6173

N94-30678

4040

The NIST Internet Time Service

Judah Levine
Joint Institute for Laboratory Astrophysics
National Institute of Standards and Technology
and University of Colorado
Boulder, Colorado 80309

Abstract

We will describe the NIST Network Time Service which provides time and frequency information over the internet. Our first time server is located in Boulder, Colorado, a second backup server is under construction there, and we plan to install a third server on the East Coast later this year. The servers are synchronized to UTC(NIST) with an uncertainty of about 0.8 ms RMS and they will respond to time requests from any client on the internet in several different formats including the DAYTIME, TIME and NTP protocols.

The DAYTIME and TIME protocols are the easiest to use and are suitable for providing time to PCs and other small computers. In addition to UTC(NIST), the DAYTIME message provides advance notice of leap seconds and of the transitions to and from Daylight Saving Time. The Daylight Saving Time notice is based on the US transition dates of the first Sunday in April and the last one in October. The NTP is a more complex protocol that is suitable for larger machines; it is normally run as a "dæmon" process in the background and can keep the time of the client to within a few milliseconds of UTC(NIST).

We will describe the operating principles of various kinds of client software ranging from a simple program that queries the server once and sets the local clock to more complex 'dæmon" processes (such as NTP) that continuously correct the time of the local clock based on periodic calibrations.

Introduction to the Internet

The current internet is based on the ARPANET, a small network started in the 1960s and originally funded by the Department of Defense. The network has grown dramatically in the last decade and now connects thousands of sites worldwide. One of the important reasons for the success of the network is its support for a group of relatively simple message protocols whose structure is largely independent of the details of the underlying transmission medium. Two of the most widely used protocols are the User Datagram Protocol (udp) and the Transmission Control Protocol (tcp); both of these are in turn built on a common base called the internet protocol (ip) and hence are usually referred to as udp/ip and tcp/ip.

An important concept supported by the internet protocols is that of a logical connection – a connection between two processes on two different machines so that messages sent by one

504

are received by the other without either end–point needing to be concerned with the details of the physical realization of the network or the topology of the intervening path. A machine may have more than one of these logical connections active at the same time, with two independent processes transmitting and receiving messages over the same physical connection. This multiplexing requires a hierarchical address – both the source and destination addresses must contain not only the address of the machine, but also some identification of the process on the machine that is the source or destination of the messages. This end-point is usually called a port, and it is identified by a port number. A process that wishes to communicate over the network must logically connect to the port to send and receive messages.

The protocols also support the concept of a server process – a process on a machine that is continuously "listening" for connections and performs some action whenever a connection request is received. There are often several such processes on any machine; they may each be actively listening for connections or they may designate a super–server that listens on behalf of all of them. The response is basically the same in either case – a request to a specific port on the machine logically activates the server process that has advertised its willingness to respond to that type of request. Once the server process has been activated, the communication between it and the client proceeds transparently through all of the intervening layers in both machines – both of them see the connection as if it was a dedicated physical circuit with full–duplex capabilities.

If a server process is to be useful, the client must know which port it is listening to, since the port number must form part of the request for service. A group of standard "well–known" port numbers have been assigned to address this issue, and all machines on the internet are expected to conform to these standard assignments. The mail–server, for example, is expected to be listening for connections on tcp/ip port 25; other well–known services have similar port assignments (Comer, 1991, pages 167 and 201).

Three ports have been assigned for time services. Port 13 is for the "daytime" service using either the tcp/ip or udp/ip protocols; port 37 is for the "time" service using either tcp/ip and udp/ip protocols, and port 123 is for the Network Time Protocol (NTP) using udp/ip only. This paper describes a server whose time is synchronized to the NIST clock ensemble; it will respond in the appropriate format to time requests on any of these three ports from any client on the internet.

Time Formats

In addition to defining a standard port number for each service, it is equally important to define a standard message format. These formats are specified in a series of documents called "Requests for Comments" or RFCs. The RFC documents are numbered sequentially in chronological order; revisions to a protocol are usually assigned a new number so that many of the lowered-numbered documents have been superseded and are obsolete. The RFC documents are available in several formats from the Network Information Center (Comer, 1991, Appendix 1). The daytime protocol is specified in RFC–867, the time protocol is in RFC–868 and the Network Time Protocol in several documents including RFC–1119, RFC–1128, RFC–1129 and RFC–1305.

1. The Time Protocol

The time protocol is the simplest one to use. The server listens on port 37 and responds to a request in either tcp/ip or udp/ip formats by replying with the time in UTC seconds since 1 January 1900. The response is an unformatted 32–bit binary number; the conversion to local civil time (if necessary) is the responsibility of the client program. The 32–bit binary format can specify times over a span of about 136 years with a resolution of 1 second; there is no provision for finer resolution or for increasing the dynamic range. The transmitted time will wrap–around through 0 in the next century and, if the protocol is still in use at that time, there will be a 136–year ambiguity in the transmitted time thereafter.

The strength of this protocol is its simplicity – many computers connected to the internet keep time internally as the number of seconds since 1 January 1970 (or sometimes since 17 November 1858) and a conversion between the received time and the internal format is a simple matter of binary arithmetic. This strength must be balanced against several serious weaknesses:

a. There are two difficulties with the handling of leap seconds – a practical one of what time to transmit at the leap second and a conceptual one of how to specify the time afterwards. The choices depend on whether or not the clients are assumed to know that a leap second is currently being inserted and whether they can remember all of the previous leap seconds to correct the transmitted time afterwards. The simplest solution is to adjust the time on both the server and the client at the time of the leap second, which is equivalent to pretending afterwards that the leap second did not happen. This raises difficulties when a client tries to compute the time interval between two epochs on opposite sides of a leap second, but this difficulty is a generic one and is not limited to computer clocks.
b. The protocol is awkward for machines that keep time internally as time of day plus the date (PCs fall into this category) since the conversion of the received message to the internal format in the client requires a full knowledge of the vagaries of the calendar and the time–zone system and of the transitions to and from daylight saving time.
c. There is no provision in the message for additional information such as the health of the server, and the protocol cannot be easily expanded without the risk of breaking the software in some clients.

2. The Daytime Protocol

The daytime protocol has many of the advantages of the time protocol and addresses many of its short–comings as well. The server listens on port 13 and responds to a request in either tcp/ip or udp/ip formats by replying with the time and date as a line of text whose exact format is not specified in the standard beyond the requirement that it be composed of human–readable standard ASCII characters. We have chosen a format for the daytime service which conforms to the very broad requirement of RFC–867 and which addresses many of the short–comings that characterize the time format we have just discussed. Our message format is very similar to the format used by our ACTS system (Levine et al., 1989). A typical message is shown below:

			D	L			
MJD	YY-MM-DD	HH:MM:SS	ST	S	H	Adv.	
49302	93 - 11 - 11	17 : 30 :42	00	0	0	50.0	UTC(NIST) *

The message consists of a single line of text; the identifying characters in the legend above the line have been added here to show the significance of each field. The first number is the Modified Julian Day. It is included for those systems that keep time as the number of seconds since some epoch, since the conversion between a Modified Julian Day number and such formats is a simple matter of binary arithmetic and does not require a knowledge of the calendar. The next 6 numbers are the UTC date and time as shown by the legend and can be directly understood by a human observer and easily used by machines that use the date and time system internally. (Although this format can transmit the time during a leap second in a natural way, I discuss below a number of reasons for not doing this.) The DST flag and LS flag give advance notice of the transitions to and from daylight saving time and of the imminent occurrence of a leap second, respectively. The format is the same as in the ACTS system:

If DST is 0, then the US is currently on Standard Time; if DST is 50 then the US is currently on Daylight Saving Time. If DST is between 49 and 1 a transition from Daylight Saving Time to Standard Time is imminent. The DST value is decremented at 0000 UTC every day and the transition will arrive at 2 am local time when the counter is 1. If dst is between 99 and 51 a transition to Daylight Saving Time is imminent. The DST value is decremented at 0000 UTC every day and the transition will arrive at 2 am local time when the counter is 51.

If LS is 0 then no leap second is imminent. If LS is 1 then a leap second is scheduled to be added after 23:59:59 UTC on the last day of the current month. That second will be called 23:59:60, and the next second will be 00:00:00 of the next day. If LS is 2 then a leap second is scheduled to be dropped at the end of the current month. The second following 23:59:58 will be 00:00:00 of the next day.

The server itself transmits UTC and therefore experiences no internal discontinuity during the transitions to and from Daylight Saving Time, but there is likely to be a discontinuity during a leap second. As pointed out above, there is an ambiguity in what to transmit during and following a leap second depending on whether or not the client is assumed to know of its existence and whether or not it knows how to parse a time of 23:59:60. After the leap second has occurred, this ambiguity is resolved in the server by adjusting the time of the server and pretending that the leap second did not happen. There are many reasons for making this adjustment gradually by slewing the clock rather than suddenly by stepping it. This adjustment will therefore take a finite time to complete, so that the time of the server may be ambiguous while it is going on. The client software will face the same problem during leap seconds and may also have a much larger version of it each Spring and Fall when Daylight Saving Time starts and ends if its internal clock is set to local time rather than to UTC. The client, too, must choose between slewing the clock and adjusting it in one step. The first alternative results in a clock that is wrong for a significant period of time, and the second may play havoc with many of the time-dependent processes on the machine. Many implementations of the client software choose the single-step alternative in both the leap-second and Daylight-Saving-Time

situations; the leap–second adjustment could be implemented by parsing both 23:59:59 and 23:59:60 as 23:59:59.

The H parameter gives an estimate of the health of the time server, with a value of 0 indicating fully healthy. Positive integers indicate increasingly poor health; both the magnitude of the possible time errors and the uncertainty with which they are known increase as this parameter increases from 0. Users who need the time with an uncertainty of less than 1 second should not use the message if the health parameter is non–zero and those who need the time with an uncertainty of 3 s or less should not use the message if the health parameter is greater than +1. A value of +2 indicates a time error of up to 5 minutes and values greater than this indicate an internal failure in which the time error may be small but cannot be determined.

The final parameter gives the time advance in milliseconds. The entire packet (not just the terminating on–time marker) leaves the server early by this amount to compensate approximately for the delay in the travel time through the internet. Within the continental US, travel times on the internet range from about 30 ms to 150 ms, so that the packet is likely to arrive within 100 ms of the correct time anywhere in the US. This parameter is fixed at the present time since our experience with ACTS indicates that this accuracy is sufficient for most users, but future enhancements to the server software may estimate this parameter dynamically as is done with ACTS at the present time.

The message ends with an asterisk for compatibility with the ACTS format, but this character has no special significance as an on–time marker since the entire message will most likely be transmitted and received as a single network packet. The time advance parameter applies to the entire packet for this reason.

3. NTP – The Network Time Protocol

NTP is the most complex and sophisticated of the time protocols, and it can provide the highest accuracy to a time client as a result. It normally runs continuously on the client as a "dæmon" (background) process; it periodically queries the server and makes small adjustments to the local time based on the data that it receives. The server responds to each query with a packet in a special NTP format that is built on the udp/ip network protocol (Mills, 1991). The client software can also be configured to query several servers and to average the responses in a statistically robust manner. In particular, it evaluates the response of each server against the average and is prepared to consider the possibility that one of the servers is broken. NTP must receive its calibration data via the noisy internet, and it will be limited by the un–modeled noise in the transmission medium. It is most likely to have problems at intermediate periods of a few hours or so, because the internet noise in this period regime is likely to have low–frequency divergences (which would appear in an Allan variance analysis as Flicker or Random–Walk Frequency Modulations) that are difficult to estimate because they are not amenable to improvement by averaging.

Server Synchronization

Our server was initially synchronized to UTC(NIST) using periodic calls to the ACTS system. We found that the time of the server could be kept within 0.8 ms RMS of UTC(NIST) by calling ACTS once every 3000 s. We are currently upgrading the server so that its clock is phase–locked to a 1 pulse/s signal received directly from the clock room. This signal is stretched to 125 s and is connected to the machine in such a way as to generate an interrupt every second. These interrupts can initiate the phase–lock task which in turn adjusts the internal clock so that its time is an exact even second. Periodic calls to ACTS are also scheduled, since the phase–lock process cannot detect a slip of an integer number of seconds. This process can keep the internal clock within 100 s RMS of UTC(NIST). The improvement in the accuracy and stability of the reference time of the server is unlikely to be noticed by the users, since the uncertainty in the received time is dominated by the uncertainty in the travel time across the network.

Both synchronization methods have advantages and disadvantages: the phase–lock system is easier to implement but requires a direct 1 pulse/s signal; the ACTS system requires much more complex software but can be used to implement a stratum–1 clock anywhere a telephone line is available. We plan to use both of these methods in the future in constructing additional time servers. This could greatly reduce the jitter due to the network delay for many users since the distance between them and a server would be much smaller.

Access to the Server

Our primary network time server is named time_a.timefreq.bldrdoc.gov, and its internet address is 132.163.135.130. A backup server named time_b.timefreq.bldrdoc.gov is being constructed; its address is 132.163.135.131. The times of both servers will be synchronized to UTC(NIST) using the phase–lock method we have outlined above.

We have written example software that can be used to set or check the time of a client machine by parsing the "daytime" format response of the server. The client software includes a routine to convert the received message to local time, if necessary; conversion to daylight saving time is also supported. This software can be adapted to run in most software and hardware environments, requiring only a network connection and standard interface software to send and receive messages on the internet. It provides a time capability similar to the ACTS system (but at lower accuracy) without the need for making toll calls. The example software is publicly available via anonymous ftp from the primary network time server in directory /pub/daytime. Both the source code and the documentation are in this directory.

We have also written example software to access the time service, but this service provides less information than the daytime service with no appreciable increase in accuracy, and we do not recommend its use. This service is often used by time programs that are supplied with commercial network software for PCs. These programs usually require external environmental information to specify the time–zone of the user and what to do about daylight saving time. Examples of how to specify this information are in the documentation in the directory specified above.

The software for the Network Time Protocol is widely available, and is often bundled with the operating system itself. It is normally distributed in source code with instructions for how to build it for many different environments.

Conclusions

The NIST Network Time Service provides time information to internet users that is directly traceable to UTC(NIST). The information is provided by a time–server located in Boulder; the server will respond to requests for time in several different formats. The simpler formats are well suited to the needs of small computers with modest accuracy requirements, while the more complex formats can provide substantially better accuracy at a substantial increase in both the size and complexity of the client software.

One of the synchronization algorithms we have developed for the server itself uses periodic calls to our ACTS service to synchronize the time of the server; it could be used to construct a stratum–1 time–server needing only a standard voice–grade telephone line for synchronization. This server might be connected to a local network that is disjoint from the internet or it might be used as a stand–alone machine wherever accurate time–stamps are required.

Acknowledgements

I am grateful to David Mills of the University of Delaware, the designer of NTP, for many helpful discussions. This work is supported in part by grant NCR–9115055 from the National Science Foundation through the University of Colorado.

References

Comer, Douglas E., 1991. *"Internetworking with TCP/IP"*, Volume 1, Second Edition. Englewood Cliffs, New Jersey, Prentice Hall.

Levine, J., M. Weiss, D. D. Davis, D. W. Allan and D. B. Sullivan, 1989. *"The NIST Automated Computer Time Service"*, J. Res. of the Natl. Inst. Stand. Tech., 94, 311–321.

Mills, David L., 1991. *"Internet Time Synchronization: The Network Time Protocol"*, IEEE Trans. Communications, 39, 1482–1493.

QUESTIONS AND ANSWERS

Dr. Winkler, USNO: This is a remarkable experiment. We have for the last half year or so made regular timings between our station at Richmond and Washington. And we find that in that network, through that connection which we have, delays are vastly different from second to second. They are in fact traffic-dependent. It is a packet-switch network and in this packet–switch network each node of course retransmits the packets, depending on its own local traffic. So for that reason the actual noise which you see is a function of time of day. So that is why we have abandoned that, actually, as a practical missile for the dissemination of time, of timing between stations; and we are also looking at circuit boards with GPS receivers which is much more economical and more precise.

J. Levine: I agree with you 100 percent. May I show another slide? What I did was say let's measure the delay between Boulder and Washington; and let's measure the outbound delay and the inbound delay. So here you have it. Boulder or Washington go through 16 gateways: Denver, St. Louis, Chicago, Cleveland, New York, etc. The delay is 59 ms. This is measured kind of average. Back delay: Washington to Boulder turns out to be a very strange delay, because it goes through NIST Gaithersburg and then in one step it makes it to NIST Boulder. The delay: 55 ms. The answer is the difference is a few ms; it is remarkably the same. Of course four ms, that is the kind of the noise you are going to get; or four, or ten or 12 or some number like that.

Dr. Winkler: May I add just one thing? During these experiments we have not advanced by 50 milliseconds but one-half of the round-slip delay as measured at that moment.

Tom Becker, Air System Technologies: Is this transfer done on a transactional request basis? Or are these two machines essentially logically connected continually?

J. Levine: All of the protocols have to put up a request to the server for the time. In the case of the time and the daytime protocol, you do it by hand; in the case of NTP, NTP schedules itself with a variable time ranging from a few minutes to a few hours. If you don't ask, you don't get.

Tom Becker: Yes, it would seem that if it were a continuous process and there were very many people doing this, the network would be overloaded just by time transfer information. What happens then in–between requests? What would a local machine do to maintain accurate time?

J. Levine: Well again that depends on what your client software is. If you have NTP, then NTP is a dæmon and it is making small adjustments to your client software. If you are using the other protocols, then nothing happens and your clock is free running.

Tom Becker: NTP does or does not run on PCs?

J. Levine: NTP is probably too big for most PCs. There is no reason why it couldn't run in principle; it is mostly a matter of size.

Tom Becker: But you are not aware of an implementation of NTP for PC?

J. Levine: It is written in C; it should run on a PC if you didn't run on a hardware speed first. But I really don't know the answer to the question.