SUN BLUEPRINTS
[illegible]
APPROVED

Using NTP to Control and Synchronize System Clocks - Part III:

NTP Monitoring and Troubleshooting

By David Deeths - Enterprise Engineering and Glenn Brunette - SunPS

Sun BluePrints™ OnLine - September 2001

`http://www.sun.com/blueprints`

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

Part No.: 816-2353-10
Revision 01, 09/05/01
Edition: September 2001

Using NTP to Control and Synchronize System Clocks- Part III: NTP Monitoring and Troubleshooting

This article is the third in a series of three articles that discuss using Network Time Protocol (NTP) to synchronize system clocks. The goal of this article is to provide an effective understanding of NTP troubleshooting and monitoring. Clock synchronization of the sort provided by NTP is critical for tracking events logged on disparate network nodes in a common time scale, therefore solving issues with an incorrect configuration is important.

Without time synchronization, it can be difficult or impossible to correlate the time sequence of events involving multiple network components, such as outages, bugs, security breaches, service problems, or network problems. While it is usually easy to set up and use NTP, it can sometimes be difficult to determine where the problem lies when the configuration does not work properly.

To perform effective troubleshooting, an in-depth understanding of the inner working of NTP is required. The first section of this article covers the inner works and algorithms of NTP to provide an understanding of how to perform effective troubleshooting. The second section offers NTP troubleshooting techniques and commands.

For readers who are unfamiliar with NTP, the first two articles in the series provide an introduction to NTP. The first article of this series covered basic NTP and time synchronization concepts. The second article covered NTP administration, architecture, security, and setup. It is recommended to read the first and second articles before beginning this one, unless the reader has extensive knowledge of NTP.

NTP Algorithms

The algorithms used in NTP have been carefully and continuously refined and are designed to quite carefully optimize accuracy as much as possible. This section outlines the important features of the algorithms most likely to concern system and network administrators. The goal of this section is not to provide an in-depth explanation of exactly how all elements of NTP work, but to provide an understanding sufficient for troubleshooting.

Many NTP errors can be easily traced down to the part of the algorithm in which a failure occurred; hence, it is important for an administrator to understand the different parts of the algorithm and the basics of what happens in each one. The details and derivation of the algorithms are much too complex to outline here, but interested readers should refer to RFC 1305 for an extremely detailed view of the control algorithms. However, readers interested in this high level of detail will also find this introduction useful, because there is no high level overview in the specifications.

NTP is designed to allow a clock to synchronize as closely as possible to a universal time standard, even in the presence of incorrect time sources, or when accurate time sources are temporarily unavailable. Keeping time synchronized requires several different procedures. First, the current time needs to be ascertained. This involves statistical methods of choosing the best source from among several possible sources.

Once a time source has been determined, the client needs to synchronize with it. In addition, to make synchronization less necessary and to allow the client to keep fairly accurate time in the absence of time servers, the client clock needs to be disciplined to adjust incorrect client clock frequency. The subsections below discuss both how NTP solves the problem of time determination and how NTP adjusts the local clock.

The Steps of the Time Algorithm

Several steps are involved in determining the correct time. Although some of these steps are not necessary when a client is only synchronizing to a single server, it is essential to understand them for more complex setups.

The following is a list of the basic steps involved in determining the time as closely as possible, which are explained in more detail below. The steps are performed in order:

1. Sanity checks

    Ensures that packets are valid

2. Filtering

   Uses a history of samples for a given clock to reduce jitter

3. Intersection algorithms

   Removes clocks that appear to be set to the wrong time

4. Candidate checks of the clustering algorithm

   Removes the worst clocks until only 10 remain (to limit processing)

5. Synchronization of source selection of the clustering algorithm

   Selects a “best” source out of those remaining

6. Clock combination

   Revises the time from the “best” source based on time and error estimates from the other clocks

Any server that survives the sanity checks is presumably configured correctly. The *Sanity Checks* section describes typical misconfigurations that could cause problems. The subsequent sections give an overview of the steps appropriate for understanding NTP logs and monitoring commands.

Sanity Checks

NTP makes sure that packets are valid before they are processed. There are several steps taken to determine if a packet is useful. Knowing how the sanity checks work is important, because if all the packets sent from a server fail the sanity checks, `ntptrace` will time-out without providing information. The `ntpq peers` subcommand will show hosts that fail the sanity checks with no symbol in front of them (hosts that pass the sanity checks will have a symbol in front of them, such as a letter “x”, a period “.”, an asterisk ”*”, a letter “o”, a pound sign “#”, or a plus sign “+”). The *Troubleshooting and Tuning* section explains how to determine when a server has failed a sanity check; however, before reading that section, it is important to understand what sanity checks are run.

The following sanity checks are performed on each packet, respectively:

1. A packet must not be a duplicate of another packet from the same server.

   This eliminates the possibility of a packet arriving through 2 routes and being processed twice.

2. A packet must contain the same client timestamp as the last packet that was sent out by the client.

This is a consistency check that the server is answering the latest client request. This provides protection against receiving packets out of order. It also may provide some protection against forged packets.

3. The server is reachable.

   An artifact of the way xntpd works is that a packet is forged on the client, even if a server is unreachable. Thus, xntpd does not recognize a server is unreachable until this step. This is important, because xntpd may do some processing on data representing parts of an expected packet, even if the packet and its useful data is never received. The fact that unreachable servers are not eliminated from the server list until the sanity checks take place, is often confusing to system administrators doing troubleshooting.

4. The calculated round trip delay and dispersion for server must be less than 16 seconds.

   If the error range of a clock (primarily determined by network latency) is more than 16 seconds, then xntpd considers the clock too inaccurate to synchronize to. This could happen in some extremely congested networks. It is unlikely that system administrators will encounter situations where this check is failed.

5. Authentication must succeed (if configured).

   If the client is only configured to accept authenticated packets, then unauthenticated packets are denied during the sanity checks. This is often very confusing to system administrators, who expect authentication to be checked before any processing is done on the packet. If you are using authentication and a reachable server is failing the sanity checks, checking the authentication configuration is usually a good starting point.

6. A server's clock must be synchronized to an external source that has been updated within the last day.

   NTP will ignore packets from a server that is not ultimately synchronized to a stratum 0 server (directly or through a tree of servers). The one day restriction allows a server to lose connectivity for up to 24 hours and still provide NTP service to its clients. The one day restriction also allows servicing to a reference clock, and prevents synchronization from occurring if the clock is permanently disabled. This avoids a client from connecting to a server that was never intended to be a time server, for instance an incorrectly configured workstation; it also means that servers that are not connected to a root time source will be ignored.

   In many cases, system administrators try to setup an NTP server without a reference clock connected to it or one of its servers. Although the setup of such a machine is as an NTP server, the machine will not serve time until it is somehow connected to a reference source. A server must either be configured to be a stratum 0 server using its internal clock for reference (not recommended), connected to a reference clock, or served by a server that can be traced to a reference clock. If none of these conditions hold, then the client will ignore packets from the server. This problem can often cause a new server installation to be unrecognized by the clients.

7. Server must be at a lower or equal stratum number than the client.

   This prevents loops in the NTP configuration.

8. The total delay and maximum error must be less than 16 seconds from the root clock.

   This prevents inaccurate NTP configurations where fairly large latencies exist at numerous levels of the hierarchy. Note that because of the clustering algorithm, the delay and maximum error of the total of all the steps will usually be less than the delay and maximum error of each step added together.

Once the packet has passed through the sanity checks, some higher-level processing can be performed.

Estimating Offset and Error

An NTP client can have a relationship with up to 64 servers, but only 10 of these can serve as potential synchronization sources. Increasing the number of servers an NTP client can connect to is good for two reasons: it allows for more accurate time, and reduces the chance of the time becoming unsynchronized from an accidentally or intentionally misconfigured time source. These advantages are realized in the clustering algorithm and the clock combination algorithm.

NTP not only tries to synchronize to true time, it also computes an error range on a client’s time. In fact, this error range is closely tied to an NTP client’s choice for a “true” time server. The maximum error in either direction is called the dispersion (this is often referred to in the NTP man pages, but never explained). NTP determines the best time source based on several factors, including the offset, the delay, and an error factor.

The offset is half the difference between the time it takes for the client/server query and the server/client response. This difference is measured from the time the packet arrives and leaves the server, and when the packet leaves and arrives at the client. The delay is the round-trip time for the client query to be received back by the client. The time the server spends processing the client query is subtracted from the round-trip timed, because it is not a network issue. The error factor represents errors related to clock reading times and frequency tolerance. The dispersion can be represented as half the delay plus the error.

FIGURE 1 provides an brief description of clock offset and dispersion.

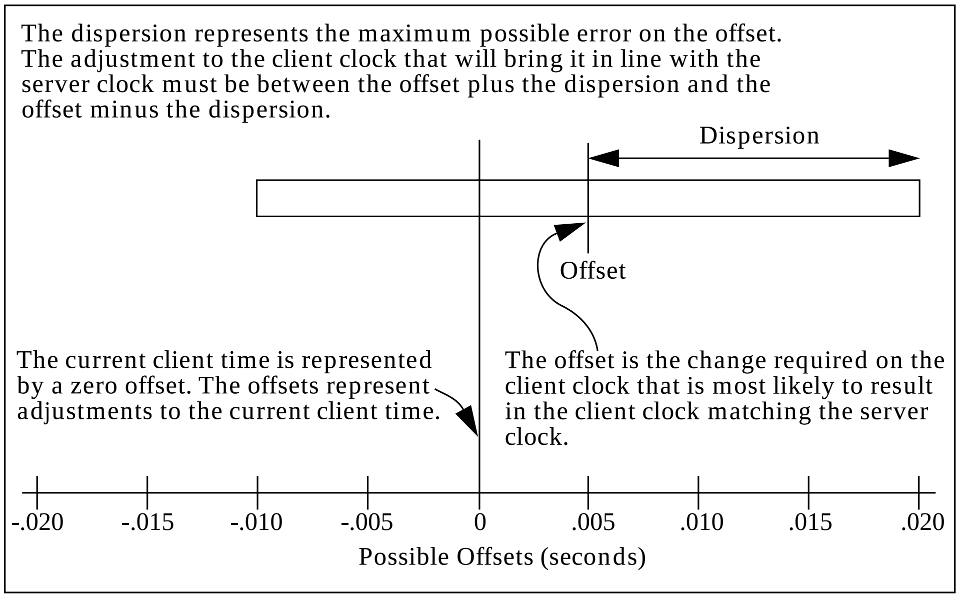
The dispersion represents the maximum possible error on the offset. The adjustment to the client clock that will bring it in line with the server clock must be between the offset plus the dispersion and the offset minus the dispersion.
Dispersion
Offset
The current client time is represented by a zero offset. The offsets represent adjustments to the current client time.
The offset is the change required on the client clock that is most likely to result in the client clock matching the server clock.
-.020
-.015
-.010
-.005
0
.005
.010
.015
.020
Possible Offsets (seconds)

FIGURE 1 Clock Offset and Dispersion

Using the previous case, the amount of time the client needs to be adjusted to match the server must be between the following:

```
offset - delay/2 - error <= actual time adjustment <= offset + delay/2 + error
```

FIGURE 2 illustrates the process NTP uses to determine offset and dispersion. First, a client sends a request to a server. This request contains the time it was sent (according to the client's clock), "A". The request experiences some amount of network delay, unknown to NTP, but recorded to be .01 seconds in this example. The server receives the request and notes the receive time (according to the server), "B". The request is then processed and stamped with the time (according to the server), "C". The response experiences some amount of network delay, unknown to NTP, but recorded to be .02 seconds in this example. The client then checks this with its time when the response is received, "D".

FIGURE 2 illustrates an NTP client/server query and response.

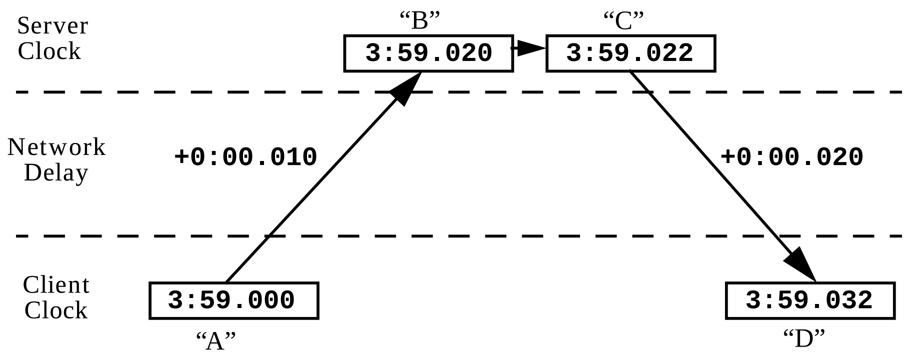

FIGURE 2 Sample Client/Server Exchange

In this example, the offset is ((B-A)-(D-C))/2, or .005 seconds. The delay is ((D-A)-(C-B)), or .03 seconds. The error will be ignored. The degree of possible client adjustment then falls within the range -.01 seconds to .02 seconds (.005 +/- .03/2).

NTP has no knowledge of the true disparity between the clocks, but it assumes (for single server configurations) that the correct client adjustment is exactly at the midpoint of this interval. In the previous example, the clock would be adjusted by .005 seconds. In reality, for the clocks to be truly synced, the adjustment should have been .01 seconds, but NTP did well, given the excessive network latency.

Note – Theoretically, the true offset must be within the error bounds set by the dispersion. However, NTP does make some assumptions that might not always be true in the real world, for instance the statistical distribution of network latencies assumed by NTP may not match the true distribution of a given network. As much as possible, correction factors are set to account for these sorts of factors, but it is possible under some circumstances to have an offset from true time that is outside the error bounds. Because this is a rare case that has only minor effects, this is not something that users need to be concerned with.

When NTP uses multiple clocks, the procedure is much more complex. NTP chooses the best clock to synchronize with, based on stratum, network latency, and claimed accuracy. If a clock is selected as preferred, it will be chosen unless its time is far enough off from the other available servers that NTP assumes it is malfunctioning. When multiple clocks are synchronized with NTP, the error bounds still determine the clock adjustment value, but several clocks are combined to determine the final adjustment value. An environment with a combination of multiple clocks can be confusing, and lead to troubleshooting problems. The following subsection describes the inner works of a clock combination.

Filtering and Intersection Algorithms

Once a packet is received, the packet data enters a queue of packet data from the same source. The queue contains 8 positions and when a new packet arrives, the oldest packet data is discarded. NTP uses a filtering algorithm to reduce the effects of small errors on the clock accuracy. The output from the filtering algorithm are values that represent the best guess at the current offset and maximum error from a given clock. Understanding the filtering algorithms is not required (although the interested reader can read RFC 1305), but the queue itself is important to know for troubleshooting.

It is important to understand the queue in order to understand the reachability fields in the loop logs and in the `ntpq` output (described in the *Troubleshooting and Tuning* section). Also, the queueing behavior affects the speed that NTP can recognize a new server. Because the queue needs to have a majority of the slots with valid data, a server is not valid for synchronization until it sends 5 valid packets. This requirement is the reason it takes `xntpd` approximately 5 minutes to sync to a new server.

Once the data is filtered, `xntpd` compares the filtered data from the various clocks to determine an intersection in which the "true" time must lie. This intersection algorithm is accomplished by using the offset from each clock, along with the maximum possible error for each clock. If two clocks don't have overlapping error bars, then one or the other clock must be incorrect. If there are several clocks, then an intersection of the error bars of the majority of them is the most believable.

Clocks that have error bars that overlap this interval are in the majority. In the NTP documentation, clocks that are in this majority are called "truechimers" and clocks that are outside this majority are called "falsetickers." In most cases, all the clocks in a configuration will be truechimers. If a clock appears to be a falseticker, then it is important to investigate to see what the problem is. While in an ideal world, all falsetickers would be the result of incorrectly set clocks, it is possible (but very unlikely) that a falseticker may have a correctly set clock. In most cases, falsetickers are the result of a server which is using its own inaccurate internal clock as a reference clock. As the internal clock wanders, the server slowly becomes a falseticker. FIGURE 3 is a diagram of truechimers and falcetickers in the intersection algorithm.

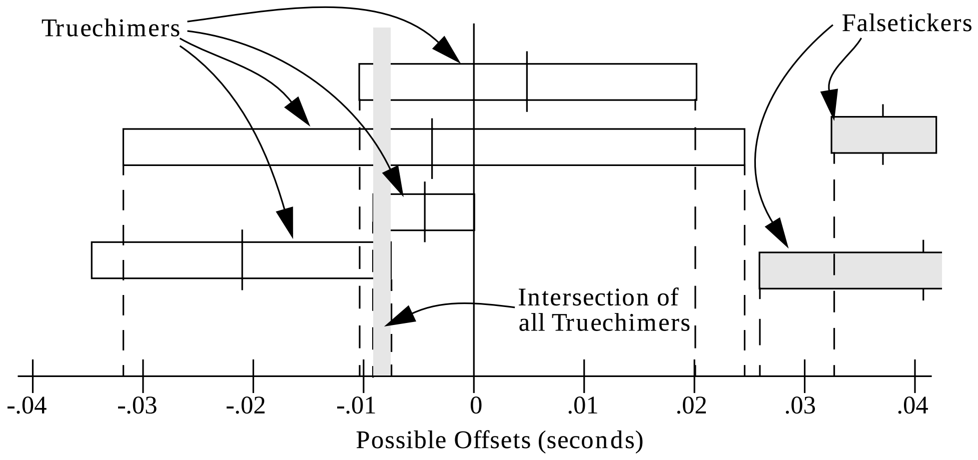

FIGURE 3 The Intersection Algorithm

If a server is a falseticker, it will have an “x” in front of the hostname in the output of the `ntpq peers` subcommand.

Clustering Algorithm

The clustering algorithm sorts all of the truechimers based on a weighted combination of several factors determining the quality of the server (stratum, dispersion from the root clock, dispersion to the client, delay from the root clock, delay to the client, and possible slewing error). Stratum will almost always dominate this selection. The list is then trimmed so only the 10 “best” clocks remain. Servers which are trimmed out of the list in this manner show up with a period "." in front of their names in the output of the `ntpq peers` subcommand. The screen output below shows a sample of the `ntpq peers` output.

```
ntpq> peers
.ahau.mex.sun.co  matins.hours.su   4 u   88  512  377    52.28  -15.444   11.23
-imix.palenque.s  lauds.hours.sun   4 u    -  128  376    82.72   33.854    9.02
+ik.tikal.sun.co  prime.hours.sun   4 u  656  512  324    82.44  -10.320    7.34
.akbal.tulum.sun  terce.hours.sun   4 u  721 1024  377   132.80  -26.504    9.37
-kan.sayil.sun.c  sext.hours.sun.   4 u    -   64  376    72.14  -59.963    2.18
+chicchan.bonamp  nones.hours.sun   4 u    -   64  333    74.94    0.308    0.85
 cimi.seibal.sun  compline.hours.   4 u    3  256  171    76.00   -2.084 8000.53
+manik.uxmal.sun  vespers.hours.s   4 u    -   64  336    75.87    7.717    1.16
 lamat.uxmal.sun  compline.hours.   4 u    -   64    2     0.00    0.000 16000.0
 muluc.tikal.sun  0.0.0.0          16 -    -   64    0     0.00    0.000 16000.0
 oc.palenque.sun  0.0.0.0          16 -    -   64    0     0.00    0.000 16000.0
 224.0.1.1        0.0.0.0          16 -    -   64    0     0.00    0.000 16000.0
+chuen.tikal.sun  lauds.hours.sun   4 u  664 1024  377    56.21   -0.620    5.17
*eb.copan.sun.co  prime.hours.sun   3 u  639 1024  377    54.76   -1.608    3.07
xben.palenque.su  lauds.hours.sun   3 u  459 1024  367    80.84  -208.84  100.78
+ix.caracol.sun.  prime.hours.sun   3 u  682 1024  377    92.48    1.055    4.14
+men.yaxchilan.s  terce.hours.sun   4 u   28  128  352    67.50   -6.308    1.80
.cib.copan.sun.c  terce.hours.sun   4 u   52   64  174    68.45   -1.023  126.74
 caban.tulum.sun  nones.hours.sun   5 u    -   64  237    61.26   -2.177  500.79
-etznab.seibal.s  compline.hours.   4 u  602  512  372    72.02  -24.098    6.68
```

The clock selection then further sorts and trims the list based on an analysis of the historical and current dispersion and error from each clock. If the current synchronization source survives, it is automatically selected, otherwise the best clock in the list is selected as the new synchronization source.

The synchronization source appears with an asterisk "*" or the letter "o" in front of it in the `ntpq peers` subcommand output. If no servers are within the maximum distance, the potential synchronization source has a pound sign "#" in front of it. Any other servers which survive the clustering algorithm have a plus sign "+" prepended to the hostname in the `ntpq peers` subcommand output.

Clock Combination

Although only one clock is chosen to be the synchronization source, the correction on the client clock is influenced by all the survivors of the clustering algorithm. The offset from each clock is weighted by its quality (determined by the factors described in the *Clustering Algorithm* section). The weighted offsets are added together and the average is used as the amount to adjust the client clock.

Synchronizing and Disciplining Clocks

NTP uses two different types of clock adjustments. For small differences, the clock is gradually adjusted. For larger differences, the clock is instantly adjusted to the correct value. For crystal clock oscillators such as the ones found in Sun workstations and servers, a small difference is anything less than or equal to 128ms difference.

The gradual adjustment is called slewing. When the clock is slewed, its phase and frequency are gradually changed to synchronize with the right time. NTP accomplishes this by calling a kernel routine which gradually adjusts the time. If the precision time kernel is disabled (by including the entry, `disable pll` in the `ntp.conf` file), the `adjtime()` function is used. Using the kernel routine is better because it provides greater accuracy and is more efficient.

The instant adjustment is called stepping. When a clock is stepped, it is simply set to a new value. Stepping is necessary because slewing could take a prohibitively long time in some circumstances. Because NTP is usually proficient keeping the time in very close synchronization, stepping is rare. In fact, unless the system's clock is wandering very badly, stepping usually only occurs after a reboot. Stepping the clock can be undesirable, because the clock can be stepped backwards, causing problems with some applications. Stepping is accomplished using the `settimeofday()` function.

Infrequent corrections are best, and occur if the clock is more stable, which allows slewing to be the primary means of time synchronization. A client does not truly sync to a server if it is stepping because the clock is drifting very fast (otherwise it would be slewing). This could indicate some sort of problem with the client's clock or network latency.

Slewing may always need to be used if sudden jumps are unacceptable, or if applications can't deal with the possibility of the clock suddenly jumping backwards. Stepping may be desired if clock adjustments need to be faster. In either case, recall that `xntpd` will usually slew, and even if stepping is needed occasionally, the steps will generally be very small (seconds or subseconds).

With the Open Source version of NTP, stepping backwards can be disabled by starting `xntpd` with the `-x` flag. In this case, the clock may still step the time forwards, but it will always slew the clock to correct a clock that is running too fast. This is important for some applications that have issues if, for example, a reply is timestamped before the original request because of clock adjustments. The versions of NTP included with the Solaris™ Operating Environment (Solaris OE) do not include this functionality. A Solaris OE patch for NTP will disable stepping entirely, but that is inappropriate in most circumstances. This is discussed in the *Slewing Effects From NTP Patches* section below. For a discussion of NTP version differences, refer to the first article in this series.

The `ntpdate` command uses the same stepping and slewing conventions as `xntpd`. However, it is possible to force `ntpdate` to always use one or the other. The `-b` flag forces `ntpdate` to slew the time until it matches the correct time, regardless of the time offset. The `-B` flag forces `ntpdate` to step the clock to the correct time.

Slewing Effects From NTP Patches

Systems using Solaris 8 OE 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. Since the maximum slew rate is about 0.5 ms per second, the time will take at least 200 times the offset to reach the correct time. Thus if the clock is off by 1.5 minutes, it will take at least five hours to correct the time. 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
```

If this line is not included, clocks will synchronize to the server, but will adjust to maintain whatever offset they originally had to the server, rather than adjusting to eliminate the offset. Because `ntpdate` (used to set the client clock to the server clock) is run prior to starting `xntpd` in the `/etc/rc2.d/S74xntpd` script, the clock may already be quite accurate, since `ntpdate` adjusts the clock to within a fraction of a second of the correct time. `xntpd` then maintains this fraction of a second difference (rather than allowing the clock to wander). However, if `ntpdate` is unable to reach the server, problems could result. The system will work with or without `ntpdate` if it does not use the above patches, or if the `ntp.conf` file includes a `disable pll` line (if the patches above are installed).

The `disable pll` entry in the `ntp.conf` file forces the system to use NTP's clock setting discipline directly, rather than having NTP interact with the kernel's clock setting discipline. This entry is necessary as a side effect of the patch, and an updated patch will fix this dependency and the slewing problem. This article will be updated to discuss the fix for this problem when the new patch is released.

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

Time Offset Aborts

A sometimes confusing feature of NTP is that the NTP daemon on a client will abort itself if the offset between its clock and the server's clock is too great. This behavior is triggered by an offset of greater than about 17 minutes. The rationale behind aborting the daemon over an excessive time error is that such a difference likely means that either the client is drifting at a very high rate, or the server has suddenly had its time changed using some method other than NTP. The other potential cause is client booting. The system clock may run at a slightly different speed when the machine is powered off, so if the machine has been powered off for a long time, this offset is possible.

This behavior can often be confusing to system administrators, especially because `xntpd` may not abort immediately. NTP has a sanity checker if it finds a very large offset. The sanity check involves waiting for several confirmations of negative offset, and may take 15 minutes or more. Thus, `xntpd` may come up and then abort after 15 minutes. The sanity interval is a protection against transient network latency that unexpectedly causes a very large delay. A related problem is that `xntpd` dies relatively silently when this occurs. Administrators who are unaware of this "feature" are likely to boot a machine with a poor time setting and later have the NTP daemon die unnoticed. They may have a false sense that the client is still being synchronized. The NTP startup script (`/etc/init.d/xntpd`) included with Solaris OE versions 2.6 and later solves this problem by calling `ntpdate` first, waiting a few seconds, and then starting `xntpd`.

The open source version of `xntpd` uses the `-g` flag to set `xntpd` so that it will not abort due to excessive time difference.

The `stats` logs used by NTP will not show that NTP has aborted, but it will show up in the `syslog` logs. However, the error does not make it clear that the daemon has aborted, so there is a risk some systems administrators may overlook or ignore it. The following is an example of the `syslog` output:

```
Aug 28 15:24:36 lefay xntpd[6086]: [ID 261039 daemon.error] time error
2050.390807 is way too large (set clock manually)
```

Troubleshooting and Tuning

There are a variety of methods available to troubleshoot an NTP installation. These include debugging output, logs, and several programs which are designed to monitor `xntpd`. All of these methods are described below.

Logging

As with any important service, logging is an important part of NTP administration. However, the standard NTP logs are not likely to be very useful for most system administrators.

NTP events that are important to the system are logged to `syslog`. This includes starting and stopping, but not much else. Particularly, `syslog` does not log problems with the clock, problems with synchronizing, or attempts by unauthorized clients to synchronize or monitor the server. The lack of this information can make it very difficult to debug NTP based on the standard logs. In fact, there is basically no choice but to resort to other means of debugging.

Starting `xntpd` with the `-d` flag will send debugging information to STDOUT and/or to `syslog`. This provides a comprehensive log of NTP actions. Multiple `-d` flags will provide more detailed levels of debugging. Unless STDOUT is redirected, this produces a screen-full of output at start time and additional output every time the machine contacts or is contacted by another machine. While it can be inconvenient to restart the NTP daemon simply to perform debugging, using the `xntpd -d` option can sometimes be the fastest and most convenient way of tracking down a problem.

In addition to the above mentioned logging mechanisms, NTP provides its own logging mechanism. One advantage of the logs is they can be statistically analyzed to discover trends or problems. Appropriate statistics scripts are included with the open source distribution, but not with the distribution that comes with the Solaris OE. The logging options native to NTP are designed with optimizing clock accuracy in mind, so they are not entirely human-readable and using them to debug problems can be difficult. However, there are several ways that administrators can extract some useful debugging information from the logs. Some ideas for using the logs are discussed below.

Enabling Logging

In order to enable logging, a statistics folder needs to be specified in `ntp.conf`. The `statdir` directive indicates the directory name. The following line in `ntp.conf` sets up `/var/ntp` as the statistics directory:

```
statsdir /var/ntp
```

Once a statistics directory is setup, files can be setup to store the various NTP logs. The configuration in the `ntp.conf` file is simple. The `filegen` directive starts the line, followed by the type of log, and then name specifies the name of the file (in the `statsdir` directory). NTP logs are archived by day, week, month, or year, according to the `type` option. The `type` option also allows setting to none, which puts everything in the same file, or process ID (PID), which separates the files according to the invocation of the `xntpd` process. If the link option is used, then the file has a hard link that always keeps the same name and always points to the current archive. The `enable` keyword at the end of the line specifies that the log administration set is enabled (the other option is `disable`). This means that rotating logs are used and the most current log always has the same name. The logging directives are described in more detail in the `xntpd` man page.

The following example shows a `peerstats` log file entry in the `ntp.conf` file to archive weekly under the name `peerstatistics_log`. Because `enable` is set, a hard link allows easy access to the current archive.

```
filegen peerstats file peerstatistics_log type week enable
```

There are three different types of `filegen` directives allowed: `peerstats`, `loopstats`, and `clockstats`. Most system administrators will only find the `peerstats` file useful. The `clockstats` file is only useful on machines attached to a reference clock and the `loopstats` file is generally only useful if precision time is required. On a server with a reference clock, all three files should be enabled. Each type of `filegen` directive is described in the following subsections.

peerstats

Logging of peer statistics can be accomplished by adding the following to `ntp.conf`:

```
filegen peerstats file peerstats type week enable
```

Lines in the `peerstats` file look like the following:

```
52058 61147.923 129.148.1.229 9014 -0.001382 0.09590 15.87503
```

The following is a brief description of each field in the `peerstats` file:

- Fields 1-3

  The first field indicates the Modified Julian Day for the log entry. Since this is fairly useless to most mortals, you can determine the number of days since Jan 1, 2001 by subtracting 51911. The second field is a time stamp for the number of seconds into the day. The bit after the decimal place generally varies very little in a given invocation of `xntpd`, this is normal behavior because the routine will generally fall in the same portion of the second. The third field shows the IP address of the host in question.

■ Field 4

The fourth field represents the status of the host with the IP address in field three. This status is not meant to be human readable, but can be translated relatively easily. The fourth field actually contains the most important information to a system administrator. Scanning this field will allow you to quickly determine the configuration and status of all the NTP servers for a given client. The field consists of 4 hex digits. The first 2 hex digits specify the server configuration and how far it progressed through the clients clock selection process. The second two hex digits indicate the number of events and the type of the last event.

TABLE 1 shows common status codes and their meanings. The first digit specifies the configuration, reachability, and authentication status for the specified server. The second digit records how well a given server passed through the various stages of the clock selection algorithms. While knowing how the server passed through clock selection may seem like an entirely academic issue, this information is actually extremely useful for debugging problems. For instance, if a server has a status code of x1xx, the server's clock was outside of the largest possible error bounds of the other clocks, a condition which almost surely indicates that the server is set to the wrong time. If a server has a status code of x0xx, this could be due to a number of problems, since there are several sanity checks that could have failed (refer to the *NTP Algorithms* section). However, several common problems will trigger this status, including the server failing to authenticate, the server having a huge (over 16 seconds) error bound, or the configured server existing on a higher stratum number than the client. A full list of conditions that could cause the x0xx status code is in the *NTP Algorithms* section. A clock that has a status value of at least x2xx is probably correctly configured, though at least one server should have a status of x6xx.

TABLE 1 Status codes in `peerstats` log

Status Code Format	Meaning
1xxx	Server has sent peer synchronization requests to local machine, but is not configured locally
7xxx	Server is peer that is not configured locally but is reachable and using proper authentication
8xxx	Server is configured, but not authenticated or reachable
9xxx	Server is configured and reachable
Cxxx	Server is configured to use authentication, but not reachable
Dxxx	Server is configured to use authentication and is reachable, but is not using a trusted key
Fxxx	Server is authenticated as a trusted server and is reachable
x0xx	Server rejected by client (didn't pass sanity checks)

TABLE 1 Status codes in `peerstats` log

Status Code Format	Meaning
x1xx	Server passed the sanity checks, but wasn't close enough to other servers to survive the intersection algorithm
x2xx	Server passed the correctness checks (intersection algorithm)
x3xx	Server passed candidate checks (not discarded because there were too many (over 10) good servers.)
x4xx	Server passed through the clustering algorithms without being discarded as an outlyer having too much dispersion (useful, but not the best)
x5xx	Server would be the synchronization source, but is too far away, this means that all the other clocks are either insane or too far away also.
x6xx	Server is the current synchronization source

The third and forth hex digits in the status field indicate events on the client. The third digit indicates the number of events that have occurred since the last time an error was returned to the console by NTP or `ntpq` queried for events. This value does not wrap and stops incrementing at 15 (a hex F). For a properly running server, the value should be xx1x, unless `ntpq` has queried the server since startup, in which case the value would be xx0x. The event that should have occurred was the server becoming reachable. If the value is anything other than xx1x or xx0x, the system administrator should check for the event causing errors. This can be determined by looking at the fourth hex digit in the field, which indicates the last event recorded. TABLE 2 indicates the event codes.

TABLE 2 Server event codes in `peerstats` log

Event Code	Event Description
`xxx0`	Unspecified event (Either no events have occurred, or some sort of special error)
`xxx1`	An IP error occurred reaching the server
`xxx2`	Unable to authenticate a server that used to be trusted (indicates that the keys changed or someone is spoofing the server)
`xxx3`	A formerly reachable server is now unreachable
`xxx4`	A formerly unreachable server became reachable
`xxx5`	The server's clock had an error

The values between x7xx to xFxx and xxx6 to xxxF refer to reserved values for various flags and should not occur in normal usage.

- Fields 5-7

  The fifth field in the `peerstat` output shows estimated offset to that particular host (in seconds), which represents how far the client's clock appears to be off that of the listed server. The sixth field shows the round trip delay to that host (in seconds), and the seventh field shows the dispersion, also in seconds. The dispersion represents the maximum possible error on the estimated offset. In other words, if the server has the correct time, the difference between the clients time and the correct time must lie between the offset minus the dispersion and the offset plus the dispersion. This gives you a good idea how accurate a given client is.

`loopstats`

Logging of loop statistics can be accomplished by adding the following to the `ntp.conf` file:

```
filegen loopstats file loopstats type week enable
```

Lines in the loop stats file look like the following:

```
52058 62985.936 0.002807 34.2335 8
```

The first two fields are the same as for `peerstats` (date and time). The third field indicates the offset, how much time (in seconds) the clock will be adjusted by in the loop cycle. The forth field indicates the estimated frequency offset in parts per million, and the final field is a constant relating to the NTP control loop. The derivation of this value is very complex, and the interested should refer to RFC 1305. If this number is small, the control loop will adjust more quickly, if it is large, the control loop will adjust more slowly. A quickly adjusting control loop will have problems with wandering if the reference clock is unavailable, while a slowly changing clock will have trouble adjusting to jitter. The NTP algorithms work to set this value to whatever will produce the best overall accuracy.

clockstats

Logging of loop statistics can be accomplished by adding the following to `ntp.conf`.

```
filegen clockstats file clockstats type week enable
```

The `clockstats` file reports information on the clock driver. The `clockstats` directive will not accomplish anything unless the machine has a reference clock attached to it. If a reference clock is attached, the format of the data in the file will depend on the clock or clocks.

However, understanding the NTP algorithms will allow an administrator to glean some useful information out of the logs. This is described further below.

Using `ntptrace`

The `ntptrace` program is the easiest way to verify connectivity to an NTP subnet. This program sends an NTP packet to the specified server. If possible, the server will respond with an NTP packet. The host `ntptrace` was run on will use this information to determine the stratum, offset, and synchronization distance, as well as the next server in the hierarchy. Since `ntptrace` now knows the next server in the hierarchy, it sends that server an NTP packet. This continues up the hierarchy. If next server is not up, then a time-out results.

However, because of the above implementation, there are some problems with `ntptrace`. If the machine `ntptrace` is run on is not authorized to get time from one of the servers, `ntptrace` will show a time-out, even though the servers in the chain may have the necessary access permission. Because `ntptrace` uses normal NTP packets (mode 3), rather than NTP queries (mode 6), or control requests (mode 7), it is less likely that this will be a problem (since few sites restrict time requests).

The following is an example of `ntptrace` being run:

```
/home/ntp % ntptrace
localhost: stratum 4, offset 0.751836, synch distance 0.08391
hour.West.Sun.COM: stratum 3, offset 0.752687, synch distance 0.07349
minute.East.Sun.COM: stratum 2, offset 0.752287, synch distance 0.06305
second.East.Sun.COM: stratum 1, offset 0.755516, synch distance 0.03098, refid ’TRUE’
```

Monitoring with ntpq

NTP can be used to remotely monitor an NTP server. This feature can be disabled on the server by using appropriate access control as described in the second article in this series. Disabling this feature is a good idea in any moderately secure environment, because attackers can find out a great deal about a network and the machines on it by using NTP queries. The `ntpq` program sends out an NTP packet with mode 6. Mode 6 indicates that the packet is a query, rather than a time synchronization operation. This packet is a UDP packet with only one retry, so `ntpq` may work unreliably over large distances.

The `ntpq` command can be operated interactively, or directly from the command line. In interactive mode, the syntax to run `ntpq` on the host `bigben` from the host `tock` is as follows. Any `ntpq` subcommand (discussed below) can be entered at the `ntpq>` prompt. In this example, the `peers` subcommand is used.

```
tock# ntpq bigben
ntpq> peers
     remote           refid      st t when poll reach   delay   offset    disp
==============================================================================
 224.0.1.1       0.0.0.0         16 -     -   64    0     0.00    0.000 16000.0
*bigben.west.sun .TRUE.           1 u  596 1024  377    99.43    2.563    3.62
 badtime.north.s 0.0.0.0         16 u  472 1024    0     0.00    0.000 16000.0
ntpq> quit
```

The `ntpq` command can also run `ntpq` subcommands directly from the UNIX® command line. The following shows the `associations` subcommand run from the command line.

```
# ntpq -c associations
ind assID status  conf reach auth condition  last_event cnt
===========================================================
  1  1348  c000   yes    no
  2  1349  9614   yes   yes  none  sys.peer   reachable  1
  3  1350  8000   yes    no
```

A number of `ntpq` subcommands exist for `ntpq`. The most useful of these are peers and associations (displayed above). The `associations` command provides more information, but the `peers` command is more user friendly. These and some of the other useful `ntpq` subcommands are listed below. Refer to the `ntpq` man page for more information on `ntpq` subcommands.

Selecting a Host

A host can be selected by using the `host` subcommand. All other commands are run with respect to this host until the `host` subcommand is used again. To select the host `tintinnabulation` as the current host, the following command can be used.

```
ntpq> host tintinnabulation
current host set to tintinnabulation.sun.com
```

An error message is returned if a host cannot be found. If no host is set on the command line or with the host subcommand, the local machine is assumed to be the host.

If authentication is being used, then `ntpq` will prompt for a `keyid` and password whenever a restricted action is attempted. These could be specified on the command line, but that is a security risk and should be avoided. Note that the Solaris OE version of `xntpd` only restricts actions which actually change the server state. Subcommands that request data can be run freely, even if authentication is on. Because data obtained with NTP requests could be useful to someone trying to gain unauthorized access to a machine, it is important to control access to the server in any sort of secure environment. This can be accomplished by following the recommendations in the second article in this series.

If the prompts are filled out incorrectly, the `keyid` subcommand can be used to specify the key identifier and the `passwd` subcommand can be used to specify the password.

The `peers` Subcommand

The `peers` subcommand displays most of the information that a system administrator would find useful with regards to NTP. Each server for the current host is listed, with information related to that server in the succeeding fields. The second field lists the servers current time source. A value of `.TRUE.` indicates that the server is attached to a reference clock and a value of `0.0.0.0` indicates that this information was not available, usually indicating the server is not synchronized. The third field indicates the stratum of the server. The fourth field indicates the mode of the server. The values are `l` for local (a reference clock), `u` for unicast (non-broadcast server), `b` for broadcast, and `m` for multicast. The fifth field indicates the number of seconds since the given server was last reached, and the sixth field indicates the polling interval for a given host in seconds.

The fifth and sixth fields allow bored system administrators to pass the hours waiting to see if `xntpd` actually receives packets at the end of each polling period. The seventh field indicates the history of reachability. This value is an 8 bit long bit stream that is set to 1 if the server is reached during a polling period and 0 otherwise. It is displayed in octal, so it maxes out at 377, which indicates that the last 8 attempts were good. The final three fields indicate the estimated delay, offset, and dispersion in milliseconds. The following is an example of the information provided by the `peers` subcommand:

```
ntpq> peers
.ahau.mex.sun.co  matins.hours.su   4 u   88  512  377    52.28  -15.444   11.23
-imix.palenque.s  lauds.hours.sun   4 u    -  128  376    82.72   33.854    9.02
+ik.tikal.sun.co  prime.hours.sun   4 u  656  512  324    82.44  -10.320    7.34
.akbal.tulum.sun  terce.hours.sun   4 u  721 1024  377   132.80  -26.504    9.37
-kan.sayil.sun.c  sext.hours.sun.   4 u    -   64  376    72.14  -59.963    2.18
+chicchan.bonamp  nones.hours.sun   4 u    -   64  333    74.94    0.308    0.85
 cimi.seibal.sun  compline.hours.   4 u    3  256  171    76.00   -2.084 8000.53
+manik.uxmal.sun  vespers.hours.s   4 u    -   64  336    75.87    7.717    1.16
 lamat.uxmal.sun  compline.hours.   4 u    -   64    2     0.00    0.000 16000.0
 muluc.tikal.sun  0.0.0.0          16 -    -   64    0     0.00    0.000 16000.0
 oc.palenque.sun  0.0.0.0          16 -    -   64    0     0.00    0.000 16000.0
 224.0.1.1        0.0.0.0          16 -    -   64    0     0.00    0.000 16000.0
+chuen.tikal.sun  lauds.hours.sun   4 u  664 1024  377    56.21   -0.620    5.17
*eb.copan.sun.co  prime.hours.sun   3 u  639 1024  377    54.76   -1.608    3.07
xben.palenque.su  lauds.hours.sun   3 u  459 1024  367    80.84  -208.84  100.78
+ix.caracol.sun.  prime.hours.sun   3 u  682 1024  377    92.48    1.055    4.14
+men.yaxchilan.s  terce.hours.sun   4 u   28  128  352    67.50   -6.308    1.80
.cib.copan.sun.c  terce.hours.sun   4 u   52   64  174    68.45   -1.023  126.74
 caban.tulum.sun  nones.hours.sun   5 u    -   64  237    61.26   -2.177  500.79
-etznab.seibal.s  compline.hours.   4 u  602  512  372    72.02  -24.098    6.68
```

The characters in front of the servers indicate how `xntpd` is using the listed server. A full description exists in the `ntpq` man page. The characters ”.”, “-”, and “+”, indicate healthy servers, provided that at least one server in the configuration has a “*” or “o” character to indicate that it is a current synchronization source. A “#” character indicates that the listed server, and probably all of the other servers in the configuration are too far away to be useful. A space indicates a server that has failed its sanity checks, which can indicate a reachability problem, an authentication problem, or other issues (see *Sanity Checks*). An “x” in front of the server name indicates that the servers clock is outside of the error bounds for the correct time, as determined by the majority of the other clocks. This generally means the clock is unsynchronized or set incorrectly.

The `associations` Subcommand

The `associations` subcommand lists the status of the servers. The order is the same as the output from the `peers` subcommand above, unless clocks have entered or left the configuration since the `peers` subcommand was run. The `associations` subcommand gives exactly the same information as the in the `peerstat` log. The associations subcommand displays the last logged status for each machine. The string in the status field can be interpreted the same way as the string in the peerstats log. The fields following the status field break down the status into its component parts. The following is an example of the information provided by the associations subcommand:

```
ind assID status  conf reach auth condition  last_event cnt
===========================================================
  1 14446  1214    no   yes  none eliminate   reachable  1
  2 14585  1314    no   yes  none   outlyer   reachable  1
  3 15054  1414    no   yes  none   synchr.   reachable  1
  4 15373  1214    no   yes  none eliminate   reachable  1
  5 15456  1314    no   yes  none   outlyer   reachable  1
  6 15459  1414    no   yes  none   synchr.   reachable  1
  7 15482  1014    no   yes  none    insane   reachable  1
  8 15485  1414    no   yes  none   synchr.   reachable  1
  9 15513  1014    no   yes  none    insane   reachable  1
 10 45852  8000   yes    no
 11 45853  8000   yes    no
 12 45854  8000   yes    no
 13 45855  94f4   yes   yes  none   synchr.   reachable 15
 14 45856  96f4   yes   yes  none  sys.peer   reachable 15
 15 45857  91f4   yes   yes  none falsetick   reachable 15
 16 45858  94f4   yes   yes  none   synchr.   reachable 15
 17 45859  94f4   yes   yes  none   synchr.   reachable 15
 18 45860  92f4   yes   yes  none eliminate   reachable 15
 19 45861  90f4   yes   yes  none    insane   reachable 15
 20 45863  93f4   yes   yes  none   outlyer   reachable 15
```

The first two fields are the index and the association ID. Both can be used to specify the server for other ntpq subcommands. For instance, to read the internal variables for the first server in the preceding example could be accessed using either of the following two subcommands:

```
ntpq> readlist &1
```

```
ntpq> readlist 14446
```

Reading NTP variables

NTP keeps track of many internal variables that can be tracked using the ntpq command. There are several subcommands which enable you to check these variables. The most useful of these are described below.

The `clockvar` subcommand lists the type of clock used by a given server. This can be used to determine if a stratum 1 server is connected to a real reference clock, or simply (improperly) using its own clock as a reference clock. If the server's internal clock is used, the device listed will be "`Undisciplined local clock`". If the server is not connected to a reference clock, the error "`***Association ID 0 unknown to server`" will be returned. The following shows the `clockvar` subcommand run on a stratum 1 server.

```
ntpq> clockvar
status=0000 clk_okay, last_clk_okay
device="Kinemetrics/TrueTime Receiver", timecode="214:01:22:06 ",
poll=67043, noreply=0, badformat=0, baddata=0, fudgetime1=8.000,
stratum=0, refid=TRUE, flags=0
```

As shown in the following example, the `readvar` subcommand lists more information about the server (if no association ID is specified) or information about the last packet received from a host (if an association ID is specified). Most of this information can be found elsewhere, so this is not particularly useful; however, some people may prefer this view of the data.

```
ntpq> readvar
status=06f4 leap_none, sync_ntp, 15 events, event_peer/strat_chg
system="SunOS", leap=00, stratum=4, rootdelay=250.21,
rootdispersion=184.71, peer=49327, refid=129.152.1.16,
reftime=bf132e78.38f3c000  Wed, Aug  1 2001 18:42:16.222, poll=6,
clock=bf132eb0.8621d000  Wed, Aug  1 2001 18:43:12.523, phase=-5.033,
freq=84142.38, error=14.95
ntpq> readvar 15054
status=1614 reach, sel_sys.peer, 1 event, event_reach
srcadr=192.168.96.46, srcport=123, dstadr=192.168.96.30, dstport=123,
keyid=0, stratum=3, precision=-15, rootdelay=91.81,
rootdispersion=80.22, refid=192.168.94.14,
reftime=bf133146.1c2ef000  Wed, Aug  1 2001 18:54:14.110,
delay=    1.02, offset=    1.48, dispersion=0.90, reach=377, valid=8,
hmode=2, pmode=1, hpoll=8, ppoll=10, leap=00, flash=0x0<OK>,
org=bf133164.1ce29000  Wed, Aug  1 2001 18:54:44.112,
rec=bf133164.1ca36000  Wed, Aug  1 2001 18:54:44.111,
xmt=bf133143.e8f62000  Wed, Aug  1 2001 18:54:11.910,
filtdelay=    1.02    1.04    1.74    0.76    1.86    1.80    0.72    1.68,
filtoffset=   1.46    1.05    1.72    0.84    1.27    1.30    0.69    1.10,
filterror=    0.52    4.43    8.33   12.24   16.14   16.14   18.10   20.05
```

Using `xntpdc`

For the most part, the `xntpdc` command is used to remotely reconfigure hosts running NTP. The `xntpdc` command can set nearly all of the values present in the `ntp.conf` file. However, changes made with `xntpdc` are not persistent, so those changes will be lost when the machine reboots. Because of this, reconfiguring NTP servers using `xntpdc` has limited use. However, `xntpdc` can also be used to monitor servers. In some cases, the information found in `xntpdc` can be more useful than the information found using `ntpq`.

Like `ntpq`, `xntpdc` uses the `host`, `keyid`, and `passwd` subcommands to choose a host on which the later subcommands will be run. The `peers` subcommand works similarly to the `peers` subcommand in `ntpq`, although the output is slightly different. The `xntpdc` command does not use the `associations` subcommand.

Some `xntpdc` subcommands use the IP address of a server or peer to specify information about it. Because the information about that server or peer is stored locally on the current host machine, only IP addresses listed in the current "peers" list will work. Any other IP addresses will result in a "`***Server reports data not found`" error. The `pstats` subcommand is one example of a command that uses this format, the syntax is as follows:

```
xntpdc> pstats 192.168.10.123
remote host:          192.168.10.123
local interface:      192.168.12.30
time last received:   92s
time until next send: 106s
reachability change:  9310s
packets sent:         56
packets received:     64
bad authentication:   0
bogus origin:         1
duplicate:            0
bad dispersion:       33
bad reference time:   0
candidate order:      2
```

This subcommand can provide some useful statistics on the server for a given host. Probably the most useful information here is the information on bad authentication. This is an easy way to find out if authentication is failing between two servers. The percentage of packets sent that had a bad dispersion can provide a clue as to the variability of the network connection. The candidate order indicates how the server is sorted in the clustering algorithm.

Various `xntpdc` subcommands also provide a wealth of other information about the server. While most of this information is not very useful for troubleshooting, except if the time needs to be extraordinarily accurate, it can be interesting (and amusing) to try some of the other `xntpdc` monitoring commands. The `kerninfo` subcommand provides kernel time statistics, errors estimates, stability estimates, and lists the precision. The `loopinfo` command lists the predicted offset and the degree of frequency drift. The `sysstats` command can provide information on the number of packets processed and the uptime of the `xntpd` process. It also indicates packets with bad packet lengths. A bad packet length might indicate a (possible malicious) malformed packet.

Summary

This article discussed the inner working of NTP in some detail, with the goal of providing enough of an understanding that a reader could understand the process well enough to trouble-shoot it. In addition, several tools for troubleshooting were discussed, along with helpful hints on how to interpret the output of certain commands. The reader should walk away with an understanding on NTP internals and a solid ability to quickly and easily troubleshoot even complex NTP problems.

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[tm] 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.