10Smith C10 5/13/02 4:19 PM Page 241

10 Maintaining Consistent Time: Time Servers

Every time you set up a computer, you must set its clock. What's worse, computer clocks are imperfect, so they drift from the true time—and they drift at different rates. The result is that, on a network of any size, a few weeks (or possibly just a few hours) after you've set all the computers' clocks to the same time, they'll show annoying differences. Daylight Savings Time can also cause problems, because you'll have to set the computers' clocks again or allow the computers to adjust their own times—a process that can itself cause problems if a computer has more than one OS installed on it. All told, keeping all your systems' clocks synchronized can be an administrative headache. Fortunately, a class of programs exists to help work around this problem: *time servers*. These programs deliver accurate time measurements to their clients, so setting up a central time server and configuring clients to use it can keep all your computers set to the same time. You can even have your central time server synchronize itself to an outside time server that sets its time using an atomic clock, thus allowing for very accurate time settings across your entire network.

WHEN TO RUN A TIME SERVER

One of the primary reasons for running a time server is to deliver accurate time settings to clients on your network. By having clients set their clocks to a time maintained by one of your systems, you can keep your computers' clocks set to reasonably consistent values. (Depending on the protocol in use, "reasonably consistent" can mean variations of just a few milliseconds.) This can avoid problems caused by time mismatches. For instance, with mismatched clocks, a client might save a file to a server, but become confused when reading the file back one minute later because the save date might appear to be two minutes in the future. Mismatched times can also be a real headache when tracking down information in log files. It can also be annoying to see different times on different computers' displays. Some tools, such as Kerberos, rely on clients and servers having consistent times, so running a time server with such tools is a practical necessity.

The time server program that's discussed at greatest length in this chapter is a bit unusual in that it functions as *both* a server *and* a client. Therefore, another reason to run this particular time server is to set the clock of the computer on which the server runs. In a network that contains several Linux or UNIX computers, you'll set one machine to obtain its time from an outside server, then configure the rest of the systems in exactly the same way, except that you'll point these systems to the first one to obtain their time locally. This multi-tiered configuration reduces the load on the ultimate sources of time signals, such as computers that are linked to atomic clocks or radios that can receive broadcast time signals from official time sources. If you don't want to run the complete server package on clients, you may be able to make do with a simpler program that functions only as a client, but you'll need to call it explicitly every now and then to keep the system's time set correctly.

SETTING UP AN NTP SERVER

One of the more popular time server protocols is known as the *Network Time Protocol (NTP)*, which is described by RFC 1305 (`http://www.ietf.org/rfc/rfc1305.txt`). Previous versions were described in RFCs 958, 1059, and 1119. The current version number is 4, although version 3 NTP servers are still common in 2002. The main NTP Web site is `http://www.eecis.udel.edu/~ntp/`. NTP allows for a hierarchical structure of time servers, so those closest to accurate clocks can serve others, which in turn serve the ultimate clients. NTP clients and servers are readily available for Linux and other OSs. Configuring NTP for Linux requires editing a single configuration file. You can monitor the server's operation using dedicated tools. Limited-functionality clients are also available for easy use if a system is to be a terminus in the time server hierarchy.

NOTE There's a simplified variant of NTP that's known as Simple NTP (SNTP). SNTP clients can synchronize to NTP servers.

UNDERSTANDING HOW A TIME SERVER FUNCTIONS

Time server functionality begins with a veridical time source, such as an atomic clock or a radio or modem receiver that can synchronize with an official time source. Global Positioning System (GPS) receivers work in part by using time signals from satellites, so these may be used as a way to acquire such an accurate time source. (The NTP Web page includes a discussion of such hardware devices at `http://www.eecis.udel.edu/~ntp/hardware.html`.)

The atomic clock, radio hardware, or similar device is known as a *reference* or a *stratum 0* time server. Unless you own such a device, you won't connect directly to it, since they don't have network connections (they usually interface through an RS-232 serial port and require special drivers). The computer that synchronizes its clock to the reference is known as a *stratum 1* server. In theory, these computers have the most accurate times available on the Internet, although in practice, surveys suggest that roughly a third of them have times that are incorrect by a second or more. Systems that synchronize to stratum 1 servers are known as *stratum 2* servers, and so on.

Synchronization involves transferring several packets (typically at least five over roughly a five-minute period) between the client and the server. The client sends a packet with the client's current time to the server. The server responds with a similar packet. By comparing the send and receive times using its local clock, the client can estimate the round-trip network delays between it and the server, and thus compensate for these delays in adjusting its own time. The client may also be configured to use several servers, and thus to compare the times, network delays, and other salient factors between these servers in order to select the best time from among these servers.

Locally, a full NTP server runs continuously, checking back with its parent servers every once in a while (initially every 64 seconds, but this interval usually increases to 1024 seconds, or possibly higher with some configurations). The NTP server program can adjust the clock on the computer on which it runs in several ways. NTP typically ignores large differences (those over a second or so) for a while, on the grounds that the large difference might be a protocol error; but after the difference has persisted for a while, NTP compensates either by resetting the time at once or by running the system clock faster or slower (that is, *slewing* the clock) until the system and external time are synchronized. Small errors are compensated by slewing the system clock until the measurements are synchronized. The NTP server also maintains a file, typically `/etc/ntp/drift`, `/var/state/ntp.drift`, or something similar, in which a number is stored that allows the server to compensate for the bulk of a computer's clock drift even if contact with lower-stratum time servers is lost or if the computer is powered off for a while.

TIP ✔

Errors of more than 1000 seconds usually cause NTP to exit, on the assumption that something is seriously amiss and requires human intervention. Thus, you may need to manually set your system clock to a value that's at least *approximately* correct before running an NTP server. Alternatively, you can use the `ntpdate` program, described in the upcoming section "Using an NTP Client Package," to set the time before running the NTP server. Some distributions call `ntpdate` in the NTP server startup script.

Normally, an NTP server on a small network synchronizes its clock to about three external time servers. (Three is a somewhat arbitrary number; you may increase it or decrease it as you see fit, but three usually provides enough redundancy and cross-checking to work well.) The external time servers for a small network should normally be stratum 2 servers; there's very little accuracy to be gained by using stratum 1 servers, and because there are so few of them, stratum 1 servers are best reserved for use by servers with over a hundred clients. Clients on the small network may then run NTP servers for near-continuous updates from the network's stratum 3 NTP server, or they may use NTP clients or other time protocol clients to update their clocks less frequently. If your network has more than a couple dozen computers and continuous service is important, you might consider running two stratum 3 time servers to reduce the chance of problems should your main time server computer go down or become unreliable. If extremely precise time is important, you might obtain a GPS-based clock and run your own stratum 1 time server. Such clocks sell for a few hundred dollars. Devices based on AM radio time broadcasts are less costly, but are also much less accurate (they're usually accurate to less than a second, but if the goal is high accuracy, that may not be enough).

NTP uses Coordinated Universal Time (UTC) as the basis for all operations. UTC is similar to Greenwich Mean Time (GMT)—the time in Greenwich, England, unadjusted for Daylight Savings Time. UTC differs from GMT in several technical details, but the main point of interest is that it's defined more precisely and in terms of highly reliable atomic reactions, rather than the rotation of the Earth, which varies by small but measurable amounts. When necessary, UTC is adjusted for variation in Earth's rotation by the addition or subtraction of a second from a day (a *leap second*). Local time is related to UTC by adding or subtracting your time zone, with Daylight Savings Time adjustments, if necessary. Linux systems use UTC internally, so UTC is a good choice for a time protocol from a Linux point of view.

Most *x*86 OSs require that the motherboard's hardware clock be set to local time, not UTC. For this reason, Linux supports motherboard clocks that are set to either local time or UTC, and maintains a separate software clock set to UTC. On a Linux-only system, UTC is the better option because Daylight Savings Time requires no adjustments to the hardware clock. On a system that dual boots Linux and Windows (or some other OS that requires a clock set to local time), you may be forced to use local time, and both OSs may attempt to make Daylight Savings Time adjustments. Using a time protocol at system boot time can help alleviate this problem, but Linux doesn't automatically adjust the hardware clock when the system clock is set, so the benefit of NTP running under Linux may not manifest itself in the other OS. You can type **`hwclock –systohc –localtime`** to set the hardware clock from the system clock on a system that sets the hardware clock to local time. Substitute `–utc` for `–localtime` if the computer stores time in UTC.

TIME SERVER PROGRAMS FOR LINUX

The main NTP server package for Linux is known as `ntp`, or some variant of that, such as `xntp`, `xntp3`, or `xntpd`. The `x` in the name stands for *experimental*, although that's something of a misnomer, since the software has been in common use for several years. NTP version 4 packages usually omit the `x`. Most Linux distributions ship with one of these packages—usually version 4, although in 2002 some distributions still ship with version 3.

The NTP package included with most distributions includes the main NTP server and several support programs, including the following:

- **`ntpd`**—This is the main NTP server program. (It's sometimes called `xntpd`, particularly with version 3 servers.) As noted earlier, although this program is referred to as a server, it functions both as a server for higher-numbered strata and as a client to one or more lower-strata NTP servers. Thus, you might run this program even if you want your system to function only as a client to other time servers.

- **`ntpdate`**—This program is a much simpler client-only package. You can call it periodically on systems for which a constant accurate time is less important, as described in the upcoming section "Using an NTP Client Package."
- **`ntptrace`**—Occasionally, you might want to know the source of a time setting. This program traces backwards, from the local computer to the NTP server to which it's currently synchronized, and so on. This can sometimes be useful diagnostic information.
- **`ntpq`**—This is the NTP monitoring program, described shortly, in the section "Monitoring NTP's Operations."
- **`xntpdc`**—This is another NTP monitoring and control program. It's used for more advanced operations than is `ntpq`.

In addition to the main NTP package, there are other time-setting programs available for Linux. The most common of these is `rdate`, which is similar to `ntpdate` in general principles—you use `rdate` to set the clock on a one-time basis. Most distributions don't install `rdate` by default, but many do ship with it. One advantage of `ntpdate` over `rdate` is that `ntpdate` provides much greater precision—to within a few milliseconds, depending upon network connections and the precision of the parent server. By contrast, `rdate` provides a precision of only about a second.

CONFIGURING ntp.conf

NTP is configured through a file called `ntp.conf`, which is usually located in the `/etc` directory. This file contains comment lines, which begin with a pound sign (`#`), and option lines that set various NTP options. Important options include the following:

- **`server` *`address`* `[key` *`key`*`] [version` *`number`*`] [prefer]`**—This option sets the name of a server to which NTP should synchronize itself. The *`address`* may be a hostname or an IP address. You may include several `server` options, one per line; your NTP server tries to contact each one and picks the best one for synchronization. (I discuss locating an NTP server shortly.) You may also include some additional information on this line. Specifically, `key` *`key`* provides an authentication key if you want to implement security to restrict access to the server, `version` *number* tells the server to use the *number* version of the protocol, and `prefer` tells the system to give this server preference over others.

- **`fudge` *`address`* `stratum` *`number`***—This option is used mainly in reference to the 127.127.1.0 server (which corresponds to the local system clock) to make NTP treat it as a stratum 7 server—that is, well below most NTP servers in priority. This allows NTP to operate even if it can't reach any other servers.
- **`driftfile` *`filename`***—The drift file contains a number that NTP uses when it starts up after having been shut down for a while, or when it can't reach a server. This number allows NTP to compensate for the computer's average clock drift, thus reducing the clock's inaccuracy when isolated.
- **`broadcast` *`address`* `[key` *`key`*`] [version` *`number`*`] [ttl` *`number`*`]`**—If you include this option, the server will periodically broadcast its current time to all clients on the network specified by *`address`*, which should be the computer's address on that network or a multicast address of `224.0.1.1`. Using broadcasts may reduce network traffic on a large local network with many NTP servers that function mainly as clients.
- **`broadcastclient [yes | no]`**—You can tell an NTP server to listen for broadcasts from other local NTP servers with this option.

There are many other `ntp.conf` options available for performing more exotic functions. Consult the documentation, which ships in HTML format and usually appears in `/usr/share/doc/xntp-`*`version`* or a similar directory.

The default `ntp.conf` file on most distributions is nearly functional; you need only edit or add one or more `server` lines to point the computer to appropriate NTP server systems. Your choice of servers is important, because a server that's very distant from you in a network topology sense, that has unreliable network connections, or that has synchronized to an unreliable source, can cause your own time to be incorrect or of highly variable accuracy. As noted earlier, unless you're setting up a time server that will serve hundreds or more clients, or unless you have some compelling reason to have the most precise time possible, you should synchronize to a stratum 2 or higher server. You can find an extended discussion of these issues at `http://www.eecis.udel.edu/~mills/ntp/servers.htm`. The bottom of this document includes links to lists of public stratum 1 and stratum 2 time servers, so you can search the stratum 2 list for suitable time server candidates. Try to find a server that's close to you in a network topology sense. Network topology corresponds only very loosely with geography—if you're in Philadelphia, a server in Canberra, Australia, is likely to be farther than one in Boston, but the Boston server might be closer than one in New York City, depending on how your network links work.

TIP ✓ You can use `ping` to get a rough idea of the network delays involved in reaching two different NTP servers. As a very rough rule of thumb, NTP servers with shorter ping times are preferable to those with longer ping times.

If the time server list indicates that a server's operators want to be notified before you use the server, be sure to do so. You might also want to investigate time server options that might be less public, but closer to you. Many large organizations, including many Internet service providers (ISPs), operate time servers. You might therefore find a time server that's very close to you on the Internet. Consult your ISP, or your network manager if you want to set up a time server for a department in a larger organization.

If you buy a GPS time receiver or other external clock hardware, you can synchronize your system to it to become a stratum 1 time server. You'll need to install special drivers for the external clock hardware. These drivers make the hardware appear to be a device on the 127.127.0.0/16 network, so you can use an ordinary `server` option line, but point it to the appropriate IP address. You must consult the documentation for your device's Linux drivers for details. The NTP Web site includes a list of hardware product manufacturers at `http://www.eecis.udel.edu/~ntp/hardware.html`.

When you're done reconfiguring `ntp.conf`, you should restart the NTP server. This is usually done through a SysV startup script, as discussed in Chapter 4, Starting Servers. Unless your startup script calls `ntpdate` prior to starting `ntpd`, you won't see your system time change dramatically, even if it's off by a few minutes. Instead, NTP will wait a while as it receives multiple time checks, then either change the time quickly or adjust it more slowly to synchronize it with its own parent server's time. You can monitor its operations using `ntpq`, as discussed in the next section.

MONITORING NTP'S OPERATIONS

Aside from watching the time with a program like `xclock`, the usual way to monitor NTP's operation is to use the `ntpq` program. You launch this program and then enter any of a large number of text-based commands at its prompt. The program then displays information in response to these commands. Some of the more important `ntpq` commands include the following:

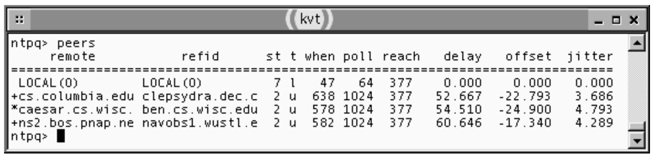
- **`host`** ***`hostname`***—By default, `ntpq` queries the server running on the localhost computer. You can change this behavior by issuing the `host` command, so you can easily monitor the functioning of NTP servers throughout your network. You can achieve the same effect by providing the target hostname when you launch `ntpq`, as in **`ntpq remote.threeroomco.com`**.
- **`hostnames [yes | no]`**—If you provide the `yes` option, `ntpq` displays hostnames rather than IP addresses when reporting on remote computers (this is the default behavior). The `no` option causes `ntpq` to display IP addresses. Using `-n` when launching `ntpq` has the same effect as typing **`hostnames no`** after launching the program.
- **`ntpversion`** ***`versionno`***—You can specify the NTP version number (*`versionno`*) to use in queries of the NTP server with this option.
- **`quit`**—When you're done, issue this command to quit from the program.
- **`peers`**—This command is one of the more useful initial diagnostic tools. It displays a list of the servers with which yours is communicating. Unless you use the `host` command, this list should include the localhost and all the servers you listed in `ntp.conf`. Additional information includes the server to which each of the listed servers is synchronized; the stratum of each server; when each server was last contacted and the interval between contacts; a numeric code indicating how reliable the connection between the systems is; and delay, offset, and jitter information, which are measures of how accurate and variable the time data are. To the left of each entry is a one-character code indicating how your server is using data from each server. A plus sign (`+`) means that the server was considered for synchronization, but was beaten out by a better server; an asterisk (`*`) indicates that the server is your own server's parent; an `x` indicates a *false ticker*—a server whose time is just plain wrong; and various other characters indicate servers that have been rejected for assorted other reasons. Variants on `peers` are `lpeers` (which may list more servers) and `opeers` (which doesn't display the names of servers to which yours is connected).
- **`associations`**—This command produces a list of association statistics for each server. Servers aren't identified by hostname or IP address, but

by an association ID that's used in some other commands. Variants on this command are `lassociations`, `passociations`, and `lpassociations`.

- **`readvar`** ***`assocID varname`***—You can read a raw variable using the `readvar` command, which takes an association ID and variable name as arguments. This command is most likely to be useful in advanced debugging sessions. Variants are `mreadvar` and `rv` (the latter is actually an exact synonym).
- **`readlist`** ***`assocID`***—This command is like `readvar`, but it produces a list of all the standard variables. A synonym is `rl`, and a variant is `mreadlist`.
- **`pstatus`** ***`assocID`***—This command requests that the system indicated by *`assocID`* return status information. In practice, the effect is very similar to the `readlist` command.
- **`writevar`** ***`assocID varname`*****`=`*****`value`***—You can alter the value of a variable with the `writevar` command. You won't normally have to do this.

The first time you configure NTP, whenever you reconfigure it, and possibly periodically thereafter, you may want to use `ntpq` to check that it's doing what you expect. Figure 10.1 shows `ntpq` in action on a server that's been running for some time. When you first run `ntpq` after starting `ntpd`, many of the fields will be blank or will hold dummy values (typically 0). Over the course of a minute or so, most of those fields should fill with values similar to those shown in Figure 10.1, but the plus signs and asterisks to the left of most of the entries in Figure 10.1 won't appear for several minutes because the server will take a while to determine which of the would-be peers is most reliable. Some other values will change over the course of several minutes, as well, and then stabilize. If you see an `x` to the left of a server name, you may want to consider removing it from your configuration, because chances are that server is seriously misconfigured.

If you notice your system time drifting or changing in some odd way, you may want to use `ntpq` to check your NTP server's configuration. It may



```
ntpq> peers
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
 LOCAL(0)        LOCAL(0)         7 l   47   64  377    0.000    0.000   0.000
+cs.columbia.edu clepsydra.dec.c  2 u  638 1024  377   52.667  -22.793   3.686
*caesar.cs.wisc. ben.cs.wisc.edu  2 u  578 1024  377   54.510  -24.900   4.793
+ns2.bos.pnap.ne navobs1.wustl.e  2 u  582 1024  377   60.646  -17.340   4.289
ntpq> █
```

Figure 10.1 The `ntpq` program lets you monitor NTP server activities.

How to Keep Precision Time

Most computers use internal clocks that are based on *oscillators*—electronic devices that produce a signal that changes in a regular manner. If the oscillator produces, say, a 100Hz signal, this means that it changes state 100 times per second. By counting the oscillations, the computer keeps time. As noted earlier, however, computer clocks are imperfect. There are several causes of such imperfections. For one thing, the oscillators may not operate at *precisely* the values they're supposed to. If a 100Hz oscillator actually operates at 100.1Hz, time measurements will be off. In fact, an error of this magnitude would cause a clock to gain over a minute a day. What's more, the oscillation rate may vary over time, as the circuit's temperature changes as a computer heats up after being turned on or as room temperature varies. The oscillation rate may also vary as the circuit ages or for assorted other reasons.

Factors outside of the oscillator can also influence the precision of the time measurement. Normally, the oscillator signals the computer that its time period has come by using an interrupt (interrupt 0 on *x*86 computers). If the computer is busy processing other important events, though, it may not be able to process this interrupt in a prompt fashion, and so the computer may miss a "tick" of the clock.

These sources of error contribute to general clock drift, and for most users, they're a modest annoyance. In some cases, though, precision time measurement is important. For instance, computers used in extremely time-sensitive scientific experiments may need extreme precision. (Linux isn't really well-suited to such functions, although the *Real-Time Linux* variant, `http://fsmlabs.com/community/`, does much better in this respect.) The Enhanced Real Time Clock kernel option (in the Character Devices menu) enables user access to a high-precision clock, should you need it.

Inserting kernel modules is one task that's particularly likely to result in missed oscillator beats, so if high timing precision is important, you should compile a kernel with as many drivers built in as possible, rather than leaving lots of options as modules. Leaving the computer powered on at all times and keeping it in a temperature-controlled room can both help minimize changes in clock drift due to temperature factors. The NTP server tracks clock drift and attempts to compensate, and periodically checks back with its parent server, so running an NTP server can help keep an accurate time—but if there's a chance that NTP will step the time at some critical point, you might want to temporarily disable it during particularly time-sensitive operations.

have lost synchronization with a server because of a change in IP address or some permanent network connectivity problem. (A temporary network outage shouldn't create noticeable problems; the server will just switch to the local clock until the network comes back online.) If your system doesn't synchronize with any time server for several minutes after starting `ntpd`, you should investigate general network connectivity. Can you ping the remote server? Do you have a firewall in place that might block NTP queries? (You may need to reconfigure the firewall to pass through UDP port 123 packets, at least to and from the servers in question—both yours and the remote servers.) Do you have authorization to use a remote server? (That server might use a firewall or an authentication key to keep out unwanted clients.)

USING AN NTP CLIENT PACKAGE

If you've configured an NTP server on one computer on your network to acquire its time from a remote system, you can install and configure NTP servers on all other systems on your network in much the same way, but point these computers to your first NTP server instead of to external servers. The result is that your entire network will generate relatively little in the way of Internet traffic related to NTP, although the entire network will be set to the correct time via local NTP queries. You'll get the full benefits of `ntpd` on each of the clients, even though most or all of them won't be used as servers. On a complex network, you might implement several strata of time servers yourself. For instance, you might run an NTP server on each subnet for the benefit of clients on that subnet, and have each of these subnet servers connect to a system on the subnet that's one closer to the Internet at large to obtain its own time. This design will minimize NTP-related network traffic within your own network.

For many systems, running `ntpd` is overkill. This program is designed to keep the time on an entire network synchronized to within a few milliseconds, and to keep the entire network's time set to well under a second of UTC. Furthermore, `ntpd` is designed to keep this time accurate constantly, with little drift from accuracy during the course of a day. Your needs might be more modest; for instance, you might not be concerned if the time on any given system drifts a few seconds each day. Furthermore, `ntpd` is a server, and as such poses a potential risk; a security-related bug could conceivably be found and exploited in `ntpd`, resulting in a potential vulnerability for all your systems, or at least all those that are exposed to the Internet at large. (In fact, such vulnerabilities have been found in `ntpd` in the past.) For these reasons, the `ntpdate` program may be useful in many situations. As noted earlier, the `rdate` program can be used in a similar manner, but it uses its own protocol and is less precise than is `ntpdate`.

NOTE The designers of NTP are working to phase out `ntpdate`, with the goal of allowing `ntpd` to be used for one-time clock settings, much as `ntpdate` can be used today. Future versions of the NTP package may omit `ntpdate` and provide some way to use `ntpd` in a more limited manner in its place. In fact, some NTP version 4 packages lack `ntpdate`.

To run `ntpdate`, you type its name followed by the name or IP address of the time server against which you want to synchronize your system's time. You can specify multiple time servers if you want the program to automatically select the best one. You can also include several optional parameters before the program name and the time server addresses:

- **`-B`**—Normally, `ntpdate` slews the system clock speed if the time difference is less than half a second, and resets the clock if the difference is greater than half a second. This option forces the program to always slew the clock speed, even with large errors.
- **`-b`**—This option forces `ntpdate` to set the clock to the value sent by the server, even if the error is small.
- **`-o`** ***`version`***—You can tell the program to use a specific version of NTP with this option.
- **`-p`** ***`samples`***—Normally, `ntpdate` sets the clock by using four time samples from the server. You can adjust this value up or down (within the range 1 to 8) using this option.
- **`-q`**—You can query the server without setting the clock by including this option. This will *not* return a human-readable time, though; it returns information on the delays and offsets to the servers.
- **`-s`**—You can force `ntpdate` to send its output through the system logger with this option. You might do this if you call the program as a cron job.
- **`-u`**—Normally, `ntpdate` uses the standard NTP port of 123 for its outgoing packets. You can have it use an unprivileged port (numbered above 1024) by using this option. This might be necessary to work around some firewalls.

When you run `ntpdate` to synchronize your clock with a server, the program reports various statistics, such as the stratum, offset, and delay associated with the server you contact. Unless you specify `-q` or there's an error, the program then either slews your clock's speed or sets the absolute time on your computer.

One common method of using `ntpdate` is to run it as a cron job. You might run it once an hour or once a day, for instance. This may be often enough for many purposes, and it can reduce network loads to run `ntpdate` on a periodic basis, as opposed to running `ntpd` constantly.

WARNING !

Do not call `ntpdate` on a regular basis at midnight to synchronize with a public time server. Such servers are often bombarded by requests at midnight, generated by people who think that's a reasonable time to synchronize their clocks. Instead, pick some unusual time that's otherwise convenient, like 1:23 PM or 3:48 AM. This will reduce unnecessary congestion on public time servers, and may result in more accurate and reliable clock setting for your system, because you won't experience delays or dropped packets from midnight congestion.

USING SAMBA TO SERVE TIME

NTP is an extremely useful protocol, and as noted earlier, it's one of the most precise methods of setting the time on a Linux computer. Other time protocols do exist, however. One of these that deserves mention is the time server functionality that's included in the Server Message Block (SMB)/Common Internet Filesystem (CIFS) file- and printer-sharing protocols used on Microsoft networks and implemented in Linux by Samba (described in Chapter 7, File and Printer Sharing via Samba). If you run NTP on a Samba server, it may be simpler to configure Samba to serve the time rather than install NTP clients on all your Windows computers. (Samba can't set your system's time from a Windows SMB/CIFS time server, though.)

SAMBA'S TIME SERVING OPTIONS

Samba's configuration file, `smb.conf`, is broken into several sections, most of which define particular directories you want to share with SMB/CIFS clients. The first section, though, is known as `[global]`, and it defines global defaults and other options that don't make sense in individual shares. One of these is the `time server` option, which you can activate by setting the following parameter:

```
time server = Yes
```

This tells Samba to respond to time queries from SMB/CIFS clients, using the SMB/CIFS time server features. You can set this option whether or not your computer uses NTP, `rdate`, or some other time-setting protocol to set its own time, but it's most useful if the Samba server's time is set correctly through such a mechanism.

NOTE The SMB/CIFS time protocols aren't as precise as are those of NTP. Windows systems may vary by a second or so after being set in this way, even before their clocks begin to drift.

CONFIGURING A WINDOWS CLIENT TO SET ITS CLOCK

To set the time on a Windows client, you can use the following command in a DOS prompt window:

```
C:\> NET TIME \\SERVER /SET /YES
```

In this command, *`SERVER`* is the NetBIOS name of the Samba server. You can manually type such a command to set the time much as you can use `ntpdate` to set the time on a Linux computer. You might want to create a Windows batch file to execute this command whenever a user logs in, though. You can do this by typing the command (without the DOS `C:\>` prompt) into a file (you might call it `SETTIME.BAT`) and copying that file to the Windows StartUp folder. This way, whenever the user boots the computer or logs in, the command will execute. Alternatively, if your network uses a domain configuration, you can include the command in your default network login script. (Because networking hasn't started when Windows executes `AUTOEXEC.BAT`, though, you should *not* include this command in that script.)

NOTE Windows 2000 and XP support NTP more directly. Specifically, the command `NET TIME /SETSNTP:`*`ntpserver`* should synchronize the time with the *`ntpserver`* system. There's even a full NTP server that ships with these systems, but its configuration is outside the scope of this book.

SUMMARY

A time server allows you to keep your computers' clocks synchronized with each other, and with an external time server that is ultimately tied to a veridical time source. Using time servers can be useful when referring to time stamps on files and in system logs. One of the more popular and powerful time server protocols is NTP, which allows for a time server (generally called `ntpd` or `xntpd`) to run constantly, periodically checking its notion of time against that of another NTP server. On a small network, you'll probably configure one server to synchronize itself to a stratum 2 server (that is, one that's two links away from a veridical time source), then synchronize all your other computers to your stratum 3 time server using `ntpd` or a client-only NTP program like `ntpdate`. A larger network might use multiple stratum 3 time servers or a veridical time source like a GPS clock.

Other options for handling time setting include the Linux `rdate` command (a time client) and the time server functionality in SMB/CIFS, including the Samba server and Windows `NET` command. You can use the latter to easily synchronize the time on Windows clients, without installing NTP software on them.