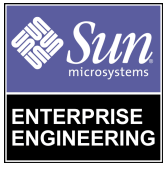

# Using NTP to Control and Synchronize System Clocks - Part I: Introduction to NTP

---

*By David Deeths - Enterprise Engineering and Glenn Brunette - Sun Professional Services*

*Sun BluePrints™ OnLine - July 2001*




**Sun Microsystems, Inc.**
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300 fax 650 969-9131

Part No.: 816-1475-10
Revision 01, 07/09/01
Edition: July 2001

Please
Recycle

# Using NTP to Control and Synchronize System Clocks - Part I: Introduction to NTP

This is Part 1 of a three-article series that discusses how to use Network Time Protocol (NTP) to synchronize system clocks. This article sets the stage by outlining why time synchronization is important and explaining the various components of the NTP distribution. This article introduces the basic concepts of NTP, and also sets expectations for the accuracy, security, and resource needs of an NTP installation. This article will be most useful for system managers and anyone who is either unfamiliar with NTP or unconvinced of the value of time synchronization, but it does cover some aspects of NTP that may be unfamilar even to experienced system administrators. The later articles in the series will provide technical information for implementing, architecting, managing, and troubleshooting NTP installations. Throughout the series, emphasis is placed on architecting a well-planned, secure, efficient, and accurate NTP solution.

Part 2 of the series will cover NTP client and server administration, access control, authentication, and architectural principles for a successful NTP infrastructure.

## Introduction

The need for synchronized time is critical for today's network environments. As organizations grow and the network services they provide continue to increase, the challenges involved with providing accurate time to their systems and applications also increase. Every aspect of managing, securing, planning, and debugging a network involves determining when events happen. Time is the critical element that allows an event on one network node to be mapped to a corresponding event on another. In many cases, these challenges can be overcome by the enterprise deployment of the NTP service.

# Time in a Networked Environment

Electronic clocks in most servers and networking devices keep inaccurate time. One of several reasons for this is that designing a computer to keep accurate time is rarely a priority for computer manufacturers because it adds cost and complexity. However, even fairly accurate computer clocks are likely to vary due to manufacturing defects, changes in temperature, electric and magnetic interference, the age of the oscillator, or even computer load. Additionally, even the smallest errors in keeping time can significantly add up over a long period. Consider two clocks that are synchronized at the beginning of the year, but one consistently takes an extra .04 milliseconds to increment itself by a second. By the end of a year, the two clocks will differ in time by more than 20 minutes. If a clock is off by just 10 parts per million, it will gain or lose almost a second a day. These measures are actually fairly optimistic examples of the accuracy of some of the clocks in modern workstations and PCs. The types of inaccuracies that exist in computer clocks are difficult to classify. Some clock variations are random, caused by environmental or electronic variations, others are systematic, caused by a miscalibrated clock.

Clearly, having any sort of meaningful time synchronization is almost impossible if clocks are allowed to run on their own. In some environments, this lack of synchronization isn't a big issue. However, in most modern networked computing environments, time synchronization is important. To reduce confusion in shared filesystems, it is crucial for the modification times to be consistent, regardless of what machine the filesystems are on. Billing services and similar applications must know the time accurately. Some financial services even require highly accurate time-keeping by law. Sorting email and other network communications can also be difficult if time stamps are incorrect. In addition, tracking security breaches, network usage, or problems affecting a large number of components can be nearly impossible if time stamps in logs are inaccurate. Time is often the critical factor in separating cause from effect.

Applications such as cyptographic key management and secure document transmission may require using accurate, encoded time stamps which match unencoded time stamps to help assure document authenticity. For instance, secure RPC needs clocks to be synced to within 15 seconds for proper operation. In addition, interactions with dynamic events such as stock market trades, aviation management, and radio and TV programming, require careful synchronization of time.

As with any complex problem that is important in a wide variety of circumstances, a number of potential solutions to the time synchronization problem exist. There are several UNIX® commands for setting and synchronizing time, as shown in TABLE 1. In addition, there are several other time synchronization protocols, including Digital Time Service (alternately known as DTS or DTSS). Many other synchronization protocols and their relative merits are discussed in detail in RFC1305, which defines

the NTP version 3 specification. Other time protocols have influenced the evolution of NTP, including `rdate` and DTS. The procedure for determining offsets evolved from a model used by telephone companies to synchronize time. NTP builds on the legacy and research efforts of these other protocols, which makes it a very robust and mature technology.

**TABLE 1** Common UNIX Commands Related to Time

| Command | Description |
|---|---|
| `date` | Displays or sets the current system date and time |
| `rdate` | Sets the current system time from a remote host using the `date` protocol (see RFC 868) |
| `adjtime()` | Adjusts the system's time of day clock gradually, to a specified value |
| `settimeofday()` | Sets the system's time of day clock instantly to a specified value |
| `ntpdate` | Sets the current systems clock from a remote NTP server |
| `xntpd` | NTP daemon |

NTP is a good choice for time synchronization in a variety of circumstances. Other schemes, such as DTS, are designed primarily for local area networks, while NTP is designed specifically for Internet environments. Although a number of UNIX commands provide setting or synchronizing time, they don't have the accuracy and robust feature set present in NTP. Flexibility of the client/server relationship and security methods allow NTP to work well in almost any environment. NTP not only corrects the current time, it can keep track of consistent time variations and automatically adjust for time drift on the client. This allows for less network traffic and keeps client clocks more stable, even if the network is unavailable. In addition, the NTP daemon can automatically adjust the time at periodic increments. NTP can also operate through firewalls and has a number of security features.

In addition, NTP operates on a wide variety of platforms. Simple Network Time Protocol (SNTP) is a lightweight variation of NTP which is compatable with NTP and popular on wintel machines. Since many platforms and networking devices support one or both of these protocols, NTP can be easily standardized throughout an enterprise.

# An Overview of NTP

Contrary to popular belief, NTP is not based on the principles of synchronizing machines with each other. NTP is based on the principles of having all machines get as close as possible to the correct time. The effects may be similar, but this is a subtle

but important distinction. The NTP algorithms and the structure of an NTP architecture clearly reflect this distinction. In addition, some of the behaviors of NTP applications can only be understood by keeping this distinction in mind. For instance, if two NTP servers are synchronized to each other as peers, what actually happens is the clocks decide among themselves which is the better source of time, and both clocks attempt to synchronize to that. The result is that the peer that is actually serving the time could change among the clocks as their accuracy changes.

At the top of any NTP hierarchy are one or more reference clocks. These are electronic clocks synchronized to a common time reference and each other using some methods outside the scope of NTP, for instance GPS signals, radio signals, or extremely accurate frequency control. Reference clocks are assumed to be accurate. The accuracy of other clocks is judged according to how "close" a clock is to a reference clock (the *stratum* of the clock, as described below), the network latency to the clock, and the claimed accuracy of the clock.

The public NTP servers available on the Internet use Coordinated Universal Time (the acronym is UTC, taken from the French translation) as the ultimate source of time. UTC evolved from Greenwich Mean Time (GMT), and still uses the Greenwich time zone as the zero offset. GMT is based on the earth's rotation, which is not constant enough to be used for detailed time measurements. UTC is based on a standard second length determined by the quantum phenomena. Time zones and Daylight Savings Time don't exist as far as NTP is concerned. Time is synchronized according to time zone neutral time (as in GMT), and then the time is projected to the time zone using the time zone environment variable (TZ), which can be set in `/etc/default/init`. The possible names for the time zones are the same as the paths to files contained under `/usr/share/lib/zoneinfo`. For instance, setting the time zone for a machine to East Saskatchewan time can be accomplished by looking under `/usr/share/lib/zoneinfo` for the appropriate timezone, and then setting the TZ variable to that value in `/etc/default/init` as follows:

```
TZ=Canada/East-Saskatchewan
```

Time zones for individual users can be changed by setting the TZ variable in their profiles.

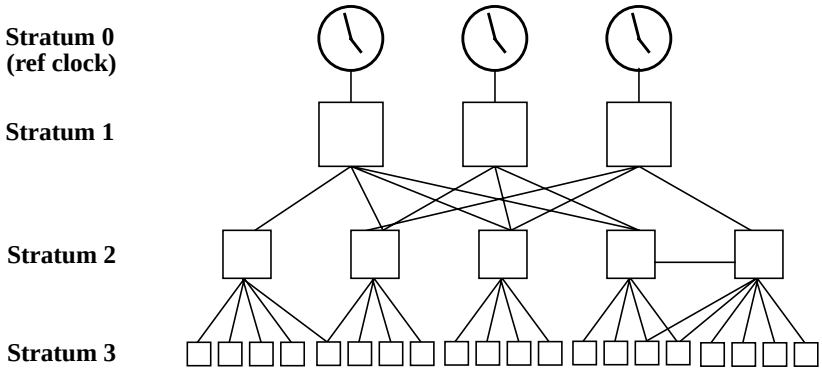
In environments that need more accurate time than an Internet link will allow (due to latency or other concerns), or environments that cannot rely on Internet time sources due to security implications, a radio time clock or GPS system (or cesium clock, if you happen to have one lying around), can be used to keep the primary NTP servers aligned with UTC. Synchronizing a few machines to an arbitrary time source, such as the internal clock on a given server, may be acceptable in a few rare cases, but in any sort of large installation it is critical to keep the clocks synchronized with some maintained time standard. Regardless of the configuration, an NTP server needs to be set up in order for clients to use it for synchronization.

# Networking and Strata

NTP uses the UDP protocol on port 123 to communicate between clients and servers. Attempts are tried at designated intervals until the server responds. The interval depends on a number of factors, and ranges from about once a minute to once every 17 minutes. Using UDP prevents retries from using up network bandwidth if a time server with a large number of clients goes down.

NTP works on a hierarchical model in which a small number of servers give time to a large number of clients. The clients on each level, or stratum, are in turn, potential servers to an even larger number of clients of a higher numbered stratum. Stratum numbers increase from the primary (stratum 1) servers to the low numbered strata at the leaves of the tree. Clients can use time information from multiple servers to automatically determine the best source of time and prevent bad time sources from corrupting their own time. FIGURE 1 illustrates the hierarchical strata model of servers used in NTP.

**FIGURE 1** NTP Strata




Servers that are directly connected to a reference clock are termed stratum 1 servers. A reference clock connected to a stratum 1 server is referred to as a stratum 0 server. Clients never communicate directly with a stratum 0 server; they always go through a stratum 1 server synchronized to a stratum 0 server. However, some vendors sell reference clocks (stratum 0 servers) which include an external network interface that acts as a stratum 1 server. Regardless of the source of time for a server, it is important to remember that the accuracy of these time signals can vary widely. Just because a server is a stratum 1 server does not necessarily mean it has accurate time. In fact, a stratum 1 server could even be configured to use its own poorly running internal clock as a reference clock. This is why it is very important to use multiple time sources and verify time sources before using them.

Clients of stratum 1 servers are referred to as stratum 2 clients. If they serve time to clients, they are also referred to as stratum 2 servers, and the clients they serve are stratum 3 clients. This continues to higher numbered strata. The maximum NTP stratum number for a client is 15; however, in practice it is rare to find clients with stratum numbers above 4 or 5 in most real-world configurations. According to a survey of NTP servers done by David Mills, more than half of the Internet-connected NTP clients are in stratum 3, with almost all the remainder in strata 2 and 4. It is usually best to follow this sort of a distribution in an NTP environment, as described in the next paragraph.

Because of the difficulty of setting up a reliable reference clock, administrators generally want to limit the number of stratum 1 servers, or use publicly available servers (when feasible). Organizations setup public NTP servers as a public service. Setting up a large number of clients to use public external servers is a poor use of their generosity (and computing resources), so it is best to have a few servers receive accurate time readings from the Internet or a WAN, and then use each of these stratum 2 or 3 servers in localized areas (a campus or LAN). All the clients at the site can then receive updates from the stratum 2 or 3 servers at the site.

An alternative to using a public stratum 1 server is to use a public stratum 2 server. In many cases, public stratum 2 servers are well synced to a set of accurate stratum 1 servers. Syncing to several accurate servers minimizes the chances of the server becoming unsynced. Using an accurate stratum 2 server which is synced to several time sources may allow for better accuracy than using an overloaded or inaccurate stratum 1 server that is not configured to communicate with other time sources. Selecting an NTP source requires careful consideration of accuracy and reliability. The process of making this choice is described further in the second article of this series.

## Resource Requirements

NTP requires little resource overhead. This allows NTP to be easily deployed on servers hosting other services, even if the servers are heavily loaded. Even a single processor NTP server can serve hundreds or even thousands of clients using only a few percent of its CPU capacity. If the server is a broadcast or multicast server, even fewer resources are required.

The bandwidth requirements for NTP are also minimal. Unencrypted NTP Ethernet packets are 90 bytes long (76 bytes long at the IP layer). A broadcast server sends out a packet about every 64 seconds. A non-broadcast client/server requires 2 packets per transaction. When first started, transactions occur about once per minute, increasing gradually to once per 17 minutes under normal conditions. Poorly synchronized clients will tend to poll more often than well synchronized clients. In NTP version 4 implementations, the minimum and maximum intervals can be extended beyond these limits, if necessary.

## Types of Clients and Servers

The relationship between NTP servers and clients can be configured to operate in several ways. Computers using NTP can operate in different modes with respect to different machines. For instance, a single machine could be a client of a machine with a lower stratum number, while being a peer to a machine on the same stratum, and a broadcast server to a number of clients at a higher stratum number.

- *Server* – An NTP server serves time to clients. Clients send a request to the server and the server sends back a time stamped response, along with information such as its accuracy and stratum.
- *Client* – An NTP client gets time responses from an NTP server or servers, and uses the information to calibrate its clock. This consists of the client determining how far its clock is off and adjusting its time to match that of the server. The maximum error is determined based on the round-trip time for the packet to be received.
- *Peer* – An NTP peer is a member of a group of NTP servers that are tightly coupled. In a group of two peers, at any given time, the most accurate peer is acting as a server and the other peers are acting as clients. The result is that peer groups will have closely synchronized times without requiring a single server to be specified.
- *Broadcast/multicast server* – An NTP server can also operate in a broadcast or multicast mode. Both work similarly; broadcast servers send periodic time updates to a broadcast address, while multicast servers send periodic updates to a multicast address. Using broadcast packets can greatly reduce the NTP traffic on a network, especially for a network with many NTP clients.
- *Broadcast/multicast client* – An NTP broadcast or multicast client listens for NTP packets on a broadcast or multicast address. When the first packet is received, it attempts to quantify the delay to the server in order to better quantify the correct time from later broadcasts. This is accomplished by a series of brief interchanges where the client and server act as a regular (non-broadcast) NTP client and server. Once these interchanges occur, the client has an idea of the network delay and thereafter can estimate the time based only on broadcast packets. If this interchange is not desirable, it can be disabled using NTP's access control features. The `-r` option can be used when `xntpd` is started to hardwire a delay if the interchange fails because of access control issues or other problems.

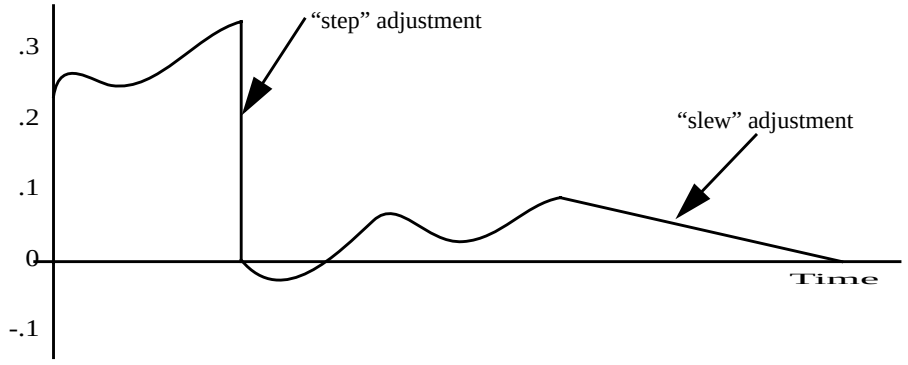
## Accuracy and Resolution

NTP may take several minutes or even hours to adjust a system's time to the ultimate degree of accuracy. There are several reasons for this. NTP averages the results of several time exchanges in order to reduce the effects of variable latency, so it may take several minutes for NTP to even reach consensus on what the average latency is. Generally this happens in about 5 minutes. In addition, It often takes

several adjustments for NTP to reach synchronization. The result is that users shouldn't expect NTP to immediately synchronize two clocks. The `ntpdate` command can be used if instant synchronization is needed. The `peers` command can be used in `ntpq` to determine if synchronization has been achieved. When a client has synchronized, the synchronization server is listed with an asterisk in front of it.

To allow clocks to quickly achieve high accuracy, yet avoid *overshooting* the time with large time adjustments, NTP uses a system where large adjustments occur quickly and small adjustments occur over time. For small time differences (less than 128 ms), NTP uses a gradual adjustment. This is called slewing. For larger time differences, the adjustment is immediate. This is called stepping. The different types of time adjustment are shown in FIGURE 2. If the accuracy of a clock becomes too insufficient (off by more than about 17 minutes), NTP aborts the NTP daemon, with the assumption that something has gone wrong with either the client or server. In order to synchronize well with a server, the client needs to avoid step adjustments.

**FIGURE 2** Time Adjustments Using Stepping and Slewing




Systems using Solaris™ Operating Environment (Solaris OE) 8 with patch 109667-01, 109667-02, or 109667-03 (109668 on x86) always slew the clock. This prevents the clock from shifting suddenly or stepping backwards, but could result in an extremely lengthy synchronization process. In addition, some additional changes are necessary for systems with this patch to synchronize correctly. Specifically, if the above patches are installed, the following line needs to be added to the `ntp.conf` file.

```
disable pll
```

This forces the system to use NTP's clock setting discipline directly, rather than having NTP interact with the kernel's clock setting discipline and is necessary as a side effect of the patch. An updated patch will fix this dependency and the slewing problem. This article will be updated to discuss the fix for this problem as soon as it is released.

Because most customers do not need to force slewing to always occur in lieu of stepping, and because slewing large time differences can lead to a number of problems, we suggest customers upgrade to the new patch as soon as it is available.

The degree of synchronization to a server is dependent primarily on network latency. The accuracy of NTP thus depends on the network environment. Because NTP uses UDP packets, traffic congestion could temporarily prevent synchronization, but the client can still self-adjust, based on its historic drift.

Under good conditions on a LAN without too many routers or other sources of network delay, synchronization to within a few milliseconds is normal. Anything that adds latency, such as hubs, switches, routers, or network traffic, will reduce this accuracy. The synchronization accuracy on a WAN is typically within the range of 10-100 ms. For the Internet, synchronization accuracy is unpredictable, so special attention is needed when configuring a client to use public NTP servers.

If synchronization accuracies better than the above estimates are required, there are several options for obtaining even higher synchronization accuracy:

- *Connecting directly to a reference clock* – If a server is attached directly to a reference clock, the accuracy is limited only by the accuracy of the reference clock and the hardware and software latencies involved in the connections to it.
- *Pulse Per Second (PPS)* – Clocks can use PPS radio receivers, which receive on-the-second radio pulses from a national standards organization. If the time is within a fraction of a second, the PPS pulses can be used to precisely synchronize to the tick of the second. This method achieves accuracies in the tens of microsecond range.
- *Precision time kernel* – The Solaris OE has a precision time kernel that allows the kernel clock to be updated to microsecond resolution (though not necessarily microsecond accuracy). By default, the precision time kernel is used by the versions of NTP which are included with the Solaris OE. Precision time kernels are defined in RFC 1589.

The maximum resolution of an NTP time stamp is about 200 picoseconds (about the time it takes an electrical pulse to travel through 2 cm of copper wire), so the ultimate accuracy of NTP is likely to be limited by hardware and latency concerns, rather than by the NTP protocol.

The important thing to remember is that NTP accuracy is always limited by the time source. A client synced to an inaccurate server, will be inaccurate. This dependency is true all the way up the chain to the reference clock. If the reference clock is miscalibrated, all the clients will be affected.

## NTP Security

The notion of accurate time is essential to determining the order in which events occur. This is a fundamental aspect of transactional integrity, system and network-wide logging and auditing, and troubleshooting and forensics. Having an accurate time source plays a critical role in tracing and debugging problems that occur on different platforms across a network. Events must be correlated with each other regardless of where they were generated.

Furthermore, the notion of time (or time ranges) is used in many forms of access control, authentication, and encryption. In some cases, these controls can be bypassed or rendered inoperative if the time source could be manipulated. For example, a payroll function could be tricked into providing access over a weekend when normally it would be restricted to normal business hours.

Many organizations have become reliant on NTP just as they are with other services such as the domain name service (DNS). This reliance can be a weakness if the service is not properly safeguarded. Therefore, it is important that these time sources are adequately protected against a wide array of threats, both internal and external, local and remote. Time is not just an extraneous service. It is fundamental to the successful operation of today's environments.

The most significant risks to NTP services are tampering and jamming. Tampering occurs when the NTP server is affected by either accidental or malicious data modification. Jamming occurs when a time server is either destroyed or prevented from providing NTP service. As with any other application, administrators must remember that NTP is not guaranteed to be secure; poor coding and other flaws in the program could allow unintended access to NTP internals or the underlying operating system.

The NTP service is capable of protecting itself against some of these threats using architectural choices such as redundancy, and configuration options such as access control and authentication. Redundancy and its impact on an NTP implementation was discussed previously. Access control is achieved by restricting what NTP functions can be accessed from specific hosts or networks. Authentication is currently provided using symmetric keys that are installed on the NTP servers and clients. Both access control and authentication are discussed in more detail in the forthcoming articles of this series.

# NTP components

Several components make up the NTP distribution. The NTP daemon, `xntpd`, is the usual method for using NTP. The `ntpdate` program can be used for fast initial synchronization before using `xntpd`, or instead of `xntpd`. The use of `ntpdate`

instead of xntpd is discouraged, for reasons discussed below. In addition, this section discusses how the ntpq, xntpdc, and ntptrace commands can be used to monitor and control NTP clients and servers.

## xntpd

The xntpd process is the NTP daemon. The same daemon is used both for client and server. The file /etc/inet/ntp.conf configures the NTP servers and clients. The advanced options enable access control, authentication, monitoring, remote management, and better time approximation.

In systems running default installations of Solaris OE versions 2.6 and later, the daemon is started automatically on startup by the /etc/init.d/xntpd file (which is linked to from /etc/rc2.d/S74xntpd) if the ntp.conf file exists. In default installations, the ntp.conf file does not exist, so the daemon never starts. A configuration can be created from scratch or copied from the ntp.servers or ntp.clients files in the same directory. Because the ntp.servers and ntp.clients files do not incorporate any authentication or access control, it is recommended to follow the examples in this article instead.

If the configuration is changed, the daemon will need to be stopped and restarted in order to update the configuration without rebooting. The xntpd process can be started or restarted at any time, and is located at /usr/lib/inet/xntpd. Alternatively, the /etc/init.d/xntpd file can be run with the start or stop options. Keep in mind that changes made to the startup files could be removed by later upgrades or patches.

While most of the configuration is taken care of by the ntp.conf file, there are a few command flags (described in the xntpd man page). While the command flags allow easy configuration from the command line, using them is discouraged, because the same can be accomplished using the configuration file. The advantage of using the configuration file is that all the configuration information is easily determined if troubleshooting is necessary. TABLE 2 describes these files.

**TABLE 2** NTP-Related Files

| Default Location | Description |
|---|---|
| /etc/inet/ntp.conf | NTP configuration file |
| /etc/inet/ntp.client | A preset multicast client file that can be copied over to ntp.conf |
| /etc/inet/ntp.server | A template server file that can be copied over to ntp.conf and modified |
| /etc/inet/ntp.drift | xntpd drift file |
| /etc/inet/ntp.keys | xntpd autentication keys file |

**TABLE 2** NTP-Related Files *(Continued)*

| Default Location | Description |
|---|---|
| no default | log file |
| `/var/ntp/ntpstats` | directory for NTP statistics files (if used) |
| no default | `pid` file |

The `xntpd` process can communicate both time and configuration information. If enabled, this allows the configuration for `xntpd` to be queried or changed remotely. There are two different types of configuration messages which can be sent: mode 6 messages and mode 7 messages. Mode 6 messages are sent by the `ntpq` program and most mode 6 messages are informational. The others deal with setting variables. Mode 7 messages are sent by the x`ntpdc` program. Mode 7 messages can be informational or can allow remote changing of the NTP configuration.

# `ntpdate`

The `ntpdate` program allows a client to set its date from an NTP server without permanently running the NTP daemon. Multiple servers can also be specified to allow a better selection for the best time source. Essentially, this allows a *one-shot* NTP transaction. Because this interaction only occurs once, the clock will eventually stray again. Repeated usage of `ntpdate` can overcome this, but is far from an ideal solution.

Some system administrators run an `ntpdate` command on an hourly, daily, or weekly basis as a `cron` job, which is capable of keeping servers in sync within a few seconds. This light-weight solution is useful in some environments because it requires one less daemon, gives the administrator more control over NTP traffic, and is easy to configure. However, in nearly all cases, the problems are likely to outweigh the benefits.

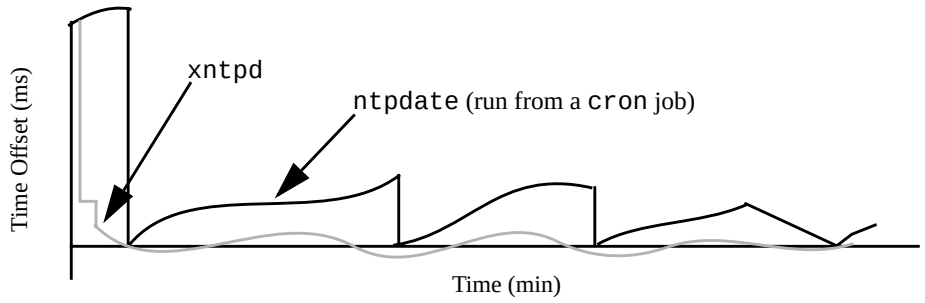
The processor load of the NTP daemon is small, so little is gained by disabling it. In addition, many clients run `cron` jobs simultaneously, which causes blasts of network traffic for every `cron` job in the span of a few seconds. This is likely to overwhelm the network, and possibly even the server (if the server is small and has a large number of clients).

The client-side `xntpd` process randomizes the delay before a time request is made, so that this overwhelming does not occur. Since both the `ntpdate` cron job and the NTP daemon are simple to configure, there is no clear management advantage to either.

A final advantage of `xntpd` over `ntpdate` is that it produces a much better time sync. This is due to several facts. Unlike `ntpdate`, `xntpd` can limit the wander of the clock based on historical data—even when a time server is unavailable. The

xntpd process does time adjustment continuously, so the changes to the time can be very small, so it rarely needs to use the *time jumps* that are likely to occur when ntpdate is run. An example of the relative accuracies of using xntpd or running ntpdate on an hourly basis is shown in FIGURE 3.

**FIGURE 3** Time Accuracy With xntpd and ntpdate Run From cron




## ntpq

The /usr/sbin/ntpq program allows querying the state of the NTP daemon on a local or remote machine. Using ntpq, an administrator can check the configuration of a remote host. If such queries are allowed on a host, this can be a useful way of choosing hosts to synchronize with, because information such as their peers and reference clock types can be determined. Since ntpq uses UDP packets, hosts may be falsely unreachable on congested networks.

## xntpdc

The /usr/sbin/xntpdc program also allows querying the state of a local or remote NTP daemon, however, xntpdc can also make runtime configuration requests to a remote machine. This allows changing the configuration on the fly. In order to make runtime configuration changes, an authentication key is needed. This requires the creation of an NTP keys file, which is described in the xntpd man page. Like ntpq, xntpdc uses UDP packets.

---

**Note –** The xntpdc program is not included with the NTP implementations which come with Solaris OE versions 2.6 and 7.

---

## ntptrace

The /usr/sbin/ntptrace is an informational command that traces the source of a given client's time. The information found by ntptrace can be determined by successive runs of ntpq on each client's preferred server. However, ntptrace allows an easy and convenient way of learning this information. This can be a useful tool for debugging. The following is an example of ntptrace output.

```
localhost: stratum 6, offset 0.000063, synch distance 1.19437
hickory: stratum 5, offset 1.903098, synch distance 0.13986
dickory: stratum 4, offset 1.900733, synch distance 0.12984
doc: stratum 3, offset 1.880003, synch distance 0.10143
clock.east: stratum 2, offset 1.904045, synch distance 0.06825
stonehenge.uk: stratum 1, offset 1.907003, synch distance 0.00797, refid 'TRUE'
```

The ntptrace output lists the client name, its stratum, its time offset from the local host, the synchronization distance, and the ID of the reference clock attached to a server, if one exists. The synchronization distance is a measure of clock accuracy, assuming that it has a correct time source. For the exact derivation, refer to the NTP specification.

# NTP Versions and Issues

NTP is perhaps one of the oldest and most used protocols on the Internet. The first version of NTP was released in 1985, but is no longer in common usage. The specifications for version 3, the latest *official* version, were released in 1992. The Internet RFC 1305 provides the specifications for version 3 of the NTP protocol. While finalized specifications for version 4 have yet to be released, implementations of the in-progress specification are already available. NTP versions can interoperate, though some features from later versions may not be available to clients using earlier versions, and the xntpdc program, which is not critical to NTP operation, may not work properly between different versions.

Implementations of NTP are included with recent versions of the Solaris OE and Open Source versions are available from http://www.ntp.org. Using the NTP distribution included with the Solaris OE distribution makes support issues easier, because Sun supports it. The open source versions may be desirable if newer features are necessary—for example, if the advanced cryptographic authentication features in NTP version 4 are needed.

There are a number of different versions of NTP available, which can be quite confusing to an NTP administrator. This section outlines some of the more recent versions available. This section also highlights a few bugs and issues that users should be aware of. NTP is available as open source from `http://www.ntp.org`. The versions included in the Solaris OE are derived from versions of the open source NTP. The mapping of open source versions to Solaris OE versions is explained in TABLE 3. The primary version number indicates the specification version and the subsequent numbers indicate the software version. Note that all current versions included with Solaris OE use the NTP version 3 specifications.

**TABLE 3** Solaris NTP Versions

| Solaris OE version | Approximate Open Source NTP version included |
|---|---|
| pre 2.5.1 | None |
| 2.5.1 | NTP 3.2 (supported on Ultra Enterprise 10000 only) |
| 2.6 | NTP 3.4y, not including `xntpdc` |
| 7 | NTP 3.4y, not including `xntpdc` |
| 8 | NTP 3-5.93e |

The NTP versions shipped with the Solaris OE are very similar to the open source versions available on the Solaris OE. The versions included with Solaris OE have some minor, Solaris OE specific modifications. In general, these modifications do not affect any user-visible features. In addition, the versions included with Solaris OE only include the core NTP applications and not the utilities that come along with the open source version. Sun supports the included NTP versions on each OS version. No other NTP versions are supported by Sun. The lack of Sun support doesn't mean a version won't work, it simply means that Sun isn't obligated to support it as part of a standard service contract. Many customers use NTP versions other than the included ones (particularly NTP v4), and simply support them locally or through the open source community.

# The History of NTP on Solaris OE

Prior to Solaris OE 2.5.1, NTP was not included in the distribution and had to be obtained separately. The first inclusion of NTP with the OS was in the Solaris OE 2.5.1 Hardware release 4/97, but Sun only supported using this NTP version on the Sun Enterprise™ 10000 server. In this version of NTP, `dosyncdr` in the `/etc/system` file had to be manually set to 0 (1 is the default).

Setting `dosyncdr` is unnecessary (and will cause problems) for later versions. Occasionally this problem pops up when customers come across out-of-date configuration guides. For versions of Solaris OE 2.6 and later, setting `dosyncdr` to 0 will prevent NTP from working. Since `dosyncdr` is 1 by default, no `/etc/system`

changes need to be made for NTP versions included in Solaris OE 2.6 and later. The `dosyncdr` change is unnecessary in 2.6 and later because the hardware clock is automatically set when the software clock is set.

In Solaris OE 2.6 and 7, Sun included an NTP version which was supported on all platforms. The new NTP also added support for multicasting. The included version was a slightly modified version of the public domain distribution of NTP 3.4y. This version included `xntpd`, `ntpdate`, `ntptrace`, and `ntpq`. Notably absent, was `xntpdc`.

In the Solaris OE 2.6 version of NTP, a race condition exists between the NTP daemon and the name service caching daemon (`nscd`). This problem has a low probability of occurring, and only occurs at boot time if name resolution is needed in the `ntp.conf` file and if `nscd` was used. If this problem does occur, it could cause `xntpd` to use very large amounts of CPU and affect other processes. Because this problem has a low probability of occurring even if all the conditions existed, some systems running Solaris OE 2.6 could be susceptible without anyone realizing it. The workaround is to load `xntpd` after `nscd` (move `/etc/rc2.d/S74xntpd` to `/etc/rc2.d/S77xntpd`), or to avoid using host names in the `ntp.conf` file. This problem was fixed in the Solaris OE 7 version of NTP.

In Solaris 8 OE, the included version is a slightly modified version of the open source distribution of NTP 3-5.93e. This version included `xntpdc`.

# Conclusion

NTP provides an excellent means of keeping a large number of nodes in close synchronization. An effectively designed NTP infrastructure can accomplish this with a minimum of network overhead and can also maintain a high level of accuracy and security. In addition, NTP solutions are relatively easy to architect and implement, making them ideal for everything from a small business to wide area enterprise deployments. The clock synchronization afforded by NTP can be useful for a variety of purposes, including keeping time stamps on different network nodes in sync, which enables easier network and security troubleshooting.

Later articles in this series will be focused on a technical level and will explain how to configure, architect, and troubleshoot NTP installations.

# References

The best source of basic information on NTP is the NTP website, at http://www.ntp.org. Included online are the NTP manual pages and the NTP FAQ, at http://www.eecis.udel.edu/~ntp/ntpfaq/NTP-a-faq.htm.

More technical information can be found on the following sites:

- http://www.eecis.udel.edu/~mills/ntp.htm
- http://www.eecis.udel.edu/~mills/papers.htm
- http://www.eecis.udel.edu/~mills/reports.htm
- http://www.eecis.udel.edu/~mills/memos.htm

The following RFCs relate to NTP. These can be found in many places throughout the internet, but are also available from the NTP home page.

- NTP version 3 specification, RFC 1305
- SNTP specification, RFC 1361
- Precision Time Kernel specification, RFC 1589

We used many of the above resources in the development of these articles, and would also like to thank Dr. David Mills for his help in answering questions and reviewing the text.

We'd also like to thank Bill Watson, Ulrich Windl, Terry Williams, Cathleen Plaziak, Regina Elling, Don Devitt, Kemer Thomson, and Alex Noordergraaf for help in reviewing and editing this article.

## *Author's Bio: David Deeths*

*David Deeths has worked for Sun's Enterprise Engineering for 4 years. His work has focused primarily on clusters and high availability systems. He is the co-author of the Sun™ Blueprints book "Sun Cluster Environment: Sun Cluster 2.2" and has published a number of online articles on a variety of topics.*

*David holds Bachelor of Science degrees in Electrical Engineering and Cognitive Science from the University of California, San Diego.*

## *Author's Bio: Glenn Brunette*

*Glenn Brunette has more than 8 years experience in the areas of computer and network security. Glenn currently works with in the Sun Professional Services organization where he is the Lead Security Architect for the North Eastern USA region. In this role, he works with many Fortune 500 companies to deliver tailored security solutions such as assessments, architecture design and implementation, as well as policy and procedure review and development. His customers have included major financial institutions, ISP, New Media, and government organizations.*

*In addition to billable services, Glenn works with the Sun Professional Services Global Security Practice and Enterprise Engineering group on the development and review of new security methodologies, best practices, and tools.*