending on the construction of the building in which you intend to operate your time server and GPS receiver, you might require an external antenna for your GPS to guarantee availability of satellite signals. See [Recommended Parts List](#)



3.4 GPS Parts List

- **GPS Receiver**

Garmin GPS III Plus, or
Garmin GPS V, or
Garmin GPS II Plus (discontinued)

<http://www.garmin.com/products/gpsIIIp/spec.html>
<http://www.garmin.com/products/gps5/spec.html>

Please see <http://www.garmin.com/products/gpsIIIp/spec.html>,
<http://www.garmin.com/products/gps5/spec.html>, <http://www.garmin.com/products/gpsII> for detailed
information. At the time of writing, most Wal-Mart stores across North America and many outdoor
outfitters stock the Garmin GPS III Plus. Please see <http://www.walmart.com> for more information. If
you wish to purchase the GPS II Plus directly from Garmin Corporation, simply phone 1-800-800-
1020 or email sales@garmin.com.

- **Data Link Cable**

In order to connect your GPS to the serial port of a personal computer you will need an interface
cable. Further, if you wish to operate the unit using a DC power supply instead of using "AA"
batteries, you will need a PC interface cable with a 12 Volt cigarette lighter adapter. Please see
<http://shop.garmin.com/> for detailed information.

The Garmin part number for the PC data interface cable is 010-10141-00

The Garmin part number for the PC interface cable with cigarette lighter adapter is 010-10165-00.

You may also visit <http://www.globalmart.com/gps/garmin/accessories/acc2/cables/> or visit <http://www.globe-mart.com/> and search for "Garmin PC Interface Cable Cigarette Lighter Adapter".

- **External Antenna**

Your GPS needs to have an unobstructed view of the sky in order to receive satellite signals. The roofs of wooden buildings sometimes allow the satellite signal to pass through. In some cases, simply placing the GPS receiver near a window will result in adequate signal strength. In all other cases, an external antenna must be mounted outdoors with a clear view towards the sky and connected to your GPS receiver. We recommend the external antennas GA-29 (Garmin part number 010-00174-00) or GA-26C (Garmin part number 010-10052-04). Please see <http://shop.garmin.com/> for detailed information.

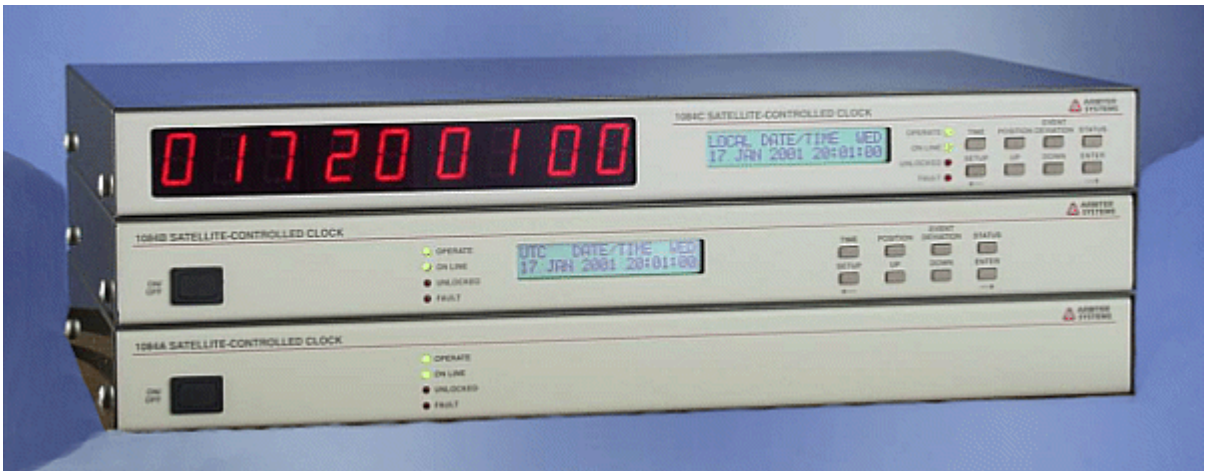
3.5 Arbiter

[Arbiter](#) sells a range of global positioning system satellite clock receivers. This driver supports model 1084, model 1088, model 1092 and model 1093. To operate these units with the PresenTense Server NTP4 plug-in, turn off the daylight savings time feature using the **D0** command and turn on initiate UTC broadcasts using the **BU** command. The NTP4 plug-in initiates communication with the clock by sending a time quality query (**TQ**). Once an echo reply including the time quality character has been received from the clock, the NTP4 plug-in turns on pulse per second timestamps by sending a **B5** command to the clock. The Arbiter clock in turn responds by broadcasting a "**format b5**" time indication once every second.

[Arbiter 1088 Satellite Controlled Clock](#)



[Arbiter 1084 A,B, & C Satellite Controlled Clocks](#)



[Arbiter 1092 A,B, & C Satellite Controlled Clocks](#)



[Arbiter 1093 A,B, & C Satellite Controlled Clocks](#)



Images reproduced with permission from Arbiter Systems Inc.

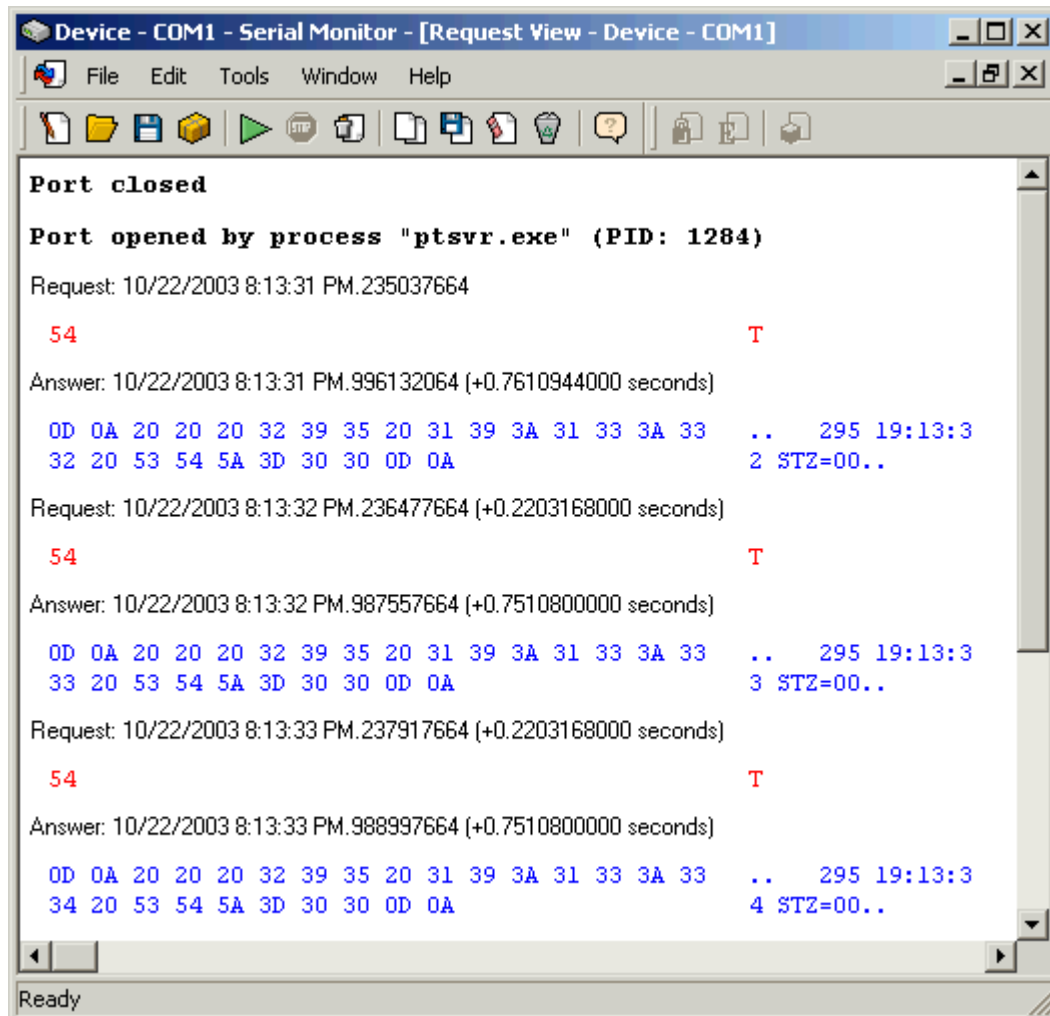
Driver tunables :

Tunable	Values/Units	Description
time1	seconds	offset calibration factor
stratum	0-15	server stratum, default 0
refid	one to four letters	server identifier, default GPS

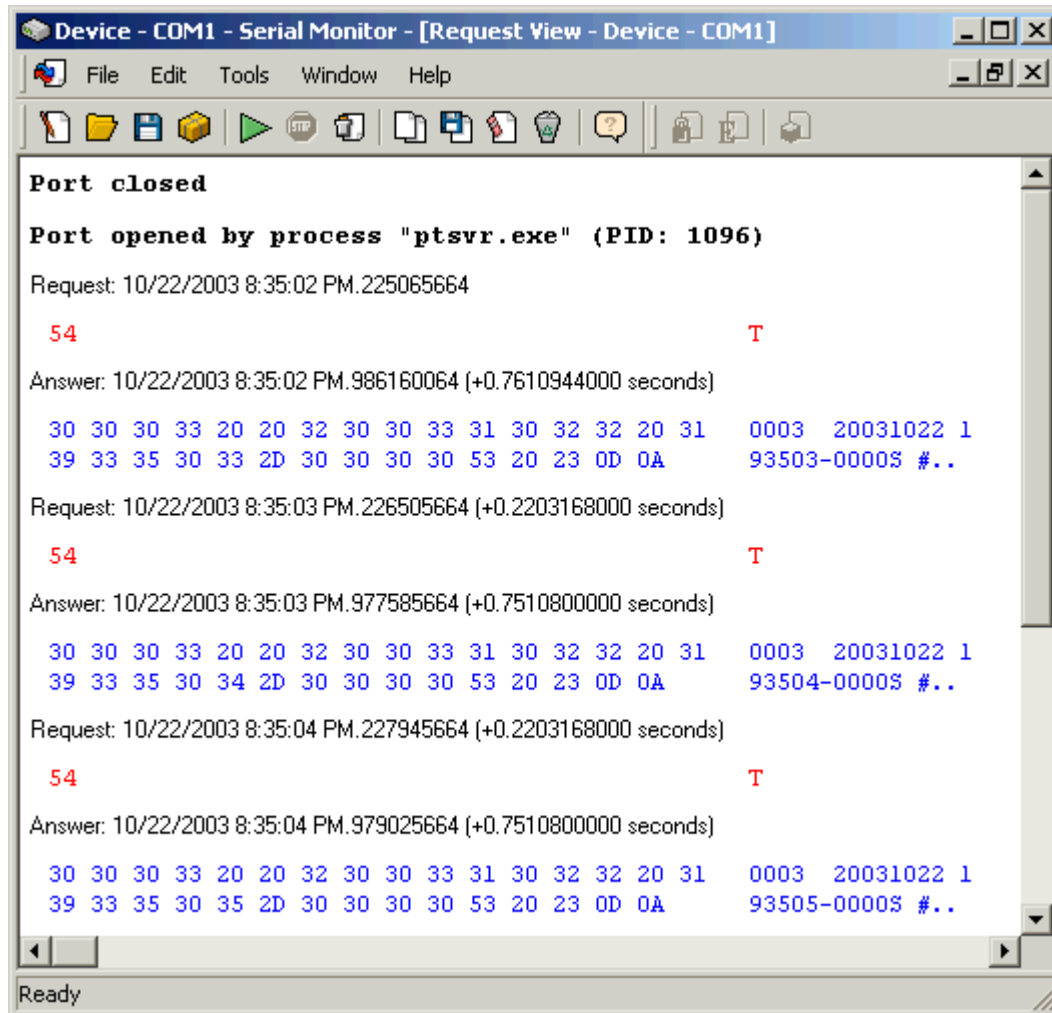
3.6 Spectracom Clock Formats

Spectracom Netclocks support a variety of data formats. The NTP4 Plug-In Driver supports format zero and format three. The screenshots below show the real-time interaction between PresenTense Server and a Spectracom Netclock using data format zero and data format three. This data is provided for reference and troubleshooting purposes only. See [NTP4 Plug-In Driver Calibration](#) and [NTP4 Plug-In Driver Troubleshooting](#) for more information.

- Data Format Zero



- Data Format Three



3.7 Spectracom

[Spectracom](#) manufactures both GPS and radio clock receivers. This driver supports the following Spectracom clock models : 8182 (WWVB), 8183 (GPS), 8183A (GPS), 8183ES (GPS) and 8189 (GPS). The driver requires the Spectracom clock to be configured to employ data format 2 using 24 hour time, no automatic daylight savings time correction, and 9600 bps. Some Spectracom clock models feature DIP switches - other models are configured via a dedicated telnet or serial port connection.

Spectracom 8189



Spectracom 8182



Spectracom 8283

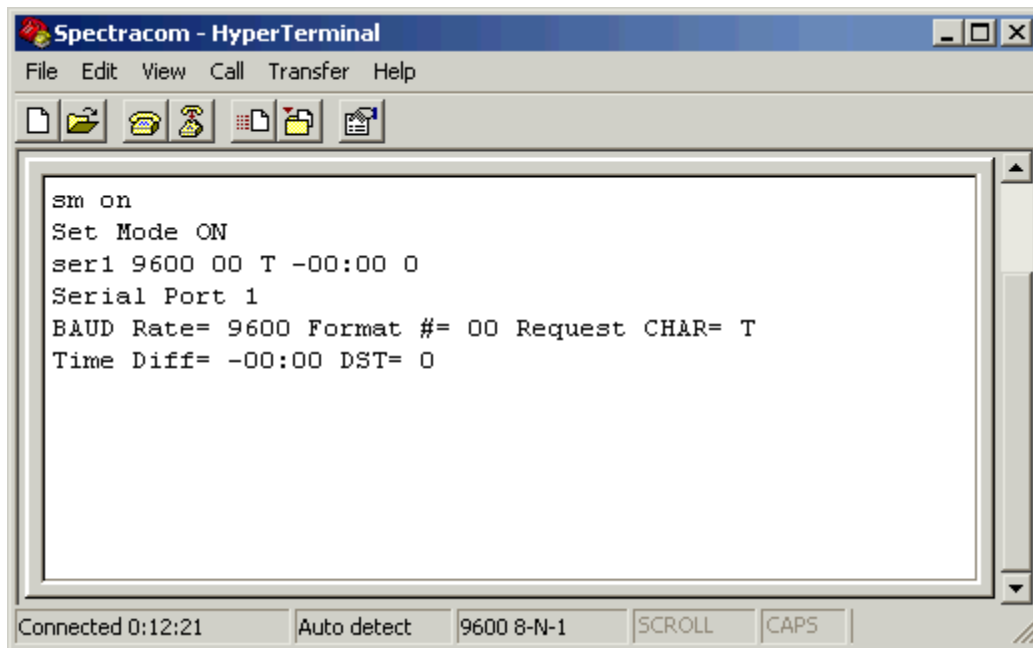


Images reproduced with permission from Spectracom Corporation.

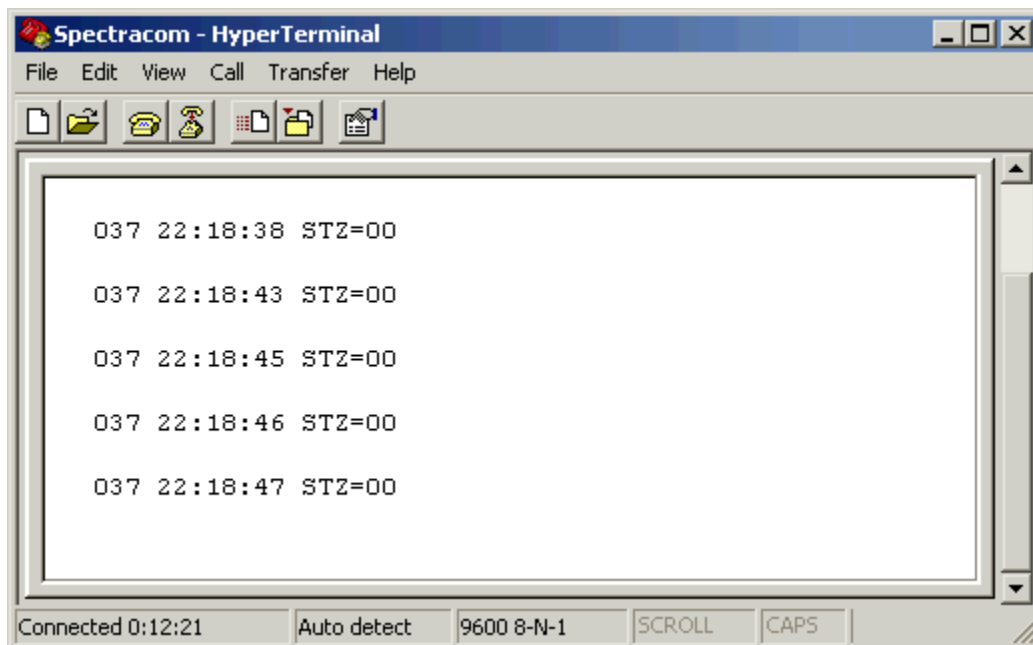
Configuration:

Connect a serial cable from your computer to the RS-232 setup port on the Spectracom 8189 and open a HyperTerminal session using the following parameters : 9600 bits per second, 8 data bits, no parity and flow control Xon / Xoff. This will allow you to configure the operation of your Spectracom clock. Type "**sm on**" and hit enter to enter setup mode for the serial interface. Then type "**ser1 9600 00 T -**

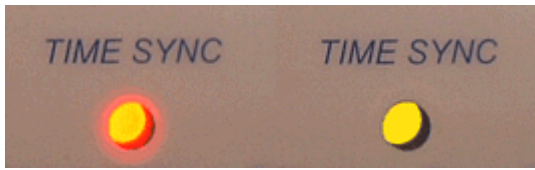
00.00 0" and hit enter to configure the clock to respond to the polling character "T" using 9600 bits per second. Close the HyperTerminal session, saving the operating parameters for your Spectracom clock.



You should now be able to verify operation of your clock by disconnecting the serial cable from the RS-232 setup port and connecting it to the RS-232 communication (COM) port. Re-open your HyperTerminal session and repeatedly hit capital "T" on your keyboard. This should prompt the clock to reply with one time stamp for each key press as shown below. Close your HyperTerminal session.



When your Spectracom clock is synchronized to satellites in geostationary orbit, the time sync LED on the front panel will change color from red to green as shown below. You are now ready to start synchronizing your computer to your satellite clock using PresenTense Server.



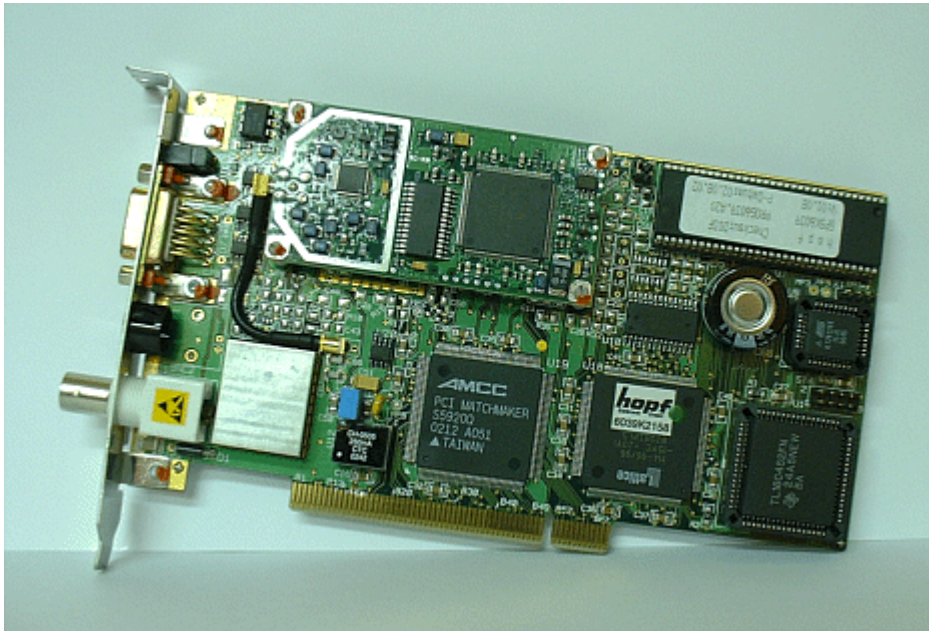
Driver tunables :

Tunable	Values/Units	Description
time1	seconds	offset calibration factor
stratum	0-15	server stratum, default 0
refid	one to four letters	server identifier, default WWVB

3.8 Hopf DCF77/GPS PCI-Bus Receiver

[Hopf Elektronik](http://www.hopf-electronik.com) manufactures GPS satellite clocks as well as radio clocks designed for the DCF77 time signal broadcast by the National Institute of Natural and Engineering Sciences in Germany. This driver supports the Hopf Elektronik clock model 6039 PCI board. Both, the DCF and GPS variants are supported. You may visit Hopf Elektronik on the web at <http://www.hopf-time.com> , or email technical support at info@hopf.com.

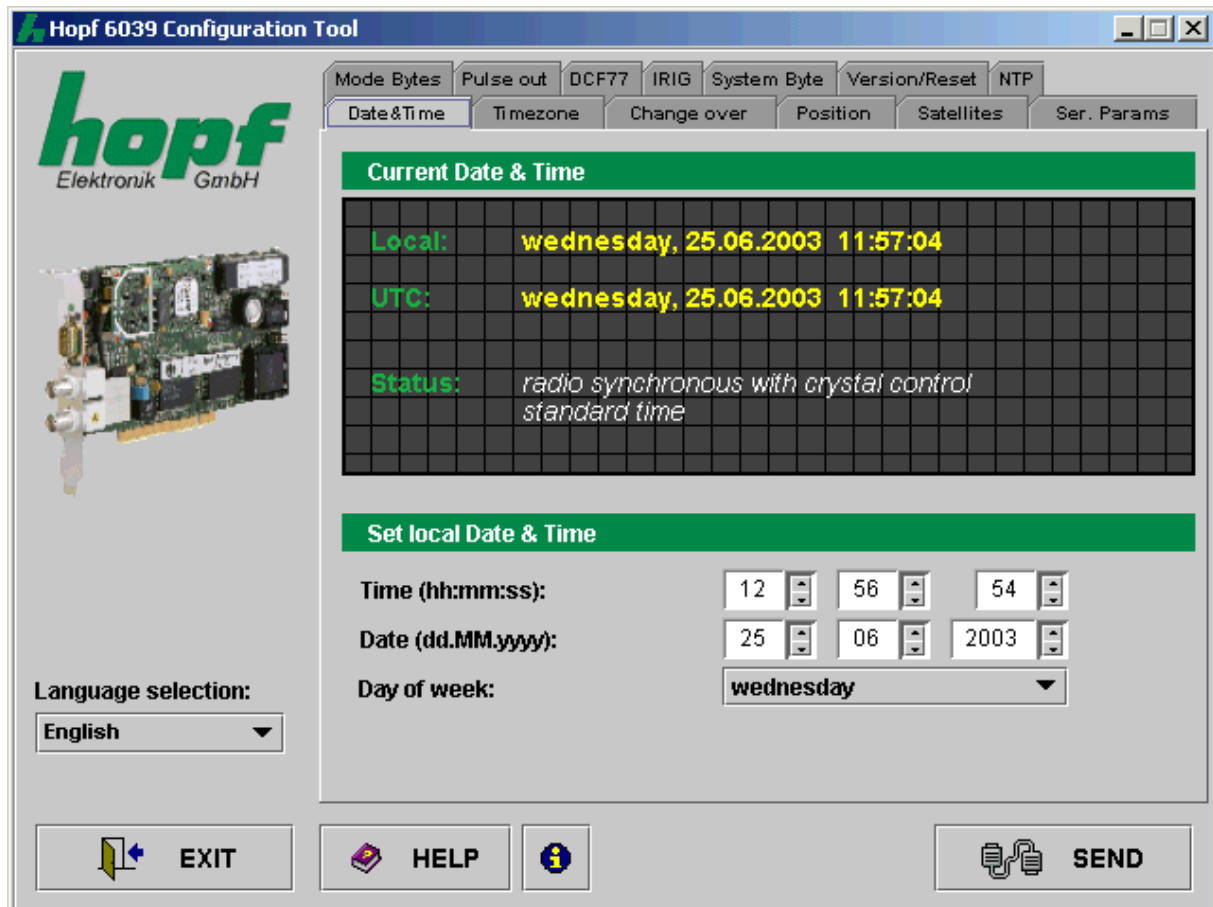
Hopf 6039 PCI-Bus Receiver



Hopf 6039 Operation

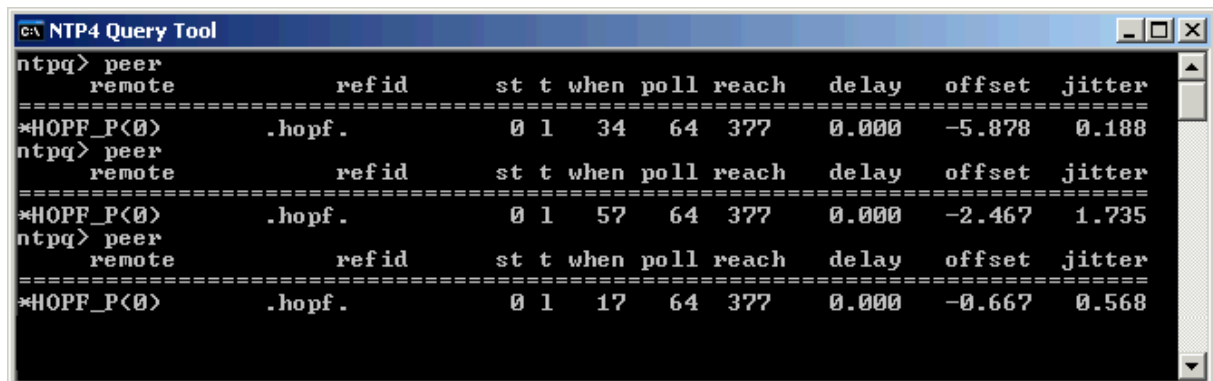
When the unit is slotted into an internal PCI bus of a personal computer and connected to its external

GPS antenna, the Hopf 6039 diagnostic software will display the synchronization status of the unit on the **Date & Time** tab in the **Status** field. Satellite signal strength is displayed on the **Satellites** tab. When the Hopf 6039 clock has successfully synchronized to satellite time or DCF77 time, the status indication changes to "**clock is radio synchronous with crystal control standard time**" as shown below. If you wish Presentense Server to use the timestamps obtained from your Hopf clock prior to this stage, see "**flag1**" under **Driver Tunables** at the bottom of this page.



Hopf 6039 Benchmark Tests

The Hopf 6039 PCI bus board has performed extremely well in benchmark tests with sub-millisecond accuracy.



Driver tunables :

Tunable	Values/Units	Description
time1	seconds	offset calibration factor
refid	one to four letters	server identifier, default GPS or DCF
flag1	0 or 1	Use value 1 to force synchronization when only "crystal driven".

3.9 Hopf 6870 & Compatibles

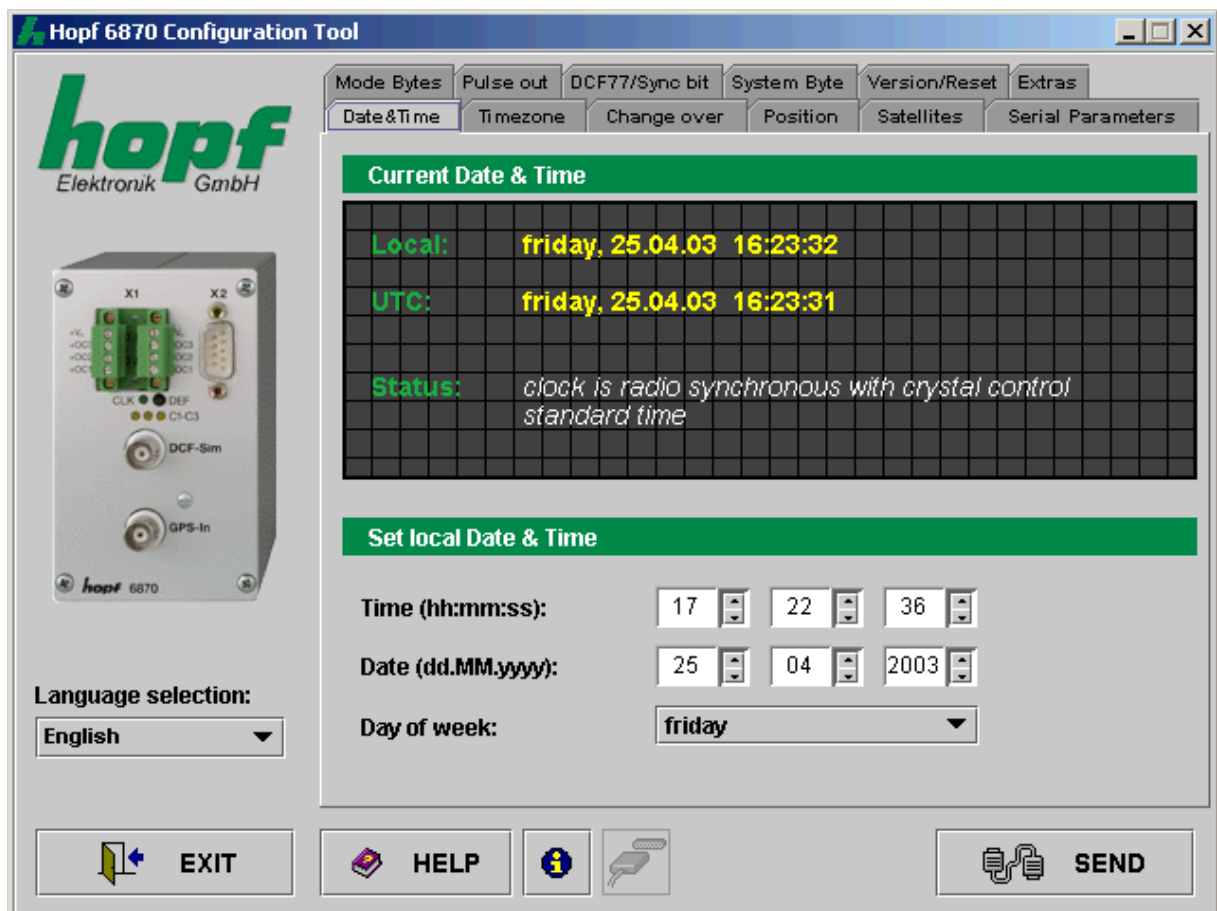
[Hopf Elektronik](http://www.hopf-time.com) manufactures GPS satellite clocks as well as radio clocks designed for the DCF77 time signal broadcast by the National Institute of Natural and Engineering Sciences in Germany. This driver supports the Hopf Elektronik clock model 6870 and compatible clocks. These rack mount units are designed to connect to a time server on a personal computer via the RS-232 serial port interface. You may visit Hopf Elektronik on the web at <http://www.hopf-time.com> , or email technical support at info@hopf.com.

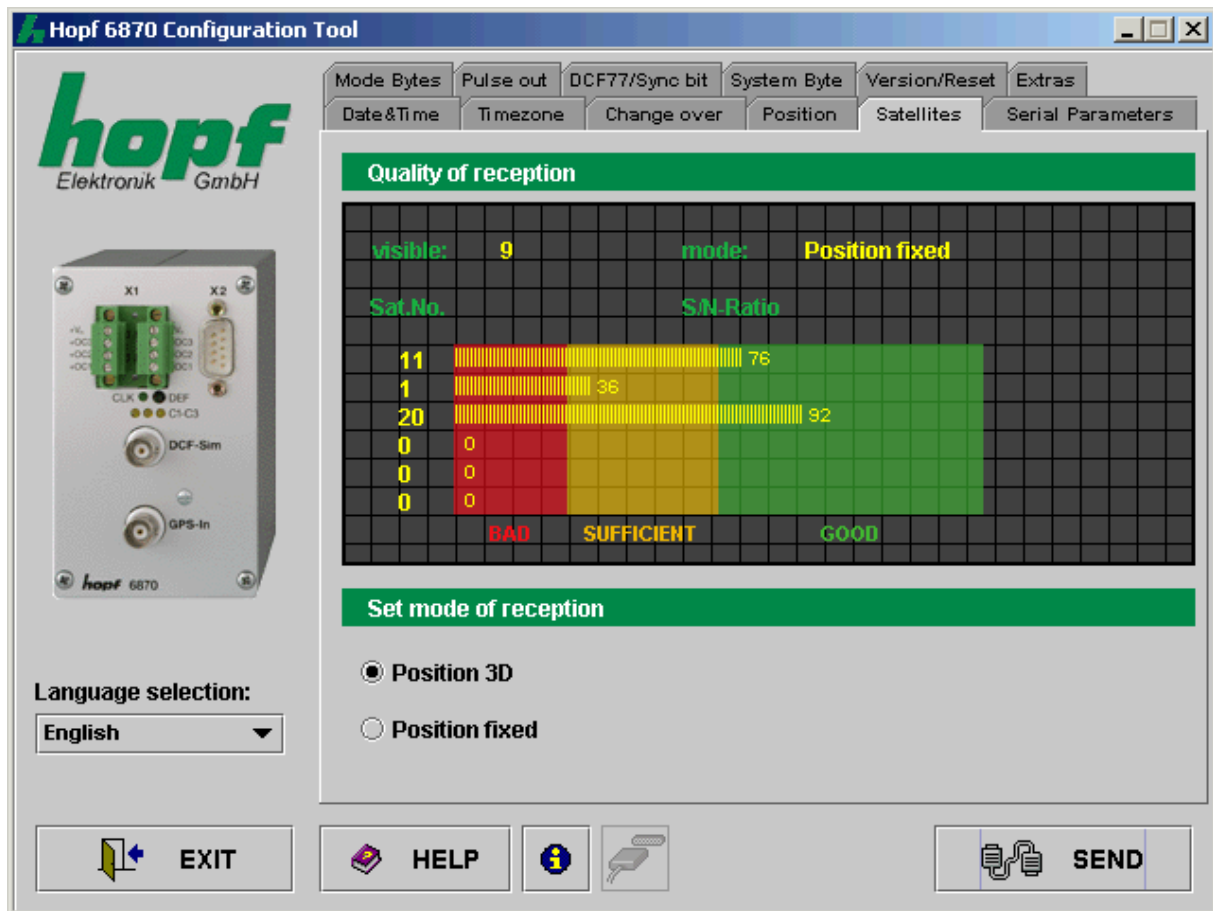
Hopf 6870



Hopf 6870 Operation

When the unit is connected to a personal computer via the RS-232 interface and connected to its external GPS antenna, the Hopf 6870 diagnostic software will display the synchronization status of the unit on the **Date & Time** tab in the **Status** field. Satellite signal strength is displayed on the **Satellites** tab. When the Hopf 6870 clock has successfully synchronized to satellite time or DCF77 time the status indication changes to "**clock is radio synchronous with crystal control standard time**". If you wish PresenTense Server to use the timestamps obtained from the Hopf clock prior to this stage, see "**flag1**" under **Driver Tunables** at the bottom of this page.





Driver tunables :

Tunable	Values/Units	Description
time1	seconds	offset calibration factor
refid	one to four letters	server identifier, default GPS or DCF
flag1	0 or 1	Use value 1 to force synchronization when only "crystal driven".

3.10 Ultralink

[Ultralink](#) manufactures precision radio time receivers. Ultralink clocks use the WWVB signal broadcast by the National Institute of Standards and Technology, Ft. Collins, Colorado, USA. The signal transmitted from Ft. Collins can be received throughout most of the continental United States as well as throughout most of Canada. This driver supports Ultralink clock models 320, 330, 331 and 332.

Driver tunables :

Tunable	Values/Units	Description
flag1	0 or 1	Use 1 for model 320. Don't set this flag if DIP switch 2 is set on model 33x series clocks.

4 RFC 868

4.1 RFC 868

Network Working Group
Request for Comments: 868

J. Postel - ISI
K. Harrenstien - SRI
May 1983

Time Protocol

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement a Time Protocol are expected to adopt and implement this standard.

This protocol provides a site-independent, machine readable date and time. The Time service sends back to the originating source the time in seconds since midnight on January first 1900.

One motivation arises from the fact that not all systems have a date/time clock, and all are subject to occasional human or machine error. The use of time-servers makes it possible to quickly confirm or correct a system's idea of the time, by making a brief poll of several independent sites on the network.

This protocol may be used either above the Transmission Control Protocol (TCP) or above the User Datagram Protocol (UDP).

When used via TCP the time service works as follows:

- S: Listen on port 37 (45 octal).
- U: Connect to port 37.
- S: Send the time as a 32 bit binary number.
- U: Receive the time.
- U: Close the connection.
- S: Close the connection.

The server listens for a connection on port 37. When the connection is established, the server returns a 32-bit time value and closes the connection. If the server is unable to determine the time at its site, it should either refuse the connection or close it without sending anything.

When used via UDP the time service works as follows:

S: Listen on port 37 (45 octal).

U: Send an empty datagram to port 37.

S: Receive the empty datagram.

S: Send a datagram containing the time as a 32 bit binary number.

U: Receive the time datagram.

The server listens for a datagram on port 37. When a datagram arrives, the server returns a datagram containing the 32-bit time value. If the server is unable to determine the time at its site, it should discard the arriving datagram and make no reply.

The Time

The time is the number of seconds since 00:00 (midnight) 1 January 1900 GMT, such that the time 1 is 12:00:01 am on 1 January 1900 GMT; this base will serve until the year 2036.

For example:

the time 2,208,988,800 corresponds to 00:00 1 Jan 1970 GMT,
 2,398,291,200 corresponds to 00:00 1 Jan 1976 GMT,
 2,524,521,600 corresponds to 00:00 1 Jan 1980 GMT,
 2,629,584,000 corresponds to 00:00 1 May 1983 GMT,
 and -1,297,728,000 corresponds to 00:00 17 Nov 1858 GMT.

5 **RFC 2030**

5.1 **RFC 2030**

Network Working Group
Request for Comments: 2030
Obsoletes: 1769
Category: Informational

D. Mills
University of Delaware
October 1996

Simple Network Time Protocol (SNTP) Version 4

for IPv4, IPv6 and OSI

Status of this Memo

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Abstract

This memorandum describes the Simple Network Time Protocol (SNTP) Version 4, which is an adaptation of the Network Time Protocol (NTP) used to synchronize computer clocks in the Internet. SNTP can be used when the ultimate performance of the full NTP implementation described in RFC-1305 is not needed or justified. When operating with current and previous NTP and SNTP versions, SNTP Version 4 involves no changes to the NTP specification or known implementations, but rather a clarification of certain design features of NTP which allow operation in a simple, stateless remote-procedure call (RPC) mode with accuracy and reliability expectations similar to the UDP/TIME protocol described in RFC-868.

The only significant protocol change in SNTP Version 4 over previous versions of NTP and SNTP is a modified header interpretation to accommodate Internet Protocol Version 6 (IPv6) [DEE96] and OSI [COL94] addressing. However, SNTP Version 4 includes certain optional extensions to the basic Version 3 model, including an anycast mode and an authentication scheme designed specifically for multicast and anycast modes. While the anycast mode extension is described in this document, the authentication scheme extension will be described in another document to be published later. Until such time that a definitive specification is published, these extensions should be considered provisional.

This memorandum obsoletes RFC-1769, which describes SNTP Version 3. Its purpose is to correct certain inconsistencies in the previous document and to clarify header formats and protocol operations for current NTP Version 3 (IPv4) and proposed NTP Version 4 (IPv6 and OSI), which are also used for SNTP. A working knowledge of the NTP Version 3 specification RFC-1305 is not required for an implementation of SNTP.

1. Introduction

The Network Time Protocol (NTP) Version 3 specified in RFC-1305 [MIL92] is widely used to synchronize computer clocks in the global Internet. It provides comprehensive mechanisms to access national time and frequency dissemination services, organize the time-synchronization subnet and adjust the local clock in each participating subnet peer. 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.

RFC-1305 specifies the NTP Version 3 protocol machine in terms of events, states, transition functions and actions and, in addition, engineered algorithms to improve the timekeeping quality and mitigate among several synchronization sources, some of which may be faulty. To achieve accuracies in the low milliseconds over paths spanning major portions of the Internet of today, these intricate algorithms, or their functional equivalents, are necessary. However, in many cases accuracies in the order of significant fractions of a second are acceptable. In such cases, simpler protocols such as the Time Protocol [POS83], have been used for this purpose. These protocols usually involve an RPC exchange where the client requests the time of day and the server returns it in seconds past some known reference epoch.

NTP is designed for use by clients and servers with a wide range of capabilities and over a wide range of network delays and jitter characteristics. Most users of the Internet NTP synchronization subnet of today use a software package including the full suite of NTP options and algorithms, which are relatively complex, real-time applications (see <http://www.eecis.udel.edu/~ntp>). 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. Accordingly, it is useful to explore alternative access strategies using simpler software appropriate for less stringent accuracy expectations.

This document describes the Simple Network Time Protocol (SNTP) Version 4, which is a simplified access strategy for servers and clients using NTP Version 3 as now specified and deployed in the Internet, as well as NTP Version 4 now under development. The access paradigm is identical to the UDP/TIME Protocol and, in fact, it should be easily possible to adapt a UDP/TIME client implementation, say for a personal computer, to operate using SNTP. Moreover, SNTP is also designed to operate in a dedicated server configuration including an integrated radio clock. With careful design and control of the various latencies in the system, which is practical in a dedicated design, it is possible to deliver time accurate to the order of microseconds.

SNTP Version 4 is designed to coexist with existing NTP and SNTP Version 3 clients and servers, as well as proposed Version 4 clients and servers. When operating with current and previous versions of NTP and SNTP, SNTP Version 4 requires no changes to the protocol or implementations now running or likely to be implemented specifically for NTP in SNTP Version 4. To a NTP or SNTP server, NTP and SNTP clients are undistinguishable; to a NTP or SNTP client, NTP and SNTP servers are undistinguishable. Like NTP servers operating in non-symmetric modes, SNTP servers are stateless and can support large numbers of clients; however, unlike most NTP clients, SNTP clients normally operate with only a single server. NTP and SNTP Version 3 servers can operate in unicast and multicast modes. In addition, SNTP Version 4 clients and servers can implement extensions to operate in anycast mode.

It is strongly recommended that SNTP be used only at the extremities of the synchronization subnet. SNTP clients should operate only at the leaves (highest stratum) of the subnet and in configurations where no NTP or SNTP client is dependent on another SNTP client for synchronization. SNTP servers should operate only at the root (stratum 1) of the subnet and then only in configurations where no other source of synchronization other than a reliable radio or modem time service is available. The full degree of reliability ordinarily expected of primary servers is possible only using the redundant sources, diverse subnet paths and crafted algorithms of a full NTP implementation. This extends to the primary source of synchronization itself in the form of multiple radio or modem sources and backup paths to other primary servers should all sources fail or the majority deliver incorrect time. Therefore, the use of SNTP rather than NTP in primary servers should be carefully considered.

An important provision in this document is the reinterpretation of certain NTP Version 4 header fields which provide for IPv6 and OSI addressing and optional anycast extensions designed specifically for multicast service. These additions are in conjunction with the proposed NTP Version 4 specification, which will appear as a separate document. The only difference between the current NTP Version 3 and proposed NTP Version 4 header formats is the interpretation of the four-octet Reference Identifier field, which is used primarily to detect and avoid synchronization loops. In Version 3 and Version 4 primary (stratum-1) servers, this field contains the four-character ASCII reference identifier defined later in this document. In Version 3 secondary servers and clients, it contains the 32-bit IPv4 address of the synchronization source. In Version 4 secondary servers and clients, it contains the low order 32 bits of the last transmit timestamp received from the synchronization source.

In the case of OSI, the Connectionless Transport Service (CLTS) is used [ISO86]. Each SNTP packet is transmitted as the TS-Userdata parameter of a T-UNITDATA Request primitive. Alternately, the header can be encapsulated in a TPDU which itself is transported using UDP [DOB91]. It is not advised that NTP be operated at the upper layers of the OSI stack, such as might be inferred from [FUR94], as this could seriously degrade accuracy. With the header formats defined in this document, it is in principle possible to interwork between servers and clients of one protocol family and another, although the practical difficulties may make this inadvisable.

In the following, indented paragraphs such as this one contain information not required by the formal

protocol specification, but considered good practice in protocol implementations.

2. Operating Modes and Addressing

SNTP Version 4 can operate in either unicast (point to point), multicast (point to multipoint) or anycast (multipoint to point) modes. A unicast client sends a request to a designated server at its unicast address and expects a reply from which it can determine the time and, optionally, the roundtrip delay and local clock offset relative to the server. A multicast server periodically sends a unsolicited message to a designated IPv4 or IPv6 local broadcast address or multicast group address and ordinarily expects no requests from clients. A multicast client listens on this address and ordinarily sends no requests. An anycast client sends a request to a designated IPv4 or IPv6 local broadcast address or multicast group address. One or more anycast servers reply with their individual unicast addresses. The client binds to the first one received, then continues operation in unicast mode.

Multicast servers should respond to client unicast requests, as well as send unsolicited multicast messages. Multicast clients may send unicast requests in order to determine the network propagation delay between the server and client and then continue operation in multicast mode.

In unicast mode, the client and server end-system addresses are assigned following the usual IPv4, IPv6 or OSI conventions. In multicast mode, the server uses a designated local broadcast address or multicast group address. An IP local broadcast address has scope limited to a single IP subnet, since routers do not propagate IP broadcast datagrams. On the other hand, an IP multicast group address has scope extending to potentially the entire Internet. The scoping, routing and group membership procedures are determined by considerations beyond the scope of this document. For IPv4, the IANA has assigned the multicast group address 224.0.1.1 for NTP, which is used both by multicast servers and anycast clients. NTP multicast addresses for IPv6 and OSI have yet to be determined.

Multicast clients listen on the designated local broadcast address or multicast group address. In the case of local broadcast addresses, no further provisions are necessary. In the case of IP multicast addresses, the multicast client and anycast server must implement the Internet Group Management Protocol (IGMP) [DEE89], in order that the local router joins the multicast group and relays messages to the IPv4 or IPv6 multicast group addresses assigned by the IANA. Other than the IP addressing conventions and IGMP, there is no difference in server or client operations with either the local broadcast address or multicast group address.

It is important to adjust the time-to-live (TTL) field in the IP header of multicast messages to a reasonable value, in order to limit the network resources used by this (and any other) multicast service. Only multicast clients in scope will receive multicast server messages. Only cooperating anycast servers in scope will reply to a client request. The engineering principles which determine the proper value to be used are beyond the scope of this document.

Anycast mode is designed for use with a set of cooperating servers whose addresses are not known beforehand by the client. An anycast client sends a request to the designated local broadcast or multicast group address as described below. For this purpose, the NTP multicast group address assigned by the IANA is used. One or more anycast servers listen on the designated local broadcast address or multicast group address. Each anycast server, upon receiving a request, sends a unicast reply message to the originating client. The client then binds to the first such message received and continues operation in unicast mode. Subsequent replies from other anycast servers are ignored.

In the case of SNTP as specified herein, there is a very real vulnerability that SNTP multicast clients can be disrupted by misbehaving or hostile SNTP or NTP multicast servers elsewhere in the Internet, since at present all such servers use the same IPv4 multicast group address assigned by the IANA. Where necessary, access control based on the server source address can be used to select only the designated server known to and trusted by the client. The use of cryptographic authentication scheme defined in RFC-1305 is optional; however, implementors should be advised that extensions to this scheme are planned specifically for NTP multicast and anycast modes.

In NTP Version 3, the reference identifier was often used to walk-back the synchronization subnet to the root (primary server) for management purposes. In NTP Version 4, this feature is not available, since the addresses are longer than 32 bits. However, the intent in the protocol design was to provide a way to detect and avoid loops. A peer could determine that a loop was possible by comparing the contents of this field with the IPv4 destination address in the same packet. A NTP Version 4 server can accomplish the same thing by comparing the contents of this field with the low order 32 bits of the originate timestamp in the same packet. There is a small possibility of false alarm in this scheme, but the false alarm rate can be minimized by randomizing the low order unused bits of the transmit timestamp.

SNTP uses the standard NTP timestamp format described in [RFC-1305](#) and previous versions of that document. In conformance with standard Internet practice, NTP data are specified as integer or fixed-point quantities, with bits numbered in big-endian fashion from 0 starting at the left, or high-order, position. Unless specified otherwise, all quantities are unsigned and may occupy the full field width with an implied 0 preceding bit 0.

It is advisable to fill the non-significant low order bits of the timestamp with a random, unbiased bit string, both to avoid systematic roundoff errors and as a means of loop detection and replay detection (see below). One way of doing this is to generate a random bit string in a 64-bit word, then perform an arithmetic right shift a number of bits equal to the number of significant bits of the timestamp, then add the result to the original timestamp.

```

      1      2      3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|-----Seconds-----|
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|-----Seconds Fraction (0-padded)-----|
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

```

As the NTP timestamp format has been in use for the last 17 years, it remains a possibility that it will be in use 40 years from now when the seconds field overflows. As it is probably inappropriate to archive

4. NTP Message Format

Below is a description of the NTP/SNTP Version 4 message format, which follows the IP and UDP headers. This format is identical to that described in RFC-1305, with the exception of the contents of the reference identifier field. The header fields are defined as follows:



completeness, the function of each field is briefly summarized below.

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

LI	Value	Meaning
00	0	no warning
01	1	last minute has 61 seconds
10	2	last minute has 59 seconds)
11	3	alarm condition (clock not synchronized)

Version Number (VN): This is a three-bit integer indicating the NTP/SNTP version number. The version number is 3 for Version 3 (IPv4 only) and 4 for Version 4 (IPv4, IPv6 and OSI). If necessary to distinguish between IPv4, IPv6 and OSI, the encapsulating context must be inspected.

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

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

In unicast and anycast modes, the client sets this field to 3 (client) in the request and the server sets it to 4 (server) in the reply. In multicast mode, the server sets this field to 5 (broadcast).

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

Stratum	Meaning
0	unspecified or unavailable
1	primary reference (e.g., radio clock)
2-15	secondary reference (via NTP or SNTP)
16-255	reserved

Poll Interval: This is an eight-bit signed integer indicating the maximum interval between successive messages, in seconds to the nearest power of two. The values that can appear in this field presently range from 4 (16 s) to 14 (16284 s); however, most applications use only the sub-range 6 (64 s) to 10 (1024 s).

Precision: This is an eight-bit signed integer indicating the precision of the local clock, in seconds to the nearest power of two. The values that normally appear in this field range from -6 for mains-frequency clocks to -20 for microsecond clocks found in some workstations.

Root Delay: This is a 32-bit signed fixed-point number indicating the total roundtrip delay to the primary reference source, in seconds with fraction point between bits 15 and 16. Note that this variable can take on both positive and negative values, depending on the relative time and frequency offsets. The

values that normally appear in this field range from negative values of a few milliseconds to positive values of several hundred milliseconds.

Root Dispersion: This is a 32-bit unsigned fixed-point number indicating the nominal error relative to the primary reference source, in seconds with fraction point between bits 15 and 16. The values that normally appear in this field range from 0 to several hundred milliseconds.

Reference Identifier: This is a 32-bit string identifying the particular reference source. In the case of NTP Version 3 or Version 4 stratum-0 (unspecified) or stratum-1 (primary) servers, this is a four-character ASCII string, left justified and zero padded to 32 bits. In NTP Version 3 secondary servers, this is the 32-bit IPv4 address of the reference source. In NTP Version 4 secondary servers, this is the low order 32 bits of the latest transmit timestamp of the reference source. NTP primary (stratum 1) servers should set this field to a code identifying the external reference source according to the following list. If the external reference is one of those listed, the associated code should be used. Codes for sources not listed can be contrived as appropriate.

Code	External Reference Source
LOCL	uncalibrated local clock used as a primary reference for a subnet without external means of synchronization
PPS	atomic clock or other pulse-per-second source individually calibrated to national standards
ACTS	NIST dialup modem service
USNO	USNO modem service
PTB	PTB (Germany) modem service
TDF	Allouis (France) Radio 164 kHz
DCF	Mainflingen (Germany) Radio 77.5 kHz
MSF	Rugby (UK) Radio 60 kHz
WWV	Ft. Collins (US) Radio 2.5, 5, 10, 15, 20 MHz
WWVB	Boulder (US) Radio 60 kHz
WWVH	Kauai Hawaii (US) Radio 2.5, 5, 10, 15 MHz
CHU	Ottawa (Canada) Radio 3330, 7335, 14670 kHz
LORC	LORAN-C radionavigation system
OMEG	OMEGA radionavigation system
GPS	Global Positioning Service
GOES	Geostationary Orbit Environment Satellite

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

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

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

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

Authenticator (optional): When the NTP authentication scheme is implemented, the Key Identifier and Message Digest fields contain the message authentication code (MAC) information defined in Appendix C of RFC-1305.

5. SNTP Client Operations

A SNTP client can operate in multicast mode, unicast mode or anycast mode. In multicast mode, the client sends no request and waits for a broadcast (mode 5) from a designated multicast server. In unicast mode, the client sends a request (mode 3) to a designated unicast server and expects a reply (mode 4) from that server. In anycast mode, the client sends a request (mode 3) to a designated local broadcast or multicast group address and expects a reply (mode 4) from one or more anycast servers. The client uses the first reply received to establish the particular server for subsequent unicast operations. Later replies from this server (duplicates) or any other server are ignored. Other than the selection of address in the request, the operations of anycast and unicast clients are identical. Requests are normally sent at intervals from 64 s to 1024 s, depending on the frequency tolerance of the client clock and the required accuracy.

A unicast or anycast client initializes the NTP message header, sends the request to the server and strips the time of day from the Transmit Timestamp field of the reply. For this purpose, all of the NTP header fields shown above can be set to 0, except the first octet and (optional) Transmit Timestamp fields. In the first octet, the LI field is set to 0 (no warning) and the Mode field is set to 3 (client). The VN field must agree with the version number of the NTP/SNTP server; however, Version 4 servers will also accept previous versions. Version 3 (RFC-1305) and Version 2 (RFC-1119) servers already accept all previous versions, including Version 1 (RFC-1059). Note that Version 0 (RFC-959) is no longer supported by any other version.

Since there will probably continue to be NTP and SNTP servers of all four versions interoperating in the Internet, careful consideration should be given to the version used by SNTP Version 4 clients. It is recommended that clients use the latest version known to be supported by the selected server in the interest of the highest accuracy and reliability. SNTP Version 4 clients can interoperate with all previous version NTP and SNTP servers, since the header fields used by SNTP clients are unchanged. Version 4 servers are required to reply in the same version as the request, so the VN field of the request also specifies the version of the reply.

While not necessary in a conforming client implementation, in unicast and anycast modes it is highly recommended that the transmit timestamp in the request is set to the time of day according to the client clock in NTP timestamp format. This allows a simple calculation to determine the propagation delay between the server and client and to align the local clock generally within a few tens of milliseconds relative to the server. In addition, this provides a simple method to verify that the server reply is in fact a legitimate response to the specific client request and avoid replays. In multicast mode, the client has no information to calculate the propagation delay or determine the validity of the server, unless the NTP authentication scheme is used.

To calculate the roundtrip delay d and local clock offset t relative to the server, the client sets the transmit timestamp in the request to the time of day according to the client clock in NTP timestamp format. The server copies this field to the originate timestamp in the reply and sets the receive timestamp and transmit timestamp to the time of day according to the server clock in NTP timestamp format.

When the server reply is received, the client determines a Destination Timestamp variable as the time of arrival according to its clock in NTP timestamp format. The following table summarizes the four timestamps.

Timestamp Name	ID	When Generated
Originate Timestamp	T1	time request sent by client
Receive Timestamp	T2	time request received by server
Transmit Timestamp	T3	time reply sent by server
Destination Timestamp	T4	time reply received by client

The roundtrip delay d and local clock offset t are defined as

$$d = (T4 - T1) - (T2 - T3) \quad t = ((T2 - T1) + (T3 - T4)) / 2.$$

The following table summarizes the SNTP client operations in unicast, anycast and multicast modes. The recommended error checks are shown in the Reply and Multicast columns in the table. The message should be considered valid only if all the fields shown contain values in the respective ranges. Whether to believe the message if one or more of the fields marked "ignore" contain invalid values is at the discretion of the implementation.

Field Name	Unicast/Anycast		Multicast
	Request	Reply	
LI	0	0-2	0-2
VN	1-4	copied from request	1-4
Mode	3	4	5
Stratum	0	1-14	1-14
Poll	0	ignore	ignore
Precision	0	ignore	ignore
Root Delay	0	ignore	ignore
Root Dispersion	0	ignore	ignore
Reference Identifier	0	ignore	ignore
Reference Timestamp	0	ignore	ignore
Originate Timestamp	0	(see text)	ignore
Receive Timestamp	0	(see text)	ignore
Transmit Timestamp	(see text)	nonzero	nonzero
Authenticator	optional	optional	optional

6. SNTP Server Operations

A SNTP Version 4 server operating with either a NTP or SNTP client of the same or previous versions retains no persistent state. Since a SNTP server ordinarily does not implement the full set of NTP algorithms intended to support redundant peers and diverse network paths, a SNTP server should be operated only in conjunction with a source of external synchronization, such as a reliable radio clock or telephone modem. In this case it always operates as a primary (stratum 1) server.

A SNTP server can operate in unicast mode, anycast mode, multicast mode or any combination of these modes. In unicast and anycast modes, the server receives a request (mode 3), modifies certain fields in the NTP header, and sends a reply (mode 4), possibly using the same message buffer as the request. In anycast mode, the server listens on the designated local broadcast or multicast group address assigned by the IANA, but uses its own unicast address in the source address field of the reply. Other than the selection of address in the reply, the operations of anycast and unicast servers are identical. Multicast messages are normally sent at poll intervals from 64 s to 1024 s, depending on the expected frequency tolerance of the client clocks and the required accuracy.

In unicast and anycast modes, the VN and Poll fields of the request are copied intact to the reply. If the Mode field of the request is 3 (client), it is set to 4 (server) in the reply; otherwise, this field is set to 2 (symmetric passive) in order to conform to the NTP specification. This allows clients configured in symmetric active (mode 1) to interoperate successfully, even if configured in possibly suboptimal ways. In multicast (unsolicited) mode, the VN field is set to 4, the Mode field is set to 5 (broadcast), and the Poll field set to the nearest integer base-2 logarithm of the poll interval.

Note that it is highly desirable that, if a server supports multicast mode, it also supports unicast mode. This is so a potential multicast client can calculate the propagation delay using a client/server exchange prior to regular operation using only multicast mode. If the server supports anycast mode,

then it must support unicast mode. There does not seem to be a great advantage to operate both multicast and anycast modes at the same time, although the protocol specification does not forbid it.

In unicast and anycast modes, the server may or may not respond if not synchronized to a correctly operating radio clock, but the preferred option is to respond, since this allows reachability to be determined regardless of synchronization state. In multicast mode, the server sends broadcasts only if synchronized to a correctly operating reference clock.

The remaining fields of the NTP header are set in the following way. Assuming the server is synchronized to a radio clock or other primary reference source and operating correctly, the LI field is set to 0 and the Stratum field is set to 1 (primary server); if not, the Stratum field is set to 0 and the LI field is set to 3. The Precision field is set to reflect the maximum reading error of the local clock. For all practical cases it is computed as the negative of the number of significant bits to the right of the decimal point in the NTP timestamp format. The Root Delay and Root Dispersion fields are set to 0 for a primary server; optionally, the Root Dispersion field can be set to a value corresponding to the maximum expected error of the radio clock itself. The Reference Identifier is set to designate the primary reference source, as indicated in the table of Section 5 of this document.

The timestamp fields are set as follows. If the server is unsynchronized or first coming up, all timestamp fields are set to zero. If synchronized, the Reference Timestamp is set to the time the last update was received from the radio clock or modem. In unicast and anycast modes, the Receive Timestamp and Transmit Timestamp fields are set to the time of day when the message is sent and the Originate Timestamp field is copied unchanged from the Transmit Timestamp field of the request. It is important that this field be copied intact, as a NTP client uses it to avoid replays. In multicast mode, the Originate Timestamp and Receive Timestamp fields are set to 0 and the Transmit Timestamp field is set to the time of day when the message is sent. The following table summarizes these actions.

Field Name	Unicast/Anycast		Multicast
	Request	Reply	
LI	ignore	0 or 3	0 or 3
VN	1-4	copied from request	4
Mode	3	2 or 4	5
Stratum	ignore	1	1
Poll	ignore	copied from request	log2 poll interval
Precision	ignore	-log2 server significant bits	-log2 server significant bits
Root Delay	ignore	0	0
Root Dispersion	ignore	0	0
Reference Identifier	ignore	source ident	source ident
Reference Timestamp	ignore	time of last radio update	time of last radio update
Originate Timestamp	ignore	copied from transmit timestamp	0
Receive Timestamp	ignore	time of day	0
Transmit Timestamp	(see text)	time of day	time of day
Authenticator	optional	optional	optional

There is some latitude on the part of most clients to forgive invalid timestamps, such as might occur when first coming up or during periods when the primary reference source is inoperative. The most important indicator of an unhealthy server is the LI field, in which a value of 3 indicates an unsynchronized condition. When this value is displayed, clients should discard the server message,

regardless of the contents of other fields.

7. Configuration and Management

Initial setup for SNTP servers and clients can be done using a configuration file if a file system is available, or a serial port if not. It is intended that in-service management of NTP and SNTP Version 4 servers and clients be performed using SNMP and a suitable MIB to be published later. Ordinarily, SNTP servers and clients are expected to operate with little or no site-specific configuration, other than specifying the IP address and subnet mask or OSI NSAP address.

Unicast clients must be provided with the designated server name or address. If a server name is used, the address of one of more DNS servers must be provided. Multicast servers and anycast clients must be provided with the TTL and local broadcast or multicast group address. Anycast servers and multicast clients may be configured with a list of address-mask pairs for access control, so that only those clients or servers known to be trusted will be used. These servers and clients must implement the IGMP protocol and be provided with the local broadcast or multicast group address as well. The configuration data for cryptographic authentication is beyond the scope of this document.

There are several scenarios which provide automatic server discovery and selection for SNTP clients with no pre-specified configuration, other than the IP address and subnet mask or OSI NSAP address. For a IP subnet or LAN segment including a fully functional NTP server, the clients can be configured for multicast mode using the local broadcast address. The same approach can be used with other servers using the multicast group address. In both cases, provision of an access control list is a good way to insure only trusted sources can be used to set the local clock.

In another scenario suitable for an extended network with significant network propagation delays, clients can be configured for anycast mode, both upon initial startup and after some period when the currently selected unicast source has not been heard. Following the defined protocol, the client binds to the first reply heard and continues operation in unicast mode. In this mode the local clock can be automatically adjusted to compensate for the propagation delay.

In still another scenario suitable for any network and where multicast service is not available, the DNS can be set up with a common CNAME, like time.domain.net, and a list of address records for NTP servers in the same domain. Upon resolving time.domain.net and obtaining the list, the client selects a server at random and begins operation in unicast mode with that server. Many variations on this theme are possible.

8. Acknowledgements

Jeff Learman was helpful in developing the OSI model for this protocol. Ajit Thyagarajan provided valuable suggestions and corrections.

9. References

[COL94] Colella, R., R. Callon, E. Gardner, Y. Rekhter, "Guidelines for OSI NSAP allocation in the Internet", RFC 1629, NIST, May 1994.

[DAR81] Postel, J., "Internet Protocol", STD 5, RFC 791, USC Information Sciences Institute, September 1981.

[DEE89] Deering, S., "Host extensions for IP multicasting", STD 5, RFC 1112, Stanford University, August 1989.

[DEE96] Deering, S., R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 1883, Xerox and Ipsilon, January 1996.

[DOB91] Dobbins, K, W. Haggerty, C. Shue, "OSI connectionless transport services on top of UDP - Version: 1", RFC 1240, Open Software Foundation, June 1991.

[EAS95] Eastlake, D., 3rd., and C. Kaufman, "Domain Name System Security Extensions", Work in Progress.

[FUR94] Furniss, P., "Octet sequences for upper-layer OSI to support basic communications applications", RFC 1698, Consultant, October 1994.

[HIN96] Hinden, R., and S. Deering, "IP Version 6 addressing Architecture", RFC 1884, Ipsilon and Xerox, January 1996.

[ISO86] International Standards 8602 - Information Processing Systems - OSI: Connectionless Transport Protocol Specification. International Standards Organization, December 1986.

[MIL92] Mills, D., "Network Time Protocol (Version 3) specification, implementation and analysis", RFC 1305, University of Delaware, March 1992.

[PAR93] Partridge, C., T. Mendez and W. Milliken, "Host anycasting service", RFC 1546, Bolt Beranek Newman, November 1993.

[POS80] Postel, J., "User Datagram Protocol", STD 6, RFC 768, USC Information Sciences Institute, August 1980.

[POS83] Postel, J., "Time Protocol", STD 26, RFC 868, USC Information Sciences Institute, May 1983.

Security Considerations

Security issues are not discussed in this memo.

Author's Address

David L. Mills
Electrical Engineering Department
University of Delaware
Newark, DE 19716

Phone: (302) 831-8247

6 RFC 1305

6.1 RFC 1305

Network Working Group
Request for Comments: 1305
Obsoletes RFC-1119, RFC-1059, RFC-958

David L. Mills
University of Delaware
March 1992

Network Time Protocol (Version 3) Specification, Implementation and Analysis

Note: This document consists of an approximate rendering in ASCII of the PostScript document of the same name. It is provided for convenience and for use in searches, etc. However, most tables, figures, equations and captions have not been rendered and the pagination and section headings are not available.

Abstract

This document describes the Network Time Protocol (NTP), specifies its formal structure and summarizes information useful for its implementation. NTP provides the mechanisms to synchronize time and coordinate time distribution in a large, diverse internet operating at rates from mundane to lightwave. It uses a returnable-time design in which a distributed subnet of time servers operating in a self-organizing, hierarchical-master-slave configuration synchronizes local clocks within the subnet and to national time standards via wire or radio. The servers can also redistribute reference time via local routing algorithms and time daemons.

Status of this Memo

This RFC specifies an IAB standards track protocol for the Internet community and requests discussion and suggestions for improvements. Please refer to the current edition of the <169>IAB Official Protocol Standards<170> for the standardization state and status of this protocol. Distribution of this memo is unlimited.

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

Preface

This document describes Version 3 of the Network Time Protocol (NTP). It supersedes Version 2 of the protocol described in RFC-1119 dated September 1989. However, it neither changes the protocol in any significant way nor obsoletes previous versions or existing implementations. The main motivation for the new version is to refine the analysis and implementation models for new applications at much higher network speeds to the gigabit-per-second regime and to provide for the enhanced stability, accuracy and precision required at such speeds. In particular, the sources of time and frequency errors have been rigorously examined and error bounds established in order to improve performance, provide a model for correctness assertions and indicate timekeeping quality to the user. The revision also incorporates two new optional features, (1) an algorithm to combine the offsets of a number of peer time servers in order to enhance accuracy and (2) improved local-clock

algorithms which allow the poll intervals on all synchronization paths to be substantially increased in order to reduce network overhead. An overview of the changes, which are described in detail in Appendix D, follows:

- **1.** In Version 3 The local-clock algorithm has been overhauled to improve stability and accuracy. Appendix G presents a detailed mathematical model and design example which has been refined with the aid of feedback-control analysis and extensive simulation using data collected over ordinary Internet paths. Section 5 of RFC-1119 on the NTP local clock has been completely rewritten to describe the new algorithm. Since the new algorithm can result in message rates far below the old ones, it is highly recommended that they be used in new implementations. Note that use of the new algorithm does not affect interoperability with previous versions or existing implementations.
- **2.** In Version 3 a new algorithm to combine the offsets of a number of peer time servers is presented in Appendix F. This algorithm is modelled on those used by national standards laboratories to combine the weighted offsets from a number of standard clocks to construct a synthetic laboratory time scale more accurate than that of any clock separately. It can be used in an NTP implementation to improve accuracy and stability and reduce errors due to asymmetric paths in the Internet. The new algorithm has been simulated using data collected over ordinary Internet paths and, along with the new local-clock algorithm, implemented and tested in the Fuzzball time servers now running in the Internet. Note that use of the new algorithm does not affect interoperability with previous versions or existing implementations.
- **3.** Several inconsistencies and minor errors in previous versions have been corrected in Version 3. The description of the procedures has been rewritten in pseudo-code augmented by English commentary for clarity and to avoid ambiguity. Appendix I has been added to illustrate C-language implementations of the various filtering and selection algorithms suggested for NTP. Additional information is included in Section 5 and in Appendix E, which includes the tutorial material formerly included in Section 2 of RFC-1119, as well as much new material clarifying the interpretation of time scales and leap seconds.
- **4.** Minor changes have been made in the Version-3 local-clock algorithms to avoid problems observed when leap seconds are introduced in the UTC time scale and also to support an auxiliary precision oscillator, such as a cesium clock or timing receiver, as a precision time base. In addition, changes were made to some procedures described in Section 3 and in the clock-filter and clock-selection procedures described in Section 4. While these changes were made to correct minor bugs found as the result of experience and are recommended for new implementations, they do not affect interoperability with previous versions or existing implementations in other than minor ways (at least until the next leap second).
- **5.** In Version 3 changes were made to the way delay, offset and dispersion are defined, calculated and processed in order to reliably bound the errors inherent in the time-transfer procedures. In particular, the error accumulations were moved from the delay computation to the dispersion computation and both included in the clock filter and selection procedures. The clock-selection procedure was modified to remove the first of the two sorting/discarding

steps and replace with an algorithm first proposed by Marzullo and later incorporated in the Digital Time Service. These changes do not significantly affect the ordinary operation of or compatibility with various versions of NTP, but they do provide the basis for formal statements of correctness as described in Appendix H.

Table of Contents

- 1. [Introduction](#)
 - 1.1. [Related Technology](#)
- 2. [System Architecture](#)
 - 2.1. [Implementation Model](#)
 - 2.2. [Network Configurations](#)
- 3. [Network Time Protocol](#)
 - 3.1. [Data Formats](#)
 - 3.2. [State Variables and Parameters](#)
 - 3.2.1. [Common Variables](#)
 - 3.2.2. [System Variables](#)
 - 3.2.3. [Peer Variables](#)
 - 3.2.4. [Packet Variables](#)
 - 3.2.5. [Clock-Filter Variables](#)
 - 3.2.6. [Authentication Variables](#)
 - 3.2.7. [Parameters](#)
 - 3.3. [Modes of Operation](#)
 - 3.4. [Event Processing](#)
 - 3.4.1. [Notation Conventions](#)
 - 3.4.2. [Transmit Procedure](#)
 - 3.4.3. [Receive Procedure](#)
 - 3.4.4. [Packet Procedure](#)
 - 3.4.5. [Clock-Update Procedure](#)
 - 3.4.6. [Primary-Clock Procedure](#)
 - 3.4.7. [Initialization Procedures](#)
 - 3.4.7.1. [Initialization Procedure](#)
 - 3.4.7.2. [Initialization-Instantiation Procedure](#)
 - 3.4.7.3. [Receive-Instantiation Procedure](#)
 - 3.4.7.4. [Primary Clock-Instantiation Procedure](#)
 - 3.4.8. [Clear Procedure](#)
 - 3.4.9. [Poll-Update Procedure](#)
 - 3.5. [Synchronization Distance Procedure](#)
 - 3.6. [Access Control Issues](#)
- 4. [Filtering and Selection Algorithms](#)
 - 4.1. [Clock-Filter Procedure](#)
 - 4.2. [Clock-Selection Procedure](#)
 - 4.2.1. [Intersection Algorithm](#)
 - 4.2.2. [Clustering Algorithm](#)
- 5. [Local Clocks](#)
 - 5.1. [Fuzzball Implementation](#)

- 5.2. [Gradual Phase Adjustments](#)
- 5.3. [Step Phase Adjustments](#)
- 5.4. [Implementation Issues](#)
- 6. [Acknowledgments](#)
- 7. [References](#)

- A. [Appendix A. NTP Data Format - Version 3](#)
- B. [Appendix B. NTP Control Messages](#)
 - NTP Control Message Format
 - Status Words
 - System Status Word
 - Peer Status Word
 - Clock Status Word
 - Error Status Word
 - Commands
- C. [Appendix C. Authentication Issues](#)
 - NTP Authentication Mechanism
 - NTP Authentication Procedures
 - Encrypt Procedure
 - Decrypt Procedure
 - Control-Message Procedures
- D. [Appendix D. Differences from Previous Versions.](#)
- E. [Appendix E. The NTP Timescale and its Chronometry](#)
 - Introduction
 - Primary Frequency and Time Standards
 - Time and Frequency Dissemination
 - Calendar Systems
 - The Modified Julian Day System
 - Determination of Frequency
 - Determination of Time and Leap Seconds
 - The NTP Timescale and Reckoning with UTC
- F. [Appendix F. The NTP Clock-Combining Algorithm](#)
 - Introduction
 - Determining Time and Frequency
 - Clock Modelling
 - Development of a Composite Timescale
 - Application to NTP
 - Clock-Combining Procedure
- G. [Appendix G. Computer Clock Modelling and Analysis](#)
 - Computer Clock Models
 - The Fuzzball Clock Model
 - The Unix Clock Model
 - Mathematical Model of the NTP Logical Clock
 - Parameter Management
 - Adjusting VCO Gain (α)
 - Adjusting PLL Bandwidth (τ)
 - The NTP Clock Model

H. [Appendix H. Analysis of Errors and Correctness Principles](#)

Introduction
Timestamp Errors
Measurement Errors
Network Errors
Inherited Errors
Correctness Principles

I. [Appendix I. Selected C-Language Program Listings](#)

Common Definitions and Variables
Clock Filter Algorithm
Interval Intersection Algorithm
Clock Selection Algorithm
Clock Combining Procedure
Subroutine to Compute Synchronization Distance

List of Figures

Figure 1. [Implementation Model](#)
Figure 2. [Calculating Delay and Offset](#)
Figure 3. [Clock Registers](#)
Figure 4. [NTP Message Header](#)
Figure 5. [NTP Control Message Header](#)
Figure 6. [Status Word Formats](#)
Figure 7. [Authenticator Format](#)
Figure 8. [Comparison of UTC and NTP Timescales at Leap](#)
Figure 9. [Network Time Protocol](#)
Figure 10. [Hardware Clock Models](#)
Figure 11. [Clock Adjustment Process](#)
Figure 12. [NTP Phase-Lock Loop \(PLL\) Model](#)
Figure 13. [Timing Intervals](#)
Figure 14. [Measuring Delay and Offset](#)
Figure 15. [Error Accumulations](#)
Figure 16. [Confidence Intervals and Intersections](#)

List of Tables

Table 1. [System Variables](#)
Table 2. [Peer Variables](#)
Table 3. [Packet Variables](#)
Table 4. [Parameters](#)
Table 5. [Modes and Actions](#)
Table 6. [Clock Parameters](#)
Table 7. [Characteristics of Standard Oscillators](#)
Table 8. [Table of Leap-Second Insertions](#)
Table 9. [Notation Used in PLL Analysis](#)
Table 10. [PLL Parameters](#)
Table 11. [Notation Used in PLL Analysis](#)

Table 12. [Notation Used in Error Analysis](#)

Author's address
David L. Mills
Electrical Engineering Department
University of Delaware
Newark, DE 19716
Phone (302) 451-1968247
EMail mills@udel.edu

6.2 1. Introduction

This document constitutes a formal specification of the Network Time Protocol (NTP) Version 3, which is used to synchronize timekeeping among a set of distributed time servers and clients. It defines the architectures, algorithms, entities and protocols used by NTP and is intended primarily for implementors. A companion document [MIL91a] summarizes the requirements, analytical models, algorithmic analysis and performance under typical Internet conditions. Another document [MIL91b] describes the NTP time scale and its relationship to other standard timescales now in use. NTP was first described in RFC-958 [MIL85c], but has since evolved in significant ways, culminating in the most recent NTP Version 2 described in RFC-1119 [MIL89]. It is built on the Internet Protocol (IP) [DAR81a] and User Datagram Protocol (UDP) [POS80], which provide a connectionless transport mechanism; however, it is readily adaptable to other protocol suites. NTP is evolved from the Time Protocol [POS83b] and the ICMP Timestamp message [DAR81b], but is specifically designed to maintain accuracy and robustness, even when used over typical Internet paths involving multiple gateways, highly dispersive delays and unreliable nets.

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

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

[Section 4](#) describes algorithms useful for deglitching and smoothing clock-offset samples collected on a continuous basis. These algorithms evolved from those suggested in [MIL85a], were refined as the results of experiments described in [MIL85b] and further evolved under typical operating conditions over the last three years. In addition, as the result of experience in operating multiple-server subnets including radio clocks at several sites in the U.S. and with clients in the U.S. and Europe, reliable algorithms for selecting good clocks from a population possibly including broken ones have been developed [DEC89], [MIL91a] and are described in [Section 4](#).

The accuracies achievable by NTP depend strongly on the precision of the local-clock hardware and stringent control of device and process latencies. Provisions must be included to adjust the software logical-clock time and frequency in response to corrections produced by NTP. [Section 5](#) describes a local-clock design evolved from the Fuzzball implementation described in [MIL83b] and [MIL88b]. This design includes offset-slewing, frequency compensation and deglitching mechanisms capable of accuracies in the order of a millisecond, even after extended periods when synchronization to primary reference sources has been lost.

Details specific to NTP packet formats used with the Internet Protocol (IP) and User Datagram Protocol (UDP) are presented in [Appendix A](#), while details of a suggested auxiliary NTP Control Message, which may be used when comprehensive network-monitoring facilities are not available, are presented in [Appendix B](#). [Appendix C](#) contains specification and implementation details of an optional authentication mechanism which can be used to control access and prevent unauthorized data modification, while [Appendix D](#) contains a listing of differences between Version 3 of NTP and previous versions. [Appendix E](#) expands on issues involved with precision timescales and calendar dating peculiar to computer networks and NTP. [Appendix F](#) describes an optional algorithm to improve accuracy by combining the time offsets of a number of clocks. [Appendix G](#) presents a detailed mathematical model and analysis of the NTP local-clock algorithms. [Appendix H](#) analyzes the sources and propagation of errors and presents correctness principles relating to the time-transfer service. [Appendix I](#) illustrates C-language code segments for the clock-filter, clock-selection and related algorithms described in [Section 4](#).

6.3 1.1 Related Technology

Other mechanisms have been specified in the Internet protocol suite to record and transmit the time at which an event takes place, including the Daytime protocol [POS83a], Time Protocol [POS83b], ICMP Timestamp message [DAR81b] and IP Timestamp option [SU81]. Experimental results on measured clock offsets and roundtrip delays in the Internet are discussed in [MIL83a], [MIL85b], [COL88] and [MIL88a]. Other synchronization algorithms are discussed in [LAM78], [GUS84], [HAL84], [LUN84], [LAM85], [MAR85], [MIL85a], [MIL85b], [MIL85c], [GUS85b], [SCH86], [TRI86], [RIC88], [MIL88a], [DEC89] and [MIL91a], while protocols based on them are described in [MIL81a], [MIL81b], [MIL83b], [GUS85a], [MIL85c], [TRI86], [MIL88a], [DEC89] and [MIL91a]. NTP uses techniques evolved from them and both linear-systems and agreement methodologies. Linear methods for digital telephone network synchronization are summarized in [LIN80], while agreement methods for

clock synchronization are summarized in [LAM85].

The Digital Time Service (DTS) [DEC89] has many of the same service objectives as NTP. The DTS design places heavy emphasis on configuration management and correctness principles when operated in a managed LAN or LAN-cluster environment, while NTP places heavy emphasis on the accuracy and stability of the service operated in an unmanaged, global-internet environment. In DTS a synchronization subnet consists of clerks, servers, couriers and time providers. With respect to the NTP nomenclature, a time provider is a primary reference source, a courier is a secondary server intended to import time from one or more distant primary servers for local redistribution and a server is intended to provide time for possibly many end nodes or clerks. Unlike NTP, DTS does not need or use mode or stratum information in clock selection and does not include provisions to filter timing noise, select the most accurate from a set of presumed correct clocks or compensate for inherent frequency errors.

In fact, the latest revisions in NTP have adopted certain features of DTS in order to support correctness principles. These include mechanisms to bound the maximum errors inherent in the time-transfer procedures and the use of a provably correct (subject to stated assumptions) mechanism to reject inappropriate peers in the clock-selection procedures. These features are described in Section 4 and Appendix H of this document.

The Fuzzball routing protocol [MIL83b], sometimes called Hellospeak, incorporates time synchronization directly into the routing-protocol design. One or more processes synchronize to an external reference source, such as a radio clock or NTP daemon, and the routing algorithm constructs a minimum-weight spanning tree rooted on these processes. The clock offsets are then distributed along the arcs of the spanning tree to all processes in the system and the various process clocks corrected using the procedure described in Section 5 of this document. While it can be seen that the design of Hellospeak strongly influenced the design of NTP, Hellospeak itself is not an Internet protocol and is unsuited for use outside its local-net environment.

The Unix 4.3bsd time daemon `timed` [GUS85a] uses a single master-time daemon to measure offsets of a number of slave hosts and send periodic corrections to them. In this model the master is determined using an election algorithm [GUS85b] designed to avoid situations where either no master is elected or more than one master is elected. The election process requires a broadcast capability, which is not a ubiquitous feature of the Internet. While this model has been extended to support hierarchical configurations in which a slave on one network serves as a master on the other [TRI86], the model requires handcrafted configuration tables in order to establish the hierarchy and avoid loops. In addition to the burdensome, but presumably infrequent, overheads of the election process, the offset measurement/correction process requires twice as many messages as NTP per update.

A scheme with features similar to NTP is described in [KOP87]. This scheme is intended for multi-server LANs where each of a set of possibly many time servers determines its local-time offset relative to each of the other servers in the set using periodic timestamped messages, then determines the local-clock correction using the Fault-Tolerant Average (FTA) algorithm of [LUN84]. The FTA algorithm, which is useful where up to k servers may be faulty, sorts

the offsets, discards the k highest and lowest ones and averages the rest. The scheme, as described in [SCH86], is most suitable to LAN environments which support broadcast and would result in unacceptable overhead in an internet environment. In addition, for reasons given in Section 4 of this paper, the statistical properties of the FTA algorithm are not likely to be optimal in an internet environment with highly dispersive delays.

A good deal of research has gone into the issue of maintaining accurate time in a community where some clocks cannot be trusted. A truechimer is a clock that maintains timekeeping accuracy to a previously published (and trusted) standard, while a falseticker is a clock that does not. Determining whether a particular clock is a truechimer or falseticker is an interesting abstract problem which can be attacked using agreement methods summarized in [LAM85] and [SRI87].

A convergence function operates upon the offsets between the clocks in a system to increase the accuracy by reducing or eliminating errors caused by falsetickers. There are two classes of convergence functions, those involving interactive-convergence algorithms and those involving interactive-consistency algorithms. Interactive-convergence algorithms use statistical clustering techniques such as the fault-tolerant average algorithm of [HAL84], the CNV algorithm of [LUN84], the majority-subset algorithm of [MIL85a], the non-Byzantine algorithm of [RIC88], the egocentric algorithm of [SCH86], the intersection algorithm of [MAR85] and [DEC89] and the algorithms in Section 4 of this document.

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

In practice it is not possible to determine the truechimers from the falsetickers on other than a statistical basis, especially with hierarchical configurations and a statistically noisy Internet. While it is possible to bound the maximum errors in the time-transfer procedures, assuming sufficiently generous tolerances are adopted for the hardware components, this generally results in rather poor accuracies and stabilities. The approach taken in the NTP design and its predecessors involves mutually coupled oscillators and maximum-likelihood estimation and clock-selection procedures, together with a design that allows provable assertions on error bounds to be made relative to stated assumptions on the correctness of the primary reference sources. From the analytical point of view, the system of distributed NTP peers operates as a set of coupled phase-locked oscillators, with the update algorithm functioning as a phase detector and the local clock as a disciplined oscillator, but with deterministic error bounds calculated at each step in the time-transfer process.

The particular choice of offset measurement and computation procedure described in Section 3 is a variant of the returnable-time system used in some digital telephone networks [LIN80].

The clock filter and selection algorithms are designed so that the clock synchronization subnet self-organizes into a hierarchical-master-slave configuration [MIT80]. With respect to timekeeping accuracy and stability, the similarity of NTP to digital telephone systems is not accidental, since systems like this have been studied extensively [LIN80], [BRA80]. What makes the NTP model unique is the adaptive configuration, polling, filtering, selection and correctness mechanisms which tailor the dynamics of the system to fit the ubiquitous Internet environment.

6.4 2. System Architecture

In the NTP model a number of primary reference sources, synchronized by wire or radio to national standards, are connected to widely accessible resources, such as backbone gateways, and operated as primary time servers. The purpose of NTP is to convey timekeeping information from these servers to other time servers via the Internet and also to cross-check clocks and mitigate errors due to equipment or propagation failures. Some number of local-net hosts or gateways, acting as secondary time servers, run NTP with one or more of the primary servers. In order to reduce the protocol overhead, the secondary servers distribute time via NTP to the remaining local-net hosts. In the interest of reliability, selected hosts can be equipped with less accurate but less expensive radio clocks and used for backup in case of failure of the primary and/or secondary servers or communication paths between them.

Throughout this document a standard nomenclature has been adopted: the stability of a clock is how well it can maintain a constant frequency, the accuracy is how well its frequency and time compare with national standards and the precision is how precisely these quantities can be maintained within a particular timekeeping system. Unless indicated otherwise, the offset of two clocks is the time difference between them, while the skew is the frequency difference (first derivative of offset with time) between them. Real clocks exhibit some variation in skew (second derivative of offset with time), which is called drift; however, in this version of the specification the drift is assumed zero.

NTP is designed to produce three products: clock offset, roundtrip delay and dispersion, all of which are relative to a selected reference clock. Clock offset represents the amount to adjust the local clock to bring it into correspondence with the reference clock. Roundtrip delay provides the capability to launch a message to arrive at the reference clock at a specified time. Dispersion represents the maximum error of the local clock relative to the reference clock. Since most host time servers will synchronize via another peer time server, there are two components in each of these three products, those determined by the peer relative to the primary reference source of standard time and those measured by the host relative to the peer. Each of these components are maintained separately in the protocol in order to facilitate error control and management of the subnet itself. They provide not only precision measurements of offset and delay, but also definitive maximum error bounds, so that the user interface can determine not only the time, but the quality of the time as well.

There is no provision for peer discovery or virtual-circuit management in NTP. Data integrity is provided by the IP and UDP checksums. No flow-control or retransmission facilities are

provided or necessary. Duplicate detection is inherent in the processing algorithms. The service can operate in a symmetric mode, in which servers and clients are indistinguishable, yet maintain a small amount of state information, or in client/server mode, in which servers need maintain no state other than that contained in the client request. A lightweight association-management capability, including dynamic reachability and variable poll-rate mechanisms, is included only to manage the state information and reduce resource requirements. Since only a single NTP message format is used, the protocol is easily implemented and can be used in a variety of solicited or unsolicited polling mechanisms.

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

6.5 2.1 Implementation Model

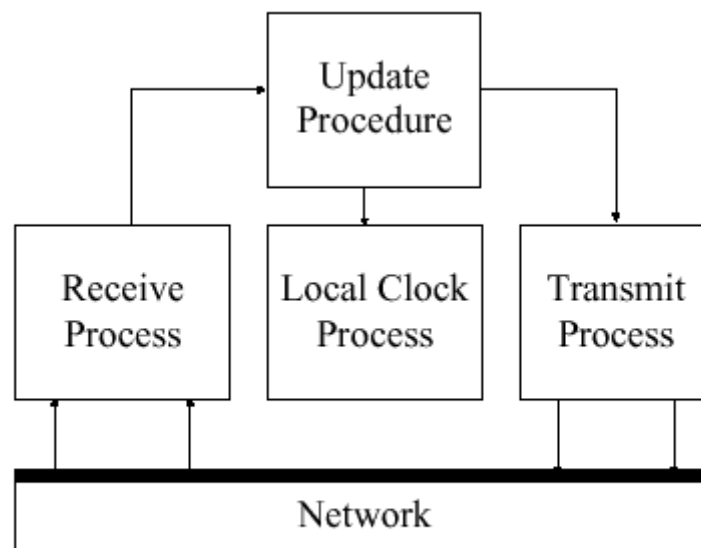


Figure 1. Implementation Model

In what may be the most common client/server model a client sends an NTP message to one or more servers and processes the replies as received. The server interchanges addresses and ports, overwrites certain fields in the message, recalculates the checksum and returns the message immediately. Information included in the NTP message allows the client to determine the server time with respect to local time and adjust the local clock accordingly. In addition, the message includes information to calculate the expected timekeeping accuracy and

reliability, as well as select the best from possibly several servers.

While the client/server model may suffice for use on local nets involving a public server and perhaps many workstation clients, the full generality of NTP requires distributed participation of a number of client/servers or peers arranged in a dynamically reconfigurable, hierarchically distributed configuration. It also requires sophisticated algorithms for association management, data manipulation and local-clock control. Throughout the remainder of this document the term host refers to an instantiation of the protocol on a local processor, while the term peer refers to the instantiation of the protocol on a remote processor connected by a network path.

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

The receive process receives NTP messages and perhaps messages in other protocols, as well as information from directly connected radio clocks. When an NTP message is received, the offset between the peer clock and the local clock is computed and incorporated into the data base along with other information useful for error determination and peer selection. A filtering algorithm described in [Section 4](#) improves the accuracy by discarding inferior data.

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

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

6.6 2.2 Network Configurations

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

accuracy at successive levels toward the leaves.

Following conventions established by the telephone industry [BEL86], the accuracy of each server is defined by a number called the stratum, with the topmost level (primary servers) assigned as one and each level downwards (secondary servers) in the hierarchy assigned as one greater than the preceding level. With current technology and available radio clocks, single-sample accuracies in the order of a millisecond can be achieved at the network interface of a primary server. Accuracies of this order require special care in the design and implementation of the operating system and the local-clock mechanism, such as described in [Section 5](#).

As the stratum increases from one, the single-sample accuracies achievable will degrade depending on the network paths and local-clock stabilities. In order to avoid the tedious calculations [BRA80] necessary to estimate errors in each specific configuration, it is useful to assume the mean measurement errors accumulate approximately in proportion to the measured delay and dispersion relative to the root of the synchronization subnet. Appendix H contains an analysis of errors, including a derivation of maximum error as a function of delay and dispersion, where the latter quantity depends on the precision of the timekeeping system, frequency tolerance of the local clock and various residuals. Assuming the primary servers are synchronized to standard time within known accuracies, this provides a reliable, deterministic specification on timekeeping accuracies throughout the synchronization subnet.

Again drawing from the experience of the telephone industry, which learned such lessons at considerable cost [ABA89], the synchronization subnet topology should be organized to produce the highest accuracy, but must never be allowed to form a loop. An additional factor is that each increment in stratum involves a potentially unreliable time server which introduces additional measurement errors. The selection algorithm used in NTP uses a variant of the Bellman-Ford distributed routing algorithm [37] to compute the minimum-weight spanning trees rooted on the primary servers. The distance metric used by the algorithm consists of the (scaled) stratum plus the synchronization distance, which itself consists of the dispersion plus one-half the absolute delay. Thus, the synchronization path will always take the minimum number of servers to the root, with ties resolved on the basis of maximum error.

As a result of this design, the subnet reconfigures automatically in a hierarchical-master-slave configuration to produce the most accurate and reliable time, even when one or more primary or secondary servers or the network paths between them fail. This includes the case where all normal primary servers (e.g., highly accurate WWVB radio clock operating at the lowest synchronization distances) on a possibly partitioned subnet fail, but one or more backup primary servers (e.g., less accurate WWV radio clock operating at higher synchronization distances) continue operation. However, should all primary servers throughout the subnet fail, the remaining secondary servers will synchronize among themselves while distances ratchet upwards to a preselected maximum infinity due to the well-known properties of the Bellman-Ford algorithm. Upon reaching the maximum on all paths, a server will drop off the subnet and free-run using its last determined time and frequency. Since these computations are expected to be very precise, especially in frequency, even extended outage periods can result in timekeeping errors not greater than a few milliseconds per day with appropriately stabilized oscillators (see [Section 5](#)).

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

6.7 3. Network Time Protocol

This section consists of a formal definition of the Network Time Protocol, including its data formats, entities, state variables, events and event-processing procedures. The specification is based on the implementation model illustrated in [Figure 1](#), but it is not intended that this model is the only one upon which a specification can be based. In particular, the specification is intended to illustrate and clarify the intrinsic operations of NTP, as well as to serve as a foundation for a more rigorous, comprehensive and verifiable specification.

6.8 3.1 Data Formats

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

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

Timestamps are determined by copying the current value of the local clock to a timestamp when some significant event, such as the arrival of a message, occurs. In order to maintain the highest accuracy, it is important that this be done as close to the hardware or software driver associated with the event as possible. In particular, departure timestamps should be redetermined for each link-level retransmission. In some cases a particular timestamp may not be available, such as when the host is rebooted or the protocol first starts up. In these cases the 64-bit field is set to zero, indicating the value is invalid or undefined.

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

6.9 3.2 State Variables and Parameters

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

6.10 3.2.1 Common Variables

The following variables are common to two or more of the system, peer and packet classes. Additional variables are specific to the optional authentication mechanism as described in Appendix C. When necessary to distinguish between common variables of the same name, the variable identifier will be used.

Peer Address (peer.peeraddr, pkt.peeraddr), Peer Port (peer.peerport, pkt.peerport): These are the 32-bit Internet address and 16-bit port number of the peer.

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

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

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

In all except the alarm condition (112), NTP itself does nothing with these bits, except pass them on to the time-conversion routines that are not part of NTP. The alarm condition occurs when, for whatever reason, the local clock is not synchronized, such as when first coming up or after an extended period when no primary reference source is available.

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

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

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

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

For comparison purposes a value of zero is considered greater than any other value. Note that the maximum value of the integer encoded as a packet variable is limited by the parameter NTP.MAXSTRATUM.

Poll Interval (sys.poll, peer.hostpoll, peer.peerpoll, pkt.poll): This is a signed integer indicating the minimum interval between transmitted messages, in seconds as a power of two. For instance, a value of six indicates a minimum interval of 64 seconds.

Precision (sys.precision, peer.precision, pkt.precision): This is a signed integer indicating the

precision of the various clocks, in seconds to the nearest power of two. The value must be rounded to the next larger power of two; for instance, a 50-Hz (20 ms) or 60-Hz (16.67ms) power-frequency clock would be assigned the value -5 (31.25 ms), while a 1000-Hz (1 ms) crystal-controlled clock would be assigned the value -9 (1.95 ms).

Root Delay (sys.rootdelay, peer.rootdelay, pkt.rootdelay): This is a signed fixed-point number indicating the total roundtrip delay to the primary reference source at the root of the synchronization subnet, in seconds. Note that this variable can take on both positive and negative values, depending on clock precision and skew.

Root Dispersion (sys.rootdispersion, peer.rootdispersion, pkt.rootdispersion): This is a signed fixed-point number indicating the maximum error relative to the primary reference source at the root of the synchronization subnet, in seconds. Only positive values greater than zero are possible.

Reference Clock Identifier (sys.refid, peer.refid, pkt.refid): This is a 32-bit code identifying the particular reference clock. In the case of stratum 0 (unspecified) or stratum 1 (primary reference source), this is a four-octet, left-justified, zero-padded ASCII string, for example (see Appendix A for comprehensive list):

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

the case of stratum 2 and greater (secondary reference) this is the four-octet Internet address of the peer selected for synchronization.

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

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

Receive Timestamp (peer.rec, pkt.rec): This is the local time, in timestamp format, when the

latest NTP message from the peer arrived. If the peer becomes unreachable the value is set to zero.

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

6.11 3.2.2 System Variables

System Variables	Name	Procedure
Leap Indicator	sys.leap	clock update
Stratum	sys.stratum	clock update
Precision	sys.precision	system
Root Delay	sys.rootdelay	clock update
Root Dispersion	sys.rootdispersion	clock update
Reference Clock Ident	sys.refid	clock update
Reference Timestamp	sys.reftime	clock update
Local Clock	sys.clock	clock update
Clock Source	sys.peer	selection
Poll Interval	sys.poll	local clock
Cryptographic Keys	sys.keys	authentication

Table 1. System Variables

Table 1 shows the complete set of system variables. In addition to the common variables described previously, the following variables are used by the operating system in order to synchronize the local clock.

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

Clock Source (sys.peer): This is a selector identifying the current synchronization source. Usually this will be a pointer to a structure containing the peer variables. The special value NULL indicates there is no currently valid synchronization source.

6.12 3.2.3 Peer Variables

Peer Variables	Name	Procedure
Configured Bit	peer.config	initialization
Peer Address	peer.peeraddress	receive
Peer Port	peer.peerport	receive
Host Address	peer.hostaddress	receive
Host Port	peer.hostport	receive
Leap Indicator	peer.leap	packet
Mode	peer.mode	packet
Stratum	peer.stratum	packet
Peer Poll Interval	peer.peerpoll	packet
Host Poll Interval	peer.hostpoll	poll update
Precision	peer.precision	packet
Root Delay	peer.rootdelay	packet
Root Dispersion	peer.rootdispersion	packet
Reference Clock Ident	peer.refid	packet
Reference Timestamp	peer.reftime	packet
Originate Timestamp	peer.org	packet, clear
Receive Timestamp	peer.rec	packet, clear
Transmit Timestamp	peer.xmt	transmit, clear
Update Timestamp	peer.update	filter, clear
Reachability Register	peer.reach	packet, transmit,clear
Peer Timer	peer.timer	receive, transmit,poll update
Filter Register	peer.filter	filter
Valid Data Counter	peer.valid	transmit
Delay	peer.delay	filter
Offset	peer.offset	filter
Dispersion	peer.dispersion	filter
Authentic Enable Bit	peer.authenable	authentication
Authenticated Bit	peer.authentic	authentication
Host Key Identifier	peer.hostkeyid	authentication
Peer Key Identifier	peer.peerkeyid	authentication

2. Peer Variables

Table 2 shows the complete set of peer variables. In addition to the common variables described previously, the following variables are used by the peer management and measurement functions.

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

Update Timestamp (peer.update): This is the local time, in timestamp format, when the most recent NTP message was received. It is used in calculating the skew dispersion.

Reachability Register (peer.reach): This is a shift register of NTP.WINDOW bits used to determine the reachability status of the peer, with bits entering from the least significant (rightmost) end. A peer is considered reachable if at least one bit in this register is set to one.

Peer Timer (peer.timer): This is an integer counter used to control the interval between transmitted NTP messages. Once set to a nonzero value, the counter decrements at one-second intervals until reaching zero, at which time the transmit procedure is called. Note that the operation of this timer is independent of local-clock updates, which implies that the timekeeping system and interval-timer system architecture must be independent of each other.

6.13 3.2.4 Packet Variables

Packet Variables	Name	Procedure
Peer Address	pkt.peeraddress	transmit
Peer Port	pkt.peerport	transmit
Host Address	pkt.hostaddress	transmit
Host Port	pkt.hostport	transmit
Leap Indicator	pkt.leap	transmit
Version Number	pkt.version	transmit
Mode	pkt.mode	transmit
Stratum	pkt.stratum	transmit
Poll Interval	pkt.poll	transmit
Precision	pkt.precision	transmit
Root Delay	pkt.rootdelay	transmit
Root Dispersion	pkt.rootdispersion	transmit
Reference Clock Ident	pkt.refid	transmit
Reference Timestamp	pkt.reftime	transmit
Originate Timestamp	pkt.org	transmit
Receive Timestamp	pkt.rec	transmit
Transmit Timestamp	pkt.xmt	transmit
Key Identifier	pkt.keyid	authentication
Crypto-Checksum	pkt.check	authentication

Table 3. Packet Variables

Table 3 shows the complete set of packet variables. In addition to the common variables described previously, the following variables are defined.

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

6.14 3.2.5 Clock-Filter Variables

When the filter and selection algorithms suggested in Section 4 are used, the following state variables are defined in addition to the variables described previously.

Filter Register (peer.filter): This is a shift register of NTP.SHIFT stages, where each stage stores a 3-tuple consisting of the measured delay, measured offset and calculated dispersion associated with a single observation. These 3-tuples enter from the most significant (leftmost) right and are shifted towards the least significant (rightmost) end and eventually discarded as new observations arrive.

Valid Data Counter (peer.valid): This is an integer counter indicating the valid samples remaining in the filter register. It is used to determine the reachability state and when the poll interval should be increased or decreased.

Offset (peer.offset): This is a signed, fixed-point number indicating the offset of the peer clock relative to the local clock, in seconds.

Delay (peer.delay): This is a signed fixed-point number indicating the roundtrip delay of the peer clock relative to the local clock over the network path between them, in seconds. Note that this variable can take on both positive and negative values, depending on clock precision and skew-error accumulation.

Dispersion (peer.dispersion): This is a signed fixed-point number indicating the maximum error of the peer clock relative to the local clock over the network path between them, in seconds. Only positive values greater than zero are possible.

6.15 3.2.6 Authentication Variables

When the authentication mechanism suggested in [Appendix C](#) is used, the following state variables are defined in addition to the variables described previously. These variables are used only if the optional authentication mechanism described in [Appendix C](#) is implemented.

Authentication Enabled Bit (peer.authenable): This is a bit indicating that the association is to operate in the authenticated mode.

Authenticated Bit (peer.authentic): This is a bit indicating that the last message received from the peer has been correctly authenticated.

Key Identifier (peer.hostkeyid, peer.peerkeyid, pkt.keyid): This is an integer identifying the cryptographic key used to generate the message-authentication code.

Cryptographic Keys (sys.key): This is a set of 64-bit DES keys. Each key is constructed as in the Berkeley Unix distributions, which consists of eight octets, where the seven low-order bits of each octet correspond to the DES bits 1-7 and the high-order bit corresponds to the DES

odd-parity bit 8.

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

6.16 3.2.7 Parameters

Parameters	Name	Value
Version Number	NTP.VERSION	3
NTP Port	NTP.PORT	123
Max Stratum	NTP.MAXSTRATUM	15
Max Clock Age	NTP.MAXAGE	86,400 sec
Max Skew	NTP.MAXSKEW	1 sec
Max Distance	NTP.MAXDISTANCE	1 sec
Min Polling Interval	NTP.MINPOLL	6 (64 sec)
Max Polling Interval	NTP.MAXPOLL	10 (1024 sec)
Min Select Clocks	NTP.MINCLOCK	1
Max Select Clocks	NTP.MAXCLOCK	10
Min Dispersion	NTP.MINDISPERSE	.01 sec
Max Dispersion	NTP.MAXDISPERSE	16 sec
Reachability Reg Size	NTP.WINDOW	8
Filter Size	NTP.SHIFT	8
Filter Weight	NTP.FILTER	1/2
Select Weight	NTP.SELECT	3/4

Table 4. Parameters

Table 4 shows the parameters assumed for all implementations operating in the Internet system. It is necessary to agree on the values for these parameters in order to avoid unnecessary network overheads and stable peer associations. The following parameters are assumed fixed and applicable to all associations.

Version Number (NTP.VERSION): This is the current NTP version number (3).

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

Maximum Stratum (NTP.MAXSTRATUM): This is the maximum stratum value that can be

encoded as a packet variable, also interpreted as infinity or unreachable by the subnet routing algorithm.

Maximum Clock Age (NTP.MAXAGE): This is the maximum interval a reference clock will be considered valid after its last update, in seconds.

Maximum Skew (NTP.MAXSKEW): This is the maximum offset error due to skew of the local clock over the interval determined by NTP.MAXAGE, in seconds. The ratio NTP.MAXSKEW over NTP.MAXAGE is interpreted as the maximum possible skew rate due to all causes.

Maximum Distance (NTP.MAXDISTANCE): When the selection algorithm suggested in Section 4 is used, this is the maximum synchronization distance for peers acceptable for synchronization.

Minimum Poll Interval (NTP.MINPOLL): This is the minimum poll interval allowed by any peer of the Internet system, in seconds to a power of two.

Maximum Poll Interval (NTP.MAXPOLL): This is the maximum poll interval allowed by any peer of the Internet system, in seconds to a power of two.

Minimum Select Clocks (NTP.MINCLOCK): When the selection algorithm suggested in [Section 4](#) is used, this is the minimum number of peers acceptable for synchronization.

Maximum Select Clocks (NTP.MAXCLOCK): When the selection algorithm suggested in [Section 4](#) is used, this is the maximum number of peers considered for selection.

Minimum Dispersion (NTP.MINDISPERSE): When the filter algorithm suggested in [Section 4](#) is used, this is the minimum dispersion increment for each stratum level, in seconds.

Maximum Dispersion (NTP.MAXDISPERSE): When the filter algorithm suggested in [Section 4](#) is used, this is the maximum peer dispersion and the dispersion assumed for missing data, in seconds.

Reachability Register Size (NTP.WINDOW): This is the size of the reachability register (peer.reach), in bits.

Filter Size (NTP.SHIFT): When the filter algorithm suggested in [Section 4](#) is used, this is the size of the clock filter (peer.filter) shift register, in stages. **Filter Weight** (NTP.FILTER): When the filter algorithm suggested in [Section 4](#) is used, this is the weight used to compute the filter dispersion.

Select Weight (NTP.SELECT): When the selection algorithm suggested in [Section 4](#) is used, this is the weight used to compute the select dispersion.

6.17 3.3 Modes of Operation

Except in broadcast mode, an NTP association is formed when two peers exchange messages and one or both of them create and maintain an instantiation of the protocol machine, called an association. The association can operate in one of five modes as indicated by the host-mode variable (peer.mode): symmetric active, symmetric passive, client, server and broadcast, which are defined as follows:

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

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

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

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

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

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

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

can be reused when the peer becomes unreachable or is operating at a higher stratum level and thus ineligible as a synchronization source.

Symmetric active mode is intended for use by time servers operating near the end nodes (highest stratum) of the synchronization subnet. Reliable time service can usually be maintained with two peers at the next lower stratum level and one peer at the same stratum level, so the rate of ongoing polls is usually not significant, even when connectivity is lost and error messages are being returned for every poll.

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

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

6.18 3.4 Event Processing

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

6.19 3.4.1 Notation Conventions

The NTP filtering and selection algorithms act upon a set of variables for clock offset θ , THETA, roundtrip delay δ , DELTA and dispersion ϵ , EPSILON. When necessary to distinguish between them, lower-case Greek letters are used for variables relative to a peer, while upper-case Greek letters are used for variables relative to the primary reference source(s), i.e., via the peer to the root of the synchronization subnet. Subscripts will be used to identify the particular peer when this is not clear from context. The algorithms are based on a quantity called the synchronization distance λ , LAMBDA, which is computed from the roundtrip delay and dispersion as described below.

As described in [Appendix H](#), the peer dispersion ϵ includes contributions due to measurement error $p = 1 \ll \text{sys.precision}$, skew-error accumulation $\phi \tau$, where $\phi =$

NTP.MAXSKEW over NTP.MAXAGE is the maximum skew rate and $\tau = \text{sys.clock} - \text{peer.update}$ is the interval since the last update, and filter (sample) dispersion $\epsilon_{\text{sub}} \sigma$ computed by the clock-filter algorithm. The root dispersion EPSILON includes contributions due to the selected peer dispersion ϵ and skew-error accumulation $\phi \tau$, together with the root dispersion for the peer itself. The system dispersion includes the select (sample) dispersion $\epsilon_{\text{sub}} \xi$ computed by the clock-select algorithm and the absolute initial clock offset $|\text{THETA}|$ provided to the local-clock algorithm. Both ϵ and EPSILON are dynamic quantities, since they depend on the elapsed time τ since the last update, as well as the sample dispersions calculated by the algorithms.

Each time the relevant peer variables are updated, all dispersions associated with that peer are updated to reflect the skew-error accumulation. The computations can be summarized as follows:

```
theta == peer.offset,
delta == peer.delay,
epsilon == peer.dispersion = rho + phi tau + epsilon sub sigma,
lambda == epsilon + | delta | over 2,
```

where τ is the interval since the original timestamp (from which θ and δ were determined) was transmitted to the present time and $\epsilon_{\text{sub}} \sigma$ is the filter dispersion (see clock-filter procedure below). The variables relative to the root of the synchronization subnet via peer i are determined as follows:

```
THETA sub i == theta sub i ,
DELTA sub i == peer.rootdelay + delta sub i,
EPSILON sub i == peer.rootdispersion + epsilon sub i + phi tau sub i ,
LAMBDA sub i == EPSILON sub i + | DELTA sub i | over 2 ,
```

where all variables are understood to pertain to the i th peer. Finally, assuming the i th peer is selected for synchronization, the system variables are determined as follows:

```
THETA = combined final offset ,
DELTA = DELTA sub i,
EPSILON = EPSILON sub i + epsilon sub xi + | THETA | ,
LAMBDA = LAMBDA sub i ,
```

where $\epsilon_{\text{sub}} \xi$ is the select dispersion (see clock-selection procedure below).

Informal pseudo-code which accomplishes these computations is presented below. Note that the pseudo-code is represented in no particular language, although it has many similarities to the C language. Specific details on the important algorithms are further illustrated in the C-language routines in [Appendix I](#).

6.20 3.4.2 Transmit Procedure

The transmit procedure is executed when the peer timer decrements to zero for all modes except client mode with a broadcast server and server mode in all cases. In client mode with a broadcast server messages are never sent. In server mode messages are sent only in response to received messages. This procedure is also called by the receive procedure when an NTP message arrives that does not result in a persistent association.

begin transmit procedure

The following initializes the packet buffer and copies the packet variables. The value skew is necessary to account for the skew-error accumulated over the interval since the local clock was last set.

```

pkt.peeraddr <-- peer.hostaddr;      /* copy system and peer variables */
pkt.peerport <-- roman peer.hostport;
pkt.hostaddr <-- roman peer.peeraddr;
pkt.hostport <-- roman peer.peerport;
pkt.leap <-- roman sys.leap;
pkt.version <-- roman NTP.VERSION;
pkt.mode <-- roman peer.mode;
pkt.stratum <-- roman sys.stratum;
pkt.poll <-- roman peer.hostpoll;
pkt.precision <-- roman sys.precision;
pkt.rootdelay <-- roman sys.rootdelay;
if (sys.leap = 112 or (sys.clock - sys.reftime) > NTP.MAXAGE)
    skew <-- NTP.MAXSKEW;
else
    skew <-- phi (sys.clock~~sys.reftime);
pkt.rootdispersion <-- sys.rootdispersion + (1 << sys.precision) +skew;
pkt.refid <-- sys.refid;
pkt.reftime <-- sys.reftime;

```

The transmit timestamp pkt.xmt will be used later in order to validate the reply; thus, implementations must save the exact value transmitted. In addition, the order of copying the timestamps should be designed so that the time to format and copy the data does not degrade accuracy.

```

pkt.org <-- peer.org;                /* copy timestamps */
pkt.rec <-- peer.rec;
pkt.xmt <-- sys.clock;
peer.xmt <-- pkt.xmt;

```

The call to encrypt is implemented only if authentication is implemented. If authentication is enabled, the delay to encrypt the authenticator may degrade accuracy. Therefore, implementations should include a system state variable (not mentioned elsewhere in this specification) which contains an offset calculated to match the expected encryption delay and

correct the transmit timestamp as obtained from the local clock.

```
#ifdef (authentication implemented)    /* see Appendix C */
    call encrypt;
#else
    send packet;
```

The reachability register is shifted one position to the left, with zero replacing the vacated bit. If all bits of this register are zero, the clear procedure is called to purge the clock filter and reselect the synchronization source, if necessary. If the association was not configured by the initialization procedure, the association is demobilized.

```
peer.reach <-- peer.reach << 1;        /* update reachability */
if (peer.reach = 0 and peer.config = 0) begin
    demobilize association;
    exit;
endif
```

If valid data have been shifted into the filter register at least once during the preceding two poll intervals (low-order bit of peer.reach set to one), the valid data counter is incremented. After eight such valid intervals the poll interval is incremented. Otherwise, the valid data counter and poll interval are both decremented and the clock-filter procedure called with zero values for offset and

peer.mode → mode ↓	sym act 1	sym pas 2	client 3	server 4	bcst 5
sym active	recv	pkt	recv ²	xmit ²	xmit ^{1,2}
sym passive	recv	error	recv ²	error	error
client	xmit ²	xmit ²	error	xmit	xmit ¹
server	recv ²	error	recv	error	error
broadcast	recv ^{1,2}	error	recv ¹	error	error

Notes:

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

Table 5. Modes and Actions

delay and NTP.MAXDISPERSE for dispersion. The clock-select procedure is called to reselect the synchronization source, if necessary.

```
if (peer.reach & 6 !=0)                /* test two low-order bits (shifted) */
```

```

        if (peer.valid < NTP.SHIFT)      /* valid data received */
            peer.valid <-- peer.valid + 1;
            else peer.hostpoll <-- peer.hostpoll + 1;
    else begin
        peer.valid <-- peer.valid - 1;    /* nothing heard */
        peer.hostpoll <-- peer.hostpoll - 1;
        call clock-filter(0, 0, NTP.MAXDISPERSE);
        call clock-select;                /* select clock source */
    endif
    call poll-update;
end transmit procedure;

```

6.21 3.4.3 Receive Procedure

The receive procedure is executed upon arrival of an NTP message. It validates the message, interprets the various modes and calls other procedures to filter the data and select the synchronization source. If the version number in the packet does not match the current version, the message may be discarded; however, exceptions may be advised on a case-by-case basis at times when the version is changed. If the NTP control messages described in [Appendix B](#) are implemented and the packet mode is 6 (control), the control-message procedure is called. The source and destination Internet addresses and ports in the IP and UDP headers are matched to the correct peer. If there is no match a new instantiation of the protocol machine is created and the association mobilized.

begin receive procedure

```

    if ( pkt.version != NTP.VERSION) exit;
    #ifdef (control messages implemented)
        if ( pkt.mode = 6) call control-message;
    #endif
    for (all associations)      /* access control goes here */
        match addresses and ports to associations;
    if (no matching association)
        call receive-instantiation procedure; /* create association */

```

The call to decrypt is implemented only if authentication is implemented.

```

    #ifdef (authentication implemented) /* see Appendix C */
        call decrypt;
    #endif

```

If the packet mode is nonzero, this becomes the value of mode used in the following step; otherwise, the peer is an old NTP version and mode is determined from the port numbers as described in [Section 3.3](#).

```

    if (pkt.mode = 0)                /* for compatibility with old versions */

```

```

        mode <-- (see Section 3.3);
    else
        mode <-- pkt.mode;

```

Table 5 shows for each combination of peer.mode and mode the resulting case labels.

```

    case (mode, peer.hostmode)          /* see Table 5 */

```

If error the packet is simply ignored and the association demobilized, if not previously configured.

```

error:    if (peer.config = 0) demobilize association; /* see no evil */
          break;

```

If rcv the packet is processed and the association marked reachable if tests five through eight (valid header) enumerated in the packet procedure succeed. If, in addition, tests one through four succeed (valid data), the clock-update procedure is called to update the local clock. Otherwise, if the association was not previously configured, it is demobilized.

```

rcv:      call packet;                  /* process packet */
          if (valid header) begin        /* if valid header, update local clock */
              peer.reach <-- peer.reach | 1;
              if (valid data) call clock-update;
              endif
          else
              if (peer.config = 0) demobilize association;
              break;

```

If xmit the packet is processed and an immediate reply is sent. The association is then demobilized if not previously configured.

```

xmit:      call packet;                  /* process packet */
          <$Eroman peer.hostpoll~<<~roman peer.peerpoll>;          /* send immediate
reply */
          call poll-update;
          call transmit;
          if (<$Eroman peer.config~==0>) demobilize association;
          break;

```

If pkt the packet is processed and the association marked reachable if tests five through eight (valid header) enumerated in the packet procedure succeed. If, in addition, tests one through four succeed (valid data), the clock-update procedure is called to update the local clock. Otherwise, if the association was not previously configured, an immediate reply is sent and the association demobilized.

```

pkt:      call packet;                  /* process packet */
          if (valid header) begin        /* if valid header, update local
clock */

```

```

        peer.reach <-- peer.reach | 1>;
        if (valid data) call clock-update;
        endif
    else if ( peer.config = 0) begin
        peer.hostpoll <-- peer.peerpoll;      /* send immediate reply */
        call poll-update;
        call transmit;
        demobilize association;
        endif
    endcase
end receive procedure;

```

6.22 3.4.4 Packet Procedure

The packet procedure checks the message validity, computes delay/offset samples and calls other procedures to filter the data and select the synchronization source. Test 1 requires the transmit timestamp not match the last one received from the same peer; otherwise, the message might be an old duplicate. Test 2 requires the originate timestamp match the last one sent to the same peer; otherwise, the message might be out of order, bogus or worse. In case of broadcast mode (5) the apparent roundtrip delay will be zero and the full accuracy of the time-transfer operation may not be achievable. However, the accuracy achieved may be adequate for most purposes. The poll-update procedure is called with argument peer.hostpoll (peer.peerpoll may have changed).

```

begin packet procedure
    peer.rec <-- sys.clock;      /* capture receive timestamp */
    if (pkt.mode != 5) begin
        test1 <-- ( pkt.xmt != peer.org); /* test 1 */

```

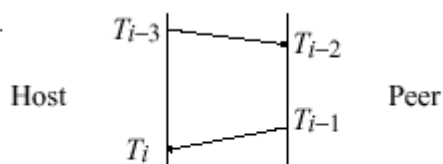


Figure 2. Calculating Delay and Offset

```

        test2 <-- (pkt.org = peer.xmt);      /* test 2 */
        endif
    else begin
        pkt.org <-- peer.rec;                  /* fudge missing timestamps */
        pkt.rec <-- pkt.xmt;
        test1 <-- true;                        * fake tests */
        test2 <-- true;
        endif

```

```

peer.org <-- pkt.xmt;           /* update originate timestamp */
peer.peerpoll <-- pkt.poll;     /* adjust poll interval */
call poll-update(peer.hostpoll);

```

Test 3 requires that both the originate and receive timestamps are nonzero. If either of the timestamps are zero, the association has not synchronized or has lost reachability in one or both directions.

```

test3 <-- ( pkt.org != 0 and pkt.rec != 0); /* test 3 */

```

The roundtrip delay and clock offset relative to the peer are calculated as follows. Number the times of sending and receiving NTP messages as shown in Figure 2<fig2> and let i be an even integer. Then T_{i-3} , T_{i-2} , T_{i-1} and T_i are the contents of the `pkt.org`, `pkt.rec`, `pkt.xmt` and `peer.rec` variables, respectively. The clock offset θ , roundtrip delay δ and dispersion ϵ of the host relative to the peer is:

$$\begin{aligned}\delta &= (T_i - T_{i-3}) - (T_{i-1} - T_{i-2}), \\ \theta &= \frac{(T_{i-2} - T_{i-3}) + (T_{i-1} - T_i)}{2}, \\ \epsilon &= (1 \ll \text{sys.precision}) + \phi(T_i - T_{i-3}),\end{aligned}$$

where, as before, $\phi = \text{NTP.MAXSKEW} / \text{NTP.MAXAGE}$. The quantity ϵ represents the maximum error or dispersion due to measurement error at the host and local-clock skew accumulation over the interval since the last message was transmitted to the peer. subsequently, the dispersion will be updated by the clock-filter procedure.

The above method amounts to a continuously sampled, returnable-time system, which is used in some digital telephone networks [BEL86]. Among the advantages are that the order and timing of the messages are unimportant and that reliable delivery is not required. Obviously, the accuracies achievable depend upon the statistical properties of the outbound and inbound data paths. Further analysis and experimental results bearing on this issue can be found in [MIL90] and in [Appendix H](#). Test 4 requires that the calculated delay be within reasonable bounds:

```

test4 <-- (|delta| < NTP.MAXDISPERSE and epsilon < NTP.MAXDISPERSE); /* test 4
*/

```

Test 5 is implemented only if the authentication mechanism described in [Appendix C](#) is implemented. It requires either that authentication be explicitly disabled or that the authenticator be present and correct as determined by the decrypt procedure.

```

#ifdef (authentication implemented) /* test 5 */
    <test5 <-- ( (peer.config = 1 and peer.authenable = 0) or peer.authentic = 1);
#endif

```


Test 6 requires the peer clock be synchronized and the interval since the peer clock was last updated be positive and less than NTP.MAXAGE. Test 7 insures that the host will not synchronize on a peer with greater stratum. Test 8 requires that the header contains reasonable values for the pkt.rootdelay and pkt.rootdispersion fields.

```
test6 <-- ( roman pkt.leap != 112 and      /* test 6 */
  pkt.reftime <= pkt.xmt < pkt.reftime + NTP.MAXAGE)
test7 <-- pkt.stratum <= sys.stratum and /* test 7 */
  pkt.stratum < NTP.MAXSTRATUM;
test8 <--( | pkt.rootdelay | < NTP.MAXDISPERSE and /* test 8 */
  pkt.rootdispersion < NTP.MAXDISPERSE);
```

With respect to further processing, the packet includes valid (synchronized) data if tests one through four succeed (test1 & test2 & test3 & test4 = 1), regardless of the remaining tests. Only packets with valid data can be used to calculate offset, delay and dispersion values. The packet includes a valid header if tests five through eight succeed (test5 & test6 & test7 & test8 = 1), regardless of the remaining tests. Only packets with valid headers can be used to determine whether a peer can be selected for synchronization. Note that test1 and test2 are not used in broadcast mode (forced to true), since the originate and receive timestamps are undefined.

The clock-filter procedure is called to produce the delay (peer.delay), offset (peer.offset) and dispersion (peer.dispersion) for the peer. Specification of the clock-filter algorithm is not an integral part of the NTP specification, since there may be other algorithms that work well in practice. However, one found to work well in the Internet environment is described in [Section 4](#) and its use is recommended.

```
if (not valid header) exit;
peer.leap <-- pkt.leap;      /* copy packet variables */
peer.stratum <-- pkt.stratum;
peer.precision <-- pkt.precision;
peer.rootdelay <-- pkt.rootdelay;
peer.rootdispersion <-- pkt.rootdispersion;
peer.refid <-- pkt.refid;
peer.reftime <-- pkt.reftime;
if (valid data) call clock-filter(theta ,delta ,epsilon); /* process sample */
end packet procedure;
```

6.23 3.4.5 Clock-Update Procedure

The clock-update procedure is called from the receive procedure when valid clock offset, delay and dispersion data have been determined by the clock-filter procedure for the current peer. The result of the clock-selection and clock-combining procedures is the final clock correction <\$ETHETA>, which is used by the local-clock procedure to update the local clock. If no candidates survive these procedures, the clock-update procedure exits without doing

anything further.

```
begin clock-update procedure
    call clock-select;           /* select clock source */
    if (sys.peer != peer) exit;
```

It may happen that the local clock may be reset, rather than slewed to its final value. In this case the clear procedure is called for every peer to purge the clock filter, reset the poll interval and reselect the synchronization source, if necessary. Note that the local-clock procedure sets the leap bits sys.leap to <169>unsynchronized<170> 112 in this case, so that no other peer will attempt to synchronize to the host until the host once again selects a peer for synchronization.

The distance procedure calculates the root delay <\$DELTA>, root dispersion <\$EPSILON> and root synchronization distance <\$LAMBDA> via the peer to the root of the synchronization subnet. The host will not synchronize to the selected peer if the distance is greater than NTP.MAXDISTANCE. The reason for the minimum clamp at NTP.MINDISPERSE is to discourage subnet route flaps that can happen with Bellman-Ford algorithms and small roundtrip delays.

```
    LAMBDA andistance (peer);           /* update system variables */
    if (LAMBDA >= NTP.MAXDISTANCE) exit;
    sys.leap <-- peer.leap;
    sys.stratum <-- roman peer.stratum + 1;
    sys.refid <-- peer.peeraddr;
    call local-clock;
    if (local clock reset) begin           /* if reset, clear state variables */
        sys.leap <-- 112;
        for (all peers) call clear;
    endif
    else begin
        sys.peer <-- peer;               /* if not, adjust local clock */
        sys.rootdelay <-- DELTA;
        sys.rootdispersion <-- EPSILON + max ( epsilonxi + | THETA |,
NTP.MINDISPERSE);
    endif
    sys.reftime <-- roman sys.clock;
end clock-update procedure;
```

In some system configurations a precise source of timing information is available in the form of a train of timing pulses spaced at one-second intervals. Usually, this is in addition to a source of timecode information, such as a radio clock or even NTP itself, to number the seconds, minutes, hours and days. In these configurations the system variables are set to refer to the source from which the pulses are derived. For those configurations which support a primary reference source, such as a radio clock or calibrated atomic clock, the stratum is set at one as long as this is the actual synchronization source and whether or not the primary-clock procedure is used.

Specification of the clock-selection and local-clock algorithms is not an integral part of the NTP specification, since there may be other algorithms which provide equivalent performance. However, a clock-selection algorithm found to work well in the Internet environment is described in [Section 4](#), while a local-clock algorithm is described in [Section 5](#) and their use is recommended. The clock-selection algorithm described in [Section 4](#) usually picks the peer at the lowest stratum and minimum synchronization distance among all those available, unless that peer appears to be a falseticker. The result is that the algorithms all work to build a minimum-weight spanning tree relative to the primary reference time servers and thus a hierarchical-master-slave synchronization subnet.

6.24 3.4.6 Primary-Clock Procedure

When a primary reference source such as a radio clock is connected to the host, it is convenient to incorporate its information into the data base as if the clock were represented as an ordinary peer. In the primary-clock procedure the clock is polled once a minute or so and the returned timecode used to produce a new update for the local clock. When `peer.timer` decrements to zero for a primary clock peer, the transmit procedure is not called; rather, the radio clock is polled, usually using an ASCII string specified for this purpose. When a valid timecode is received from the radio clock, it is converted to NTP timestamp format and the peer variables updated. The value of `peer.leap` is set depending on the status of the leap-warning bit in the timecode, if available, or manually by the operator. The value for `peer.peeraddr`, which will become the value of `sys.refid` when the clock-update procedure is called, is set to an ASCII string describing the clock type (see [Appendix A](#)).

```
begin primary-clock-update procedure
    peer.leap <-- "from" radio or operator;           /* copy variables */
    peer.peeraddr <-- ASCII identifier;
    peer.rec <-- radio timestamp;
    peer.reach <-- 1;
    call clock-filter(sys.clock - peer.rec, 0, 1 <= peer.precision); /* process sample */
    call clock-update;                                /* update local clock */
end primary-clock-update procedure;
```

6.25 3.4.7 Initialization Procedures

The initialization procedures are used to set up and initialize the system, its peers and associations.

6.26 3.4.7.1 Initialization Procedure

The initialization procedure is called upon reboot or restart of the NTP daemon. The local clock is presumably undefined at reboot; however, in some equipment an estimate is available from the reboot environment, such as a battery-backed clock/calendar. The precision variable is determined by the intrinsic architecture of the local hardware clock. The authentication variables are used only if the authentication mechanism described in [Appendix C](#) is implemented. The values of these variables are determined using procedures beyond the scope

of NTP itself.

```

begin initialization procedure
    #ifdef (authentication implemented)    /* see Appendix C */
        sys.keys <-- as required;
    #endif;
    sys.leap <-- 112;                      /* copy variables */
    sys.stratum <-- 0(undefined);
    sys.precision <-- host precision;
    sys.rootdelay <-- 0 (undefined);
    sys.rootdispersion <-- 0 (undefined);
    sys.refid <-- 0 (undefined);
    sys.reftime <-- 0 (undefined);
    sys.clock <-- external reference;
    sys.peer <-- NULL;
    sys.poll <-- NTP.MINPOLL;
    for (all configured peers)              /* create configured associations */
        call initialization-instantiation procedure;
end initialization procedure;

```

6.27 3.4.7.2 Initialization-Instantiation Procedure

This implementation-specific procedure is called from the initialization procedure to define an association. The addresses and modes of the peers are determined using information read during the reboot procedure or as the result of operator commands. The authentication variables are used only if the authentication mechanism described in [Appendix C](#) is implemented. The values of these variables are determined using procedures beyond the scope of NTP itself. With the authentication bits set as suggested, only properly authenticated peers can become the synchronization source.

```

begin initialization-instantiation procedure
    peer.config <-- 1;
    #ifdef (authentication implemented)          /* see Appendix C */
        peer.authenable <-- 1 (suggested);
        peer.authentic <-- 0;
        peer.hostkeyid <-- as required;
        peer.peerkeyid <-- 0;
    #endif;
    peer.peeraddr <-- peer IP address;           /* copy variables */
    peer.peerport <-- roman NTP.PORT;
    peer.hostaddr <-- host IP address;
    peer.hostport <-- roman NTP.PORT;
    peer.mode <-- host mode;
    peer.peerpoll <-- 0 (undefined);
    peer.timer <-- 0;
    peer.delay <-- 0 (undefined);

```

```

    peer.offset <-- 0(undefined);
    call clear;
end initialization-instantiation procedure;
/* initialize association */

```

6.28 3.4.7.3 Receive-Instantiation Procedure

The receive-instantiation procedure is called from the receive procedure when a new peer is discovered. It initializes the peer variables and mobilizes the association. If the message is from a peer operating in client mode (3), the host mode is set to server mode (4); otherwise, it is set to symmetric passive mode (2). The authentication variables are used only if the authentication mechanism described in [Appendix C](#) is implemented. If implemented, only properly authenticated non-configured peers can become the synchronization source.

```

begin receive-instantiation procedure
  #ifdef (authentication implemented)
    peer.authenable <-- 0;
    peer.authentic <-- 0;
    peer.hostkeyid <-- as required;
    peer.peerkeyid <-- 0;
  #endif
  peer.config <-- 0;
  peer.peeraddr <-- pkt.peeraddr;
  peer.peerport <-- pkt.peerport;
  peer.hostaddr <-- roman pkt.hostaddr;
  hostport <-- pkt.hostport;
  if (pkt.mode = 3)
    peer.mode <-- 4;
  else
    peer.mode <-- 2;
  peer.peerpoll <-- 0 (undefined);
  peer.timer <-- 0;
  peer.delay <-- 0 (undefined);
  peer.offset <-- 0 (undefined);
  call clear;
end receive-instantiation procedure;
/* see Appendix C */
/* copy variables */
/* determine mode */
/* initialize association */

```

6.29 3.4.7.4 Primary Clock-Instantiation Procedure

This procedure is called from the initialization procedure in order to set up the state variables for the primary clock. The value for peer.precision is determined from the radio clock specification and hardware interface. The value for peer.rootdispersion is nominally ten times the inherent maximum error of the radio clock; for instance, 10 us for a calibrated atomic clock, 10 ms for a WWVB or GOES radio clock and 100 ms for a less accurate WWV radio clock.

```

begin clock-instantiation procedure
    peer.config <-- ~1>;                /* copy variables */
    peer.peeraddr <-- 0 undefined;
    peer.peerport <-- 0 (not used);
    peer.hostaddr <-- 0 (not used);
    peer.hostport <-- 0 (not used);
    peer.leap <-- 112;
    peer.mode <-- 0 (not used);
    peer.stratum <-- 0;
    peer.peerpoll <-- 0 (undefined);
    peer.precision <-- clock precision;
    peer.rootdelay <-- 0;
    peer.rootdispersion <-- clock dispersion;
    peer.refid <-- 0 (not used);
    peer.reftime <-- 0 (undefined);
    peer.timer <-- 0;
    peer.delay <-- 0 (undefined);
    peer.offset <-- 0 (undefined);
    call clear;                          /* initialize association */
end clock-instantiation procedure;

```

In some configurations involving a calibrated atomic clock or LORAN-C receiver, the primary reference source may provide only a seconds pulse, but lack a complete timecode from which the numbering of the seconds, etc., can be derived. In these configurations seconds numbering can be derived from other sources, such as a radio clock or even other NTP peers. In these configurations the primary clock variables should reflect the primary reference source, not the seconds-numbering source; however, if the seconds-numbering source fails or is known to be operating incorrectly, updates from the primary reference source should be suppressed as if it had failed.

6.30 3.4.8 Clear Procedure

The clear procedure is called when some event occurs that results in a significant change in reachability state or potential disruption of the local clock.

```

begin clear procedure
    peer.org <-- 0 (undefined);          /* mark timestamps undefined */
    peer.rec <-- 0 (undefined);
    peer.xmt <-- 0 (undefined);
    peer.reach <-- 0;                    /* reset state variables */
    peer.filter <-- [0, ,0,NTP.MAXDISPERSE]; /* all stages */
    peer.valid <-- 0;
    peer.dispersion <-- NTP.MAXDISPERSE;
    peer.hostpoll <-- NTP.MINPOLL;       /* reset poll interval */
    call poll-update;
    call clock-select;                  /* select clock source */

```

end clear procedure;

6.31 3.4.9 Poll-Update Procedure

The poll-update procedure is called when a significant event occurs that may result in a change of the poll interval or peer timer. It checks the values of the host poll interval (peer.hostpoll) and peer poll interval (peer.peerpoll) and clamps each within the valid range. If the peer is selected for synchronization, the value is further clamped as a function of the computed compliance (see [Section 5](#)).

```
begin poll-update procedure
  temp <-- peer.hostpoll;           /* determine host poll interval */
  if (peer = sys.peer)
    temp <-- min (temp, sys.poll, NTP.MAXPOLL);
  else
    temp <-- min (temp, NTP.MAXPOLL);
  peer.hostpoll <-- max (temp, NTP.MINPOLL);
  temp <-- 1 << min ( peer.hostpoll, max (peer.peerpoll, NTP.MINPOLL));
```

If the poll interval is unchanged and the peer timer is zero, the timer is simply reset. If the poll interval is changed and the new timer value is greater than the present value, no additional action is necessary; otherwise, the peer timer must be reduced. When the peer timer must be reduced it is important to discourage tendencies to synchronize transmissions between the peers. A prudent precaution is to randomize the first transmission after the timer is reduced, for instance by the sneaky technique illustrated.

```
    if (peer.timer = 0)               /* reset peer timer */
      peer.timer <-- temp;
    else if ( peer.timer > temp)
      peer.timer <-- ( sys.clock & (temp -1))+ 1;
end poll-update procedure;
```

6.32 3.5 Synchronization Distance Procedure

The distance procedure calculates the synchronization distance from the peer variables for the peer.

```
begin distance(peer) procedure;
  DELTA <-- peer.rootdelay + |peer.delay|;
  EPSILON <-- peer.rootdispersion + peer.dispersion + phi (sys.clock - peer.update);
  LAMBDA <-- EPSILON + | DELTA | over 2 ;
end distance procedure;
```

Note that, while DELTA may be negative in some cases, both EPSILON and LAMBDA are always positive.

6.33 3.6 Access Control Issues

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

While not a required feature of NTP itself, some implementations may include an access-control feature that prevents unauthorized access and controls which peers are allowed to update the local clock. For this purpose it is useful to distinguish between three categories of access: those that are preauthorized as trusted, preauthorized as friendly and all other (non-preauthorized) accesses. Presumably, preauthorization is accomplished by entries in the configuration file or some kind of ticket-management system such as Kerberos [STE88]. In this model only trusted accesses can result in the peer becoming the synchronization source. While friendly accesses cannot result in the peer becoming the synchronization source, NTP messages and timestamps are returned as specified.

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

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

6.34 4. Filtering and Selection Algorithms

A most important factor affecting the accuracy and reliability of time distribution is the complex of algorithms used to reduce the effect of statistical errors and false ticks due to failure of various subnet components, reference sources or propagation media. The algorithms suggested in this section were developed and refined over several years of operation in the Internet under widely varying topologies, speeds and traffic regimes. While these algorithms are believed the best available at the present time, they are not an integral part of the NTP specification, since other algorithms with similar or superior performance may be devised in future.

However, it is important to observe that not all time servers or clients in an NTP synchronization subnet must implement these algorithms. For instance, simple workstations may dispense with one or both of them in the interests of simplicity if accuracy and reliability requirements justify. Nevertheless, it would be expected that an NTP server providing synchronization to a sizable community, such as a university campus or research laboratory, would be expected to implement these algorithms or others proved to have equivalent functionality. A comprehensive discussion of the design principles and performance is given in [MIL91a].

In order for the NTP filter and selection algorithms to operate effectively, it is useful to have a measure of recent sample variance recorded for each peer. The measure adopted is based on first-order differences, which are easy to compute and effective for the purposes intended. There are two measures, one called the filter dispersion ϵ_{sigma} and the other the select dispersion ϵ_{xi} . Both are computed as the weighted sum of the clock offsets in a temporary list sorted by synchronization distance. If θ_i ($0 \leq i < n$) is the offset of the i^{th} entry, then the sample difference ϵ_{ij} of the i^{th} entry relative to the j^{th} entry is defined $\epsilon_{ij} = |\theta_i - \theta_j|$. The dispersion relative to the j^{th} entry is defined ϵ_j and computed as the weighted sum

$$\epsilon_j = \sum_{i=0}^{n-1} \epsilon_{ij} w^{i+1},$$

where w is a weighting factor chosen to control the influence of synchronization distance in the dispersion budget. In the NTP algorithms w is chosen less than $1/2$: $w = \text{NTP.FILTER}$ for filter dispersion and $w = \text{NTP.SELECT}$ for select dispersion. The (absolute) dispersion ϵ_{sigma} and ϵ_{xi} as used in the NTP algorithms are defined relative to the 0th entry ϵ_{00} .

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

6.35 4.1 Clock-Filter Procedure

The clock-filter procedure is executed upon arrival of an NTP message or other event that results in new data samples. It takes arguments of the form θ , δ , ϵ , where θ is a sample clock offset measurement and δ and ϵ are the associated roundtrip delay and dispersion. It determines the filtered clock offset (peer.offset), roundtrip delay (peer.delay) and dispersion (peer.dispersion). It also updates the dispersion of samples already recorded and saves the current time (peer.update).

The basis of the clock-filter procedure is the filter shift register (peer.filter), which consists of NTP.SHIFT stages, each stage containing a 3-tuple $[\theta_{\text{sub } i}, \delta_{\text{sub } i}, \epsilon_{\text{sub } i}]$, with indices numbered from zero on the left. The filter is initialized with the value $[0, 0, \text{NTP.MAXDISPERSE}]$ in all stages by the clear procedure. New data samples are shifted into the filter at the left end, causing first NULLs then old samples to fall off the right end. The packet procedure provides samples of the form $[\theta, \delta, \epsilon]$ as new updates arrive, while the transmit procedure provides samples of the form $[0, 0, \text{NTP.MAXDISPERSE}]$ when two poll intervals elapse without a fresh update. While the same symbols (θ, δ, ϵ) are used here for the arguments, clock-filter contents and peer variables, the meaning will be clear from context. The following pseudo-code describes this procedure.

begin clock-filter procedure (θ, δ, ϵ)

The dispersion $\epsilon_{\text{sub } i}$ for all valid samples in the filter register must be updated to account for the skew-error accumulation since the last update. These samples are also inserted on a temporary list with entry format $[\text{distance}, \text{index}]$. The samples in the register are shifted right one stage, with the overflow sample discarded and the new sample inserted at the leftmost stage. The temporary list is then sorted by increasing distance. If no samples remain in the list, the procedure exits without updating the peer variables.

```

    for (i from NTP.SIZE - 1 to 1) begin      /* update dispersion */
        [  $\theta_{\text{sub } i}, \delta_{\text{sub } i}, \epsilon_{\text{sub } i}$  ] <-- [  $\theta_{\text{sub } \{i-1\}}, \delta_{\text{sub } \{i-1\}}, \epsilon_{\text{sub } \{i-1\}}$  ];

                                                    /* shift stage right */
         $\epsilon_{\text{sub } i} = \epsilon_{\text{sub } i} + \phi \tau$ ;
        add [  $\lambda_{\text{sub } i} == \epsilon_{\text{sub } i} + \{|\delta_{\text{sub } i}|\} \text{ over } 2, i$  ] to temporary list;
        endfor;
        [  $\theta_{\text{sub } 0}, \delta_{\text{sub } 0}, \epsilon_{\text{sub } 0}$  ] <-- [  $\theta, \delta, \epsilon$  ];      /* insert new
sample */
        add [  $\lambda == \epsilon + \{|\delta|\} \text{ over } 2, 0$  ] to temporary list;
         $\text{peer.update} <-- \text{sys.clock}$ ;      /* reset base time */
        sort temporary list by increasing [distance || index];

```

where $[\text{distance} || \text{index}]$ represents the concatenation of the distance and index fields and distance is the high-order field. The filter dispersion $\epsilon_{\text{sub } \sigma}$ is computed and included in the peer dispersion. Note that for this purpose the temporary list is already sorted.

$\epsilon_{\text{sub } \sigma} <-- 0$;

```

    for (i from NTP.SHIFT - 1 to 0)          /* compute filter dispersion */
        if (peer.dispersion sub index[i] >= NTP.MAXDISPERSE or
            | theta sub i - theta sub 0 | > NTP.MAXDISPERSE
            epsilon sub sigma <-- epsilon sub sigma + NTP.MAXDISPERSE) times
NTP.FILTER;
        else
            epsilon sub sigma <-- ( epsilon sub sigma + | theta sub i - theta sub 0 | ) times
NTP.FILTER;

```

The peer offset theta sub 0, delay delta sub 0 and dispersion epsilon sub 0 are chosen as the values corresponding to the minimum-distance sample; in other words, the sample corresponding to the first entry on the temporary list, here represented as the 0th subscript.

```

    peer.offset <-- theta sub 0;              /* update peer variables */
    peer.delay <-- delta sub 0;
    peer.dispersion <-- min ( epsilon sub 0 + epsilon sub sigma , NTP.MAXDISPERSE);
end clock-filter procedure

```

The peer.offset and peer.delay variables represent the clock offset and roundtrip delay of the local clock relative to the peer clock. Both of these are precision quantities and can usually be averaged over long intervals in order to improve accuracy and stability without bias accumulation (see [Appendix H](#)). The peer.dispersion variable represents the maximum error due to measurement error, skew-error accumulation and sample variance. All three variables are used in the clock-selection and clock-combining procedures to select the peer clock(s) used for synchronization and to maximize the accuracy and stability of the indications.

6.36 4.2 Clock-Selection Procedure

The clock-selection procedure uses the peer variables THETA, DELTA, EPSILON and tau and is called when these variables change or when the reachability status changes. It consists of two algorithms, the intersection algorithm and the clustering algorithm. The intersection algorithm constructs a list of candidate peers eligible to become the synchronization source, computes a confidence interval for each and casts out falsetickers using a technique adapted from Marzullo and Owicki [MAR85]. The clustering algorithm sorts the list of surviving candidates in order of stratum and synchronization distance and repeatedly casts out outliers on the basis of select dispersion until only the most accurate, precise and stable survivors are left. A bit is set for each survivor to indicate the outcome of the selection process. The system variable sys.peer is set as a pointer to the most likely survivor, if there is one, or to the NULL value if not.

6.37 4.2.1 Intersection Algorithm

begin clock-selection procedure

Each peer is examined in turn and added to an endpoint list only if it passes several sanity checks designed to avoid loops and use of exceptionally noisy data. If no peers survive the sanity checks, the procedure exits without finding a synchronization source. For each of m survivors three entries of the form [endpoint,type] are added to the endpoint list: [THETA-LAMBDA ,-1], [THETA ,0] and [THETA + LAMBDA ,1]. There will be 3 m entries on the list, which is then sorted by increasing endpoint.

```

m <-- 0;
for (each peer)                /* calling all peers */
  if (peer.reach != 0 and peer.dispersion < NTP.MAXDISPERSE and
      not (peer.stratum > 1 and peer.refid = peer.hostaddr)) begin
    LAMBDA andistance (peer);    /* make list entry */
    add [THETA - LAMBDA ,-1] to endpoint list;
    add [THETA ,0] to endpoint list;
    add [THETA + LAMBDA ,1]> to endpoint list;
    m <-- m+1;
  endif
endfor
if (m = 0) begin                /* skeddaddle if no candidates */
  sys.peer <-- NULL;
  sys.stratum <-- 0 (undefined);
  exit;
endif
sort endpoint list by increasing endpoint||type;

```

The following algorithm is adapted from DTS [DEC89] and is designed to produce the largest single intersection containing only truechimers. The algorithm begins by initializing a value f and counters i and c to zero. Then, starting from the lowest endpoint of the sorted endpoint list, for each entry [endpoint,type] the value of type is subtracted from the counter i , which is the number of intersections. If type is zero, increment the value of c , which is the number of falsetickers (see [Appendix H](#)). If $i \geq m - f$ for some entry, endpoint of that entry becomes the lower endpoint of the intersection; otherwise, f is increased by one and the above procedure is repeated. Without resetting f or c , a similar procedure is used to find the upper endpoint, except that the value of type is added to the counter.. If after both endpoints have been determined $c \leq f$, the procedure continues having found $m - f$ truechimers; otherwise, f is increased by one and the entire procedure is repeated.

```

for (f from 0 to f >= m over 2) begin    /* calling all truechimers */
  c <-- 0;
  i <-- 0;
  for (each [endpoint, type] from lowest) begin /* find low endpoint */
    i <-- i - type;
    low <-- endpoint;
    if (i >= m - f) break;
    if (type = 0) c <-- c + 1 ;
  endfor;
  i <-- 0;

```

```

    for (each [endpoint, type] from highest) begin /* find high endpoint */
        i <-- i + type;
        high <-- endpoint;
        if ( i >= m - f ) break;
        if ( type = 0) c <-- c + 1;
    endfor;
    if ( c <= f) break;          /* continue until all falsetickers found */
    endfor;
    if ( low > high) begin      /* quit if no intersection found */
        sys.peer <- NULL;
        exit;
    endif;

```

Note that processing continues past this point only if there are more than m over 2 intersections. However, it is possible, but not highly likely, that there may be fewer than m over 2 truechimers remaining in the intersection.

6.38 4.2.2. Clustering Algorithm

In the original DTS algorithm the clock-selection procedure exits at this point with the presumed correct time set midway in the computed intersection [low,high]. However, this can lead to a considerable loss in accuracy and stability, since the individual peer statistics are lost. Therefore, in NTP the candidates that survived the preceding steps are processed further. The candidate list is rebuilt with entries of the form [distance,index], where distance is computed from the (scaled) peer stratum and synchronization distance LAMBDA. The scaling factor provides a mechanism to weight the combination of stratum and distance. Ordinarily, the stratum will dominate, unless one or more of the survivors has an exceptionally high distance. The list is then sorted by increasing distance.

```

m <-- 0;
for (each peer) begin          /* calling all peers */
    if (low <= theta <= high) begin
        LAMBDA <-- distance (peer); /* make list entry */
        ist <-- peer.stratum * NTP.MAXDISPERSE + LAMBDA >
        add [ dist ,peer] to candidate list;
        m <-- m + 1;
    endif;
endfor;
sort candidate list by increasing dist;

```

The next steps are designed to cast out outliers which exhibit significant dispersions relative to the other members of the candidate list while minimizing wander, especially on high-speed LANs with many time servers. Wander causes needless network overhead, since the poll interval is clamped at sys.poll as each new peer is selected for synchronization and only slowly increases when the peer is no longer selected. It has been the practical experience that the

number of candidates surviving to this point can become quite large and can result in significant processor cycles without materially

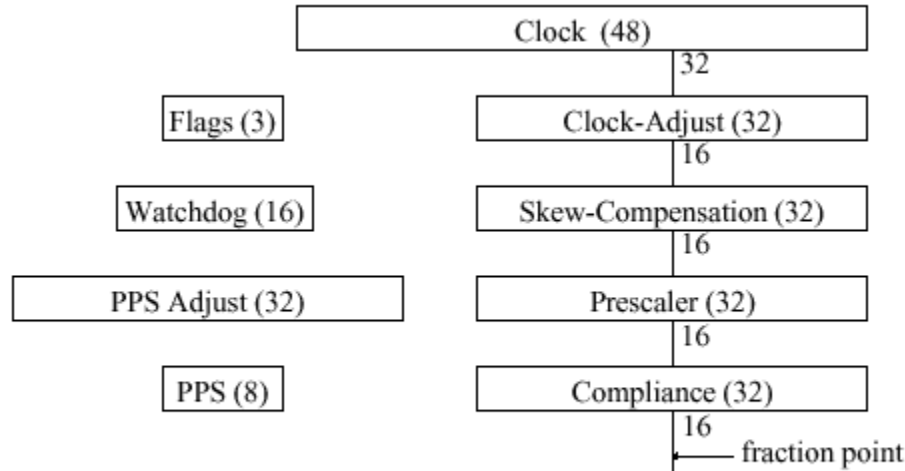


Figure 3. Clock Registers

enhancing stability and accuracy. Accordingly, the candidate list is truncated at NTP.MAXCLOCK entries.

Note ϵ_{xi} is the select (sample) dispersion relative to the i th peer represented on the candidate list, which can be calculated in a manner similar to the filter dispersion described previously. The ϵ_j is the dispersion of the j th peer represented on the list and includes components due to measurement error, skew-error accumulation and filter dispersion. If the maximum ϵ_{xi} is greater than the minimum ϵ_j and the number of survivors is greater than NTP.MINCLOCK, the i th peer is discarded from the list and the procedure is repeated. If the current synchronization source is one of the survivors and there is no other survivor of lower stratum, then the procedure exits without doing anything further. Otherwise, the synchronization source is set to the first survivor on the candidate list. In the following i, j, k, l are peer indices, with k the index of the current synchronization source (NULL if none) and l the index of the first survivor on the candidate list.

```
while begin
  for (each survivor [distance,index]) begin /* compute dispersions */
    find index i for max  $\epsilon_{xi}$ ;
    find index j for min  $\epsilon_j$ ;
  endfor
  if ( $\epsilon_{xi} \leq \epsilon_j$  or  $m \leq \text{NTP.MINCLOCK}$ ) break;
  peer.survivor[i] <-- 0; /* discard ith peer */
  if (i = k) sys.peer <-- NULL;
  delete the ith peer from the candidate list;
  m <-- m - 1;
```

```

    endwhile
    if (peer.survivor [k] = 0 or peer.stratum[k] > peer.stratum [l]>) begin
        sys.peer <-- l;                /* new clock source */
    end

```

Parameter	Name	Crystal	Mains
Adjustment Interval	CLOCK.ADJ	4 sec	1 sec
PPS Timeout	CLOCK.PPS	60 sec	60 sec
Step Timeout	CLOCK.MINSTEP	900 sec	900 sec
Maximum Aperture	CLOCK.MAX	± 128 ms	± 512 ms
Frequency Weight	CLOCK.FREQ	16	16
Phase Weight	CLOCK.PHASE	8	9
Compliance Weight	CLOCK.WEIGHT	13	13
Compliance Maximum	CLOCK.COMP	4	4
Compliance Multiplier	CLOCK.MULT	4	4

Table 6. Clock Parameters

```

        call poll-update;
    endif
end clock-select procedure;

```

The algorithm is designed to favor those peers near the head of the candidate list, which are at the lowest stratum and distance and presumably can provide the most accurate and stable time. With proper selection of weight factor v (also called NTP.SELECT), entries will be trimmed from the tail of the list, unless a few outliers disagree significantly with respect to the remaining entries, in which case the outliers are discarded first. The termination condition is designed to avoid needless switching between synchronization sources when not statistically justified, yet maintain a bias toward the low-stratum, low-distance peers.

6.39 5. Local Clocks

In order to implement a precise and accurate local clock, the host must be equipped with a hardware clock consisting of an oscillator and interface and capable of the required precision and stability. A logical clock is then constructed using these components plus software components that adjust the apparent time and frequency in response to periodic updates computed by NTP or some other time-synchronization protocol such as Hellospeak [MIL83b] or the Unix 4.3bsd TSP [GUS85a]. This section describes the Fuzzball local-clock model and implementation, which includes provisions for precise time and frequency adjustment and can maintain time to within 15 ns and frequency to within 0.3 ms per day. The model is suitable for use with both compensated and uncompensated quartz oscillators and can be adapted to power-frequency oscillators. A summary of the characteristics of these and other types of oscillators can be found in [Appendix E](#), while a comprehensive mathematical analysis of the NTP local-clock model can be found in

Appendix G.

It is important to note that the particular implementation described is only one of possibly many implementations that provide equivalent functionality. However, it is equally important to note that the clock model described in [Appendix G](#) and which is the basis of the implementation involves a particular kind of control-feedback loop that is potentially unstable if the design rules are broken. The model and parameter described in [Appendix G](#) are designed to provide accurate and stable time under typical operating conditions using conventional hardware and in the face of disruptions in hardware or network connectivity. The parameters have been engineered for reliable operation in a multi-level hierarchical subnet where unstable operation at one level can disrupt possibly many other levels.

6.40 5.1 Fuzzball Implementation

The Fuzzball local clock consists of a collection of hardware and software registers, together with a set of algorithms, which implement a logical clock that functions as a disciplined oscillator and synchronizes to an external source. Following is a description of its components and manner of operation. Note that all arithmetic is two's complement integer and all shifts "<<" and ">>" are arithmetic (sign-fill for right shifts and zero-fill for left shifts). Also note that $x \ll n$ is equivalent to $x \gg -n$.

The principal components of the local clock are shown in Figure 3, in which the fraction points shown are relative to whole milliseconds. The 48-bit Clock register and 32-bit Prescaler function as a disciplined oscillator which increments in milliseconds relative to midnight at the fraction point. The 32-bit Clock-Adjust register is used to adjust the oscillator phase in gradual steps to avoid discontinuities in the indicated time scale. Its contents are designated x in the following. The 32-bit Skew-Compensation register is used to trim the oscillator frequency by adding small phase increments at periodic adjustment intervals and can compensate for frequency errors as much as .01% or ± 100 ppm. Its contents are designated y in the following. The 16-bit Watchdog counter and 32-bit Compliance register are used to determine validity, as well as establish the PLL bandwidth and poll interval (see [Appendix G](#)). The contents of the Compliance register are designated z in the following. The 32-bit PPS-Adjust register is used to hold a precision time adjustment when a source of 1-pps pulses is available, while the 8-bit PPS counter is used to verify presence of these pulses. The two-bit Flags register contains the two leap bits described elsewhere (leap).

All registers except the Prescaler register are ordinarily implemented in memory. In typical clock interface designs such as the DEC KVV11-C, the Prescaler register is implemented as a 16-bit buffered counter driven by a quartz-controlled oscillator at some multiple of 1000 Hz. A counter overflow is signalled by an interrupt, which results in an increment of the Clock register at the bit corresponding to the overflow. The time of day is determined by reading the Prescaler register, which does not disturb the counting process, and adding its value to that of the Clock register with fraction points aligned as shown and with unimplemented low-order bits set to zero. In other interface designs, such as the LSI-11 event-line mechanism, each tick of the clock is signalled by an interrupt at intervals of 16-2/3 ms or 20 ms, depending on interface and mains frequency. When this occurs the appropriate increment

in fractional milliseconds is added to the Clock register.

The various parameters used are summarized in Table 6, in which certain parameters have been rescaled from those given in [Appendix G](#) due to the units here being in milliseconds. When the system is initialized, all registers and counters are cleared and the leap bits set to 112 (unsynchronized). At adjustment intervals of CLOCK.ADJ seconds CLOCK.ADJ is subtracted from the PPS counter, but only if the previous contents of the PPS counter are greater than zero. Also, CLOCK.ADJ is added to the Watchdog counter, but the latter is clamped not to exceed NTP.MAXAGE divided by CLOCK.ADJ (one full day). In addition, if the Watchdog counter reaches this value, the leap bits are set to 112 (unsynchronized).

In some system configurations a precise source of timing information is available in the form of a train of timing pulses spaced at one-second intervals. Usually, this is in addition to a source of timecode information, such as a radio clock or even NTP itself, to number the seconds, minutes, hours and days. In typical clock interface designs such as the DEC KVV11-C, a special input is provided which can trigger an interrupt as each pulse is received. When this happens the PPS counter is set to CLOCK.PPS and the current time offset is determined in the usual way. Then, the PPS-Adjust register is set to the time offset scaled to milliseconds. Finally, if the PPS-Adjust register is greater than or equal to 500, 1000 is subtracted from its contents. As described below, the PPS-Adjust register and PPS counters can be used in conjunction with an ordinary timecode to produce an extremely accurate local clock.

6.41 5.2 Gradual Phase Adjustments

Left uncorrected, the local clock runs at the offset and frequency resulting from its last update. An update is produced by an event that results in a valid clock selection. It consists of a signed 48-bit integer in whole milliseconds and fraction, with fraction point to the left of bit 32. If the magnitude is greater than the maximum aperture CLOCK.MAX, a step adjustment is required, in which case proceed as described later. Otherwise, a gradual phase adjustment is performed. Normally, the update is computed by the NTP algorithms described previously; however, if the PPS counter is greater than zero, the value of the PPS-Adjust register is used instead. Let u be a 32-bit quantity with bits 0-31 set as bits 16-47 of the update. If some of the low-order bits of the update are unimplemented, they are set as the value of the sign bit. These operations move the fraction point of u to the left of bit 16 and minimize the effects of truncation and roundoff errors. Let b be the number of leading zeros of the absolute value of the Compliance register and let c be the number of leading zeros of the Watchdog counter, both of which are easily computed by compact loops. Then, set b to

$$b = b - 16 + \text{CLOCK.COMP}$$

and clamp it to be not less than zero. This represents the logarithm of the loop time constant. Then, set c to

$$c = 10 - c$$

and clamp it to be not greater than NTP.MAXPOLL - NTP.MINPOLL. This represents the

logarithm of the integration interval since the last update. The clamps insure stable operation under typical conditions encountered in the Internet. Then, compute new values for the Clock-Adjust and Skew-Compensation registers

$$\begin{aligned} x &= u \gg b, \\ y &= y + (u \gg (b + b - c)). \end{aligned}$$

Finally, compute the exponential average

$$z = z + (u \ll (b + \text{CLOCK.MULT}) - z) \gg \text{CLOCK.WEIGHT},$$

where the left shift realigns the fraction point for greater precision and ease of computation.

At each adjustment interval the final clock correction consisting of two components is determined. The first (phase) component consists of the quantity

$$x \gg \text{CLOCK.PHASE},$$

which is then subtracted from the previous contents of the Clock-Adjust register to form the new contents of that register. The second (frequency) component consists of the quantity

$$y \gg \text{CLOCK.FREQ}.$$

The sum of the phase and frequency components is the final clock correction, which is then added to the Clock register. Finally, the Watchdog counter is set to zero. Operation continues in this way until a new correction is introduced.

The value of b computed above can be used to update the poll interval system variable (`sys.poll`). This functions as an adaptive parameter that provides a very valuable feature which reduces the polling overhead, especially if the clock-combining algorithm described in Appendix F is used:

$$\text{sys.poll} \leftarrow b + \text{NTP.MINPOLL}.$$

Under conditions when update noise is high or the hardware oscillator frequency is changing relatively rapidly due to environmental conditions, the magnitude of the compliance increases. With the parameters specified, this causes the loop bandwidth (reciprocal of time constant) to increase and the poll interval to decrease, eventually to `NTP.MINPOLL` seconds. When noise is low and the hardware oscillator very stable, the compliance decreases, which causes the loop bandwidth to decrease and the poll interval to increase, eventually to `NTP.MAXPOLL` seconds.

The parameters in Table 6 have been selected so that, under good conditions with updates in the order of a few milliseconds, a precision of a millisecond per day (about .01 ppm or 10^{-8}), can be achieved. Care is required in the implementation to insure monotonicity of the Clock register and to preserve the highest precision while minimizing the propagation of roundoff

errors. Since all of the multiply/divide operations (except those involved with the 1-pps pulses) computed in real time can be approximated by bitwise-shift operations, it is not necessary to implement a full multiply/divide capability in hardware or software.

In the various implementations of NTP for many Unix-based systems it has been the common experience that the single most important factor affecting local-clock stability is the matching of the phase and frequency coefficients to the particular kernel implementation. It is vital that these coefficients be engineered according to the model values, for otherwise the PLL can fail to track normal oscillator variations and can even become unstable.

6.42 5.3 Step Phase Adjustments

When the magnitude of a correction exceeds the maximum aperture `CLOCK.MAX`, the possibility exists that the clock is so far out of synchronization with the reference source that the best action is an immediate and wholesale replacement of Clock register contents, rather than in gradual adjustments as described above. However, in cases where the sample variance is extremely high, it is prudent to disbelieve a step change, unless a significant interval has elapsed since the last gradual adjustment. Therefore, if a step change is indicated and the Watchdog counter is less than the preconfigured value `CLOCK.MINSTEP`, the update is ignored and the local-clock procedure exits. These safeguards are especially useful in those system configurations using a calibrated atomic clock or LORAN-C receiver in conjunction with a separate source of seconds-numbering information, such as a radio clock or NTP peer.

If a step change is indicated the update is added directly to the Clock register and the Clock-Adjust register and Watchdog counter both set to zero, but the other registers are left undisturbed. Since a step change invalidates data currently in the clock filters, the leap bits are set to 112 (unsynchronized) and, as described elsewhere, the clear procedure is called to purge the clock filters and state variables for all peers. In practice, the necessity to perform a step change is rare and usually occurs when the local host or reference source is rebooted, for example. This is fortunate, since step changes can result in the local clock apparently running backward, as well as incorrect delay and offset measurements of the synchronization mechanism itself.

Considerable experience with the Internet environment suggests the values of `CLOCK.MAX` tabulated in Table 6 as appropriate. In practice, these values are exceeded with a single time-server source only under conditions of the most extreme congestion or when multiple failures of nodes or links have occurred. The most common case when the maximum is exceeded is when the time-server source is changed and the time indicated by the new and old sources exceeds the maximum due to systematic errors in the primary reference source or large differences in path delays. It is recommended that implementations include provisions to tailor `CLOCK.MAX` for specific situations. The amount that `CLOCK.MAX` can be increased without violating the monotonicity requirement depends on the Clock register increment. For an increment of 10 ms, as used in many workstations, the value shown in Table 6 can be increased by a factor of five.

6.43 5.4 Implementation Issues

The basic NTP robustness model is that a host has no other means to verify time other than NTP itself. In some equipment a battery-backed clock/calendar is available for a sanity check. If such a device is available, it should be used only to confirm sanity of the timekeeping system, not as the source of system time. In the common assumption (not always justified) that the clock/calendar is more reliable, but less accurate, than the NTP synchronization subnet, the recommended approach at initialization is to set the Clock register as determined from the clock/calendar and the other registers, counters and flags as described above. On subsequent updates if the time offset is greater than a configuration parameter (e.g., 1000 seconds), then the update should be discarded and an error condition reported. Some implementations periodically record the contents of the Skew-Compensation register in stable storage such as a system file or NVRAM and retrieve this value at initialization. This can significantly reduce the time to converge to the nominal stability and accuracy regime.

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

On the day prior to the insertion of a leap second the leap bits (sys.leap) are set at the primary servers, presumably by manual means. Subsequently, these bits show up at the local host and are passed to the local-clock procedure. This causes the modulus of the time variable, which is the length of the current day, to be increased or decreased by one second as appropriate. Immediately following insertion the leap bits are reset. Additional discussion on this issue can be found in [Appendix E](#).

Lack of a comprehensive mechanism to administer the leap bits in the primary servers is presently an awkward problem better suited to a comprehensive network-management mechanism yet to be developed. As a practical matter and unless specific provisions have been made otherwise, currently manufactured radio clocks have no provisions for leap seconds, either automatic or manual. Thus, when a leap actually occurs, the radio must resynchronize to the broadcast timecode, which may take from a few minutes to some hours. Unless special provisions are made, a primary server might leap to the new time scale, only to be yanked back to the previous time scale when it next synchronizes to the radio. Subsequently, the server will be yanked forward again when the radio itself resynchronizes to the broadcast timecode.

This problem can not be reliably avoided using any selection algorithm, since there will always exist an interval of at least a couple of minutes and possibly as much as some hours when some or all radios will be out of synchronization with the broadcast timecode and only after the majority of them have resynchronized will the subnet settle down. The CLOCK.MINSTEP delay is designed to cope with this problem by forcing a minimum interval since the last gradual adjustment was made before allowing a step change to occur. Therefore, until the radio resynchronizes, it will continue on the old time scale, which is one second off the local

clock after the leap and outside the maximum aperture CLOCK.MAX permitted for gradual phase adjustments. When the radio eventually resynchronizes, it will almost certainly come up within the aperture and again become the reference source. Thus, even in the unlikely case when the local clock incorrectly leaps, the server will go no longer than CLOCK.MINSTEP seconds before resynchronizing.

6.44 6. Acknowledgments

Many people contributed to the contents of this document, which was thoroughly debated by electronic mail and debugged using two different prototype implementations for the Unix 4.3bsd operating system, one written by Louis Mamakos and Michael Petry of the University of Maryland and the other by Dennis Ferguson of the University of Toronto. Another implementation for the Fuzzball operating system [MIL88b] was written by the author. Many individuals too numerous to mention meticulously tested the several beta-test prototype versions and ruthlessly smoked out the bugs, both in the code and the specification. Especially useful were comments from Dennis Ferguson and Bill Sommerfeld, as well as discussions with Joe Comuzzi and others at Digital Equipment Corporation.

6.45 7. References

[ABA89]

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

[ALL74a]

Allan, D.W., J.H. Shoaf and D. Halford. Statistics of time and frequency data analysis. In: Blair, B.E. (Ed.). Time and Frequency Theory and Fundamentals. National Bureau of Standards Monograph 140, U.S. Department of Commerce, 1974, 151-204.

[ALL74b]

Allan, D.W., J.E. Gray and H.E. Machlan. The National Bureau of Standards atomic time scale: generation, stability, accuracy and accessibility. In: Blair, B.E. (Ed.). Time and Frequency Theory and Fundamentals. National Bureau of Standards Monograph 140, U.S. Department of Commerce, 1974, 205-231.

[BEL86]

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

[BER87]

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

[BLA74]

Blair, B.E. Time and frequency dissemination: an overview of principles and techniques. In: Blair, B.E. (Ed.). Time and Frequency Theory and Fundamentals. National Bureau of Standards Monograph 140, U.S. Department of Commerce, 1974, 233-314.

[BRA80]

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

[COL88]

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

[DAR81a]

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

[DAR81b]

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

[DEC89]

Digital Time Service Functional Specification Version T.1.0.5. Digital Equipment Corporation, 1989.

[DER90]

Dershowitz, N., and E.M. Reingold. Calendrical Calculations. Software Practice and Experience 20, 9 (September 1990), 899-928.

[FRA82]

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

[GUS84]

Gusella, R., and S. Zatti. TEMPO - A network time controller for a distributed Berkeley UNIX system. IEEE Distributed Processing Technical Committee Newsletter 6, NoSI-2 (June 1984),

7-15. Also in: Proc. Summer
USENIX Conference (June 1984, Salt Lake City).

[GUS85a]

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

[GUS85b]

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

[HAL84]

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

[JOR85]

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

[KOP87]

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

[LAM78]

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

[LAM85]

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

[LIN80]

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

[LUN84]

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

[MAR85]

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

[MIL81a]

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

[MIL81b]

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

[MIL83a]

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

[MIL83b]

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

[MIL85a]

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

[MIL85b]

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

[MIL85c]

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

[MIL88a]

Mills, D.L. Network Time Protocol (version 1) - specification and implementation. DARPA

Network Working Group Report RFC-1059, University of Delaware, July 1988.

[MIL88b]

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

[MIL89]

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

[MIL90]

Mills, D.L. Measured performance of the Network Time Protocol in the Internet system. ACM Computer Communication Review 20, 1 (January 1990), 65-75.

[MIL91a]

Mills, D.L. Internet time synchronization: the Network Time Protocol. IEEE Trans. Communications 39, 10 (October 1991), 1482-1493.

[MIL91b]

Mills, D.L. On the chronology and meterology of computer network timescales and their application to the Network Time Protocol. ACM Computer Communications Review 21, 5 (October 1991), 8-17.

[MIT80]

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

[NBS77]

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

[NBS79]

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

[NBS80]

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

[POS80]

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

[POS83a]

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

[POS83b]

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

[RIC88]

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

[SCH86]

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

[SMI86]

Smith, J. Modern Communications Circuits. McGraw-Hill, New York, NY, 1986.

[SRI87]

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

[STE88]

Steiner, J.G., C. Neuman, and J.I. Schiller. Kerberos: an authentication service for open network systems. Proc. Winter USENIX Conference (February 1988).

[SU81]

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

[TRI86]

Tripathi, S.K., and S.H. Chang. ETempo: a clock synchronization algorithm for hierarchical

LANs - implementation and measurements. Systems Research Center Technical Report TR-86-48, University of Maryland, 1986.

[VAN84]

Van Dierendonck, A.J., and W.C. Melton. Applications of time transfer using NAVSTAR GPS. In: Global Positioning System, Papers Published in Navigation, Vol. II, Institute of Navigation, Washington, DC, 1984.

[VAS78]

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

6.46 Appendix A

NTP Data Format - Version 3

The format of the NTP Message data area, which immediately follows the UDP header, is shown in Figure 4. Following is a description of its fields.

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

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

Version Number (VN): This is a three-bit integer indicating the NTP version number, currently three (3).

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

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

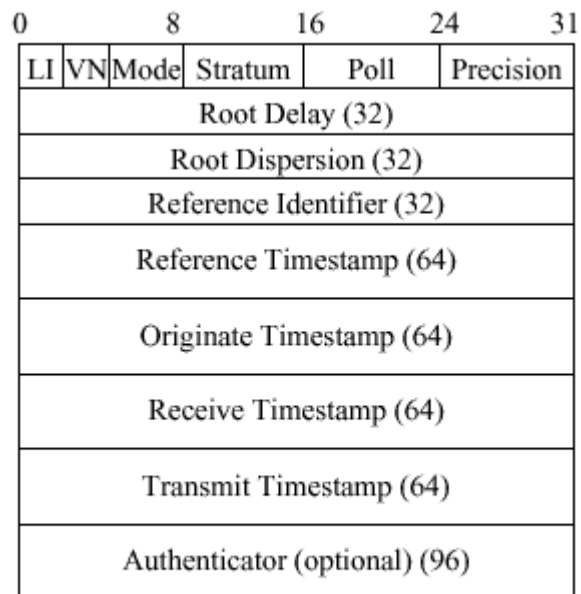


Figure 4. NTP Message Header

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

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

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

Poll Interval: This is an eight-bit signed integer indicating the maximum interval between

successive messages, in seconds to the nearest power of two. The values that can appear in this field range from NTP.MINPOLL to NTP.MAXPOLL inclusive.

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

Root Delay: This is a 32-bit signed fixed-point number indicating the total roundtrip delay to the primary reference source, in seconds with fraction point between bits 15 and 16. Note that this variable can take on both positive and negative values, depending on clock precision and skew.

Root Dispersion: This is a 32-bit signed fixed-point number indicating the maximum error relative to the primary reference source, in seconds with fraction point between bits 15 and 16. Only positive values greater than zero are possible.

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

Stratum	Code	Meaning
0	DCN	DCN routing protocol
0	NIST	NIST public modem
0	TSP	TSP time protocol
0	DTS	Digital Time Service
1	ATOM	Atomic clock (calibrated)
1	VLF	VLF radio (OMEGA,, etc.)
1	callsign	Generic radio
1	LORC	LORAN-C radionavigation
1	GOES	GOES UHF environment satellite
1	GPS	GPS UHF satellite positioning

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

Reference Timestamp: This is the local time at which the local clock was last set or corrected,

in 64-bit timestamp format.

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

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

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

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

6.47 Appendix B

NTP Control Messages

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

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

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

Most control functions involve sending a command and receiving a response, perhaps

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

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

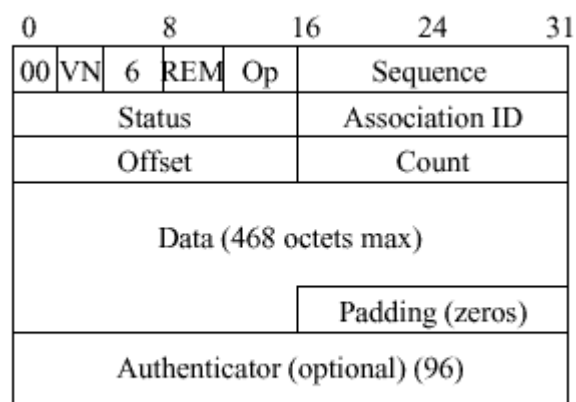


Figure 5. NTP Control Message Header

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

Commands need not necessarily be sent by an NTP peer, so ordinary access-control procedures may not apply; however, the optional mask/match mechanism suggested elsewhere in this document provides the capability to control access by mode number, so this could be

used to limit access for control messages (mode 6) to selected address ranges.

NTP Control Message Format

The format of the NTP Control Message header, which immediately follows the UDP header, is shown in Figure 5. Following is a description of its fields. Bit positions marked as zero are reserved and should always be transmitted as zero.

Version Number (VN): This is a three-bit integer indicating the NTP version number, currently three (3).

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

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

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

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

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

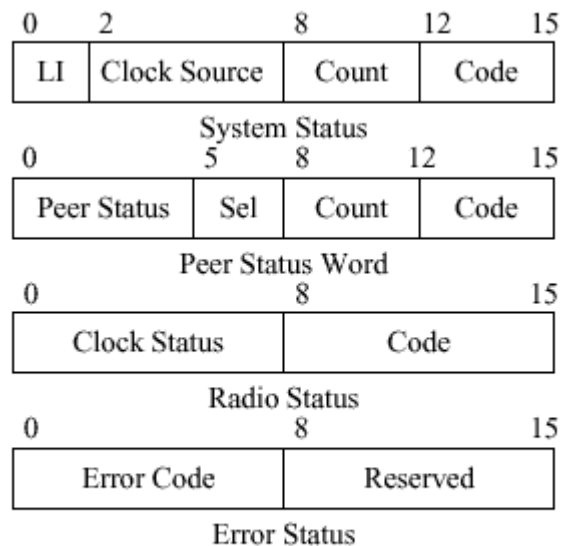


Figure 6. Status Word Formats

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

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

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

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

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

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

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

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

Status Words

Status words indicate the present status of the system, associations and clock. They are designed to be interpreted by network-monitoring programs and are in one of four 16-bit formats shown in Figure 6 and described in this section. System and peer status words are associated with responses for all commands except the read clock variables, write clock variables and set trap address/port commands. The association identifier zero specifies the system status word, while a nonzero identifier specifies a particular peer association. The status word returned in response to read clock variables and write clock variables commands indicates the state of the clock hardware and decoding software. A special error status word is used to report malformed command fields or invalid values.

System Status Word

The system status word appears in the status field of the response to a read status or read

variables command with a zero association identifier. The format of the system status word is as follows:

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

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

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

0	unspecified or unknown
1	Calibrated atomic clock (e.g., HP 5061)
2	VLF (band 4) or LF (band 5) radio (e.g., OMEGA,, WWVB)
3	HF (band 7) radio (e.g., CHU,, MSF,, WWV/H)
4	UHF (band 9) satellite (e.g., GOES,, GPS)
5	local net (e.g., DCN,, TSP,, DTS)
6	UDP/NTP
7	UDP/TIME
8	eyeball-and-wristwatch
9	telephone modem (e.g., NIST)
10-63	reserved

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

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

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

Peer Status Word

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

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

0	configured (peer.config)
1	authentication enabled (peer.authenable)
2	authentication okay (peer.authentic)
3	reachability okay (peer.reach <F128M>Ö<F255D> 0)
4	reserved

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

- 0 rejected
- 1 passed sanity checks (tests 1 through 8 in Section 3.4.3)
- 2 passed correctness checks (intersection algorithm in Section 4.2.1)
- 3 passed candidate checks (if limit check implemented)
- 4 passed outlier checks (clustering algorithm in Section 4.2.2)
- 5 current synchronization source; max distance exceeded (if limit check implemented)
- 6 current synchronization source; max distance okay
- 7 reserved

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

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

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

Clock Status Word

There are two ways a reference clock can be attached to a NTP service host, as an dedicated device managed by the operating system and as a synthetic peer managed by NTP. As in the read status command, the association identifier is used to identify which one, zero for the system clock and nonzero for a peer clock. Only one system clock is supported by the protocol, although many peer clocks can be supported. A system or peer

clock status word appears in the status field of the response to a read clock variables or write clock variables command. This word can be considered an extension of the system status word or the peer status word as appropriate. The format of the clock status word is as follows:

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

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

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

Error Status Word

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

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

Commands

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

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

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

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

Commands are interpreted as follows:

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

```
<<association identifier>> <<status word>>,
```

one for each currently defined association. **Read Variables (2):** The command data field is empty or contains a list

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

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

Variables command.

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

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

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

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

6.48 Appendix C

Authentication Issues

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

The access-control mechanism suggested in the NTP specification responds to these requirements by limiting access to trusted peers. The various sanity checks resist most replay and spoofing attacks by discarding old duplicates and using the originate timestamp as a one-time pad, since it is unlikely that even a synchronized peer can predict future timestamps with the precision required on the basis of past observations alone. In addition, the protocol

environment resists jamming attacks by employing redundant time servers and diverse network paths. Resistance to stochastic disruptions, actual or manufactured, are minimized by careful design of the filtering and selection algorithms.

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

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

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

A packet format and crypto-checksum procedure appropriate for NTP is specified in the following sections. The cryptographic information is carried in an authenticator which follows the (unmodified) NTP header fields. The crypto-checksum procedure uses the Data Encryption Standard (DES) [NBS77]; however, only the DES encryption algorithm is used and the decryption algorithm is not necessary. This feature is specifically targeted toward governmental sensitivities on the export of cryptographic technology, since the DES decryption algorithm need not be included in NTP software distributions and thus cannot be extracted and used in other applications to avoid message data disclosure.

NTP Authentication Mechanism

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

Authentication Enabled Bit (peer.authenable): This is a bit indicating that the association is to operate in the authenticated mode. For configured peers this bit is determined from the startup environment. For non-configured peers, this bit is set to one if an arriving message includes the authenticator and set to zero otherwise.

Authenticated Bit (peer.authentic): This is a bit indicating that the last message received from the peer has been correctly authenticated.

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

Cryptographic Keys (sys.key): This is a set of 64-bit DES keys. Each key is constructed as in the Berkeley Unix distributions, which consists of eight octets, where the seven low-order bits of each octet correspond to the DES bits 1-7 and the high-order bit corresponds to the DES

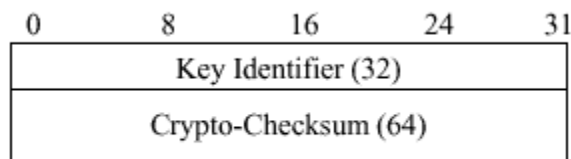


Figure 7. Authenticator Format

odd-parity bit 8. By convention, the unspecified key 0 (zero), consisting of eight odd-parity

zero octets, is used for testing and presumed known throughout the NTP community. The remaining keys are distributed using methods outside the scope of NTP.

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

The authenticator field consists of two subfields, one consisting of the pkt.keyid variable and the other the pkt.check variable computed by the encrypt procedure, which is called by the transmit procedure described in the NTP specification, and by the decrypt procedure, which is called by the receive procedure described in the NTP specification. Its presence is revealed by the fact the total datagram length according to the UDP header is longer than the NTP message length, which includes the header plus the data field, if present. For authentication purposes, the

NTP message is zero-padded if necessary to a 64-bit boundary, although the padding bits are not considered part of the NTP message itself. The authenticator format shown in Figure 7<\$&fig7> has 96 bits, including a 32-bit key identifier and 64-bit crypto-checksum, and is aligned on a 32-bit boundary for efficient computation. Additional information required in some implementations, such as certificate authority and encryption algorithm, can be inserted between the (padded) NTP message and the key identifier, as long as the alignment conditions are met. Like the authenticator itself, this information is not included in the crypto-checksum. Use of these data are beyond the scope of this specification. These conventions may be changed in future as the result of other standardization activities.

NTP Authentication Procedures

When authentication is implemented there are two additional procedures added to those described in the NTP specification. One of these (encrypt) constructs the crypto-checksum in transmitted messages, while the other (decrypt) checks this quantity in received messages. The procedures use a variant of the cipher-block chaining method described in [NBS80] as applied to DES. In principal, the procedure is independent of DES and requires only that the encryption algorithm operate on 64-bit blocks. While the NTP authentication mechanism specifies the use of DES, other algorithms could be used by prior arrangement.

Encrypt Procedure

For ordinary NTP messages the encryption procedure operates as follows. If authentication is not enabled, the procedure simply exits. If the association is active (modes 1, 3, 5), the key is determined from the system key identifier. If the association is passive (modes 2, 4) the key is determined from the peer key identifier, if the authentic bit is set, or as the default key (zero) otherwise. These conventions allow further protection against replay attacks and keying errors, as well as facilitate testing and migration to new versions. The crypto-checksum is calculated using the 64-bit NTP header and data words, but not the authenticator or padding bits.

begin encrypt procedure

```

    if (peer.authenable = 0) exit;          /* do nothing if not enabled */
    if (peer.hostmode = 1 or peer.hostmode = 3 or peer.hostmode = 5)
        keyid <-- peer.hostkeyid;          /* active modes use system key */

```

```

else
  if (peer.authentic = 1)          /* passive modes use peer key */
    keyid <-- peer.peerkeyid;
  else
    keyid <-- 0;                   /* unauthenticated use key 0 */
  temp <-- 0;                      /* calculate crypto-checksum */
  for (each 64-bit header and data word) begin
    temp <-- temp xor word;
    temp <-- DES (temp,keyid);
  endfor;
  pkt.keyid <-- keyid;             /* insert packet variables */
  pkt.check <-- temp;
end encrypt procedure;

```

Decrypt Procedure

For ordinary messages the decryption procedure operates as follows. If the peer is not configured, the data portion of the message is inspected to determine if the authenticator fields are present. If so, authentication is enabled; otherwise, it is disabled. If authentication is enabled and the authenticator fields are present and the crypto-checksum succeeds, the authentication bit is set to one; otherwise, it is set to zero.

```

begin decrypt procedure
  peer.authentic <-- 0;
  if (peer.config = 0)             /* if not configured, enable per packet */
    if (authenticator present)
      peer.authenable <-- 1;
    else
      peer.authenable <-- 0;
  if (peer.authenable = 0 or authenticator not present)) exit;
  peer.peerkeyid <-- pkt.keyid;     /* use peer key */
  temp <-- 0;                      /* calculate crypto-checksum */
  for (each 64-bit header and data word) begin
    temp <-- temp xor word;
    temp <-- DES (temp, peer.peerkeyid);
  endfor;
  if (temp == pkt.check) peer.authentic <-- 1; /* declare result */
end decrypt procedure;

```

Control-Message Procedures

In anticipation that the functions provided by the NTP control messages will eventually be subsumed by a comprehensive network-management function, the peer variables are not used for control message authentication. If an NTP command message is received with an authenticator field, the crypto-checksum is computed as in the decrypt procedure and the response message includes the authenticator field as computed by the encrypt procedure. If the received authenticator is

correct, the key for the response is the same as in the command; otherwise, the default key (zero) is used. Commands causing a change to the peer data base, such as the write variables and set trap address/port commands, must be correctly authenticated; however, the remaining commands are normally not authenticated in order to minimize the encryption overhead.

6.49 Appendix D

Differences from Previous Versions

The original NTP, later called NTP Version 0, was described in RFC-958 [MIL85c]. Subsequently, Version 0 was superseded by Version 1 (RFC-1059 [MIL88a]), and Version 2 (RFC-1119 [MIL89]). The Version-2 description was split into two documents, RFC-1119 defining the architecture and specifying the protocol and algorithms, and another [MIL90b] describing the service model, algorithmic analysis and operating experience. In previous versions these two objectives were combined in one document. While the architecture assumed in Version 3 is identical to Version 2, the protocols and algorithms differ in minor ways. Differences between NTP Version 3 and previous versions are described in this Appendix. Due to known bugs in very old implementations, continued support for Version-0 implementations is not recommended. It is recommended that new implementations follow the guidelines below when interoperating with older implementations.

Version 3 neither changes the protocol in any significant way nor obsoletes previous versions or existing implementations. The main motivation for the new version is to refine the analysis and implementation models for new applications at much higher network speeds to the gigabit-per-second regime and to provide for the enhanced stability, accuracy and precision required at such speeds. In particular, the sources of time and frequency errors have been rigorously examined and error bounds established in order to improve performance, provide a model for correctness assertions and indicate timekeeping quality to the user. Version 3 also incorporates two new optional features, (1) an algorithm to combine the offsets of a number of peer time servers in order to enhance accuracy and (2) improved local-clock algorithms which allow the poll intervals on all synchronization paths to be substantially increased in order to reduce network overhead. Following is a summary of previous versions of the protocol together with details of the Version 3 changes.

- **1.** Version 1 supports no modes other than symmetric-active and symmetric-passive, which are determined by inspecting the port-number fields of the UDP packet header. The peer mode can be determined explicitly from the packet-mode variable (`pkt.mode`) if it is nonzero and implicitly from the source port (`pkt.peerport`) and destination port (`pkt.hostport`) variables if it is zero. For the case where `pkt.mode` is zero the mode is determined as follows:

pkt.peerport	pkt.hostport	Mode
NTP.PORT	NTP.PORT	symmetric active
NTP.PORT	not NTP.PORT	server
not NTP.PORT	NTP.PORT	client
not NTP.PORT	not NTP.PORT	not possible

Note that it is not possible in this case to distinguish between symmetric active and symmetric passive modes. Use of the pkt.mode and NTP.PORT variables in this way is not recommended and may not be supported in future versions of the protocol. The low-order three bits of the first octet, specified as zero in Version 1, are used for the mode field in Version 2. Version-2 and Version-3 implementations interoperating with Version-1 implementations should operate in a passive mode only and use the value one in the version number (pkt.version) field and zero in the mode (pkt.mode) field in transmitted messages.

- **2.** Version 1 does not support the NTP control message described in Appendix B. Certain old versions of the Unix NTP daemon ntpd use the high-order bits of the stratum field (pkt.stratum) for control and monitoring purposes. While these bits are never set during normal Version-1, Version-2 or Version-3 operations, new implementations may use the NTP reserved mode 6 described in Appendix B and/or private reserved mode 7 for special purposes, such as remote control and monitoring, and in such cases the format of the packet following the first octet can be arbitrary. While there is no guarantee that different implementations can interoperate using private reserved mode 7, it is recommended that vanilla ASCII format be used whenever possible.
- **3.** Version 1 does not support authentication. The key identifiers, cryptographic keys and procedures described in Appendix C are new to Version 2 and continued in Version 3, along with the corresponding variables, procedures and authenticator fields. In the NTP message described in Appendix A and NTP control message described in Appendix B the format and contents of the header fields are independent of the authentication mechanism and the authenticator itself follows the header fields, so that previous versions will ignore the authenticator.
- **4.** In Version 1 the total dispersion (pkt.rootdispersion) field of the NTP header was called the estimated drift rate, but not used in the protocol or timekeeping procedures. Implementations of the Version-1 protocol typically set this field to the current value of the skew-compensation register, which is a signed quantity. In a Version 2 implementation apparent large values in this field may affect the order considered in the clock-selection procedure. Version-2 and Version-3 implementations interoperating with older implementations should assume this field is zero, regardless of its actual contents.
- **5.** Version 2 and Version 3 incorporate several sanity checks designed to avoid disruptions due to unsynchronized, duplicate or bogus timestamp information. The checks in Version 3 are specifically designed to detect lost or duplicate packets and resist invalid timestamps. The leap-indicator bits are set to show the unsynchronized state if updates are not received

from a reference source for a considerable time or if the reference source has not received updates for a considerable time. Some Version-1 implementations could claim valid synchronization indefinitely following loss of the reference source.

- **6.** The clock-selection procedure of Version 2 was considerably refined as the result of accumulated experience with the Version-1 implementation. Additional sanity checks are included for authentication, range bounds and to avoid use of very old data. The candidate list is sorted twice, once to select a relatively few robust candidates from a potentially large population of unruly peers and again to order the resulting list by measurement quality. As in Version 1, The final selection procedure repeatedly casts out outliers on the basis of weighted dispersion.
- **7.** The local-clock procedure of Version 2 were considerably improved over Version 1 as the result of analysis, simulation and experience. Checks have been added to warn that the oscillator has gone too long without update from a reference source. The compliance register has been added to improve frequency stability to the order of a millisecond per day. The various parameters were retuned for optimum loop stability using measured data over typical Internet paths and with typical local-clock hardware. In version 3 the phase-lock loop model was further refined to provide an adaptive-bandwidth feature that automatically adjusts for the inherent stabilities of the reference clock and local clock while providing optimum loop stability in each case.
- **8.** Problems in the timekeeping calculations of Version 1 with high-speed LANs were found and corrected in Version 2. These were caused by jitter due to small differences in clock rates and different precisions between the peers. Subtle bugs in the Version-1 reachability and polling-rate control were found and corrected. The `peer.valid` and `sys.hold` variables were added to avoid instabilities when the reference source changes rapidly due to large dispersive delays under conditions of severe network congestion. The `peer.config`, `peer.authenable` and `peer.authentic` bits were added to control special features and simplify configuration.
- **9.** In Version 3 The local-clock algorithm has been overhauled to improve stability and accuracy. Appendix G presents a detailed mathematical model and design example which has been refined with the aid of feedback-control analysis and extensive simulation using data collected over ordinary Internet paths. Section 5 of RFC-1119 on the NTP local clock has been completely rewritten to describe the new algorithm. Since the new algorithm can result in message rates far below the old ones, it is highly recommended that they be used in new implementations. Note that this algorithm is not integral to the NTP protocol specification itself and its use does not affect interoperability with previous versions or existing implementations; however, in order to insure overall NTP subnet stability in the Internet, it is essential that the local-clock characteristics of all NTP time servers conform to the analytical models presented previously and in this document.
- **10.** In Version 3 a new algorithm to combine the offsets of a number of peer time servers is presented in Appendix F. This algorithm is modelled on those used by national standards laboratories to combine the weighted offsets from a number of standard clocks to construct a

synthetic laboratory time scale more accurate than that of any clock separately. It can be used in an NTP implementation to improve accuracy and stability and reduce errors due to asymmetric paths in the Internet. The new algorithm has been simulated using data collected over ordinary Internet paths and, along with the new local-clock algorithm, implemented and tested in the Fuzzball time servers now running in the Internet. Note that this algorithm is not integral to the NTP protocol specification itself and its use does not affect interoperability with previous versions or existing implementations.

- **11.** Several inconsistencies and minor errors in previous versions have been corrected in Version 3. The description of the procedures has been rewritten in pseudo-code augmented by English commentary for clarity and to avoid ambiguity. Appendix I has been added to illustrate C-language implementations of the various filtering and selection algorithms suggested for NTP. Additional information is included in Section 5 and in Appendix E, which includes the tutorial material formerly included in Section 2 of RFC-1119, as well as much new material clarifying the interpretation of timescales and leap seconds.
- **12.** Minor changes have been made in the Version-3 local-clock algorithms to avoid problems observed when leap seconds are introduced in the UTC time scale and also to support an auxiliary precision oscillator, such as a cesium clock or timing receiver, as a precision time base. In addition, changes were made to some procedures described in Section 3 and in the clock-filter and clock-selection procedures described in Section 4. While these changes were made to correct minor bugs found as the result of experience and are recommended for new implementations, they do not affect interoperability with previous versions or existing implementations in other than minor ways (at least until the next leap second).
- **13.** In Version 3 changes were made to the way delay, offset and dispersion are defined, calculated and processed in order to reliably bound the errors inherent in the time-transfer procedures. In particular, the error accumulations were moved from the delay computation to the dispersion computation and both included in the clock filter and selection procedures. The clock-selection procedure was modified to remove the first of the two sorting/discarding steps and replace with an algorithm first proposed by Marzullo and later incorporated in the Digital Time Service. These changes do not significantly affect the ordinary operation of or compatibility with various versions of NTP, but they do provide the basis for formal statements of correctness as described in Appendix H.

6.50 Appendix E

The NTP Timescale and its Chronometry

Introduction

Following is an extended discussion on computer network chronometry, which is the precise determination of computer time and frequency relative to international standards and the determination of conventional civil time and date according to the modern calendar. It describes the methods conventionally used to establish civil time and date and the various

timescales now in use. In particular, it characterizes the Network Time Protocol (NTP) time scale relative to the Coordinated Universal Time (UTC) time scale, and establishes the precise interpretation of UTC leap seconds in NTP.

In the following discussion the terms time, oscillator, clock, epoch, calendar, date and time scale are used in a technical sense. Strictly speaking, the time of an event is an abstraction which determines the ordering of events in some given frame of reference. An oscillator is a generator capable of precise frequency (relative to the given frame of reference) to a specified tolerance. A clock is an oscillator together with a counter which records the (fractional) number of cycles since being initialized with a given value at a given time. The value of the counter at any given time is called its epoch at that time. In general, epoches are not continuous and depend on the precision of the counter.

A calendar is a mapping from epoch in some frame of reference to the times and dates used in everyday life. Since multiple calendars are in use today and sometimes disagree on the dating of the same events in the past, the chronometry of past and present events is an art practiced by historians. One of the goals of this discussion is to provide a standard chronometry for precision dating of present and future events in a global networking community. To synchronize frequency means to adjust the oscillators in the network to run at the same frequency, to synchronize time means to set the clocks so that all agree at a particular epoch with respect to UTC, as provided by international standards, and to synchronize clocks means to synchronize them in both frequency and time.

In order to synchronize clocks, there must be some way to directly or indirectly compare them in time and frequency. The ultimate frame of reference for our world consists of the cosmic oscillators: the Sun, Moon and other galactic orbiters. Since the frequencies of these oscillators are relatively unstable and not known exactly, the ultimate reference standard oscillator has been chosen by international agreement as a synthesis of many observations of an atomic transition of exquisite stability. The epoches of each heavenly and Earthbound oscillator defines a distinctive time scale, not necessarily always continuous, relative to the standard oscillator. Another goal of this presentation is to describe a standard chronometry to rationalize conventional computer time and UTC; in particular, how to handle leap seconds.

Primary Frequency and Time Standards

A primary frequency standard is an oscillator that can maintain extremely precise frequency relative to a physical phenomenon, such as a transition in the orbital states of an electron. Presently available atomic oscillators are based on the transitions of the hydrogen, cesium and rubidium atoms. Table

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

Table 7. Characteristics of Standard Oscillators

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

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

As the telephone networks of the world are evolving rapidly to digital technology, consideration should be given to the methods used for frequency synchronization in digital networks. A network of clocks in which each oscillator is phase-locked to a single frequency standard is called isochronous, while a network in which some oscillators are phase-locked to different master oscillators, but with the master oscillators closely synchronized in frequency (not necessarily phase locked), to a single frequency standard is called plesiochronous. In plesiochronous systems the phase of some oscillators can slip relative to others and cause occasional data errors in synchronous transmission systems.

The industry has agreed on a classification of clock oscillators as a function of minimum accuracy, minimum stability and other factors [ALL74a]. There are three factors which determine the classification: stability, jitter and wander. Stability refers to the systematic variation of frequency with time and is synonymous with aging, drift, trends, etc. Jitter (also called timing jitter) refers to short-term variations in frequency with components greater than 10 Hz, while wander refers to long-term variations in frequency with components less than 10 Hz. The classification determines the oscillator stratum (not to be confused with the NTP stratum), with the more accurate oscillators assigned the lower strata and less accurate oscillators the higher strata:

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

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

Time and Frequency Dissemination

In order that atomic and civil time can be coordinated throughout the world, national administrations operate primary time and frequency standards and coordinate them cooperatively by observing various radio broadcasts and through occasional use of portable atomic clocks. Most seafaring nations of the world operate some sort of broadcast time service for the purpose of calibrating chronographs, which are used in conjunction with ephemeris data to determine navigational position. In many countries the service is primitive and limited to seconds-pips broadcast by marine communication stations at certain hours. For instance, a chronograph error of one second represents a longitudinal position error of about 0.23 nautical mile at the Equator.

The U.S. National Institute of Standards and Technology (NIST - formerly National Bureau of Standards) operates three radio services for the dissemination of primary time and frequency information. One of these uses high-frequency (HF or CCIR band 7) transmissions on frequencies of 2.5, 5, 10, 15 and 20 MHz from Fort Collins, CO (WWV), and Kauai, HI (WWVH). Signal propagation is usually by reflection from the upper ionospheric layers, which vary in height and composition throughout the day and season and result in unpredictable delay variations at the receiver. The timecode is transmitted over a 60-second interval at a data rate of 1 bps using a 100-Hz subcarrier on the broadcast signal. The timecode information includes UTC time-day information, but does not currently include year or leap-second warning. While these transmissions and those of Canada from Ottawa, Ontario (CHU), and other countries can be received over large areas in the western hemisphere, reliable frequency comparisons can be made only to the order of 10^{-7} and time accuracies are limited to the order of a millisecond [BLA74]. Radio clocks which operate with these transmissions include the Traconex 1020, which provides accuracies to about ten milliseconds and is priced in the \$1,500 range.

A second service operated by NIST uses low-frequency (LF or CCIR band 5) transmissions on 60 kHz from Boulder, CO (WWVB), and can be received over the continental U.S. and adjacent coastal areas. Signal propagation is via the lower ionospheric layers, which are relatively stable and have predictable diurnal variations in height. The timecode is transmitted over a 60-second interval at a rate of 1 pps using periodic reductions in carrier power. With appropriate receiving and averaging techniques and corrections for diurnal and seasonal propagation effects, frequency comparisons to within 10-11 are possible and time accuracies of from a few to 50 microseconds can be obtained [BLA74]. Some countries in western Europe operate similar services which use transmissions on 60 kHz from Rugby, U.K. (MSF), and on 77.5 kHz from Mainflingen, West Germany (DCF77). The timecode information includes UTC time-day-year information and leap-second warning. Radio clocks which operate with these transmissions include the Spectracom 8170 and Kinometrics/TrueTime 60-DC and LF-DC, which provide accuracies to a millisecond or less and are priced in the \$2,500 range. However, these receivers do not extract the year information and leap-second warning.

The third service operated by NIST uses ultra-high frequency (UHF or CCIR band 9) transmissions on about 468 MHz from the Geosynchronous Orbit Environmental Satellites (GOES), three of which cover the western hemisphere. The timecode is interleaved with messages used to interrogate remote sensors and consists of 60 4-bit binary-coded decimal words transmitted over an interval of 30 seconds. The timecode information includes UTC time-day-year information and leap-second warning. Radio clocks which operate with these transmissions include the Kinometrics/TrueTime 468-DC, which provides accuracies to 0.5 ms and is priced in the \$6,000 range. However, this receiver does not extract the year information and leap-second warning.

The U.S. Department of Defense is developing the Global Positioning System (GPS) for worldwide precision navigation. This system will eventually provide 24-hour worldwide coverage using a constellation of 24 satellites in 12-hour orbits. For time-transfer applications GPS has a potential accuracy in the order of a few nanoseconds; however, various considerations of defense policy may limit accuracy to hundreds of nanoseconds [VAN84]. The timecode information includes GPS time and UTC correction; however, there appears to be no leap-second warning. Radio clocks which operate with these transmissions include the Kinometrics/TrueTime GPS-DC, which provides accuracies to 200 μ s and is priced in the \$12,000 range. However, since only about half the satellites have been launched, expensive rubidium or quartz oscillators are necessary to preserve accuracy during outages. Also, since this is a single-channel receiver, it must be supplied with geographic coordinates within a degree from an external source before operation begins.

The U.S. Coast Guard, along with agencies of other countries, has operated the LORAN-C [FRA82] radionavigation system for many years. It currently provides time-transfer accuracies of less than a microsecond and eventually may achieve 100 ns within the ground-wave coverage area of a few hundred kilometers from the transmitter. Beyond the ground wave area signal propagation is via the lower ionospheric layers, which decreases accuracies to the order of 50 μ s. With the recent addition of the Mid-Continent Chain, the deployment of LORAN-C transmitters now provides complete coverage of the U.S. LORAN-C timing receivers, such as the Austron 2000, are specialized and extremely expensive (up to \$20,000). They are used primarily to monitor local cesium clocks and are not suited for unattended, automatic

operation. While the LORAN-C system provides a highly accurate frequency and time reference within the ground wave area, there is no timecode modulation, so the receiver must be supplied with UTC time to within a few tens of seconds from an external source before operation begins.

The OMEGA [VAS78] radionavigation system operated by the U.S. Navy and other countries consists of eight very-low-frequency (VLF or CCIR band 4) transmitters operating on frequencies from 10.2 to 13.1 kHz and providing 24-hour worldwide coverage. With appropriate receiving and averaging techniques and corrections for propagation effects, frequency comparisons and time accuracies are comparable to the LF systems, but with worldwide coverage [BLA74]. Radio clocks which operate with these transmissions include the Kinemetrics/TrueTime OM-DC, which provides accuracies to 1 ms and is priced in the \$3,500 range. While the OMEGA system provides a highly accurate frequency reference, there is no timecode modulation, so the receiver must be supplied with geographic coordinates within a degree and UTC time within five seconds from an external source before operation begins. There are several other VLF services intended primarily for worldwide data communications with characteristics similar to OMEGA. These services can be used in a manner similar to OMEGA, but this requires specialized techniques not suited for unattended, automatic operation.

Note that not all transmission formats used by NIST radio broadcast services [NBS79] and no currently available radio clocks include provisions for year information and leap-second warning. This information must be determined from other sources. NTP includes provisions to distribute advance warnings of leap seconds using the leap-indicator bits described in the NTP specification. The protocol is designed so that these bits can be set manually or by the radio timecode at the primary time servers and then automatically distributed throughout the synchronization subnet to all other time servers.

Calendar Systems

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

The ancient Egyptian lunisolar calendar had twelve 30-day lunar months, but was guided by the seasonal appearance of the star Sirius (Sothis). In order to reconcile this calendar with the solar year, a civil calendar was invented by adding five intercalary days for a total of 365 days. However, in time it was observed that the civil year was about one-fourth day shorter than the actual solar year and thus would precess relative to it over a 1460-year cycle called the Sothic cycle. Along with the Shang Chinese, the ancient Egyptians had thus established the solar year at 365.25 days, or within about 11 minutes of the present measured value. In 432 BC, about a

century after the Chinese had done so, the Greek astronomer Meton calculated there were 110 lunar months of 29 days and 125 lunar months of 30 days for a total of 235 lunar months in 6940 solar days, or just over 19 years. The 19-year cycle, called the Metonic cycle, established the lunar month at 29.532 solar days, or within about two minutes of the present measured value.

The Roman republican calendar was based on a lunar year and by 50 BC was eight weeks out of step with the solar year. Julius Caesar invited the Alexandrian astronomer Sosigenes to redesign the calendar, which led to the adoption in 46 BC of the Julian calendar. This calendar is based on a year of 365 days with an intercalary day inserted every four years. However, for the first 36 years an intercalary day was mistakenly inserted every three years instead of every four. The result was 12 intercalary days instead of nine, and a series of corrections that was not complete until 8 AD.

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

By 1545 the discrepancy in the Julian year relative to the solar year had accumulated to ten days. In 1582, following suggestions by the astronomers Christopher Clavius and Luigi Lilio, Pope Gregory XIII issued a papal bull which decreed, among other things, that the solar year would consist of 365.2422 days. In order to more closely approximate the new value, only those centennial years divisible by 400 would be leap years, while the remaining centennial years would not, making the actual value 365.2425, or within about 26 seconds of the current measured value. Since the beginning of the Common Era and prior to 1990 there were 474 intercalary days inserted in the Julian calendar, but 14 of these were removed in the Gregorian calendar. While the Gregorian calendar is in use throughout most of the world today, some countries did not adopt it until early in the twentieth century. While it remains a fascinating field for time historians, the above narrative provides conclusive evidence that conjugating calendar dates of significant events and assigning NTP timestamps to them is approximate at best. In principle, reliable dating of such events requires only an accurate count of the days relative to some globally alarming event, such as a comet passage or supernova explosion; however, only historically persistent and politically stable societies, such as the ancient Chinese and Egyptian, and especially the classic Maya, possessed the means and will to do so.

The Modified Julian Day System

In order to measure the span of the universe or the decay of the proton, it is necessary to have a standard day-numbering plan. Accordingly, the International Astronomical Union has adopted the use of the standard second and Julian Day Number (JDN) to date cosmological events and related phenomena. The standard day consists of 86,400 standard seconds, where time is expressed as a fraction of the whole day, and the standard year consists of 365.25

standard days.

In the scheme devised in 1583 by the French scholar Joseph Julius Scaliger and named after his father, Julius Caesar Scaliger, JDN 0.0 corresponds to 12h (noon) on the first day of the Julian Era, 1 January 4713 BC. The years prior to the Common Era (BC) are reckoned according to the Julian calendar, while the years of the Common Era (AD) are reckoned according to the Gregorian calendar. Since 1 January 1 AD in the Gregorian calendar corresponds to 3 January 1 in the Julian calendar [DER90], JDN 1,721,426.0 corresponds to 12h on the first day of the Common Era, 1 January 1 AD. The Modified Julian Date (MJD), which is sometimes used to represent dates near our own era in conventional time and with fewer digits, is defined as $MJD = JD - 2,400,000.5$. Following the convention that our century began at 0h on 1 January 1900, at which time the tropical year was already 12h old, that eclectic instant corresponds to MJD 15,020.0. Thus, the Julian time scale ticks in standard (atomic) 365.25-day centuries and was set to a given value at the approximate epoch of a cosmic event which apparently synchronized the entire human community, the origin of the Common Era.

Determination of Frequency

For many years the most important use of time and frequency information was for worldwide navigation and space science, which depend on astronomical observations of the Sun, Moon and stars [JOR85]. Sidereal time is based on the transit of stars across the celestial meridian of an observer. The mean sidereal day is 23 hours, 56 minutes and 4.09 seconds, but varies about ± 30 ms throughout the year due to polar wandering and orbit variations. Ephemeris time is based on tables with which a standard time interval such as the tropical year - one complete revolution of the Earth around the Sun - can be determined through observations of the Sun, Moon and planets. In 1958 the standard second was defined as $1/31,556,925.9747$ of the tropical year that began this century. On this scale the tropical year is 365.2421987 days and the lunar month - one complete revolution of the Moon around the Earth - is 29.53059 days; however, the actual tropical year can be determined only to an accuracy of about 50 ms and has been increasing by about 5.3 ms per year.

Of the three heavenly oscillators readily apparent to ancient mariners and astronomers - the Earth rotation about its axis, the Earth revolution around the Sun and the Moon revolution around the Earth - none of the three have the intrinsic stability, relative to modern technology, to serve as a standard reference oscillator. In 1967 the standard second was redefined as $9,192,631,770$ periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the cesium-133 atom. Since 1972 the time and frequency standards of the world have been based on International Atomic Time (TAI), which is defined and maintained using multiple cesium-beam oscillators to an accuracy of a few parts in 10^{13} , or better than a microsecond per day. Note that, while this provides an extraordinarily precise time scale, it does not necessarily agree with conventional solar time and may not in fact even be absolutely uniform, unless subtle atomic conspiracies can be ruled out.

Determination of Time and Leap Seconds

The International Bureau of Weights and Measures (IBWM) uses astronomical observations provided by the U.S. Naval Observatory and other observatories to determine UTC. Starting from apparent mean solar time as observed, the UT0 time scale is determined using corrections for Earth

UTC Date	MJD	NTP Time	Offset
01 Jan 72	41,317	2,272,060,800	0
30 Jun 72	41,498	2,287,785,600	1
31 Dec 72	41,682	2,303,683,200	2
31 Dec 73	42,047	2,335,219,200	3
31 Dec 74	42,412	2,366,755,200	4
31 Dec 75	42,777	2,398,291,200	5
31 Dec 76	43,143	2,429,913,600	6
31 Dec 77	43,508	2,461,449,600	7
31 Dec 78	43,873	2,492,985,600	8
31 Dec 79	44,238	2,524,521,600	9
30 Jun 81	44,785	2,571,782,400	10
30 Jun 82	45,150	2,603,318,400	11
30 Jun 83	45,515	2,634,854,400	12
30 Jun 85	46,246	2,698,012,800	13
31 Dec 87	47,160	2,776,982,400	14
31 Dec 89	47,891	2,840,140,800	15
31 Dec 90	48,256	2,871,676,800	16

Table 8. Table of Leap-Second Insertions

orbit and inclination (the Equation of Time, as used by sundials), the UT1 (navigator's) time scale by adding corrections for polar migration and the UT2 time scale by adding corrections for known periodicity variations. While standard frequencies are based on TAI, conventional civil time is based on UT1, which is presently slowing relative to TAI by a fraction of a second per year. When the magnitude of correction approaches 0.7 second, a leap second is inserted or deleted in the TAI time scale on the last day of June or December.

For the most precise coordination and time stamping of events since 1972, it is necessary to know when leap seconds are implemented in UTC and how the seconds are numbered. As specified in CCIR Report 517, which is reproduced in [BLA74], a leap second is inserted following second 23:59:59 on the last day of June or December and becomes second 23:59:60 of that day. A leap second would be deleted by omitting second 23:59:59 on one of these days, although this has never happened. Leap seconds were inserted prior to 1 January 1991 on the occasions listed in Table 8 (courtesy U.S. Naval Observatory). Published IBWM corrections consist not only of leap seconds, which result in step discontinuities relative to TAI, but 100-ms UT1 adjustments called DUT1, which provide increased accuracy for navigation and space science.

Note that the NTP time column actually shows the epoch following the last second of the day

given in the UTC date and MJD columns (except for the first line), which is the precise epoch of insertion. The offset column shows the cumulative seconds offset between the uncoordinated (Julian) time scale and the UTC time scale; that is, the number of seconds to add to the Julian clock in order to maintain nominal agreement with the UTC clock. Finally, note that the epoch of insertion is relative to the time scale immediately prior to that epoch; e.g., the epoch of the 31 December 90 insertion is determined on the time scale in effect following the 31 December 1990 insertion, which means the actual insertion relative to the Julian clock is fourteen seconds later than the apparent time on the UTC time scale.

The UTC time scale thus ticks in standard (atomic) seconds and was set to the value 0h MJD 41,317.0 at the epoch determined by astronomical observation to be 0h on 1 January 1972 according to the Gregorian calendar; that is, the inaugural tick of the UTC Era. In fact, the inaugural tick which synchronized the cosmic oscillators, Julian clock, UTC clock and Gregorian calendar forevermore was displaced about ten seconds from the civil clock then in use, while the GPS clock is ahead of the UTC clock by six seconds in late 1990. Subsequently, the UTC clock has marched backward relative to the Julian time scale exactly one second on scheduled occasions at monumental epoches embedded in the institutional memory of our civilization. Note in passing that leap-second adjustments affect the number of seconds per day and thus the number of seconds per year. Apparently, should we choose to worry about it, the UTC clock, Julian clock and various cosmic clocks will inexorably drift apart with time until rationalized by some future papal bull.

The NTP Timescale and Reckoning with UTC The NTP time scale is based on the UTC time scale, but not necessarily always coincident with it. At 0h on 1 January 1972 (MJD 41,317.0), the first tick of the UTC Era, the NTP clock was set to 2,272,060,800, representing the number of standard seconds since 0h on 1 January 1900 (MJD 15,020.0). The insertion of leap seconds in UTC and subsequently into NTP does not affect the UTC or NTP oscillator, only the conversion to conventional civil UTC time. However, since the only institutional memory available to NTP are the UTC timecode broadcast services, the NTP time scale is in effect reset to UTC as each timecode is received. Thus, when a leap second is inserted in UTC and subsequently in NTP, knowledge of all previous leap seconds is lost.

Another way to describe this is to say there are as many NTP timescales as historic leap seconds. In effect, a new time scale is established after each new leap second. Thus, all previous leap seconds, not to mention the apparent origin of the time scale itself, lurch backward one second as each new time scale is established. If a clock synchronized to NTP in 1990 was used to establish the UTC epoch of an event that occurred in early 1972 without correction, the event would appear fifteen seconds late relative to UTC. However, NTP primary time servers resolve the epoch using the broadcast timecode, so that the NTP clock is set to the broadcast value on the current time scale. As a result, for the most precise determination of epoch relative to the historic UTC clock, the user must subtract from the apparent NTP epoch the offsets shown in Table 8 at the relative epoches shown. This is a feature of almost all present day time-distribution mechanisms.

The chronometry involved can be illustrated with the help of Figure 8, which shows the details of seconds numbering just before, during and after the last scheduled leap insertion at 23:59:59 on 31 December 1989. Notice the NTP leap bits are set on the day prior to insertion,

as indicated by the <169>+<170> symbols on the figure. Since this makes the day one second longer than usual, the NTP day rollover will not occur until the end of the first occurrence of second 800. The UTC time conversion routines must notice the apparent time and the leap bits and handle the time scale conversions accordingly. Immediately after the leap insertion both timescales resume ticking the seconds as if

	UTC		NTP	
	hours	seconds	kiloseconds	seconds
31 Dec 90	23:59	:59	2,871,590	,399 +
(leap)	23:59	:60	2,871,590	,400 +
1 Jan 91	00:00	:00	2,871,590	,400
	00:00	:01	2,871,590	,401

Figure 8. Comparison of UTC and NTP Timescales at Leap

the leap had never happened. The chronometric correspondence between the UTC and NTP timescales continues, but NTP has forgotten about all past leap insertions. In NTP chronometric determination of UTC time intervals spanning leap seconds will thus be in error, unless the exact times of insertion are known.

It is possible that individual systems may use internal data formats other than the NTP timestamp format, which is represented in seconds to a precision of about 200 picoseconds; however, a persuasive argument exists to use a two-part representation, one part for whole days (MJD or some fixed offset from it) and the other for the seconds (or some scaled value, such as milliseconds). This not only facilitates conversion between NTP and conventional civil time, but makes the insertion of leap seconds much easier. All that is required is to change the modulus of the seconds counter, which on overflow increments the day counter. This design insures that continuity of the time scale is assured, even if outside synchronization is lost before, during or after leap-second insertion. Since timestamp data are unaffected, synchronization is assured, even if timestamp data are in flight at the instant and originated before or at that instant.

6.51 Appendix F

The NTP Clock-Combining Algorithm

Introduction

A common problem in synchronization subnets is systematic time-offset errors resulting from asymmetric transmission paths, where the networks or transmission media in one direction are substantially different from the other. The errors can range from microseconds on high-speed ring networks to large fractions of a second on satellite/landline paths. It has been found experimentally that these errors can be considerably reduced by combining the apparent

offsets of a number of time servers to produce a more accurate working offset. Following is a description of the combining method used in the NTP implementation for the Fuzzball [MIL88b]. The method is similar to that used by national standards laboratories to determine a synthetic laboratory time scale from an ensemble of cesium clocks [ALL74b]. These procedures are optional and not required in a conforming NTP implementation.

In the following description the stability of a clock is how well it can maintain a constant frequency, the accuracy is how well its frequency and time compare with national standards and the precision is how precisely these quantities can be maintained within a particular timekeeping system. Unless indicated otherwise, The offset of two clocks is the time difference between them, while the skew is the frequency difference (first derivative of offset with time) between them. Real clocks exhibit some variation in skew (second derivative of offset with time), which is called drift.

Determining Time and Frequency

Figure 9 shows the overall organization of the NTP time-server model. Timestamps exchanged with possibly many other subnet peers are used to determine individual roundtrip delays and clock offsets relative to each peer as described in the NTP specification. As shown in the figure, the computed delays and offsets are processed by the clock filter to reduce incidental timing noise and the most accurate and reliable subset determined by the clock-selection algorithm. The resulting offsets of this subset are first combined as described below and then processed by the phase-locked loop (PLL). In the PLL the combined effects of the filtering, selection and combining operations is to produce a phase-correction term. This is processed by the loop filter to control the local clock, which functions as a voltage-controlled oscillator (VCO). The VCO furnishes the timing (phase) reference to produce the timestamps used in all calculations.

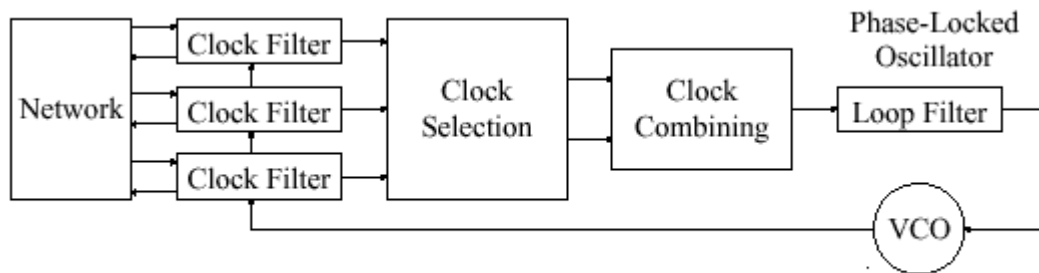


Figure 9. Network Time Protocol

Clock Modelling

The International Standard (SI) definition of time interval is in terms of the standard second: the duration of 9,192,631,770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the cesium-133 atom. Let u represent the standard unit of time interval so defined and $v = 1$ over u be the standard unit of frequency. The epoch, denoted by t , is defined as the reading of a counter that runs at frequency v and

began counting at some agreed initial epoch t_0 , which defines the standard or absolute time scale. For the purposes of the following analysis, the epoch of the standard time scale, as well as the time indicated by a clock will be considered continuous. In practice, time is determined relative to a clock constructed from an atomic oscillator and system of counter/dividers, which defines a time scale associated with that particular oscillator. Standard time and frequency are then determined from an ensemble of such timescales and algorithms designed to combine them to produce a composite time scale approximating the standard time scale.

Let $T(t)$ be the time displayed by a clock at epoch t relative to the standard time scale:

$$T(t) = 1/2 D(t \text{ sub } 0) [t - t \text{ sub } 0]^2 + R(t \text{ sub } 0) [t - t \text{ sub } 0] + T(t \text{ sub } 0) + x(t),$$

where $D(t \text{ sub } 0)$ is the fractional frequency drift per unit time, $R(t \text{ sub } 0)$ the frequency and $T(t \text{ sub } 0)$ the time at some previous epoch t_0 . In the usual stationary model these quantities can be assumed constant or changing slowly with epoch. The random nature of the clock is characterized by $x(t)$, which represents the random noise (jitter) relative to the standard time scale. In the usual analysis the second-order term $D(t \text{ sub } 0)$ is ignored and the noise term $x(t)$ modelled as a normal distribution with predictable spectral density or autocorrelation function.

The probability density function of time offset $p(t - T(t))$ usually appears as a bell-shaped curve centered somewhere near zero. The width and general shape of the curve are determined by $x(t)$, which depends on the oscillator precision and jitter characteristics, as well as the measurement system and its transmission paths. Beginning at epoch t_0 the offset is set to zero, following which the bell creeps either to the left or right, depending on the value of $R(t \text{ sub } 0)$ and accelerates depending on the value of $D(t \text{ sub } 0)$.

Development of a Composite Timescale

Now consider the time offsets of a number of real clocks connected by real networks. A display of the offsets of all clocks relative to the standard time scale will appear as a system of bell-shaped curves slowly precessing relative to each other, but with some further away from nominal zero than others. The bells will normally be scattered over the offset space, more or less close to each other, with some overlapping and some not. The problem is to estimate the true offset relative to the standard time scale from a system of offsets collected routinely between the clocks.

A composite time scale can be determined from a sequence of offsets measured between the n clocks of an ensemble at nominal intervals τ . Let $R_{\text{sub } i}(t \text{ sub } 0)$ be the frequency and $T_{\text{sub } i}(t \text{ sub } 0)$ the time of the i th clock at epoch t_0 relative to the standard time scale and let $\hat{}$ designate the associated estimates. Then, an estimator for T_i computed at t_0 for epoch $t \text{ sub } 0 + \tau$ is

$$\hat{T}_i(t_0 + \tau) = \hat{R}_i(t_0)\tau + T_i(t_0),$$

neglecting second-order terms. Consider a set of n independent time-offset measurements

made between the clocks at epoch $t_0 + \tau$ and let the offset between clock i and clock j at that epoch be $T_{ij}(t_0 + \tau)$, defined as

$$T_{ij}(t_0 + \tau) \equiv T_i(t_0 + \tau) - T_j(t_0 + \tau).$$

Note that $T_{ij} = -T_{ji}$ and $T_{ii} = 0$. Let $w_i(\tau)$ be a previously determined weight factor associated with the i th clock for the nominal interval τ . The basis for new estimates at epoch $t_0 + \tau$ is

$$T_j(t_0 + \tau) = \sum_{i=1}^n w_i(\tau) [\hat{T}_i(t_0 + \tau) + T_{ji}(t_0 + \tau)].$$

That is, the apparent time indicated by the j th clock is a weighted average of the estimated time of each clock at epoch $t_0 + \tau$ plus the time offset measured between the j th clock and that clock at epoch $t_0 + \tau$.

An intuitive grasp of the behavior of this algorithm can be gained with the aid of a few examples. For instance, if $w_i(\tau)$ is unity for the i th clock and zero for all others, the apparent time for each of the other clocks is simply the estimated time $\hat{T}_i(t_0 + \tau)$. If $w_i(\tau)$ is zero for the i th clock, that clock can never affect any other clock and its apparent time is determined entirely from the other clocks. If $w_i(\tau) = 1/n$ for all i , the apparent time of the i th clock is equal to the average of the time estimates computed at t_0 plus the average of the time offsets measured to all other clocks. Finally, in a system with two clocks and $w_i(\tau) = 1/2$ for each, and if the estimated time at epoch $t_0 + \tau$ is fast by 1 s for one clock and slow by 1 s for the other, the apparent time for both clocks will coincide with the standard time scale.

In order to establish a basis for the next interval τ , it is necessary to update the frequency estimate $\hat{R}_i(t_0 + \tau)$ and weight factor $w_i(\tau)$. The average frequency assumed for the i th clock during the previous interval τ is simply the difference between the times at the beginning and end of the interval divided by τ . A good estimator for $\hat{R}_i(t_0 + \tau)$ has been found to be the exponential average of these differences, which is given by

$$\hat{R}_i(t_0 + \tau) = \hat{R}_i(t_0) + \alpha_i [\hat{R}_i(t_0) - \{T_{ij}(t_0 + \tau) - T_{ji}(t_0)\} / \tau],$$

where α_i is an experimentally determined weight factor which depends on the estimated frequency error of the i th clock. In order to calculate the weight factor $w_i(\tau)$, it is necessary to determine the expected error $\epsilon_i(\tau)$ for each clock. In the following, braces " $\{ \}$ " indicate absolute value and brackets " $\langle \rangle$ " indicate the infinite time average. In practice, the infinite averages are computed as exponential time averages. An

estimate of the magnitude of the unbiased error of the i th clock accumulated over the nominal interval τ is

$$\epsilon_i(\tau) = |\hat{T}_i(t_0 + \tau) - T_i(t_0 + \tau)| + \frac{0.8 \langle \epsilon_e^2(\tau) \rangle}{\sqrt{\langle \epsilon_i^2(\tau) \rangle}},$$

where $\epsilon_i(\tau)$ and $\epsilon_e(\tau)$ are the accumulated error of the i th clock and entire clock ensemble, respectively. The accumulated error of the entire ensemble is

$$\langle \epsilon_e^2(\tau) \rangle = \left[\sum_{i=1}^n \frac{1}{\langle \epsilon_i^2(\tau) \rangle} \right]^{-1}.$$

Finally, the weight factor for the i th clock is calculated as

$$w_i(\tau) = \frac{\langle \epsilon_e^2(\tau) \rangle}{\langle \epsilon_i^2(\tau) \rangle}.$$

When all estimators and weight factors have been updated, the origin of the estimation interval is shifted and the new value of t_0 becomes the old value of $t_0 + \tau$.

While not entering into the above calculations, it is useful to estimate the frequency error, since the ensemble clocks can be located some distance from each other and become isolated for some time due to network failures. The frequency-offset error in R_i is equivalent to the fractional frequency y_i ,

$$y_i = \{ \nu_i - \nu_I \} / \nu_I$$

measured between the i th time scale and the standard time scale I . Temporarily dropping the subscript i for clarity, consider a sequence of N independent frequency-offset samples $y(j)$ ($j = 1, 2, \dots, N$) where the interval between samples is uniform and equal to T . Let τ be the nominal interval over which these samples are averaged. The Allan variance $\sigma_y^2(N, T, \tau)$ [ALL74a] is defined as

$$\langle \sigma_y^2(N, T, \tau) \rangle = \left\langle \frac{1}{N-1} \left[\sum_{j=1}^N y(j)^2 - \frac{1}{N} \left(\sum_{j=1}^N y(j) \right)^2 \right] \right\rangle ,$$

A particularly useful formulation is $N = 2$ and $T = \tau$:

$$\langle \sigma_y^2(N = 2, T = \tau, \tau) \rangle \equiv \sigma_y^2(\tau) = \left\langle \frac{[y(j+1) - y(j)]^2}{2} \right\rangle ,$$

so that

$$\sigma_y^2(\tau) = \frac{1}{2(N-1)} \sum_{j=1}^{n-1} [y(j+1) - y(j)]^2 .$$

While the Allan variance has found application when estimating errors in ensembles of cesium clocks, its application to NTP is limited due to the computation and storage burden. As described in the next section, it is possible to estimate errors with some degree of confidence using normal byproducts of NTP processing algorithms.

Application to NTP

The NTP clock model is somewhat less complex than the general model described above. For instance, at the present level of development it is not necessary to separately estimate the time and frequency of all peer clocks, only the time and frequency of the local clock. If the timekeeping reference is the local clock itself, then the offsets available in the peer.offset peer variables can be used directly for the T sub ij quantities above. In addition, the NTP local-clock model incorporates a type-II phase-locked loop, which itself reliably estimates frequency errors and corrects accordingly. Thus, the requirement for estimating frequency is entirely eliminated.

There remains the problem of how to determine a robust and easily computable error estimate epsilon sub i. The method described above, although analytically justified, is most difficult to implement. Happily, as a byproduct of the NTP clock-filter algorithm, a useful error estimate is available in the form of the dispersion. As described in the NTP specification, the dispersion includes the absolute value of the weighted average of the offsets between the chosen offset sample and the $n-1$ other samples retained for selection. The effectiveness of this estimator was compared with the above estimator by simulation using observed timekeeping data and found to give quite acceptable results.

The NTP clock-combining algorithm can be implemented with only minor modifications to the algorithms as described in the NTP specification. Although elsewhere in the NTP specification the use of general-purpose multiply/divide routines has been successfully avoided, there seems to be no way to avoid them in the clock-combining algorithm. However, for best performance the local-clock algorithm described elsewhere in this document should be implemented as well, since the combining algorithms result in a modest increase in phase noise which the revised local-clock algorithm is designed to suppress.

Clock-Combining Procedure

The result of the NTP clock-selection procedure is a set of survivors (there must be at least one) that represent truechimers, or correct clocks. When clock combining is not implemented, one of these peers, chosen as the most likely candidate, becomes the synchronization source and its computed offset becomes the final clock correction. Subsequently, the system variables are adjusted as described in the NTP clock-update procedure. When clock combining is implemented, these actions are unchanged, except that the final clock correction is computed by the clock-combining procedure.

The clock-combining procedure is called from the clock-select procedure. It constructs from the variables of all surviving peers the final clock correction THETA. The estimated error required by the algorithms previously described is based on the synchronization distance LAMBDA computed by the distance procedure, as defined in the NTP specification. The reciprocal of LAMBDA is the weight of each clock-offset contribution to the final clock correction. The following pseudo-code describes the procedure.

```

begin clock-combining procedure
    temp1 <-- 0;
    temp2 <-- 0;
    for (each peer remaining on the candidate list)      /* scan all survivors */
        LAMBDA <-- distance (peer);
        temp <-- 1 over peer.stratum * NTP.MAXDISPERSE + LAMBDA;
        temp1 <-- temp1 + temp;          /* update weight and offset */
        temp2 <-- temp2 + temp * peer.offset;
    endif;
    THETA <-- temp2 over temp1;
    /* compute final correction */
end clock-combining procedure;

```

The value THETA is the final clock correction used by the local-clock procedure to adjust the clock.

6.52 Appendix G

Computer Clock Modelling and Analysis

A computer clock includes some kind of reference oscillator, which is stabilized by a quartz

crystal or some other means, such as the power grid. Usually, the clock includes a prescaler, which divides the oscillator frequency to a standard value, such as 1 MHz or 100 Hz, and a counter, implemented in hardware, software or some combination of the two, which can be read by the processor. For systems intended to be synchronized to an external source of standard time, there must be some means to correct the phase and frequency by occasional vernier adjustments produced by the timekeeping protocol. Special care is necessary in all timekeeping system designs to insure that the clock indications are always monotonically increasing; that is, system time never "runs backwards."

Computer Clock Models

The simplest computer clock consists of a hardware latch which is set by overflow of a hardware counter or prescaler, and causes a processor interrupt or tick. The latch is reset when acknowledged by the processor, which then increments the value of a software clock counter. The phase of the clock is adjusted by adding periodic corrections to the counter as necessary. The frequency of the clock can be adjusted by changing the value of the increment itself, in order to make the clock run faster or slower. The precision of this simple clock model is limited to the tick interval, usually in the order of 10 ms; although in some systems the tick interval can be changed using a kernel variable.

This software clock model requires a processor interrupt on every tick, which can cause significant overhead if the tick interval is small, say in the order less 1 ms with the newer RISC processors. Thus, in order to achieve timekeeping precisions less than 1 ms, some kind of hardware assist is required. A straightforward design consists of a voltage-controlled oscillator (VCO), in which the

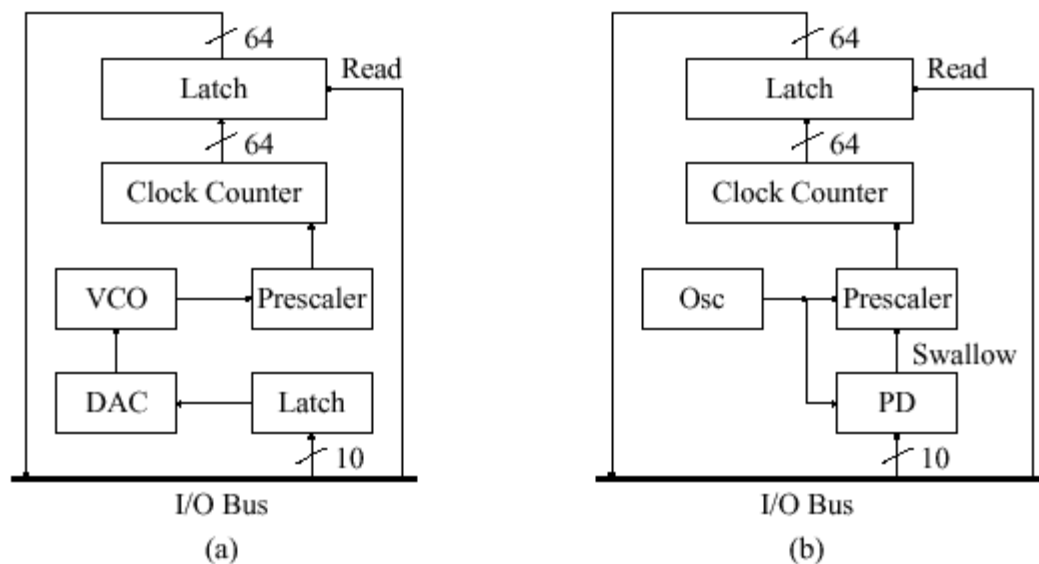


Figure 10. Hardware Clock Models

frequency is controlled by a buffered, digital/analog converter (DAC). Under the assumption

that the VCO tolerance is 10⁻⁴ or 100 parts-per-million (ppm) (a reasonable value for inexpensive crystals) and the precision required is 100 us (a reasonable goal for a RISC processor), the DAC must include at least ten bits.

A design sketch of a computer clock constructed entirely of hardware logic components is shown in Figure 10a. The clock is read by first pulsing the read signal, which latches the current value of the clock counter, then adding the contents of the clock-counter latch and a 64-bit clock-offset variable, which is maintained in processor memory. The clock phase is adjusted by adding a correction to the clock-offset variable, while the clock frequency is adjusted by loading a correction to the DAC latch. In principle, this clock model can be adapted to any precision by changing the number of bits of the prescaler or clock counter or changing the VCO frequency. However, it does not seem useful to reduce precision much below the minimum interrupt latency, which is in the low microseconds for a modern RISC processor.

If it is not possible to vary the oscillator frequency, which might be the case if the oscillator is an external frequency standard, a design such as shown in Figure 10b may be used. It includes a fixed-frequency oscillator and prescaler which includes a dual-modulus swallow counter that can be operated in either divide-by-10 or divide-by-11 modes as controlled by a pulse produced by a programmable divider (PD). The PD is loaded with a value representing the frequency offset. Each time the divider overflows a pulse is produced which switches the swallow counter from the divide-by-10 mode to the divide-by-11 mode and then back again, which in effect "swallows" or deletes a single pulse of the prescaler pulse train.

The pulse train produced by the prescaler is controlled precisely over a small range by the contents of the PD. If programmed to emit pulses at a low rate, relatively few pulses are swallowed per second and the frequency counted is near the upper limit of its range; while, if programmed to emit pulses at a high rate, relatively many pulses are swallowed and the frequency counted is near the lower limit. Assuming some degree of freedom in the choice of oscillator frequency and prescaler ratios, this design can compensate for a wide range of oscillator frequency tolerances.

In all of the above designs it is necessary to limit the amount of adjustment incorporated in any step to insure that the system clock indications are always monotonically increasing. With the software clock model this is assured as long as the increment is never negative. When the magnitude of a phase adjustment exceeds the tick interval (as corrected for the frequency adjustment), it is necessary to spread the adjustments over multiple tick intervals. This strategy amounts to a deliberate frequency offset sustained for an interval equal to the total number of ticks required and, in fact, is a feature of the Unix clock model discussed below.

In the hardware clock models the same considerations apply; however, in these designs the tick interval amounts to a single pulse at the prescaler output, which may be in the order of 1 ms. In order to avoid decreasing the indicated time when a negative phase correction occurs, it is necessary to avoid modifying the clock-offset variable in processor memory and to confine all adjustments to the VCO or prescaler. Thus, all phase adjustments must be performed by means of programmed frequency adjustments in much the same way as with the software clock model described previously.

It is interesting to conjecture on the design of a processor assist that could provide all of the above functions in a compact, general-purpose hardware interface. The interface might consist of a multifunction timer chip such as the AMD 9513A, which includes five 16-bit counters, each with programmable load and hold registers, plus an onboard crystal oscillator, prescaler and control circuitry. A 48-bit hardware clock counter would utilize three of the 16-bit counters, while the fourth would be used as the swallow counter and the fifth as the programmable divider. With the addition of a programmable-array logic device and architecture-specific host interface, this compact design could provide all the functions necessary for a comprehensive timekeeping system.

The Fuzzball Clock Model

The Fuzzball clock model uses a combination of hardware and software to provide precision timing with a minimum of software and processor overhead. The model includes an oscillator, prescaler and hardware counter; however, the oscillator frequency remains constant and the hardware counter produces only a fraction of the total number of bits required by the clock counter. A typical design uses a 64-bit software clock counter and a 16-bit hardware counter which counts the prescaler output. A hardware-counter overflow causes the processor to increment the software counter at the bit corresponding to the frequency $2^N f_p$, where N is the number of bits of the hardware counter and f_p is the counted frequency at the prescaler output. The processor reads the clock counter by first generating a read pulse, which latches the hardware counter, and then adding its contents, suitably aligned, to the software counter.

The Fuzzball clock can be corrected in phase by adding a (signed) adjustment to the software clock counter. In practice, this is done only when the local time is substantially different from the time indicated by the clock and may violate the monotonicity requirement. Vernier phase adjustments determined in normal system operation must be limited to no more than the period of the counted frequency, which is 1 kHz for LSI-11 Fuzzballs. In the Fuzzball model these adjustments are performed at intervals of 4 s, called the adjustment interval, which provides a maximum frequency adjustment range of 250 ppm. The adjustment opportunities are created using the interval-timer facility, which is a feature of most operating systems and independent of the time-of-day clock. However, if the counted frequency is increased from 1 kHz to 1 MHz for enhanced precision, the adjustment frequency must be increased to 250 Hz, which substantially increases processor overhead. A modified design suitable for high precision clocks is presented in the next section.

In some applications involving the Fuzzball model, an external pulse-per-second (pps) signal is available from a reference source such as a cesium clock or GPS receiver. Such a signal generally provides much higher accuracy than the serial character string produced by a radio timecode receiver, typically in the low nanoseconds. In the Fuzzball model this signal is processed by an interface which produces a hardware interrupt coincident with the arrival of the pps pulse. The processor then reads the clock counter and computes the residual modulo 1 s of the clock counter. This represents the local-clock error relative to the pps signal.

Assuming the seconds numbering of the clock counter has been determined by a reliable source, such as a timecode receiver, the offset within the second is determined by the residual computed above. In the NTP local-clock model the timecode receiver or NTP establishes the

time to within ± 128 ms, called the aperture, which guarantees the seconds numbering to within the second. Then, the pps residual can be used directly to correct the oscillator, since the offset must be less than the aperture for a correctly operating timecode receiver and pps signal.

The above technique has an inherent error equal to the latency of the interrupt system, which in modern RISC processors is in the low tens of microseconds. It is possible to improve accuracy by latching the hardware time-of-day counter directly by the pps pulse and then reading the counter in the same way as usual. This requires additional circuitry to prioritize the pps signal relative to the pulse generated by the program to latch the counter.

The Unix Clock Model

The Unix 4.3bsd clock model is based on two system calls, `settimeofday` and `adjtime`, together with two kernel variables `tick` and `tickadj`. The `settimeofday` call unceremoniously resets the kernel clock to the value given, while the `adjtime` call slews the kernel clock to a new value numerically equal to the sum of the present time of day and the (signed) argument given in the `adjtime` call. In order to understand the behavior of the Unix clock as controlled by the Fuzzball clock model described above, it is helpful to explore the operations of `adjtime` in more detail.

The Unix clock model assumes an interrupt produced by an onboard frequency source, such as the clock counter and prescaler described previously, to deliver a pulse train in the 100-Hz range. In principle, the power grid frequency can be used, although it is much less stable than a crystal oscillator. Each interrupt causes an increment called `tick` to be added to the clock counter. The value of the increment is chosen so that the clock counter, plus an initial offset established by the `settimeofday` call, is equal to the time of day in microseconds.

The Unix clock can actually run at three different rates, one corresponding to `tick`, which is related to the intrinsic frequency of the particular oscillator used as the clock source, one to `tick + tickadj` and the third to `tick - tickadj`. Normally the rate corresponding to `tick` is used; but, if `adjtime` is called, the argument `delta` given is used to calculate an interval $\Delta t = \text{delta} \cdot \text{tick} / \text{tickadj}$ during which one or the other of the two rates are used, depending on the sign of `delta`. The effect is to slew the clock to a new value at a small, constant rate, rather than incorporate the adjustment all at once, which could cause the clock to be set backward. With common values of `tick` = 10 ms and `tickadj` = 5 μ s, the maximum frequency adjustment range is $\pm \text{tickadj} / \text{tick} = \pm \{5 \times 10^{-6}\} / \{10^{-2}\}$ or ± 500 ppm. Even larger ranges may be required in the case of some workstations (e.g., SPARC stations) with extremely poor component tolerances.

When precisions not less than about 1 ms are required, the Fuzzball clock model can be adapted to the Unix model by software simulation, as described in Section 5 of the NTP specification, and calling `adjtime` at each adjustment interval. When precisions substantially better than this are required, the hardware microsecond clock provided in some workstations can be used together with certain refinements of the Fuzzball and Unix clock models. The particular design described below is appropriate for a maximum oscillator frequency tolerance of 100 ppm (.01%), which can be

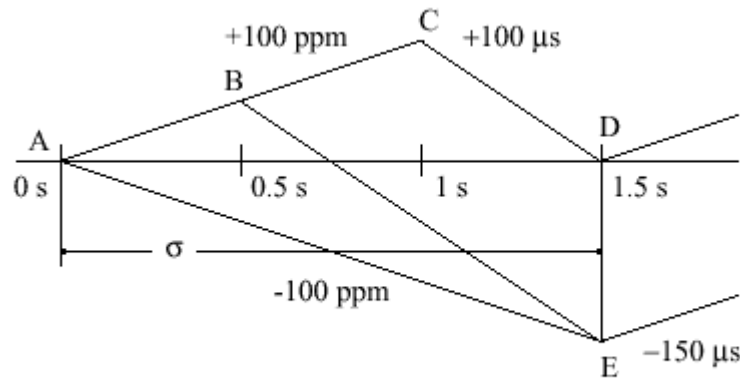


Figure 11. Clock Adjustment Process

obtained using a relatively inexpensive quartz crystal oscillator, but is readily scalable for other assumed tolerances.

The clock model requires the capability to slew the clock frequency over the range ± 100 ppm with an intrinsic oscillator frequency error as great as ± 100 ppm. Figure 11 shows the timing relationships at the extremes of the requirements envelope. Starting from an assumed offset of nominal zero and an assumed error of $+100$ ppm at time 0 s, the line AC shows how the uncorrected offset grows with time. Let σ represent the adjustment interval and a the interval AB, in seconds, and let r be the slew, or rate at which corrections are introduced, in ppm. For an accuracy specification of 100 μ s, then

$$\sigma \leq \frac{100 \mu\text{s}}{100 \text{ ppm}} + \frac{100 \mu\text{s}}{(r - 100) \text{ ppm}} = \frac{r}{r - 100}.$$

The line AE represents the extreme case where the clock is to be steered -100 ppm. Since the slew must be complete at the end of the adjustment interval,

$$a \leq \frac{(r - 200) \sigma}{r}.$$

These relationships are satisfied only if $r > 200$ ppm and $\sigma < 2$ s. Using $r = 300$ ppm for convenience, $\sigma = 1.5$ s and $a < 0.5$ s. For the Unix clock model with $\text{tick} = 10$ ms, this results in the value of $\text{tickadj} = 3\mu\text{s}$.

One of the assumptions made in the Unix clock model is that the period of adjustment computed in the `adjtime` call must be completed before the next call is made. If not, this

results in an error message to the system log. However, in order to correct for the intrinsic frequency offset of the clock oscillator, the NTP clock model requires `adjtime` to be called at regular adjustment intervals of σ s. Using the algorithms described here and the architecture constants in the NTP specification, these adjustments will always complete.

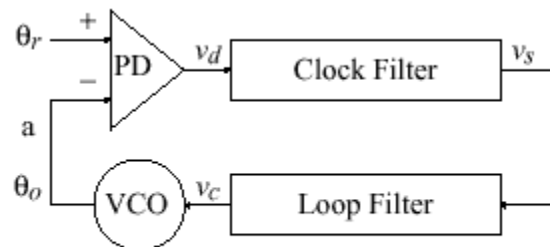


Figure 12. NTP Phase-Lock Loop (PLL) Model

Variable	Description
v_d	phase detector output
v_s	clock filter output
v_c	loop filter output
θ_r	reference phase
θ_o	VCO phase
ω_c	PLL crossover frequency
ω_z	PLL corner frequency

Table 9. Notation Used in PLL Analysis

Parameter	Value	Description
α	2^{-2}	VCO gain
σ	2^2	adjustment interval
τ	2^6	PLL time constant
T	2^3	clock-filter delay
K_f	2^{22}	frequency weight
K_g	2^8	phase weight

Table 10. PLL Parameters

Mathematical Model of the NTP Logical Clock

The NTP logical clock can be represented by the feedback-control model shown in Figure 12. The model consists of an adaptive-parameter, phase-lock loop (PLL), which continuously adjusts the phase and frequency of an oscillator to compensate for its intrinsic jitter, wander and drift. A mathematical analysis of this model developed along the lines of [SMI86] is

presented in following sections, along with a design example useful for implementation guidance in operating-systems environments such as Unix and Fuzzball. Table 9 summarizes the quantities ordinarily treated as variables in the model. By convention, v is used for internal loop variables, θ for phase, ω for frequency and τ for time. Table 10 summarizes those quantities ordinarily fixed as constants in the model. Note that these are all expressed as a power of two in order to simplify the implementation.

In Figure 12 the variable θ_r represents the phase of the reference signal and θ_o the phase of the voltage-controlled oscillator (VCO). The phase detector (PD) produces a voltage v_d representing the phase difference $\theta_r - \theta_o$. The clock filter functions as a tapped delay line, with the output v_s taken at the tap selected by the clock-filter algorithm described in the NTP specification. The loop filter, represented by the equations given below, produces a VCO correction voltage v_c , which controls the oscillator frequency and thus the phase θ_o .

The PLL behavior is completely determined by its open-loop, Laplace transfer function $G(s)$ in the s domain. Since both frequency and phase corrections are required, an appropriate design consists of a type-II PLL, which is defined by the function

$$G(s) = \frac{\omega_c^2}{\tau^2 s^2} \left(1 + \frac{\tau s}{\omega_z}\right),$$

where ω_c is the crossover frequency (also called loop gain), ω_z is the corner frequency (required for loop stability) and τ determines the PLL time constant and thus the bandwidth. While this is a first-order function and some improvement in phase noise might be gained from a higher-order function, in practice the improvement is lost due to the effects of the clock-filter delay, as described below.

The open-loop transfer function $G(s)$ is constructed by breaking the loop at point a on Figure 12 and computing the ratio of the output phase $\theta_o(s)$ to the reference phase $\theta_r(s)$. This function is the product of the individual transfer functions for the phase detector, clock filter, loop filter and VCO. The phase detector delivers a voltage $v_d(t) = \theta_r(t) - \theta_o(t)$, so its transfer function is simply $F_d(s) = 1$, expressed in V/rad. The VCO delivers a frequency change $\Delta\omega = \{d\theta_o(t)/dt\} = \alpha \{v_c(t)\}$, where α is the VCO gain in rad/V-sec and $\theta_o(t) = \alpha \int v_c(t) dt$. Its transfer function is the Laplace transform of the integral, $F_o(s) = \alpha/s$, expressed in rad/V. The clock filter contributes a stochastic delay due to the clock-filter algorithm; but, for present purposes, this delay will be assumed a constant T , so its transfer function is the Laplace transform of the delay, $F_s(s) = e^{-Ts}$. Let $F(s)$ be the transfer function of the loop filter, which has yet to be determined. The open-loop transfer function $G(s)$ is the product of these four individual transfer functions:

$$G(s) = \frac{\omega_c^2}{\tau^2 s^2} \left(1 + \frac{\tau s}{\omega_z}\right) = F_d(s)F_s(s)F(s)F_o(s) = 1e^{-Ts} F(s) \frac{\alpha}{s}.$$

For the moment, assume that the product Ts is small, so that $e^{-Ts} \approx 1$. Making the following substitutions,

$$\omega_c^2 = \frac{\alpha}{K_f} \quad \text{and} \quad \omega_z = \frac{K_g}{K_f}$$

and rearranging yields

$$F(s) = \frac{1}{K_g \tau} + \frac{1}{K_f \tau^2 s},$$

which corresponds to a constant term plus an integrating term scaled by the PLL time constant τ . This form is convenient for implementation as a sampled-data system, as described later.

With the parameter values given in Table 10, the Bode plot of the open-loop transfer function $G(s)$ consists of a -12 dB/octave line which intersects the 0-dB baseline at $\omega_c = 2 \times 10^{-12}$ rad/s, together with a +6 dB/octave line at the corner frequency $\omega_z = 2 \times 10^{14}$ rad/s. The damping factor $\zeta = \omega_c / \omega_z = 2 \times 10^{-14}$ suggests the PLL will be stable and have a large phase margin together with a low overshoot. However, if the clock-filter delay T is not small compared to the loop delay, which is approximately equal to $1/\omega_c$, the above analysis becomes unreliable and the loop can become unstable. With the values determined as above, T is ordinarily small enough to be neglected.

Assuming the output is taken at v_s , the closed-loop transfer function $H(s)$ is

$$H(s) \equiv \frac{v_s(s)}{\theta_r(s)} = \frac{F_d(s)e^{-Ts}}{1 + G(s)}.$$

If only the relative response is needed and the clock-filter delay can be neglected, $H(s)$ can be written

$$H(s) = \frac{1}{1 + G(s)} = \frac{s^2}{s^2 + \frac{\omega_c^2}{\omega_z \tau} s + \frac{\omega_c^2}{\tau^2}}.$$

For some input function $I(s)$ the output function $I(s)H(s)$ can be inverted to find the time response. Using a unit-step input $I(s) = 1$ over s and the values determined as above, This yields a PLL rise time of about 52 minutes, a maximum overshoot of about 4.8 percent in about 1.7 hours and a settling time to within one percent of the initial offset in about 8.7 hours.

Parameter Management

A very important feature of the NTP PLL design is the ability to adapt its behavior to match the prevailing stability of the local oscillator and transmission conditions in the network. This is done using the α and τ parameters shown in Table 10. Mechanisms for doing this are described in following sections.

Adjusting VCO Gain

The α parameter is determined by the maximum frequency tolerance of the local oscillator and the maximum jitter requirements of the timekeeping system. This parameter is usually an architecture constant and fixed during system operation. In the implementation model described below, the reciprocal of α , called the adjustment interval σ , determines the time between corrections of the local clock, and thus the value of α . The value of σ can be determined by the following procedure.

The maximum frequency tolerance for board-mounted, uncompensated quartz-crystal oscillators is probably in the range of 10^{-4} (100 ppm). Many if not most Internet timekeeping systems can tolerate jitter to at least the order of the intrinsic local-clock resolution, called precision in the NTP specification, which is commonly in the range from one to 20 ms. Assuming 10-3 s peak-to-peak as the most demanding case, the interval between clock corrections must be no more than $\sigma = 10 \sup -3 \text{ over } \{2 \times 10 \sup -4\} = 5 \text{ sec}$. For the NTP reference model $\sigma = 4 \text{ sec}$ in order to allow for known features of the Unix operating-system kernel. However, in order to support future anticipated improvements in accuracy possible with faster workstations, it may be useful to decrease σ to as little as one-tenth the present value.

Note that if σ is changed, it is necessary to adjust the parameters K_f and K_g in order to retain the same loop bandwidth; in particular, the same ω_c and ω_z . Since α varies as the reciprocal of σ , if σ is changed to something other than 22, as in Table 10, it is necessary to divide both K_f and K_g by σ over 4 to obtain the new values.

Adjusting PLL Bandwidth

A key feature of the type-II PLL design is its capability to compensate for the intrinsic frequency errors of the local oscillator. This requires a initial period of adaptation in order to refine the frequency estimate (see later sections of this appendix). The tau parameter determines the PLL time constant and thus the loop bandwidth, which is approximately equal to $\{\omega_{\text{sub } c}\}$ over tau. When operated with a relatively large bandwidth small tau, as in the analysis above, the PLL adapts quickly to changes in the input reference signal, but has poor long term stability. Thus, it is possible to accumulate substantial errors if the system is deprived of the reference signal for an extended period. When operated with a relatively small bandwidth large tau, the PLL adapts slowly to changes in the input reference signal, and may even fail to lock onto it. Assuming the frequency estimate has stabilized, it is possible for the PLL to coast for an extended period without external corrections and without accumulating significant error.

In order to achieve the best performance without requiring individual tailoring of the loop bandwidth, it is necessary to compute each value of tau based on the measured values of offset, delay and dispersion, as produced by the NTP protocol itself. The traditional way of doing this in precision timekeeping systems based on cesium clocks, is to relate tau to the Allan variance, which is defined

Variable	Value	Description
μ		update interval
ρ		poll interval
f		frequency error
g		phase error
h		compliance
K_h	2^{13}	compliance weight
K_s	2^4	compliance maximum
K_t	2^{14}	compliance multiplier
K_u	2^0	poll-interval factor

Table 11. Notation Used in PLL Analysis

as the mean of the first-order differences of sequential samples measured during a specified interval tau,

$$\sigma_y^2(\tau) = \frac{1}{2(N-1)} \sum_{i=1}^{N-1} [y(i+1) - y(i)]^2,$$

where y is the fractional frequency measured with respect to the local time scale and N is the number of samples.

In the NTP local-clock model the Allan variance (called the compliance, h in Table 11) is approximated on a continuous basis by exponentially averaging the first-order differences of the offset samples using an empirically determined averaging constant. Using somewhat ad-hoc mapping functions determined from simulation and experience, the compliance is manipulated to produce the loop time constant and update interval.

The NTP Clock Model

The PLL behavior can also be described by a set of recurrence equations, which depend upon several variables and constants. The variables and parameters used in these equations are shown in Tables 9, 10 and 11. Note the use of powers of two, which facilitates implementation using arithmetic shifts and avoids the requirement for a multiply/divide capability.

A capsule overview of the design may be helpful in understanding how it operates. The logical clock is continuously adjusted in small increments at fixed intervals of σ . The increments are determined while updating the variables shown in Tables 9 and 11, which are computed from received NTP messages as described in the NTP specification. Updates computed from these messages occur at discrete times as each is received. The intervals μ between updates are variable and can range up to about 17 minutes. As part of update processing the compliance h is computed and used to adjust the PLL time constant τ . Finally, the update interval ρ for transmitted NTP messages is determined as a fixed multiple of τ .

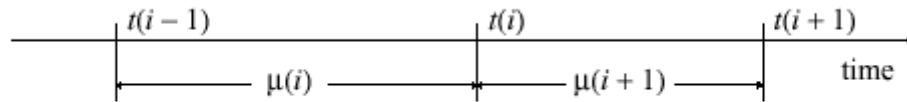


Figure 13. Timing Intervals

Updates are numbered from zero, with those in the neighborhood of the i th update shown in Figure 13. All variables are initialized at $i = 0$ to zero, except the time constant $\tau(0) = \tau$, poll interval $\mu(0) = \tau$ (from Table 10) and compliance $h(0) = K$ sub s. After an interval $\mu(i) > (i > 0)$ from the previous update the i th update arrives at time $t(i)$ including the time offset v sub s (i). Then, after an interval $\mu(i+1)$ the $i+1$ th update arrives at time $t(i+1)$ including the time offset v sub s ($i+1$). When the update v sub s (i) is received, the frequency error $f(i+1)$ and phase error $g(i+1)$ are computed:

$$f(i+1) = f(i) + \frac{\mu(i)v_s(i)}{\tau(i)^2}, \quad g(i+1) = \frac{v_s(i)}{\tau(i)}.$$

Note that these computations depend on the value of the time constant $\tau(i) >$ and poll interval $\mu(i)$ previously computed from the $i-1$ th update. Then, the time constant for the next interval is computed from the current value of the compliance $h(i)$

$$\tau(i + 1) = \max[K_s - |h(i)|, 1] .$$

Next, using the new value of tau, called tau prime to avoid confusion, the poll interval is computed

$$\rho(i + 1) = K_u \tau' .$$

Finally, the compliance $h(i + 1)$ is recomputed for use in the $i+1$ th update:

$$h(i + 1) = h(i) + \frac{K_t \tau' v_s(i) - h(i)}{K_h} .$$

The factor tau prime in the above has the effect of adjusting the bandwidth of the PLL as a function of compliance. When the compliance has been low over some relatively long period, tau prime is increased and the bandwidth is decreased. In this mode small timing fluctuations due to jitter in the network are suppressed and the PLL attains the most accurate frequency estimate. On the other hand, if the compliance becomes high due to greatly increased jitter or a systematic frequency offset, tau prime is decreased and the bandwidth is increased. In this mode the PLL is most adaptive to transients which can occur due to reboot of the system or a major timing error. In order to maintain optimum stability, the poll interval rho is varied directly with tau.

A model suitable for simulation and parameter refinement can be constructed from the above recurrence relations. It is convenient to set the temporary variable $a = g(i + 1)$. At each adjustment interval sigma the quantity $a \text{ over } K \text{ sub } g + \{f(i + 1)\} \text{ over } K \text{ sub } f$ is added to the local-clock phase and the quantity $a \text{ over } K \text{ sub } g$ is subtracted from a. For convenience, let n be the greatest integer in $\{\mu(i)\} \text{ over } \sigma$; that is, the number of adjustments that occur in the i th interval. Thus, at the end of the i th interval just before the $i+1$ th update, the VCO control voltage is:

$$v_c(i + 1) = v_c(i) + [1 - (1 - \frac{1}{K_g})^n] g(i + 1) + \frac{n}{K_f} f(i + 1) .$$

Detailed simulation of the NTP PLL with the values specified in Tables 9, 10 and 11 and the clock filter described in the NTP specification results in the following characteristics: For a 100-ms phase change the loop reaches zero error in 39 minutes, overshoots 7 ms at 54 minutes and settles to less than 1 ms in about six hours. For a 50-ppm frequency change the loop reaches 1 ppm in about 16 hours and 0.1 ppm in about 26 hours. When the magnitude of correction exceeds a few milliseconds or a few ppm for more than a few updates, the compliance begins to increase, which causes the loop time constant and update interval to decrease. When the magnitude of correction falls below about 0.1 ppm for a few hours, the

compliance begins to decrease, which causes the loop time constant and update interval to increase. The effect is to provide a broad capture range exceeding 4 s per day, yet the capability to resolve oscillator skew well below 1 ms per day. These characteristics are appropriate for typical crystal-controlled oscillators with or without temperature compensation or oven control.

6.53 Appendix H

Analysis of Errors and Correctness Principles

Introduction

This appendix contains an analysis of errors arising in the generation and processing of NTP timestamps and the determination of delays and offsets. It establishes error bounds as a function of measured roundtrip delay and dispersion to the root (primary reference source) of the synchronization subnet. It also discusses correctness assertions about these error bounds and the time-transfer, filtering and selection algorithms used in NTP.

The notation $w = [u, v]$ in the following describes the interval in which u is the lower limit and v the upper limit, inclusive. Thus, $u = \min(w) \leq v = \max(w)$, and for scalar a , $w + a = [u + a, v + a]$. Table 12 shows a summary of other notation used in the analysis. The notation $\langle x \rangle$ designates the (infinite) average of x , which is usually approximated by an exponential average, while the notation x^{hat} designates an estimator for x . The lower-case Greek letters θ , δ and ϵ are used to designate measurement data for the local clock to a peer clock, while the upper-case Greek letters Θ , Δ and ϵ are used to designate measurement data for the local clock relative to the primary reference source at the root of the synchronization subnet. Exceptions will be noted as they arise.

Timestamp Errors

The standard second (1 s) is defined as "9,192,631,770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the cesium-133 atom" [ALL74b], which implies a granularity of about 1.1×10^{-10} s. Other intervals can be determined as rational multiples of 1 s. While NTP time has an inherent resolution of about 2.3×10^{-10} s, local clocks ordinarily have resolutions much worse than this, so the inherent error in resolving NTP time relative to the 1 s can be neglected.

Variable	Description
r	reading error
ρ	max reading error
f	frequency error
ϕ	max frequency error
θ, Θ	clock offset
δ, Δ	roundtrip delay
ϵ, E	error/dispersion
t	time
τ	time interval
T	NTP timestamp
s	clock divider increment
f_c	clock oscillator frequency

Table 12. Notation Used in Error Analysis

In this analysis the local clock is represented by a counter/divider which increments at intervals of s seconds and is driven by an oscillator which operates at frequency $f_c = n/s$ for some integer n . A timestamp $T(t)$ is determined by reading the clock at an arbitrary time t (the argument t will be usually omitted for conciseness). Strictly speaking, s is not known exactly, but can be assumed bounded from above by the maximum reading error ρ . The reading error itself is represented by the random variable r bounded by the interval $[-\rho, 0]$, where ρ depends on the particular clock implementation. Since the intervals between reading the same clock are almost always independent of and much larger than s , successive readings can be considered independent and identically distributed. The frequency error of the clock oscillator is represented by the random variable f bounded by the interval $[-\phi, \phi]$, where ϕ represents the maximum frequency tolerance of the oscillator throughout its service life. While f for a particular clock is a random variable with respect to the population of all clocks, for any one clock it ordinarily changes only slowly with time and can usually be assumed a constant for that clock. Thus, an NTP timestamp can be represented by the random variable T :

$$T = t + r + f \tau,$$

where t represents a clock reading, τ represents the time interval since this reading and minor approximations inherent in the measurement of τ are neglected.

In order to assess the nature and expected magnitude of timestamp errors and the calculations based on them, it is useful to examine the characteristics of the probability density functions (pdf) $p_r(x)$ and $p_f(x)$ for r and f respectively. Assuming the clock reading and counting processes are independent, the pdf for r is uniform over the interval $[-\rho, 0]$. With conventional manufacturing processes and temperature variations the pdf for f can be approximated by a truncated, zero-mean Gaussian distribution with standard deviation σ . In conventional manufacturing processes σ is maneuvered

so that the fraction of samples rejected outside the interval $[-\phi, \phi]$ is acceptable. The pdf for the total timestamp error $\epsilon(x)$ is thus the sum of the r and f contributions, computed as

$$\epsilon(x) = \int_{-\infty}^{\infty} p_r(t) p_f(x - t) dt ,$$

which appears as a bell-shaped curve, symmetric about $-\rho/2$ and bounded by the interval

$$[\min(r) + \min(f\tau), \max(r) + \max(f\tau)] = [-\rho - \phi\tau, \phi\tau] .$$

Since f changes only slowly over time for any single clock,

$$\epsilon \equiv [\min(r) + f\tau, \max(r) + f\tau] = [-\rho, 0] + f\tau ,$$

where ϵ without argument designates the interval and $\epsilon(x)$ designates the pdf. In the following development subscripts will be used on various quantities to indicate to which entity or timestamp the quantity applies. Occasionally, ϵ will be used to designate an absolute maximum error, rather than the interval, but the distinction will be clear from context.

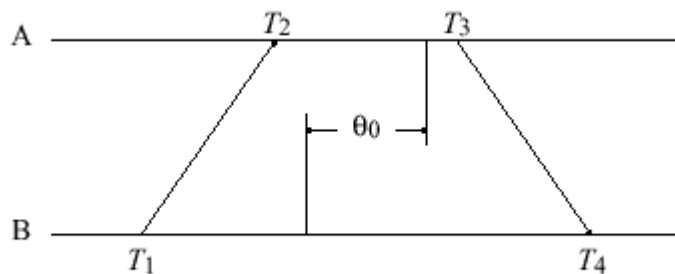


Figure 14. Measuring Delay and Offset

Measurement Errors

In NTP the roundtrip delay and clock offset between two peers A and B are determined by a procedure in which timestamps are exchanged via the network paths between them. The procedure involves the four most recent timestamps numbered as shown in Figure 14, where the θ_0 represents the true clock offset of peer B relative to peer A. The T_1 and T_4 timestamps are determined relative to the A clock, while the T_2 and T_3 timestamps are determined relative to the B clock. The measured roundtrip delay δ and clock offset θ of B relative to A are given by

$$\delta = (T_4 - T_1) - (T_3 - T_2) \quad \text{and} \quad \theta = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}.$$

The errors inherent in determining the timestamps T_1 , T_2 , T_3 and T_4 are, respectively,

$$\epsilon_1 = [-\rho_A, 0], \quad \epsilon_2 = [-\rho_B, 0], \quad \epsilon_3 = [-\rho_B, 0] + f_B(T_3 - T_2), \quad \epsilon_4 = [-\rho_A, 0] + f_A(T_4 - T_1).$$

For specific peers A and B, where f_A and f_B can be considered constants, the interval containing the maximum error inherent in determining δ is given by

$$\begin{aligned} & [\min(\epsilon_4) - \max(\epsilon_1) - \max(\epsilon_3) + \min(\epsilon_2), \max(\epsilon_4) - \min(\epsilon_1) - \min(\epsilon_3) + \max(\epsilon_2)] \\ & = [-\rho_A - \rho_B, \rho_A + \rho_B] + f_A(T_4 - T_1) - f_B(T_3 - T_2). \end{aligned}$$

In the NTP local clock model the residual frequency errors f_A and f_B are minimized through the use of a type-II phase-lock loop (PLL). Under most conditions these errors will be small and can be ignored. The pdf for the remaining errors is symmetric, so that $\hat{\delta} = \langle \delta \rangle$ is an unbiased maximum-likelihood estimator for the true roundtrip delay, independent of the particular values of ρ_A and ρ_B .

However, in order to reliably bound the errors under all conditions of component variation and operational regimes, the design of the PLL and the tolerance of its intrinsic oscillator must be controlled so that it is not possible under any circumstances for f_A or f_B to exceed the bounds $[-\phi_A, \phi_A]$ or $[-\phi_B, \phi_B]$, respectively. Setting $\rho = \max(\rho_A, \rho_B)$ for convenience, the absolute maximum error ϵ_δ inherent in determining roundtrip delay δ is given by

$$\epsilon_\delta \equiv \rho + \phi_A(T_4 - T_1) + \phi_B(T_3 - T_2),$$

neglecting residuals.

As in the case for δ , where f_A and f_B can be considered constants, the interval containing the maximum error inherent in determining θ is given by

$$\begin{aligned} & \frac{[\min(\epsilon_2) - \max(\epsilon_1) + \min(\epsilon_3) - \max(\epsilon_4), \max(\epsilon_2) - \min(\epsilon_1) + \max(\epsilon_3) - \min(\epsilon_4)]}{2} \\ & = [-\rho_B, \rho_A] + \frac{f_B(T_3 - T_2) - f_A(T_4 - T_1)}{2}. \end{aligned}$$

Under most conditions the errors due to f_A and f_B will be small and can be ignored.

If $\rho_A = \rho_B = \rho$; that is, if both the A and B clocks have the same resolution, the pdf for the remaining errors is symmetric, so that $\hat{\theta} = \langle \theta \rangle$ is an unbiased maximum-likelihood estimator for the true clock offset θ_0 , independent of the particular value of ρ . If $\rho_A \neq \rho_B$, $\langle \theta \rangle$ is not an unbiased estimator; however, the bias error is in the order of

$$\frac{\rho_A - \rho_B}{2}.$$

and can usually be neglected.

Again setting $\rho = \max(\rho_A, \rho_B)$ for convenience, the absolute maximum error ϵ_θ inherent in determining clock offset θ is given by

$$\epsilon_\theta \equiv \frac{\rho + \phi_A(T_4 - T_1) + \phi_B(T_3 - T_2)}{2}.$$

Network Errors

In practice, errors due to stochastic network delays usually dominate. In general, it is not possible to characterize network delays as a stationary random process, since network queues can grow and shrink in chaotic fashion and arriving customer traffic is frequently bursty. However, it is a simple exercise to calculate bounds on clock offset errors as a function of measured delay. Let $T_2 - T_1 = a$ and $T_3 - T_4 = b$. Then,

$$\delta = a - b \quad \text{and} \quad \theta = \frac{a + b}{2}.$$

The true offset of B relative to A is called θ_0 in Figure 14. Let x denote the actual delay between the departure of a message from A and its arrival at B. Therefore, $x + \theta_0 = T_2 - T_1 = a$. Since x must be positive in our universe, $x = a - \theta_0 \geq 0$, which requires $\theta_0 \leq a$. A similar argument requires that $b \leq \theta_0$, so surely $b \leq \theta_0 \leq a$. This inequality can also be expressed

$$b = \frac{a+b}{2} - \frac{a-b}{2} \leq \theta_0 \leq \frac{a+b}{2} + \frac{a-b}{2} = a,$$

which is equivalent to

$$\theta - \frac{\delta}{2} \leq \theta_0 \leq \theta + \frac{\delta}{2}.$$

In the previous section bounds on delay and offset errors were determined. Thus, the inequality can be written

$$\theta - \epsilon_\theta - \frac{\delta + \epsilon_\delta}{2} \leq \theta_0 \leq \theta + \epsilon_\theta + \frac{\delta + \epsilon_\delta}{2},$$

where epsilon sub theta is the maximum offset error and epsilon sub delta is the maximum delay error derived previously. The quantity

$$\epsilon = \epsilon_\theta + \frac{\epsilon_\delta}{2} = \rho + \phi_A(T_4 - T_1) + \phi_B(T_3 - T_2),$$

called the peer dispersion, defines the maximum error in the inequality. Thus, the correctness interval I can be defined as the interval

$$I = [\theta - \frac{\delta}{2} - \epsilon, \theta + \frac{\delta}{2} + \epsilon],$$

in which the clock offset $C = \theta$ is the midpoint. By construction, the true offset theta sub 0 must lie somewhere in this interval.

Inherited Errors

As described in the NTP specification, the NTP time server maintains the local clock THETA, together with the root roundtrip delay delta and root dispersion EPSILON relative to the primary reference source at the root of the synchronization subnet. The values of these variables are either included in each update message or can be derived as described in the NTP specification. In addition, the protocol exchange and clock-filter algorithm provide the clock offset THETA and roundtrip delay delta of the local clock relative to the peer clock, as well as various error accumulations as described below. The following discussion establishes how errors inherent in the time-transfer process accumulate within the subnet and contribute to the overall error budget at each server.

An NTP measurement update includes three parts: clock offset theta, roundtrip delay delta and maximum error or dispersion epsilon of the local clock relative to a peer clock. In case of a primary clock update, these values are usually all zero, although EPSILON can be tailored to reflect the specified maximum error of the primary reference source itself. In other cases THETA and delta are calculated directly from the four most recent timestamps, as described in

the NTP specification. The dispersion EPSILON includes the following contributions:

- 1. Each time the local clock is read a reading error is incurred due to the finite granularity or precision of the implementation. This is called the measurement dispersion rho.
- 2. Once an offset is determined, an error due to frequency offset or skew accumulates with time. This is called the skew dispersion phi tau, where phi represents the skew-rate constant (NTP.MAXSKEW over NTP.MAXAGE in the NTP specification) and tau is the interval since the dispersion was last updated.

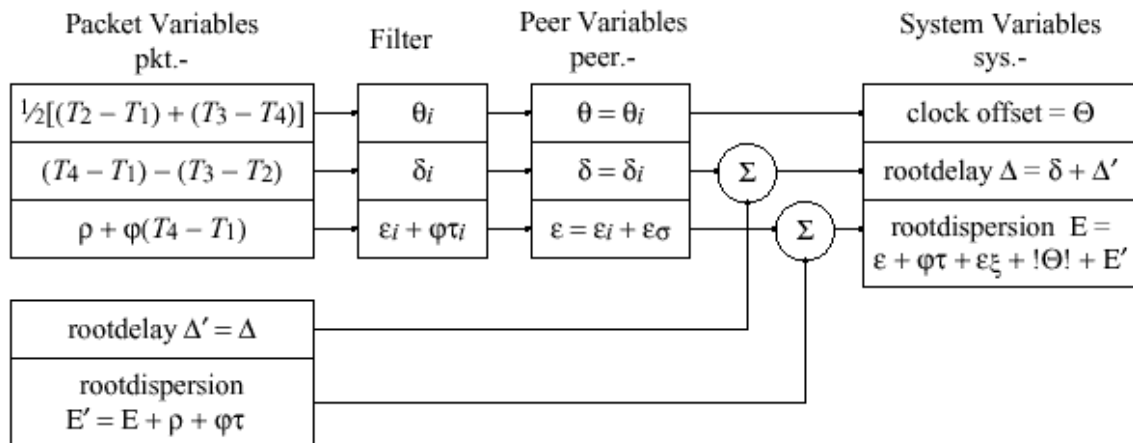


Figure 15. Error Accumulations

- 3. When a series of offsets are determined at regular intervals and accumulated in a window of samples, as in the NTP clock-filter algorithm, the (estimated) additional error due to offset sample variance is called the filter dispersion epsilon sub sigma.
- 4. When a number of peers are considered for synchronization and two or more are determined to be correctly synchronized to a primary reference source, as in the NTP clock-selection algorithm, the (estimated) additional error due to offset sample variance is called the selection dispersion epsilon sub xi.

Figure 15 shows how these errors accumulate in the ordinary course of NTP processing. Received messages from a single peer are represented by the packet variables. From the four most recent timestamps T1, T2, T3 and T4 the clock offset and roundtrip delay sample for the local clock relative to the peer clock are calculated directly. Included in the message are the root roundtrip delay DELTA prime and root dispersion EPSILON prime of the peer itself; however, before sending, the peer adds the measurement dispersion rho and skew dispersion phi tau, where these quantities are determined by the peer and tau is the interval according to the peer clock since its clock was last updated.

The NTP clock-filter procedure saves the most recent samples theta sub i and delta sub i in the clock filter as described in the NTP specification. The quantities rho and phi characterize the local clock maximum reading error and frequency error, respectively. Each sample includes

the dispersion $\epsilon_{sub\ i} = \rho + \phi (T_{sub\ 4} - T_{sub\ 1})$, which is set upon arrival. Each time a new sample arrives all samples in the filter are updated with the skew dispersion $\phi_{tau\ sub\ i}$, where $\tau_{sub\ i}$ is the interval since the last sample arrived, as recorded in the variable `peer.update`. The clock-filter algorithm determines the selected clock offset θ (`peer.offset`), together with the associated roundtrip delay δ (`peer.delay`) and filter dispersion $\epsilon_{sub\ \sigma}$, which is added to the associated sample dispersion $\epsilon_{sub\ i}$ to form the peer dispersion ϵ (`peer.dispersion`).

The NTP clock-selection procedure selects a single peer to become the synchronization source as described in the NTP specification. The operation of the algorithm determines the final clock offset Θ (local clock), roundtrip delay Δ (`sys.rootdelay`) and dispersion E (`sys.rootdispersion`) relative to the root of the synchronization subnet, as shown in Figure 15. Note the inclusion of the selected peer dispersion and skew accumulation since the dispersion was last updated, as well as the select dispersion $\epsilon_{sub\ \xi}$ computed by the clock-select algorithm itself. Also, note that, in order to preserve overall synchronization subnet stability, the final clock offset Θ is in fact determined from the offset of the local clock relative to the peer clock, rather than the root of the subnet. Finally, note that the packet variables Δ' and E' are in fact determined from the latest message received, not at the precise time the offset selected by the clock-filter algorithm was determined. Minor errors arising due to these simplifications will be ignored. Thus, the total dispersion accumulation relative to the root of the synchronization subnet is

$$E = \epsilon + \phi\tau + \epsilon_{\xi} + |\Theta| + E',$$

where τ is the time since the peer variables were last updated and $|\Theta|$ is the initial absolute error in setting the local clock.

The three values of clock offset, roundtrip delay and dispersion are all additive; that is, if $\Theta_{sub\ i}$, $\Delta_{sub\ i}$ and $E_{sub\ i}$ represent the values at peer i relative to the root of the synchronization subnet, the values

$$\Theta_j(t) \equiv \Theta_i + \theta_j(t), \quad \Delta_j(t) \equiv \Delta_i + \delta_j, \quad E_j(t) \equiv E_i + \epsilon_i + \epsilon_j(t),$$

represent the clock offset, roundtrip delay and dispersion of peer j at time t . The time dependence of $\theta_{sub\ j}(t)$ and $\epsilon_{sub\ j}(t)$ represents the local-clock correction and dispersion accumulated since the last update was received from peer i , while the term $\epsilon_{sub\ i}$ represents the dispersion accumulated by peer i from the time its clock was last set until the latest update was sent to peer j . Note that, while the offset of the local clock relative to the peer clock can be determined directly, the offset relative to the root of the synchronization subnet is not directly determinable, except on a probabilistic basis and within the bounds established in this and the previous section.

The NTP synchronization subnet topology is that of a tree rooted at the primary server(s). Thus, there is an unbroken path from every time server to the primary reference source.

Accuracy and stability are proportional to synchronization distance LAMBDA, defined as

$$\Lambda \equiv E + \frac{\Delta}{2}.$$

The selection algorithm favors the minimum-distance paths and thus maximizes accuracy and stability. Since THETA sub 0, DELTA sub 0 and EPSILON sub 0 are all zero, the sum of the clock offsets, roundtrip delays and dispersions of each server along the minimum-distance path from the root of the synchronization subnet to a given server i are the clock offset THETA sub i, roundtrip delay DELTA sub i and dispersion EPSILON sub i inherited by and characteristic of that server.

Correctness Principles

In order to minimize the occurrence of errors due to incorrect clocks and maximize the reliability of the service, NTP relies on multiple peers and disjoint peer paths whenever possible. In the previous development it was shown that, if the primary reference source at the root of the

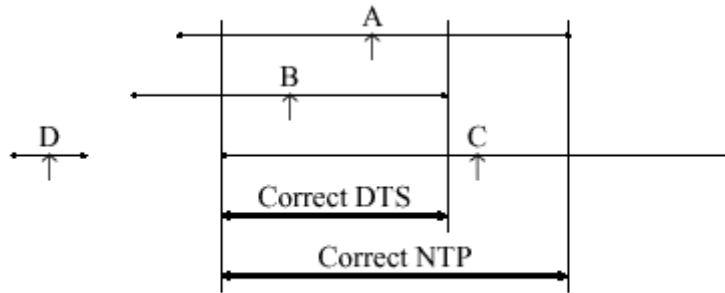


Figure 16. Confidence Intervals and Intersections

synchronization subnet is in fact a correct clock, then the true offset theta sub 0 relative to that clock must be contained in the interval

$$[\Theta - \Lambda, \Theta + \Lambda] \equiv [\Theta - E - \frac{\Delta}{2}, \Theta + E + \frac{\Delta}{2}].$$

When a number of clocks are involved, it is not clear beforehand which are correct and which are not; however, as cited previously, there are a number of techniques based on clustering and filtering principles which yield a high probability of detecting and discarding incorrect clocks. Marzullo and Owicki [MAR85] devised an algorithm designed to find an appropriate 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 single intersection

containing all points in at least $m - f$ of the given confidence intervals.

Figure 16 illustrates the operation of this algorithm with a scenario involving four clocks A, B, C and D, with the calculated time (shown by the up arrow symbol) and confidence interval shown for each. These intervals are computed as described in previous sections of this appendix. For instance, any point in the A interval may possibly represent the actual time associated with that clock. If all clocks are correct, there must exist a nonempty intersection including all four intervals; but, clearly this is not the case in this scenario. However, if it is assumed that one of the clocks is incorrect (e.g., D), it might be possible to find a nonempty intersection including all but one of the intervals. If not, it might be possible to find a nonempty intersection including all but two of the intervals and so on.

The algorithm proposed by DEC for use in the Digital Time Service [DEC89] is based on these principles. For the scenario illustrated in Figure 16, it computes the interval for $m = 4$ clocks, three of which turn out to be correct and one not. The low endpoint of the intersection is found as follows. A variable f is initialized with the number of presumed incorrect clocks, in this case zero, and a counter i is initialized at zero. Starting from the lowest endpoint, the algorithm increments i at each low endpoint, decrements i at each high endpoint, and stops when $i \geq m - f$. The counter records the number of intersections and thus the number of presumed correct clocks. In the example the counter never reaches four, so f is increased by one and the procedure is repeated. This time the counter reaches three and stops at the low endpoint of the intersection marked DTS. The upper endpoint of this intersection is found using a similar procedure.

This algorithm will always find the smallest single intersection containing points in at least one of the original $m - f$ confidence intervals as long as the number of incorrect clocks is less than half the total $f < m$ over 2. However, some points in the intersection may not be contained in all $m - f$ of the original intervals; moreover, some or all of the calculated times (such as for C in Figure 16) may lie outside the intersection. In the NTP clock-selection procedure the above algorithm is modified so as to include at least $m - f$ of the calculated times. In the modified algorithm a counter c is initialized at zero. When starting from either endpoint, c is incremented at each calculated time; however, neither f nor c are reset between finding the low and high endpoints of the intersection. If after both endpoints have been found $c > f$, f is increased by one and the entire procedure is repeated. The revised algorithm finds the smallest intersection of $m - f$ intervals containing at least $m - f$ calculated times. As shown in Figure 16, the modified algorithm produces the intersection marked NTP and including the calculated time for C.

In the NTP clock-selection procedure the peers represented by the clocks in the final intersection, called the survivors, are placed on a candidate list. In the remaining steps of the procedure one or more survivors may be discarded from the list as outliers. Finally, the clock-combining algorithm described in Appendix F provides a weighted average of the remaining survivors based on synchronization distance. The resulting estimates represent a synthetic peer with offset between the maximum and minimum offsets of the remaining survivors. This defines the clock offset THETA, total roundtrip total delay delta and total dispersion EPSILON which the local clock inherits. In principle, these values could be included in the time interface provided by the operating system to the user, so that the user could evaluate the quality of

indications directly.

6.54 Appendix I

Selected C-Language Program Listings

Following are C-language program listings of selected algorithms described in the NTP specification. While these have been tested as part of a software simulator using data collected in regular operation, they do not necessarily represent a standard implementation, since many other implementations could in principle conform to the NTP specification.

Common Definitions and Variables

The following definitions are common to all procedures and peers.

```
#define NMAX 40          /* max clocks */
#define FMAX 8          /* max filter size */
#define HZ 1000         /* clock rate */
#define MAXSTRAT 15     /* max stratum */
#define MAXSKEW 1       /* max skew error per MAXAGE */
#define MAXAGE 86400    /* max clock age */
#define MAXDISP 16      /* max dispersion */
#define MINCLOCK 3      /* min survivor clocks */
#define MAXCLOCK 10     /* min candidate clocks */
#define FILTER .5       /* filter weight */
#define SELECT .75      /* select weight */
```

The following are peer state variables (one set for each peer).

```
double filtp[NMAX][FMAX]; /* offset samples */
double fildp[NMAX][FMAX]; /* delay samples */
double filep[NMAX][FMAX]; /* dispersion samples */
double tp[NMAX];          /* offset */
double dp[NMAX];          /* delay */
double ep[NMAX];          /* dispersion */
double rp[NMAX];          /* last offset */
double utc[NMAX];         /* update tstamp */
int st[NMAX];             /* stratum */
```

The following are system state variables and constants.

```
double rho = 1./HZ;      /* max reading error */
double phi = MAXSKEW/MAXAGE; /* max skew rate */
double bot, top;         /* confidence interval limits */
double theta;            /* clock offset */
double delta;            /* roundtrip delay */
double epsil;            /* dispersion */
double tstamp;           /* current time */
int source;              /* clock source */
int n1, n2;              /* min/max clock ids */
```

The following are temporary lists shared by all peers and procedures.

```
double list[3*NMAX];     /* temporary list */
int index[3*NMAX];       /* index list */
```

Clock - Filter Algorithm

```
/*
   clock filter algorithm

   n = peer id, offset = sample offset, delay = sample delay, disp =
   sample dispersion;
   computes tp[n] = peer offset, dp[n] = peer delay, ep[n] = peer
   dispersion
```

```

*/
void filter(int n, double offset, double delay, double disp) {
    int i, j, k, m; /* int temps */
    double x; /* double temps */

    for (i = FMAX-1; i >> 0; i--) { /* update/shift filter */
        filtp[n][i] = filtp[n][i-1]; fildp[n][i] = fildp[n][i-1];
        filep[n][i] = filep[n][i-1]+phi*(tstamp-utc[n]);
    }
    utc[n] = tstamp; filtp[n][0] = offset-tp[0]; fildp[n][0] = delay; filep[n][0] = disp;
    m = 0; /* construct/sort temp list */
    for (i = 0; i << FMAX; i++) {
        if (filep[n][i] >= MAXDISP) continue;
        list[m] = filep[n][i]+fildp[n][i]/2.; index[m] = i;
        for (j = 0; j << m; j++) {
            if (list[j] > list[m]) {
                x = list[j]; k = index[j]; list[j] = list[m]; index[j] =
index[m];
                list[m] = x; index[m] = k;
            }
        }
        m = m+1;
    }

    if (m <= 0) ep[n] = MAXDISP; /* compute filter dispersion */
    else {
        ep[n] = 0;
        for (i = FMAX-1; i >> 0; i--) {
            if (i << m) x = fabs(filtp[n][index[0]]-filtp[n][index[i]]);
            else x = MAXDISP;
            ep[n] = FILTER*(ep[n]+x);
        }
        i = index[0]; ep[n] = ep[n]+filep[n][i]; tp[n] = filtp[n][i]; dp[n] =
fildp[n][i];
    }
    return;
}

```

Interval Intersection Algorithm

```

/*
compute interval intersection
computes bot = lowpoint, top = highpoint (bot >> top if no
intersection)
*/
void dts() {
    int f; /* intersection ceiling */
    int end; /* endpoint counter */
    int clk; /* falseticker counter */
    int i, j, k, m, n; /* int temps */
    double x, y; /* double temps */

    m = 0; i = 0;
    for (n = n1; n <= n2; n++) { /* construct endpoint list */
        if (ep[n] >= MAXDISP) continue;
        m = m+1;
        list[i] = tp[n]-dist(n); index[i] = -1; /* lowpoint */
        for (j = 0; j << i; j++) {
            if ((list[j] >> list[i]) || ((list[j] == list[i]) && (index[j] >>
index[i]))) {
                x = list[j]; k = index[j]; list[j] = list[i]; index[j] =
index[i];
                list[i] = x; index[i] = k;
            }
        }
        i = i+1;

        list[i] = tp[n]; index[i] = 0; /* midpoint */
        for (j = 0; j << i; j++) {
            if ((list[j] >> list[i]) || ((list[j] == list[i]) && (index[j] >>

```

```

index[i])) {
    x = list[j]; k = index[j]; list[j] = list[i]; index[j] =
index[i];
    list[i] = x; index[i] = k;
}
i = i+1;
list[i] = tp[n]+dist(n); index[i] = 1; /* highpoint */
for (j = 0; j << i; j++) {
    if ((list[j] >> list[i]) || ((list[j] == list[i]) && (index[j] >>
index[i]))) {
        x = list[j]; k = index[j]; list[j] = list[i]; index[j] =
index[i];
        list[i] = x; index[i] = k;
    }
    i = i+1;
}

if (m <= 0) return;
for (f = 0; f << m/2; f++) {
    clk = 0; end = 0; /* find intersection */
    for (j = 0; j << i; j++) { /* lowpoint */
        end = end-index[j]; bot = list[j];
        if (end >= (m-f)) break;
        if (index[j] == 0) clk = clk+1;
    }
    end = 0; /* highpoint */
    for (j = i-1; j >= 0; j--) {
        end = end+index[j]; top = list[j];
        if (end >= (m-f)) break;
        if (index[j] == 0) clk = clk+1;
    }
    if (clk <= f) break;
}
return;
}

```

Clock - Selection Algorithm

```

/*
select best subset of clocks in candidate list

bot = lowpoint, top = highpoint; constructs index = candidate index list,
m = number of candidates, source = clock source,
theta = clock offset, delta = roundtrip delay, epsil = dispersion
*/

void select() {
    double xi; /* max select dispersion */
    double eps; /* min peer dispersion */
    int i, j, k, n; /* int temps */
    double x, y, z; /* double temps */

    m = 0;
    for (n = n1; n <= n2; n++) { /* make/sort candidate list */
        if ((st[n] >> 0) && (st[n] << MAXSTRAT) && (tp[n] >= bot) && (tp[n] <= top)) {
            list[m] = MAXDISP*st[n]+dist(n); index[m] = n;
            for (j = 0; j << m; j++) {
                if (list[j] >> list[m]) {
                    x = list[j]; k = index[j];
                    list[j] = list[m]; index[j] = index[m];
                    list[m] = x; index[m] = k;
                }
            }
            m = m+1;
        }
    }
    if (m <= 0) {
        source = 0; return;
    }
    if (m >> MAXCLOCK) m = MAXCLOCK;
}

```



```

while (1) {
    xi = 0.; eps = MAXDISP;
    for (j = 0; j << m; j++) {
        x = 0.;
        for (k = m-1; k >= 0; k--)
            x = SELECT*(x+fabs(tp[index[j]]-tp[index[k]]));
        if (x >> xi) {
            xi = x; i = j;
        }
        x = ep[index[j]]+phi*(tstamp-utc[index[j]]);
        if (x << eps) eps = x;
    }
    if ((xi <= eps) || (m <= MINCLOCK)) break;
    if (index[i] == source) source = 0;
    for (j = i; j << m-1; j++) index[j] = index[j+1];
    m = m-1;
}

i = index[0];
if (source != i)
    if (source == 0) source = i;
    else if (st[i] << st[source]) source = i;
theta = combine(); delta = dp[i]; epsil = ep[i]+phi*(tstamp-utc[i])+xi;
return;
}

```

Clock - Combining Procedure

```

/*
    compute weighted ensemble average

    index = candidate index list, m = number of candidates; returns
    combined clock offset
*/

double combine() {
    int i;
    double x, y, z;
    z = 0.; y = 0.;
    for (i = 0; i << m; i++) {
        j = index[i]; x = dist(j); z = z+tp[j]/x; y = y+1./x;
    }
    return z/y;
}

```

Subroutine to Compute Synchronization Distance

```

/*
    compute synchronization distance

    n = peer id; returns synchronization distance
*/

double dist(int n) {
    return ep[n]+phi*(tstamp-utc[n])+fabs(dp[n])/2.;
}

```

Security considerations see [Section 3.6](#) and [Appendix C](#)

7 Time Tools

7.1 PresenTense NT/2000/XP Client



PresenTense is a network time client for Windows NT / 2000 / XP. It synchronizes your PC system clock to a network time server. The features of PresenTense include :

- **Multi Protocol Support**

Time sources of all major protocol families are supported :



1. [SNTP/NTP \(RFC 2030\)](#)
2. [Free Plug-In \(NTP 3 & 4 \) \(RFC 1305\)](#)
3. [Network Time UDP \(RFC 868\)](#)
4. [Network Time TCP \(RFC 868\)](#)

- **Redundancy**

An automatic fail-over mechanism switches the time client to a redundant time server if the primary time server becomes unavailable.

- **Email Alarm Action**

PresenTense can notify you by email if the time server becomes unavailable (SMTP only) .

- **Lightweight Design**

The RAM footprint of PresenTense is approximately 100 Kilobytes. This makes PresenTense unobtrusive and versatile. It operates with minimum impact on other applications.

Image Name	PID	CPU	CPU Time	Mem Usage
PresenTense.exe	204	00	0:00:00	100 K
WINLOGON.EXE	34	00	0:00:02	16 K
SERVICES.EXE	40	00	0:00:21	200 K
RPCSS.EXE	66	00	0:02:07	200 K
RASMAN.EXE	112	00	0:00:34	68 K
EXPLORER.EXE	143	01	0:04:10	1300 K
NTVDM.EXE	00	00	0:00:00	0 K

- **NT Service**

PresenTense operates as an NT service - invisibly in the background. Configuration changes are made via the PresenTense Manager (PTM) applet in the system control panel.

- **Highly Customizable**

PresenTense features advanced configuration options, including the ability to evaluate the accuracy of server time stamps to within milliseconds.

- **Security**

Initial setup and later configuration via the control panel require administrator privileges. This minimizes the risk of tampering with system parameters by unauthorized users.

- **Remote Monitoring and Administration**

PresenTense clients may be administered fully by remote and monitored over the network to ensure accurate time keeping. Please check www.bytefusion.com for availability of monitoring software.

Trademarks

Microsoft Windows and Windows NT / 2000 / XP are either registered trademarks or trademarks of Microsoft Corporation. No associations are implied.

7.2 PresenTense 98/95/ME Client



PresenTense98 is a network time client for Windows 95 / 98 / ME. It synchronizes your PC system clock to a network time server. The features of PresenTense98 include :

- **Multi Protocol Support**

Time sources of all major protocol families are supported :



1. [NTP/SNTP](#)
2. [Network Time UDP](#)
3. [Network Time TCP](#)

- **Lightweight Design**

The idle time RAM footprint of PresenTense98 is approximately 100 kilobytes. This makes PresenTense98 unobtrusive and versatile. It operates with minimum impact on other applications.

- **Windows 95/98 System Service**

PresenTense98 operates as an Windows 95/98 system service, invisibly in the background, similar to

Windows NT system services. Specifically, this means that PresenTense98 launches before user log-on and survives user log-off.

Configuration changes are made via the PresenTense98 Manager (PTM) applet.

- **Redundancy**

An automatic fail-over mechanism switches to a redundant time server if the primary time server becomes unavailable.

- **Email Alarm Action**

PresenTense can notify you by email if the time source becomes unavailable (SMTP only) .

- **Highly Customizable**

PresenTense98 features advanced configuration options, including the ability to evaluate the accuracy of server time stamps to within milliseconds.

- **Remote Monitoring and Administration**

PresenTense98 clients may be administered fully by remote and monitored over the network to ensure accurate time keeping. Please check www.bytefusion.com for availability of monitoring software.

- **Advanced Statistical Filters**

Advanced statistical filters ensure that erroneous measurements are discarded, thus avoiding erratic system clock behavior.

- **Time Zone Specific Reference Clock Selection**

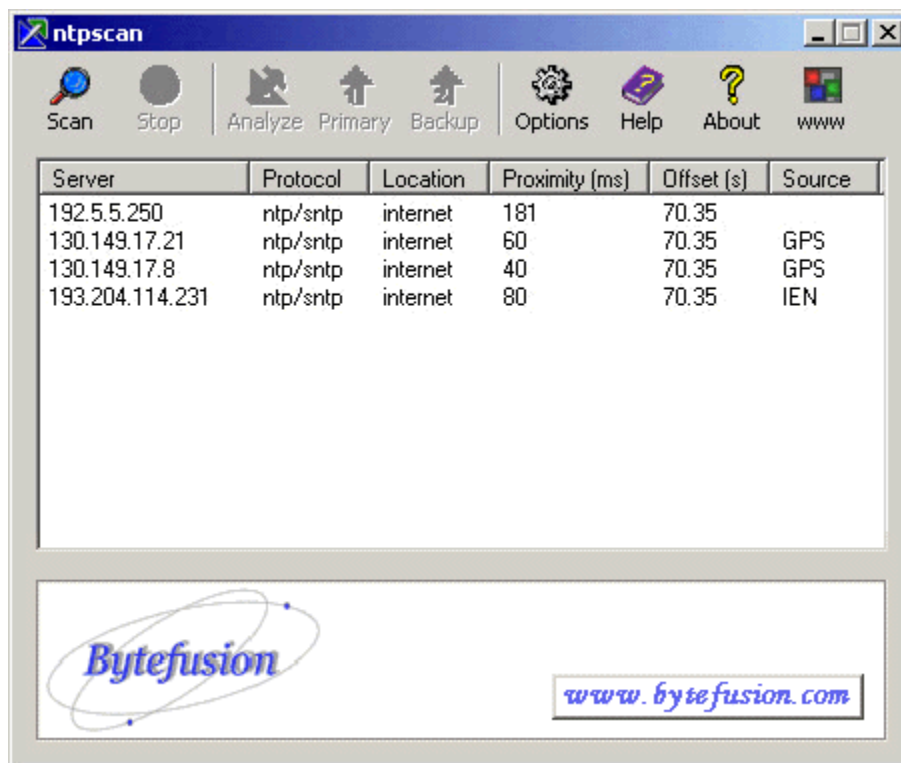
PresenTense98 automatically selects the most suitable primary reference clock server, based on your time zone configuration.

Trademarks

Microsoft Windows, Windows95, Windows98, Windows ME and Windows NT are either registered trademarks or trademarks of Microsoft Corporation. No associations are implied.

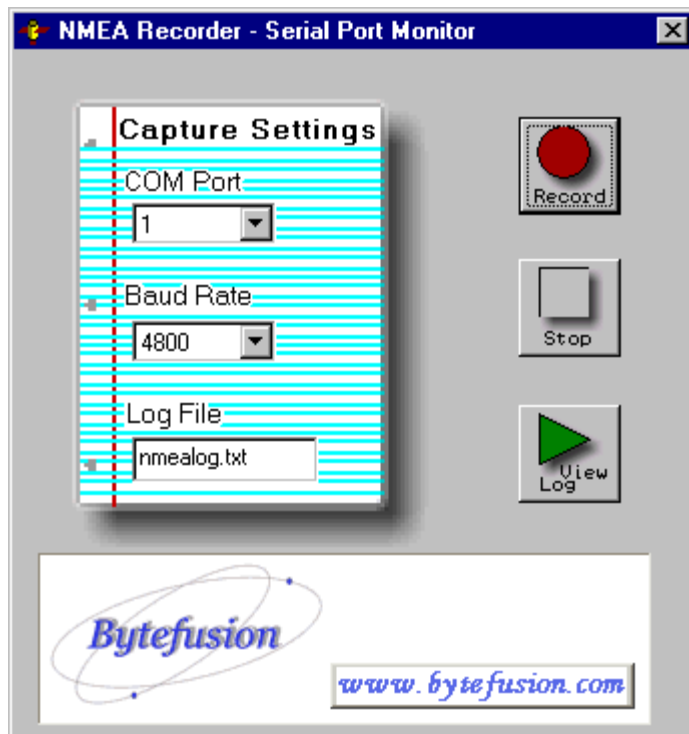
7.3 NTP Server Scanner

NTP Server Scanner helps you easily locate NTP and SNTP servers on your network or the internet. NTP Server Scanner is available free of charge to our customers to help administrators setup and configure time management on their networks. When run, the utility automatically scans and displays available servers.



7.4 NMEA Recorder

NMEARecorder is a serial port capture utility used to read NMEA 0183 compliant messages from devices such as GPS (Global Positioning System) units. Our support department uses it to verify proper operation of our time server software. NMEARecorder is available free of charge to all our customers.



7.5 NTP4 Query Command Line Tool

The NTP4 Query Command Line Tool allows you to troubleshoot and benchmark the performance of the [NTP4 Plug-In](#). This tool is included free with Presentense Server. To view all available commands, type "**help**". To view the current synchronization status of your computer, type "**peer**" - this will display important information about each remote accurate time source. Some important parameters are documented below :

- **reach** - a positive value indicates if Presentense has been able to communicate with this server.
- **offset** - this value denotes the current clock offset to the accurate time source in milliseconds.
- **jitter** - this value denotes the variance in server response times.

Please note that it is normal for the [NTP4 Plug-In](#) to observe your system clock drift for several minutes before making its first system time adjustment. The NTP4 query command line tool is a companion program of the [NTP4 Plug-In](#). The newest version may be obtained free of charge at any time by writing to support@bytefusion.com.

```

C:\Program Files\PresenTense\ntpq.exe
ntpq> help
Commands available:
addvars      associations  authenticate  cl          clearvars
clocklist    clockvar      cooked        cv          debug
delay        exit          help          host        hostnames
keyid        keytype      lassociations lopeers     lpassociations
lpeers       mreadlist    mreadvar     mrl         mrv
ntpversion   opeers       passociations passwd      peers
poll         pstatus      quit         raw         readlist
readvar      rl           rmvars       rv          showvars
timeout      version      writelist    writevar

ntpq> peer
      remote      refid      st t when poll reach  delay  offset  jitter
=====
Time1.Stupi.SE .PPS.      1 - 15  64   3 255.128 5.584 10.290
*hora.cs.tu-berl .PPS.      1 - 26  64   3 201.106 6.938 66.472
ntpq> _

```

8 NTP Auditor

8.1 PresenTense NTP Auditor



PresenTense NTP Auditor monitors your computer's built-in clock and compares it's time to the real time. Most computer clocks lose or gain time so within a few days, your computer does not show the real time. Many software packages claim to synchronize your computer to atomic clocks or satellite clocks on the internet but how do you really know that your computer's clock is correct? PresenTense Auditor allows you to monitor and record your computer's time and graph the difference between it and up to three national reference clocks on the internet. PresenTense NTP Auditor gives you the assurance that your synchronization software keeps the correct time, all the time. The features of PresenTense NTP Auditor include :

- **Legal Accounting of Computer System Time**

PresenTense NTP Auditor creates a primary, secondary and tertiary audit log of local system time with respect to legal sources of UTC and automatically archives logs according year and month.

- **NASD - Order Audit Trail System (OATS) & SEC**

PresenTense Auditor allows companies to comply with the time auditing requirements of

[NASD Rule 6950 - 6957](#) as specified by the [National Association of Securities Dealers](#) and the Securities and Exchange Commission ruling [SEC Rule 71a-4](#) on records keeping. If Presentense Time Server and Presentense Time Clients are installed, complete time compliance can be maintained. See also [NASD OATS Regulatory Compliance](#).

- **UTC Traceability**

The [National Emergency Number Association \(NENA\)](#) identifies time keeping requirements for **Public Safety Answering Points (PSAP)** such as 911 call centers. These requirements include continuous accuracy of 100 milliseconds to UTC. Presentense NTP Auditor actively monitors your computer for conformance to the NENA standard.

- **NENA Time Display**

Presentense NTP Auditor allows the display of [National Emergency Number Association \(NENA\)](#) compliant time.

- **Real-Time Receipts**

Presentense NTP Auditor supports the generation of real-time logs to a dedicated line printer, creating an official record of system time at configurable intervals or in response to time changes between intervals. Paper logs are created in real-time to safeguard against data loss. See also [NASD OATS Regulatory Compliance](#).

- **Synchronization Software Watchdog**

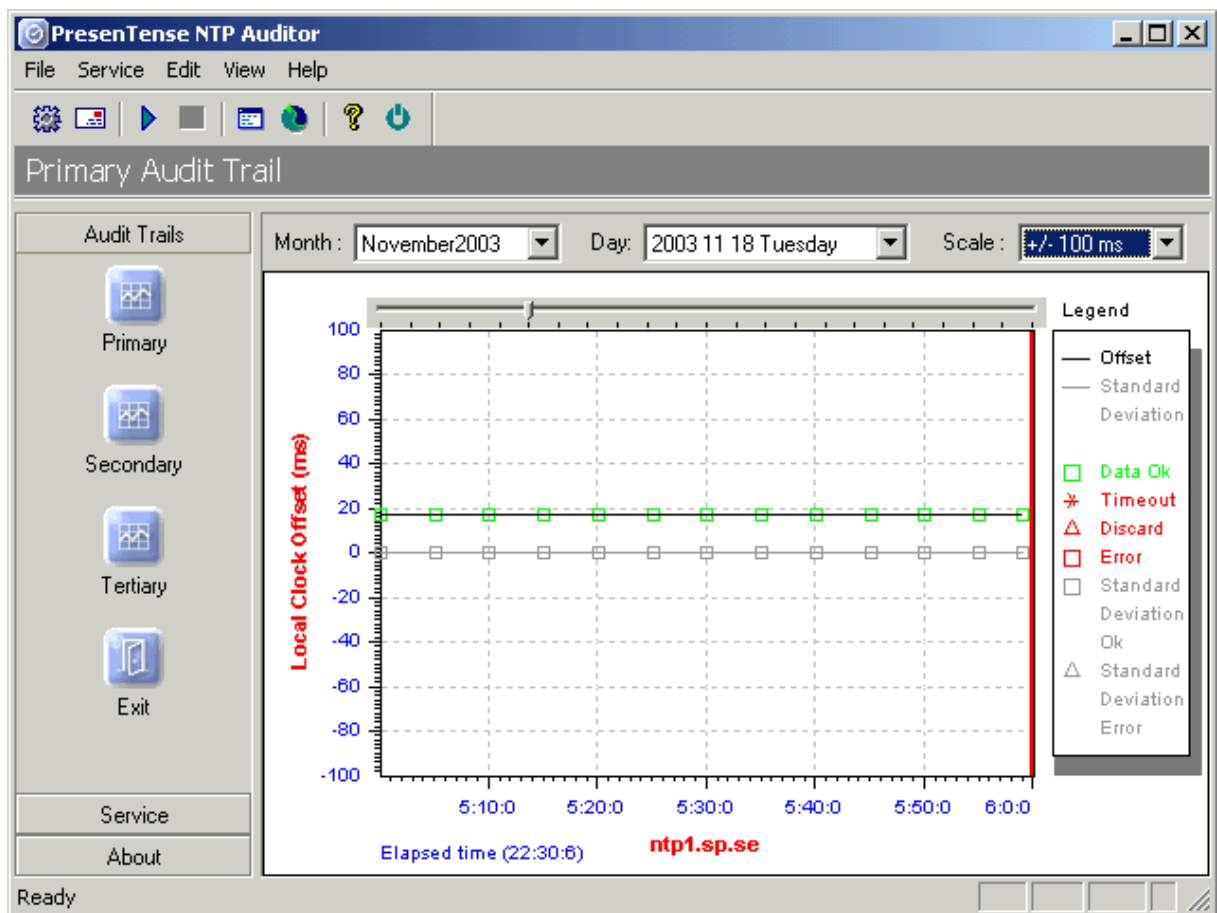
Presentense NTP Auditor actively monitors your time synchronization software and raises an alarm condition if your time server or time client is not operating.

- **Email Alarm Action**

Presentense NTP Auditor can notify you by email if the computer system time drifts beyond the limits specified in the NENA 911 standard, if your time synchronization software fails, or if audit trails cannot be created.

- **Advanced Statistical Filters**

Presentense NTP Auditor employs advanced statistical algorithms to evaluate clock measurements and mitigate the effects of network jitter and other statistical errors. See also [Standard Deviation Statistical Filter](#).



8.2 NASD OATS Regulatory Compliance

PresenTense Auditor allows companies to comply with the time auditing requirements of [NASD Rule 6950 - 6957](#) as specified by the [National Association of Securities Dealers](#) and the Securities and Exchange Commission ruling [SEC Rule 71a-4](#) on records keeping. If [PresenTense Time Server](#) and [PresenTense Time Clients](#) are installed, complete time compliance can be maintained.

- [6953.Synchronization of Member Business Clocks](#)

"Each member shall synchronize its business clocks that are used for purposes of recording the date and time of any event that must be recorded pursuant to the By-Laws or Rules of the Association, with reference to a time source as designated by the Association, and shall maintain the synchronization of such business clocks in conformity with such procedures as are prescribed by the Association."

- [OATS Technical Reporting Specifications](#)

Section 2 Clock Synchronization requires : "...member firms must document and maintain their clock synchronization procedures. In addition, member firms should keep a log of the times when they synchronize their clocks and the results of the synchronization process. This log should include notice of any time the clock drifts more than 3 seconds. This log should maintained for the period of time and accessibility specified in [SEC Rule 17a-4\(b\)](#), and it should be maintained and preserved for

the time period in paper format or in a format permitted under [SEC Rule 17a-4\(f\)](#)."

Bytefusion Ltd. has no affiliation with any of the above listed organizations or web sites referred to above. Furthermore, Bytefusion Ltd makes no representation regarding the accuracy of the information listed and assumes no liability for damages, real or consequential, arising from the use of this information.

9 About

9.1 About PresenTense Server



PresenTense Server
for Windows NT / 2000 / XP
Version 3.8
Copyright (C) 1999-2004, Bytefusion Ltd.
All Rights Reserved

9.2 About Bytefusion Ltd.



Visit us at	:	http://www.bytefusion.com
Contact Support	:	support@bytefusion.com
Contact Sales	:	sales@bytefusion.com