Using NTP to Control and Synchronize System Clocks - Part II: Basic NTP Administration and Architecture

---

By David Deeths - Enterprise Engineering and Glenn Brunette - SunPS

Sun BluePrints™ OnLine - August 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.: 816-0092-10
Revision 01, 08/23/01
Edition: August 2001

Using NTP to Control and Synchronize System Clocks - Part II:
Basic NTP Administration and Architecture

This is Part 2 of a three-article series that discusses how to use Network Time Protocol (NTP) to synchronize system clocks. This article explains the basics of client and server administration, covering various client/server configurations, as well as authentication and access control mechanisms. This article also provides a number of suggestions for an effective NTP architecture. Throughout this article, emphasis is placed on architecting a well-planned, secure, efficient, and accurate NTP solution.

The first article in the series covered basic NTP and time synchronization concepts. It is suggested reading for system managers or system administrators who don't already understand basic NTP principles like strata, reference clocks, slewing, and server types (broadcast, non-broadcast, and peer). The first article also set expectations for NTP resource usage and accuracy.

The final article in the series will discuss the internal algorithms of NTP and also provide tips on monitoring and troubleshooting NTP installations.

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.

As discussed in the first article of this series, an NTP client operates by communicating with one or more NTP servers. The NTP system is setup as a hierarchy, and each level of the hierarchy is called a stratum. The “trunks” of the hierarchy are the low number strata (strata 0 and 1), while the leaves are the higher numbered strata (3,4, 5, etc.). The NTP daemon, `xntpd`, is responsible for adjusting a machine’s clock by periodically communicating with servers on lower stratum numbers.

Communication between machines running NTP uses several different modes. In broadcast mode, a broadcast server sends periodic broadcast NTP packets to any broadcast clients configured to listen. In non-broadcast mode, an NTP client is configured to send periodic time queries to one or more NTP servers. In peer mode, two or more machines running NTP each determine which machine among them is the most accurate, and synchronize to it.

Basic NTP Configuration

The NTP configuration file contains the most important configuration data for the NTP daemon. This file is read whenever the daemon is started. By default this file is found at `/etc/inet/ntp.conf`, and will be referred to as the `ntp.conf` file through out the rest of this article. A few other files contain some less important configuration data. For instance, the drift file contains information on the rate of drift of a client clock, and the key file contains the cryptographic keys. Besides the configuration file, configuration changes can be made remotely using the `ntpq` and `xntpdc` programs. Simple configurations can also be specified at the command line. This section only covers basic configuration, so unless otherwise specified, the examples shown affect only the `ntp.conf` file.

Basic Client Configuration

An NTP client can have a number of servers, and broadcast and non-broadcast servers can be used by the same client. NTP clients synchronize their time to match NTP servers, while NTP servers never synchronize their time to match NTP clients. However, recall that NTP clients can also be NTP servers to clients of their own. There are several points to consider for various client configurations.

Note – Normal NTP clients are set up with the *server* keyword, while broadcast and multicast clients are setup with the *broadcastclient* and *multicastclient* keywords. There is no *client* keyword.

Setting up a computer as an NTP client requires simply adding several lines like the following to the `ntp.conf` file and restarting `xntpd`.

```
server address1
server address2
server hostname3
```

In this case *address1*, *address2*, and *hostname3* are the addresses (or host names) of three NTP servers, which the client can use as potential synchronization partners. It is always best to synchronize with multiple servers to help protect the client from an incorrect or inoperational server. This is important, since in many environments, it is unlikely that an NTP server failure will be noticed promptly.

The client servers should be as independent as possible. The dependency chain of an NTP server can be determined by running the `ntptrace` command with each server as the argument. If possible, servers should not share common NTP parents.

A server can be set to be preferred by use of the `prefer` keyword after the server name or address. A preferred server will be used as a synchronization source if it meets a minimum accuracy level, even if there are other more accurate servers. However, if a preferred server is far outside of the accuracy bounds determined by consulting other servers, it will be discarded.

In general, a server will not fall outside these accuracy bounds unless it or the majority of other servers are misconfigured. Using preferred servers allows setting up clients to prefer a clock that is known to be very accurate. In the following example, the client is setup to prefer the host stonehenge.

```
server stonehenge prefer
server machupicchu
server kukulcan
```

The time exchange between the client and server can also be protected with an authentication key, as described briefly in the “Basic Authentication” section.

Broadcast and Multicast Clients

Clients can also be configured to respond to broadcast or multicast packets sent by a broadcast or multicast NTP server. Note that no additional steps are needed to allow a node running `xntpd` to be a non-broadcast NTP server (assuming access and authentication are set appropriately). However, an NTP node needs to be explicitly configured in order to send out broadcast or multicast packets for clients. Setting up a broadcast or multicast server is described in the next section. Note that, since the clients are listening for broadcast packets, no server is specified after the keyword.

Broadcast and multicast are generally used as the primary server on a LAN or subnet. Using broadcast or multicast allows synchronizing a large number of clients without creating large amounts of NTP traffic. In addition, servers can be changed easily, since clients do not listen for a specific server when they are in broadcast mode. Because the links are mostly local, this allows accurate synchronization for the clients. The important NTP servers within an enterprise (generally low stratum numbers) and any servers separated by large distances or low latency links should use non-broadcast mode in order to maintain close synchronization

A computer can be configured to be a broadcast client by adding a line with `broadcastclient` to the `ntp.conf` file as shown in the following example.

```
broadcastclient
```

A client synchronizes with any server which sends out broadcast NTP messages, so it is important to use authentication when configuring broadcast clients, as described in the “Basic Authentication” section.

Configuring a multicast client is accomplished the same way, but the line in `ntp.conf` begins with the entry, `multicastclient` instead, as shown in the following example.

```
multicastclient
```

The reserved multicast address for NTP is 224.0.1.1, and this is the default multicast address. Other addresses can be specified after the `multicastclient` keyword, which is suggested for large configurations. Note that the server and client must be configured to use the same multicast address.

If the clients are not configured to synchronize with a specific group of servers, a rogue system can influence the synchronization process by broadcasting invalid time information. To defend against this threat, authentication and access control should be used to help limit potential synchronization sources. This is described in the “Basic Authentication” section.

Basic Server Configuration

No special configuration is required for a machine with a running NTP daemon to be used by other network nodes as a *standard* server (as opposed to a broadcast server or peer). However, access control is needed to prevent a machine from acting as an NTP server to clients. Further, operating as a broadcast server or a peer server involves additional configuration.

Setting up a peer can be accomplished by adding the `peer` command to the `ntp.conf` file. The configuration of a peer is basically the same as setting up a client: an address or host name needs to be specified, along with a key and possibly the `prefer` keyword. Peers also have an associated polling interval which can be set in the `ntp.conf` file. While a set of peers can use different polling intervals, true peers use the same polling interval. The defaults should be acceptable except when peers are connected by very slow links. Setting the polling range is described in the `xntpd` man page.

Generally, peer connections are used to improve the time accuracy at the base of the NTP tree (low numbered strata), or provide additional redundancy at the leaves of the NTP tree (high numbered strata). Using peer connections allows both of these without resorting to creating a new level of hierarchy.

The following shows the configuration of a set of peers:

```
peer notredame
peer chartres
```

Configuring a broadcast or multicast server is slightly different. Both broadcast and multicast servers are defined using the `broadcast` command in the `ntp.conf` file. The broadcast address is specified after the `broadcast` command. Broadcast addresses are generally the subnet address with the host field set to maximum value (all binary ones). The official multicast address for NTP is 224.0.1.1, though other multicast addresses can be used. Authentication keys can (and should) be used. For broadcast servers, a time-to-live (TTL) value (the maximum number of hops) for the NTP packets should also be specified. This value defaults to 127, but should be changed to something appropriate to the network. It is important that the packets survive long enough to reach all the clients, but if long-lived packets are necessary, it may make more sense to add an additional stratum of servers in the appropriate subnets.

The following shows the configuration for a broadcast server on subnet 192.72.23.0:

```
broadcast 192.72.23.255 ttl 6
```

The following shows the configuration for a multicast server. Note that the configuration still uses the `broadcast` keyword, but the address has changed to a multicast address. Any multicast address could be used.

```
broadcast 224.0.1.1 ttl 6
```

As a rule, authentication should always be used with broadcast or multicast clients. This is not just a security measure, but also prevents an accidentally configured broadcast server from disrupting the client's time synchronization. This is described in the "Basic Authentication" section.

Key Administration

The NTP service provides the capability for NTP clients and servers to authenticate each other. This is accomplished with *symmetric* authentication keys and key identifiers. The term symmetric means that the keys must be the same on both the client and the server. Because NTP keys are stored outside of the `ntp.conf` file, the NTP keys file must be specified in the `ntp.conf` file for any configuration that will use keys. This is accomplished using the `keys` keyword, followed by the absolute path to the file. An example of this addition to `ntp.conf` is shown below.

```
keys        /etc/inet/ntp.keys
```

With NTP version 3, authentication keys must be manually distributed to each of the client systems (NTP version 4 can use an automatic public key distribution, which is fully described in the NTP version 4 documentation). Caution must be exercised when transferring these keys to each client system. Be sure to use a protocol that supports strong authentication and encryption.

The key file consists of the following three columns, which are outlines in TABLE 1

TABLE 1 Content of the Three Columns of the `ntp.key` File

Field	Content
Identifier or key number	The identifier must be a positive integer in the range: 1 to 2^{32}-1. Each entry in the NTP keys file must have a unique identifier.
Key type	The key type is a single character that determines which algorithm will be used to compute the digital signature for an NTP transaction. This signature, once verified by a peer, authenticates the transaction. The available algorithms are ASCII, DES and MD5. Each of the options will be discussed in more detail below.
Key data	The key data represents the actual shared secret that will be used as input to the algorithm specified by the key type parameter. The actual format of the data depends on the algorithm selected.

NTP version 3 supports four different key types: 'S', 'N', 'A', and 'M'. These key types allow an organization to select the type that most suits their needs and policies. Each of these types, and their respective key data requirements are listed below.

- *Key Type 'S'* – 64-bit hexadecimal number in DES (standard) format

  In this format, the high order 7 bits of each octet are used to form the 56-bit key while the low order bit of each octet is given a value such that odd parity is maintained for each octet Leading zeros must be specified so that the key will be exactly 16 hexadecimal digits long and will have an odd parity.

  For example, a zero key, in standard format, would be given as:

  ```
  0101010101010101
  ```

  The following key is comprised of the sequence '12345678' embedded in the standard format while still maintaining odd parity for each octet.

  ```
  020407080b0d0e10
  ```

- *Key Type 'N'* – 64-bit hexadecimal number in NTP format

  This format is the same as the 'S' type format, described above, except the bits in each octet have been rotated one bit right so that the parity bit is now the high order bit of the octet. Again, leading zeros must be specified and odd parity must be maintained.

  For example, a zero key, in NTP format would be specified as:

  ```
  8080808080808080
  ```

  Consequently, the sequence '12345678' embedded into a key in NTP format would result in the following sequence.

  ```
  0102830485860708
  ```

- *Key Type 'A'* – 1-8 character ASCII string

  In this format, the key is created by using the lower order 7 bits of the ASCII representation of each character in the supplied string. Zeros are added on the right when necessary to form a full width 56 bit key.

  For example, a small string can be used, such as:

  ```
  trueTime
  ```

- *Key Type 'M'* – 1-8 character ASCII string using MD5

  This format is similar to the 'A' type above, except that the MD5 algorithm is first applied to the ASCII character string before the key is computed.

The following is a sample key file for a system:

```
# cat /etc/inet/ntp.keys
1 M tick
3 M tock
9 M truetime
```

This key file indicates that key identifier 9 uses authentication based on a message digest (or digital signature) computed using the MD5 algorithm. The string `truetime` is the actual authentication key. The key file should be owned and readable only by the root user.

Specifying a key file doesn't actually make the keys usable. In order to specify what keys can be used for what purposes, directives must be added to the `ntp.conf` file. In this article, only time synchronization will be considered. Keys can also be specified to be trusted for the purposes of monitoring and remote administration. This is covered in the `xntpd` man page.

To specify a key as trusted for time synchronization, include the `trustedkey` directive, along with the key number. Multiple key numbers can be included if they are separated by a space. Once a key is specified as trusted, the machine will trust requests for time synchronization or replies to time synchronization requests that use that key. A client without a trusted key configured will not trust the response from a server, while a server without a trusted key configured will ignore the client and never respond to it.

Adding the following to `ntp.conf` will declare keys 1 and 3 as trusted.

```
trustedkey 1 3
```

Basic Authentication

Establishing authenticated communication between a client and server requires configuration on both the client and the server. In order for authentication to work, both the client and the server must have a `keys.conf` file specified in `ntp.conf` which contains the same key with the same key ID. In other words, both the client and the server should have a line in the `keys.conf` file that is identical.

Because the client/server configuration is reversed for broadcast and non-broadcast, it is best to think of the configuration in terms of the initiating machine and the non-initiating machine. Non-broadcast *clients* initiate a transaction with the non-broadcast server. Broadcast *servers* initiate a transaction with the broadcast client.

The initiating party for the transaction should have a key specified on the `ntp.conf` line which defines the relationship. The `trustedkey` directive also needs to exist for each key used so that the client will trust the encrypted replies from the servers. Thus, the following would be the configuration for a non-broadcast client authenticating to the servers `stonehenge` and `machupicchu` using key 1 and `kukulcan` using key 4.

```
keys /etc/inet/ntp.keys
server stonehenge prefer key 1
server machupicchu key 1
server kukulcan key 4
trustedkey 1 4
```

For a broadcast configuration, the server is the initiator of the transaction. This means that for a broadcast server to authenticate with a broadcast client, a key should be specified on the `broadcast` line in the servers `ntp.conf` file. The following configuration will broadcast packets on subnet 192.72.23.0 to be authenticated with key 10.

```
keys /etc/inet/ntp.keys
broadcast 192.72.23.255 key 10 ttl 6
```

Multicasting works the same way, but with a multicasting address:

```
keys /etc/inet/ntp.keys
broadcast 224.0.1.1 key 12 ttl 6
```

For broadcast and multicast servers, in order for the server to respond to clients during the initial interchange where clients try to judge the network latency, the server must include a `trustedkey` directive. This is not necessary for broadcasting to work properly, but allowing clients to communicate with the server in the initial exchange does increase their accuracy.

As for peers, they are all potential initiators, so they all use the same syntax. All keys used also need to be declared as trusted:

```
keys /etc/inet/ntp.keys
trustedkey 1 2
peer notredame key 1
peer chartres key 2
```

For the non-initiating machine in an NTP transaction the only necessary configuration is using the `trustedkey` keyword to specify keys that will be trusted. However, in order to assure that *only* authenticated packets are accepted, the clients should also use the `enable auth` directive. If the `enable auth` directive is not used, authenticated packets will work, but so will unauthenticated packets. Since this defeats the purpose of authentication, the `enable auth` directive should always be used.

When a transaction is initiated, the initiating machine uses the appropriate key to encrypt a checksum of the packet. If the same trusted key is used by the non-initiating machine, the machine will be able to verify the correctness of the encrypted checksum. Otherwise, it will ignore the packet. That means that a non-broadcast *server* will only listen to non-broadcast clients communicating using one of its trusted keys and a broadcast *client* will only listen to broadcast servers using one of its trusted keys.

The trusted keys are a subset of all the keys defined in the keys file. The trusted key mechanism permits a simple key installation and revocation process. Only keys that have been declared trusted can be used for authenticating synchronization attempts.

The following are brief examples of configuring authentication for non-initiating machines.

In the following example, key identifier 9 has been declared as trusted.

```
# cat /etc/inet/ntp.conf
server 192.168.1.254 prefer

enable auth
keys        /etc/inet/ntp.keys
trustedkey  9
```

Clients attempting to synchronize with this server using key identifier 9 will be allowed to (assuming their keys match). Other clients will be ignored.

Note that this example also uses a server (192.168.1.254) for which there is no authentication being used. The `trustedkey` statement doesn't affect this server, because the local machine is the initiator for a client/server transaction. Since it

didn't initiate the session with authentication, it doesn't expect authentication on the reply. Such a setup where authentication is required for lower levels, but not for upper levels is most often the case for NTP services on the border of a network where time is being synchronized with external systems owned by another party. This example is intended to show how authentication can still be done within a company's network even if time is being synchronized to external parties. Keep in mind that external time sources could be insecure. It is always best to use multiple border clients each receiving time from a different external source. Authentication should be used where possible.

To complete this process, the following provides an example client configuration that uses authentication with key identifier 9 to communicate with the above server.

```
# cat /etc/inet/ntp.conf
server 192.168.1.1 key 9
trustedkey 9
enable auth
keys        /etc/inet/ntp.keys
```

The `/etc/inet/ntp.keys` file must contain an authentication key with an identifier 9 and a value `truetime` (or whatever was set in the server's `ntp.keys` file) for the authentication to succeed.

To verify the state of our server, the `ntpq` command is used as follows.

```
# ntpq
ntpq> associations
ind assID status  conf reach auth condition  last_event cnt
===========================================================
  1 13668  9614   yes   yes  none  sys.peer   reachable  1
```

Note that the word "none" occurs under the `auth` column. This indicates that authentication is not being used to communicate with the server with index 1 (under the `ind` column). The `pstatus` command can be used with the association ID at the `ntpq` prompt to determine the hostname or IP address of the server (in this case,

192.168.1.254). Since ntpq shows that authentication is not being used, this is consistent with our configuration as defined above. Similarly, on our client, the following information was obtained from ntpq.

```
# ntpq
ntpq> associations
ind assID status  conf reach auth condition  last_event cnt
===========================================================
  1 12940  f614   yes   yes   ok   sys.peer   reachable  1
```

Since the word “ok” appears under the auth column here, the client has synchronized with the server with index 1 (192.168.1.1) and that authentication was successful. The fact that the client lists auth as “ok” indicates that the configuration for both the client and server was correct and the key data matched.

In order to assure that a broadcast client only listens to authenticated packets, the enable auth keyword must be used and a trusted key must be configured on the client. The following example will configure a client to only listen to multicast packets from a server using multicast encryption with key 12.

```
keys /etc/inet/ntp.keys
enable auth
trustedkey 12
multicastclient
```

Note that authentication can be used for more than just time synchronization for peers. Authentication can also be used with the ntpq and xntpdc commands. For more information on this, refer to the xntpd man page.

Protecting the NTP Configuration Files

All of the NTP configuration files (ntp.conf, ntp.keys, ntp.drift, and the statistics files) should be owned and readable only by the root user. While only the ntp.keys file contains sensitive data, the others could be reconfigured by an attacker to cause problems with the server. The following example outlines the permissions that should be used for the ntp.conf and ntp.keys files:

```
-r--------   1 root     root         415 Apr 15 02:40 ntp.conf
-r--------   1 root     root          33 Apr 15 02:41 ntp.keys
```

Prior to Solaris™ 8 Operating Enviornment (Solaris OE), the default file creation mask for daemons started using run-control scripts was 0. As a result, files created by those daemons have world (and group) readable and writable permissions. Since NTP can be configured to create files such as the drift file or as a result of statistics collection, it is important that NTP use stronger permissions. For information on how to set the default file creation mask to a stronger value, refer to the Sun BluePrints™ OnLine article *Solaris™ Operating Environment Security* (there is an updated version for Solaris 8 OE).

Another option is to edit the NTP startup script (`/etc/init.d/xntpd`) and enter the `umask` command as the first entry. For instance, to enforce a default `umask` of 022 for all files created by NTP, simply add the following to the top of the file:

```
umask 022
```

Note that, unlike the method described in the *Solaris™ Operating Environment Security* article, using this method will not affect other services started at system boot time, so they may still use a possibly incorrect file creation mask.

Suggestions for NTP Architecture

This section discusses the basic guidelines for architecting an NTP solution, in the following sections:

- Limiting single points of failure (SPOFs) and maximizing independence
- Controlling network impact
- Enabling access control
- Selecting appropriate reference clocks

Limiting SPOFs and Maximizing Independence

Single points of failure can be reduced by assuring that client servers are as independent as possible. Using a number of independent servers reduces the effectiveness of an incorrectly configured server spoofing the time, and thus increases security. Verifying NTP server independence can be difficult if the server configuration is incoherent. To effectively map the dependencies on an NTP subnet, each of the peers and servers must be mapped. This quickly becomes unwieldy in a large configuration. A more workable solution is to use `ntptrace` to determine the hierarchy of time sources used by a client. It can then be easily identified if two machines share common time servers. This is less ideal, because it only shows the

currently used time source, as opposed to all configured peers. Regardless, a client should always receive time from at least 4 servers. This will reduce the chances of it losing synchronization when a server fails. If fewer than four servers are used, the agreement algorithm cannot reliably detect a clique including a majority of trusted sources. An easy solution is to use three servers from a lower stratum number and one unrelated peer from the same stratum.

Controlling Network Impact

It is important for NTP architects to understand how NTP and the network relate. The goals of an NTP architect are two-fold: to limit NTP's network activity and increase the accuracy of the clocks. To achieve high clock accuracy, the network latency needs to be low. NTP can achieve a high level of accuracy and remain a good network citizen if local NTP servers are used and NTP servers use the appropriate modes.

The easiest way to increase accuracy in an NTP configuration is to reduce the latency between the connections by putting NTP servers on the same LAN as their clients. If a LAN is very large, it is a good idea to have multiple servers in different geographic or network segments. Reducing the latency of a single NTP connection does not necessarily reduce the overall network latency. This is because the latency from the root time source could be increased by putting an additional server in the way, even though the latency to that server is reduced. However, if several independent servers are used, the NTP clock selection algorithms will probably help mitigate the effects of any increased latency. Another advantage to using local servers is that they tend to reduce the load on the WAN, though NTP is unlikely to be a big source of network load.

Another way of reducing NTP traffic, while keeping clock accuracy, is to use appropriate server modes. Central servers (generally stratum 1 and 2 servers) should use non-broadcast server/client mode or peer mode, which allows more accurate time distribution. These servers are generally geographically distributed; therefore, the accuracy of the time distribution is critical.

Broadcasting over high latency links can lead to very inaccurate time, both because of the latency and because it is likely the latency will be variable and unpredictable. Using broadcasting or multicasting over relatively local connections is acceptable. In fact, for a local server with a large number of clients and a fairly constant network latency broadcasting or multicasting is likely to be nearly as accurate as using a non-broadcast server. Multicasting is preferable to broadcasting because it makes identifying NTP traffic easier and does not affect non-NTP clients on the network.

Broadcasting or multicasting is a good fit in some environments, however, it is not appropriate for all environments. In particular, architectural or security concerns may preclude the use of broadcasting or multicasting. Broadcasting and multicasting are less acceptable than non-broadcast NTP transactions in many organizations due

to the impact of these choices on their security architecture. Many companies are not willing to permit broadcast traffic through their firewalls. The use of broadcast or multicast NTP transactions also opens the network to possible denial of service attacks. Architectural or managerial constraints could also prevent the use of broadcasting and multicasting.

Enabling Access Control

For security conscious environments, monitoring should be turned off because it could allow an attacker to obtain sensitive information about hosts or networks. In most other environments, disabling monitoring is an unnecessary restriction and only makes it more difficult to solve NTP problems. Requiring authorization keys for queries is another option. Limiting access to NTP queries prevents intruders from probing for information using the `ntpq` command. While the information obtained from `ntpq` may seem trivial, an intruder could discover sensitive information, including network delays (which could lead to determining network architecture), hostnames, IP addresses, and OS versions.

In addition to authenticated transactions, NTP also provides the capability to restrict access to its services. This function is provided using the `restrict` keyword. This keyword is defined in the `ntp.conf` file and has the following syntax.

```
restrict address [ mask numeric_mask ] [ flag ] [ ...  ]
```

The address and mask are both dotted octet representations of the IP address and network subnet mask to be restricted. The flags indicate what function is to be controlled. For example, if all communication from IP address 192.168.1.2 is to be ignored, the following access restriction can be used.

```
restrict 192.168.1.2 ignore
```

The `restrict` command is often used with the `default` keyword, which limits the specified access from all IP addresses.

```
restrict default ignore
```

After this default policy of denial, additional `restrict` statements can be used to increase the access. Any `restrict` statements which do not contain a keyword will enable (rather than restrict) access. For example, to only accept requests from the 192.168.1.0/24 network (the 24 indicates 24 bits of netmask), the following commands could be used.

```
restrict default ignore
restrict 192.168.1.0 mask 255.255.255.0
```

The following syntax can be used to further refine this configuration if, for example, no system on the 192.168.1.0/24 network is permitted to modify the run-time configuration of this server.

```
restrict default ignore
restrict 192.168.1.0 mask 255.255.255.0 nomodify
```

Additional keywords such as `noquery` are also useful. The keyword, `noquery`, restricts who can query the run time configuration of the time server. While having the ability to query the server would appear to be relatively harmless, the query function can be useful in mapping network time architectures and exercising potential security weaknesses. For example, with the `ntpsweep` command (supplied from the open-source NTP software distribution), it is possible to determine the operating system and processor type of NTP peers in a recursive manner. This function can be restricted using the `noquery` option, otherwise queries are allowed.

The version information seen by outsiders could be modified to mask the OS version, but few administrators take the time to obscure this information. This allows hackers to easily obtain information about the operating system platforms and versions.

A sample access control mechanism for a server could be something like as follows:

```
# cat /etc/inet/ntp.conf

# Prohibit general access to this service.
restrict default ignore

# Permit systems on this network to synchronize with this
# time service.  Do not permit those systems to modify the
# configuration of this service.  Also, do not use those
# systems as peers for synchronization.
restrict 192.168.1.0 mask 255.255.255.0 notrust nomodify notrap

# Permit time synchronization with our time source, but do not
# permit the source to query or modify the service on this system.
restrict 192.168.1.254 noquery nomodify notrap

# Permit all access over the loopback interface.  This could
# be tightened as well, but to do so would effect some of
# the administrative functions.
restrict 127.0.0.1
```

Conversely, a client could use the following controls.

```
# cat /etc/inet/ntp.conf

# Prohibit general access to this service.
restrict default ignore

# Permit time synchronization with our time source, but do not
# permit the source to query or modify the service on this system.
restrict 192.168.1.1 noquery nomodify notrap

# Permit all access over the loopback interface. This could
# be tightened as well, but to do so would effect some of
# the administrative functions.
restrict 127.0.0.1
```

It should be noted that as with any service founded on the UDP protocol, it is highly susceptible to IP spoofing. Using access control directives may not always provide sufficient protection for an organization's time services. Combining access control with authentication significantly improves the security of the NTP implementation.

As with any service, however, security is only as strong as the weakest link. For NTP, this means not only its access control and authentication but also the platform security of its servers and clients. If a platform cannot be sufficiently protected, it is possible that the NTP configuration and authentication key files can be stolen or compromised.

Selecting Appropriate Reference Clocks

A reference clock is needed to synchronize an NTP network to a useful standard. Clients cannot sync to a potential NTP server unless a reference clock exists in the server's synchronization path. There are three different ways to set up an NTP server for a large number of clients:

- Set up a reference clock on a secured network that uses accurate public NTP servers
- Set up a reference clock directly on a secured network
- Use a server's local clock as a reference clock (generally not a good idea)

Synchronizing the server to an accurate public NTP server is the most common route for most small installations. Assuring server accuracy is beyond the scope of this article, although `ntptrace` can give a general idea of the server's quality. It is important to find a server that is peered with several other servers to provide robustness. The NTP protocol is designed as a hierarchy to prevent large numbers of clients from accessing the same primary time sources. This hierarchy should be adhered to, and a large number of clients should not be configured to hit a busy stratum 1 time server. Networks should be designed to minimize the number of servers that interact with public NTP servers.

In addition, because public stratum 1 servers are often overloaded, stratum 2 servers should be used except for large (over 100 clients) NTP configurations where highly accurate time is critical. A list of public NTP servers (along with a list of things to consider when using them) is available at `http://www.eecis.udel.edu/~mills/ntp/servers.htm`. The administrators of these servers should always be contacted first before using them as NTP servers.

For secure environments where synchronized time is critical, it may not be appropriate to use a public reference clock. However, it is still important to use an external time source; otherwise, if the primary clock in the data center wanders, it causes all of the NTP clients connected to it to wander with it. While the primary clock could be adjusted to the true time occasionally, this would cause all of the clients to jump when the server adjusts. In addition, this time setting would have to be done manually; otherwise, it would compromise the security gained by not using a public NTP server. If a clock is ever adjusted to shift more than 17 minutes, all of the NTP daemons on its clients abort due to the sudden time shift.

NTP Server on a Secured Network With External Resources

Another option is to place the main NTP sources for the enterprise on secure management networks and have them receive time from external servers. However, as with any externally provided service, it is also an entry point for attackers. Therefore it is important to keep the servers independent and well secured. A layered security approach should be used that encompasses isolated network segments and systems, in addition to platform and NTP security measures. For example, the NTP servers could be deployed on independent platforms running only the NTP service. These systems should be hardened based on the recommendations in the Sun BluePrints OnLine article, *Solaris™ Operating Environment Security* (the original version of this article covered Solaris OE 2.5.1, 2.6, and 7, but an updated version covers Solaris OE 8).

In addition, the servers should use the access control and authentication facilities in NTP to further restrict access to the service. If possible, only authenticated NTP packets should be accepted. The server should also only accept packets from known, approved sources. For additional security, the NTP packets could be tunneled between the NTP sources and their external servers over encrypted connections such as those provided by IPSEC in the Solaris OE.

Reference Clocks on a Secured Network

A proper approach for using reference clocks is to setup reference clocks on appropriately secured management networks. Choosing the best reference clock for a particular network is beyond the scope of this article, although details can be found at `http://www.ntp.org`. The NTP site provides suggestions for choosing reference clocks, and also points to lists of reference clock hardware and vendors such as the ones available at
`http://www.eecis.udel.edu/~ntp/hardware.html` and
`http://www.boulder.nist.gov/timefreq/general/receiverlist.htm`.

A true reference clock can be expensive to obtain and maintain, but a variety of devices that sync to the correct time using long wave radio signals, CDMA technology, or GPS transmissions are also available. These devices are available for about US$600 and up. These are generally much cheaper and still provide time within a few milliseconds. In most datacenter environments, these are the most appropriate solution. GPS receivers need to be mounted on the roof or in a window, and even radio receivers may have problems operating in a shielded room. Additionally, to prevent a clock failure or an incorrect clock from affecting the clients time, it is best to have at least 3 independent time sources. If no public NTP servers are used, this means at least 3 reference clocks.

Because of these restrictions, it may be difficult to have reference clocks available behind the firewall unless a site is willing to invest a great deal in infrastructure and design. Another option for obtaining secure externally synchronized time is using

NTP's dial-up functionality to set a primary clock to a standard. Of course, the system should use a dial-out only modem and be protected with appropriate firewalls and other security methods.

Local Clock as a Reference Clock

Using the server's local clock as a reference clock is almost always a bad idea, especially if the server is in broadcast mode, because its time will wander and could eventually differ significantly from UTC. For details on how badly local clocks can wander, refer to the section, "Time in a Networked Environment" in the first article of this series.

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.

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[tm] 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.