BRAD KNOWLES

building scalable NTP server infrastructures

Brad has been a contributor to the NTP Public Services Project for over five years, in addition to working as a UNIX and Internet system administrator for almost two decades, specializing in Internet email and DNS administration for more than fifteen years.

knowles@ntp.org

IN THIS ARTICLE I DISCUSS HOW TO take the concept of a simple NTP service configuration for a small number of clients and then expand that to be able to serve many thousands, tens of thousands, hundreds of thousands, or even millions of clients. By choosing the right type of scalable service infrastructure, you should be able to handle a virtually unlimited number of clients with a relatively small number of servers, but it will take some work to get there.

I start out with some discussion about various typical traps that you might tend to fall into, if you're not deeply familiar with the NTP protocol and the Reference Implementation. I follow that by discussing the three different major modes of operation you can choose for your servers, highlighting weaknesses in each mode and other factors that need to be considered. Next, I mention some specific hardware recommendations for good-quality NTP servers, as well as some warnings about OS issues, and touch on the subject of "reference clocks." Finally, I come to some conclusions about which modes are the most scalable and under what circumstances. If you get lost, you may want to consult the list of NTP-related definitions [1].

I assume that you are already familiar with NTP in general and building simple NTP server configurations and that you know where the official NTP documentation is located [2], as well as the unofficial Community Supported Documentation as made available by the NTP Public Services Project [3]. You know what the difference is between the NTPv4 [4] Reference Implementation [5] and the older NTPv3 [6] clients that may be available from other sources (e.g., xntpd). You are also assumed to understand the difference between NTP and SNTP—the Simple Network Time Protocol [7].

Of course, before a machine can be an NTP server, it must first be an NTP client. In NTP parlance, the only difference between a server and a client is that an NTP server is a machine that has NTP clients that are depending on it for time, whereas an NTP client depends on NTP servers but does not have any other NTP clients depending on it. Any NTP client is a potential NTP server, and all NTP servers are also NTP clients. Therefore, I assume that you already know how to build a robust NTP client configuration, including things like knowing how many upstream NTP servers to configure and why defining two upstream servers for your client is the worst possible configuration, proper use of the "iburst" keyword, proper use of the "tos minclock" and "tos minsane" parameters, etc.

Common Misconceptions

For people who don't understand how NTP works, it is very easy to assume that the way you build a scalable NTP service infrastructure is exactly the same way you would build a scalable infrastructure for any other kind of typical Internet application such as Apache or Sendmail.

In other words, throw the biggest, baddest, honkingest, multi-core, multi-thread, multi-CPU boxes at the problem, and then front-end them with a whole array of proxy servers, load-balancing switches, and clustering, playing tricks with the network layer to make the same service IP address visible from a variety of servers, and so on.

For NTP, this is the worst possible thing you could do. NTP is UDP, not TCP. It does not have a fork()/exec() model of execution. It is single-threaded and essentially does a huge select() loop on incoming UDP packets. Doing TCP or a fork()/exec() model of execution would add unnecessary and unpredictable latency to the process of trying to handle the packets, and it would defeat the entire purpose of accurate time-serving.

However, NTP is not stateless. In fact, it is about as stateful as you can get with a UDP protocol, since it tracks both short- and long-term variations in clock stability for all configured upstream servers, based on the smallest possible statistical samples of information for each system. None of what you would think of as the "standard scaling rules" can be applied to NTP.

The goal of NTP is to try to synchronize your system clock to the "One True Time" known as UTC (Universal Coordinated Time [8]), or at least always move your system clock closer to UTC, within certain statistical error boundaries. Since there is one and only one system clock per computer, you do not want to run more than one NTP daemon on a machine, because they will both be trying to modify the system clock at the same time and they will certainly cause the system to be highly unstable, if not frequently suffering from kernel panics. The Reference Implementation includes code that works hard to prevent more than one copy of ntpd running at the same time.

The algorithms inside NTP are extremely sensitive to the most minor changes, and over the 20+ years they've been in development, they have been tuned to seek out and eliminate the tiniest statistical errors they can find, whether the variation is short- or long-term. They also need to do loop detection and elimination, and for that they depend on a one-to-one correspondence between the system clock and the IP address. For multi-homed machines, this can pose a problem, since they don't have just one IP address.

All of the NTP algorithms are also built on top of the basic assumption that if you contact a client or server at a given IP address, it will always be exactly the same machine with exactly the same system clock. So, for example, playing "anycast" games at the routing layer and making the same IP address available from multiple servers is a recipe for disaster. The same holds for using load-balancing switches or clustering.

NTP already has extensive capabilities for doing explicit failover between multiple upstream servers, and anything you do to try to hide the upstream servers behind something else will only get you worse reliability and worse quality of service.

Configuration Modes Between Servers and Clients

There are three basic modes of NTP server configuration that we are concerned with, and a variation on one of those, for a total of four modes of operation.

The simplest mode, "unicast" [9], is the classic client/server configuration for NTP. That is, each NTP client periodically sends UDP packets on port 123 to the servers they are configured to use, gets UDP responses back, goes through the NTP algorithms to select which of the designated servers is "best," and then tries to synchronize its clock to it. Note that there is no client authentication or authorization in the NTP protocol, but you do have the option to enable cryptographic server authentication to the client.

Unicast mode does have the advantage that it gives the client(s) the best possible chance for getting good time service from the upstream servers, if the upstream servers are not overloaded. This is because a unicast-mode client is involved in a periodic but ongoing long-term bidirectional conversation with each of the upstream servers, and it is able to gather the maximum amount of information possible regarding which time server currently has the "best" time, etc.

Unfortunately, even with good hardware and good configurations on both sides, the simple version of this type of configuration (without cryptographic server authentication) may tend to start having problems when handling more than about 500–1,000 clients per NTP server. Of course, if you have configured your clients to each use multiple upstream servers, then you magnify the problem of how many clients hit how many upstream servers. Overloaded servers start dropping too many queries, too many retransmissions are required, and the servers are providing reduced quality of service to all clients.

Unicast was the first mode invented oh-so-many years ago, and it should be supported by all NTP clients. In the official documentation on this mode, the terminology may differ somewhat from what I have used here but the concepts are the same. Note that you can build purely hierarchical relationships among the NTP servers themselves, or you can build them as symmetric active/passive peers, but either of these server-to-server infrastructures still operates in unicast mode [10].

Next, we have "broadcast" mode [11, 12]. The basic concept is that each client is configured to passively listen for broadcasts from the designated server(s), go through the NTP algorithms to try to select the best-quality time server, and then synchronize the clock to that.

However, in this mode the clients will actually operate in unicast mode during startup (something called the "startup dance"), and then settle down to passive listening. The startup dance allows the NTP client to determine what the "broadcast latency" is between the server and the client and to make suitable adjustments so that it can make a better determination as to which of the designated upstream servers is best, which results in the NTP client getting better-quality time service.

If there is no response from the broadcast server to the unicast packets from the client during the startup dance, the client will fill in a default value for the broadcast latency. Alternatively, the server administrator can hard-code its own choice for broadcast latency. Clients can also be configured to avoid the startup dance altogether, through the "authdelay" command—in effect, making them broadcast-only clients.

Unfortunately, broadcasts don't cross MAC-layer segments, which means you end up needing at least one broadcast NTP server on every subnet. This naturally leads to the concept of running a broadcast NTP server on your network gear, which is generally a bad idea and discussed below.

Broadcast mode should be supported even by older NTPv3 clients.

Up next, we have the multicast variant of broadcast mode, the only difference being that the UDP packets sent by the servers are addressed to a specific multicast address (defined by IANA to be 224.0.1.1) to which all multicast clients listen. However, other than the address being different, the operations are otherwise the same as broadcast mode.

Also note that this requires support at the network level. By default, most network devices are not set up to support routing multicast traffic, so they would need to have their configurations updated. Moreover, each multicast server will see packets from all multicast clients during their respective startup dances, and all clients will see packets from all servers, and again this will tend to bring problems with congestion, drops, retransmits, etc. As with broadcast mode, multicast mode should be supported even by older NTPv3 clients.

With NTPv4, there is a new mode based on multicast networking called "manycast" [13]. This is an automatic discovery mechanism used by clients to find their closest server(s). The client sends UDP packets to the configured multicast address, but it starts with a packet TTL of zero, to see whether there are any manycast servers on the local segment. If the client doesn't get a response in a given period of time, it retransmits with a TTL of one, to see whether there are any manycast servers that are just one hop away. This process continues until either the TTL is set to the maximum and no servers ever respond or the client finally discovers and sets up one or more relationships with servers somewhere on the network.

Manycast mode allows clients to automatically detect their nearest NTP servers and then set up unicast associations with them. The load will automatically be distributed throughout the infrastructure as you put multicast servers in strategic places. You will minimize as much as possible the number of servers that see traffic from too many clients, the number of clients that will see traffic from too many servers, and the number of router hops traffic has to cross in order to get from the starting point to the destination.

Of course, manycast has the same problem as multicast, in that it needs support at the network layer. However, manycast is new with NTPv4, and support for it will most likely not be found in older NTPv3 clients.

I won't discuss "pool" mode, since it is intended to allow sites to make better use of the NTP Pool Project [14] as opposed to helping you set up your own infrastructure (pool-style or not), and it's a new enough addition to the system that I'm not sure it will be covered by the upcoming NTPv4 RFCs that the IETF NTP Working Group [4] is putting together.

But assuming your version of the ntpd code is new enough to include this option, there is official documentation on how you could potentially configure your NTP clients to use the NTP pool [15].

For example, this might be a useful configuration option to use on your NTP servers to help ensure that they get an adequate number of upstream servers and a better chance at good-quality time service, even if you don't configure any of your own internal NTP clients to make use of the pool.

Running NTP Servers on Network Devices

The concept of running in broadcast mode naturally leads to the idea of people running NTP servers right on the networking gear itself. Unfortunately, although most networking gear has specialized hardware for performing the important switching and routing functions, noncore functions (such as NTP) end up getting shunted over to a "general-purpose" processor. Of course, there's lots of things that this general-purpose processor is supposed to be doing in addition to NTP, so you get a competition: the other stuff suffers, or NTP suffers, or— the most likely outcome—both suffer.

In addition, most network devices have the cheapest possible clock circuits—even on the really expensive routers where a single line card can cost a hundred thousand dollars or more. The design of NTP is such that it can only compensate for a certain amount of error in the underlying clock circuits before it just gives up. Most network devices tend to have clock circuits that have so much error inherent in them that they usually run right on the ragged edge of the amount of error that can be compensated for within the NTP protocol. As such, you should always configure them to be NTP clients: they tend to make pretty poor NTP servers even if all the clients are configured to only passively listen to broadcasts.

Moreover, network devices used as broadcast-mode NTP servers probably won't support the cryptographic server authentication methods, which would make them triply poor choices for NTP servers. This issue is also discussed in the Community Supported Documentation [16].

Lost Clock Interrupts

With NTP, you don't ever want to see clock interrupts get lost. This is one of the fastest ways to kill your NTP accuracy, and it will very likely cause the NTP daemon to quit completely. There can be many causes of lost clock interrupts, the two most common being hardware problems [17] and OS problems [18].

When the server is running on complex hardware configurations, you are likely to see excessive amounts of jitter and other statistical errors in terms of servicing clock interrupts. For NTP, the more precise and accurate the servicing of clock interrupts, the better. Typically, this means more simply configured machines are better—you'll never have a throughput issue, so you don't need to throw in really fast network cards with things such as TCP Offload Engines, and since the application is single-threaded you'll never have a CPU load problem that can be resolved by throwing more CPU cores at it.

In other words, you can probably throw out every single modern machine you've got.

Indeed, currently one of the best NTP server hardware platforms you can buy is the Soekris net4501 Single-Board Computer [19]. Poul-Henning Kamp [20] has done wonders with these boxes, and the official NTP time servers for Denmark are running on them. These machines are about as dead simple as you can get, and they can handle hundreds or thousands of clients as easily as or better than pretty much anything else on the planet [21].

There is a comparable SBC configuration that has become more common in the pool.ntp.org project. If you go to the Web site [14], you can find out more about this project, and maybe you can get the current coordinator of the project, Bjorn Hansen, to send you a link to their alternative configuration.

Note that pool.ntp.org operates in unicast mode, and by playing games with DNS-based load balancing and geographically aware names that can be chosen by the admin of the NTP client (which might also be a local NTP server), it ends up scaling to handle several million clients across the world. If the pool were able to operate in manycast mode, I have to believe that this could reduce server hardware requirements by at least one or two orders of magnitude.

Then there are the OS configuration issues. In addition to making sure that the hardware services clock interrupts in a precise and accurate manner, you need to make sure that the OS is configured to do the same.

Unfortunately, owing to the internal architecture of Windows-based OSes, the best they can do is ~50 ms accuracy, and for a good-quality NTP server you really want to get down into the single-digit millisecond ranges, if not lower.

Likewise, you will also see problems on modern versions of Linux or other freely available OSes that try to handle 1000 clock interrupts per second on lower-end hardware (e.g., with kernels configured with "Hz=1000"). On higher-end hardware that can handle these settings, or where you can tune the kernel settings to a more appropriate level, you should be able to get good-quality time service from just about any UNIX or UNIX-like OS available currently or in the past 20 years. My laptop regularly stays in the single-digit millisecond range of accuracy relative to UTC, and it's not anything particularly special in this regard.

Reference Clocks

If you're running a network of NTP servers, you may want to have one or more of your own internal "reference clocks" [22] configured so that you can provide the best quality of time to your clients. This would also give you a good measure of additional robustness in case your Internet connection goes down.

NTP actually depends on these external clocks to provide a reference of UTC against which everything else can be measured. One key measurement in NTP is your logical distance from your closest refclock: a machine directly connected to a refclock is operating at Stratum 1 (the refclock itself is Stratum 0), a machine is Stratum 2 if it is a client of a Stratum 1 server, etc. The lower your stratum number, the better the quality of time service you can potentially provide to your clients, if your refclock is good enough.

There are many different types of reference clocks, including GPS-based devices, radio-based equipment (using WWVB, DCF, or one of the other radio broadcasting stations around the world), rubidium or even cesium-based atomic clocks, and CDMA or GSM mobile telephone–based equipment. Heck, you can even use a dial-up modem to connect to a time service via POTS telephone lines.

I won't discuss any of them in detail, but as an NTP server administrator the primary thing you need to know is that they are configured in a way that is very similar to unicast mode (using the "server" keyword), but instead of listing a hostname or regular IP address, you use the appropriate pseudo-IP address in the 127.127.0.0 range. Which specific address you use will vary depending on which particular refclock you have and which driver it requires.

I will say that GPS-based refclocks are very popular, and if you're willing to build them yourself or if you can find someone willing to build one for you, they can be had for relatively small amounts of money—on the order of $100 or so. If that's too expensive, then radio refclocks can be had for as little as $20, if you're willing to put in some work or you can find someone who will do that for you.

Unfortunately, most vendors ship with their standard NTP client/server software without support compiled in for refclocks. Therefore, if you want to directly connect one or more of your NTP servers to a refclock, you will probably have to recompile and reinstall the NTP daemon, ntpd, from the source code. An alternative would be to buy an appliance that provides both refclock and NTP Stratum 1 service in a turnkey device, although that can get expensive. You'll need to decide which option is right for you.

Conclusions

So, this is what we have for the modes of operation we're concerned about:

- Unicast
- Broadcast
  - Multicast
  - Manycast

Generally speaking, multicast and manycast are the most scalable, with different issues that your infrastructure will have to support or deal with.

Multicast mode will allow a smaller number of servers to support a larger number of clients, but there may tend to be network "storms" of traffic resulting from excessive numbers of clients going through the startup dance around the same time. (Some randomization is built into the startup process, but it can only do so much to alleviate load on the server.) You will also get into issues with too many servers trying to talk on the same multicast channel at the same time.

Multicast mode also tends to encourage centralizing resources for ease of management, which will increase the number of router hops that traffic has to pass through in order to get from the server(s) to the clients and thus will reduce the quality of the time service provided to the clients. Even if multicast servers are located in close proximity to their multicast clients, the clients will not be able to get the best-quality time service, because of the asymmetric nature of the communications between them and the server, and once the startup dance is done, they may be unable to adapt to changing network conditions.

Manycast is still pretty new. The NTP community doesn't have that much experience with it yet, but it may require more server(s) to support the same number of clients. However, since it distributes these servers closer to the clients and minimizes the number of router hops, it helps to increase the quality of time service provided and the probability that clients will continue to obtain tolerable time service in the event that their subnet is temporarily disconnected from the rest of the world.

Overall, manycast mode is considered to be the most scalable and robust NTP server infrastructure. Quoting from the official documentation on manycast:

> It is possible and frequently useful to configure a host as both manycast client and manycast server. A number of hosts configured this way and sharing a common multicast group address will automatically organize themselves in an optimum configuration based on stratum and synchronization distance.

If you can't run multicast or manycast, then your next most scalable option would be broadcast.

In any event, these more scalable techniques tend to increase the exposure your time servers and their IP addresses get to the network and therefore to increase the number of clients dependent on them. That increases the probability that someone else will want to spoof packets from them, which they will probably be able to do quite easily since all communications are via UDP or similar methods.

Therefore, regardless of whether you use broadcast, multicast, or manycast, you should configure your systems to provide cryptographic server authentication to their clients, and ideally you should do the same for unicast mode as well.

RESOURCES

[1] http://support.ntp.org/bin/view/Support/NTPRelatedDefinitions.

[2] http://www.eecis.udel.edu/~mills/ntp/html/index.html.

[3] http://support.ntp.org/.

[4] http://www.ietf.org/html.charters/ntp-charter.html.

[5] http://support.ntp.org/download.

[6] http://www.ietf.org/rfc/rfc1305.txt?number=1305.

[7] http://www.ietf.org/rfc/rfc2030.txt?number=2030.

[8] http://www.eecis.udel.edu/~mills/y2k.html#utc.

[9] http://www.eecis.udel.edu/~mills/ntp/html/assoc.html#client.

[10] http://www.eecis.udel.edu/~mills/ntp/html/assoc.html#symact.

[11] http://www.eecis.udel.edu/~mills/ntp/html/assoc.html#broad.

[12] http://www.eecis.udel.edu/~mills/ntp/html/manyopt.html#bcst.

[13] http://www.eecis.udel.edu/~mills/ntp/html/manyopt.html#mcst.

[14] http://www.pool.ntp.org/.

[15] http://www.eecis.udel.edu/~mills/ntp/html/manyopt.html#poolt.

[16] http://support.ntp.org/bin/view/Support/
DesigningYourNTPNetwork#Section_5.6.

[17] http://support.ntp.org/bin/view/Support/KnownHardwareIssues.

[18] http://support.ntp.org/bin/view/Support/KnownOsIssues.

[19] http://www.soekris.com/net4501.htm.

[20] http://people.freebsd.org/~phk/.

[21] http://phk.freebsd.dk/soekris/pps/.

[22] http://www.eecis.udel.edu/~mills/ntp/html/refclock.html.