(Preprint) AAS 13-519

TIME, TIMESTAMPS, AND TIMESCALES

Harlan Stenn*

This paper focuses on the distribution and dissemination of Time and the various aspects of what that really means. We look at what information needs to be in a timestamp to make it much more generally useful, and also the various choices for timescales. Finally, we discuss what is needed to compare and convert timestamps that may be from completely different timescales. To date, there hasn't been a complete or portable way to deal with timestamps and timescales. Network Time Foundation is ready and plans to address these problems.

INTRODUCTION

The future of UTC is a bit of a thorny issue, with individual perspectives as to what aspects are “signal” and what aspects are “noise”. Perhaps uncharitably, people may be quick to understand the severity of the problems they see with the issues around UTC while being less tolerant of the severity of the different problems others have.

Einstein is reputed to have said (I read it on the internet so it must be true) “No problem can be solved from the same level of consciousness that created it.” One way to approach this is to say, “Oh look, we've discovered a problem—let's come up with a solution.” When the solution is implemented, folks are happy for a while until consciousness / awareness expands to illuminate new problems. Another approach is to say, “Let's live with the current problem because anything we do to solve it will create new, bigger problems.” Basically, no matter what one does there will be problems.

It seems to me the issues with UTC are political, not technical, because to me it’s obviously true. But on a somewhat more empirical basis, let’s take a look at air-traffic control. Society in general likes UTC, as people have an expectation that “high noon” is the middle of the day, and “midnight” is the middle of the night. So UTC makes sense for civil timekeeping. Leap seconds, on the other hand, make life more difficult for programmers. If air-traffic radar is monitoring planes that travel 300 meters per second at a rate of 128 times per second, having a leap second get inserted in an “imperfect” way across all of the involved systems is, literally, cause for alarm. It is technically possible to implement leap seconds well. And it is more difficult to implement air traffic anti-collision taking leap seconds into account. Additional problems exist when different subsystems may or may not be upgraded at different times. So I’ve just argued that there are, indeed, technical issues. Later I’ll talk about tools proposed by the Network Time Foundation (NTF) to mitigate these programming difficulties. To lend credence to my original assertion, if the collision avoidance systems used a timescale that was not based on UTC, they would not suf-

* NTP Project Manager, Founder/President of Network Time Foundation, PO Box 918, Talent OR 97540, USA.

fer from problems associated with leap seconds. Choosing UTC instead of another timescale for these applications is, therefore, a political issue.

Once upon a time, programmers used ordinary arithmetic on "time" numbers. They didn't place a high-priority on writing portable code. Many were not even aware of the importance of "defensive programming" and knew that if they subtracted two "time" numbers that they had no reason worry about overflow. When `difftime()` came along, it took a while for the majority of programmers to see why it was needed. Similarly, for a very long time many programmers did leap-year calculations using a simple "modulo 4" calculation. Some folks did slightly better. I think it wasn't until the so-called "Y2K" issue hit that people did a proper job of finding and using complete leap-year calculations.

To date, there hasn't been a complete or portable way to deal with timestamps and timescales. This paper focuses on the distribution and dissemination of Time and what the various aspects of that really mean, looking at what information needs to be in a timestamp to make it much more generally useful, and also the various choices for timescales. Finally, we discuss what is needed to compare and convert timestamps that may be in completely different timescales. Time, timestamps and timescales are issues I've been thinking about for several years' time, and while writing this paper and preparing for Network Time Foundation's participation in Google's Summer of Code, 2013 I had a prospective GSoC student tell me how interesting he thought the project was. So we worked on a proposal and it has been accepted for GSoC 2013. If all goes as expected, we'll have some interesting things to report soon.

TIME

We each know what "time" means to us. We also have social agreements with others about time. Sometimes the parties agree. It's easy to get an answer to the question "What time is it?" That answer, however, is usually short and does not contain some significant information. Specifically, the answer doesn't usually contain either the expected error or the timescale.

Time in an operating system can be an interesting beast. Database systems, at least, require monotonic time for consistent ordering of transactions. This can lead to problems if the system time is ahead of the correct time.

Changing the system time brings up issues with time-related events. POSIX uses absolute time for these. Scheduling an alarm-clock event for 10 seconds in the future actually schedules the event to fire when the clock strikes (the current time plus 10 seconds). If the system time is moved forward, all of the events that should trigger during that period will "fire". If the system time is moved backward, scheduled events will trigger at the original designated time. Sometimes this is a good thing, sometimes not. It's not easy to know when some number of seconds have elapsed, regardless of changes to the system clock.

In my opinion, it was suboptimal for POSIX to limit events to absolute time instead of also allowing relative time events, and not provide a hook to allow applications to be notified in the event of a system time change. Network Time Foundation wants to work on an API and timer library improvements to allow for "relative" as well as "absolute" timers, and also a comprehensive mechanism to notify processes when the system time changes.

TIMESTAMPS

In POSIX-compliant systems, the usual ways to get the time are by calling `gettimeofday()` or `clock_gettime()`. In these cases we get seconds and sub-seconds since an epoch. We also get an implicit timescale and no idea of what the potential error is around that timestamp. This is

the *status quo*; the vast majority of software out there is designed to handle these arguably incomplete and insufficient timestamps. We propose a new "timestamp", containing the following elements:

- The current reported system time.
- The expected difference between the system time and "true" time.
- The expected error in the time.
- The timescale used.

The above list could be extended further—it could contain a "boot-count index" and even a "host ID". The boot count index might be useful when comparing timestamps on a specific host, given that the system may have been rebooted one or more times in between the first and second timestamps in question, and there may be significant information that was learned in between these events. This is somewhat analogous to IERS Bulletin A.[*] Similarly, a "host ID" might be useful when comparing timestamps taken on different machines, in order to see if any additional information might be available for either system that would affect the interpretation of the given timestamps.

Political decisions and consequent new problems will need to be solved at the next level of consciousness. With the above timestamp format, forward time-steps could be made as before—monotonically increasing time—by adding the adjustment directly to the reported system time. If directly advancing the system time might have undesired consequences (remember the way POSIX events are handled—this might cause scheduled events to fire earlier than expected) one could instead add the correction to the "expected difference" field and let the Operating System apply the correction.

If the time ever needs to be stepped backwards, the desired offset gets put into the "expected difference" field and we stop incrementing the current system time. As time advances, we appropriately reduce the desired offset. If one needs the "current" time we just add a tiny value to the current reported time and also increase the expected difference by that same amount. If one needs a "full" timestamp we return the current structure. When the desired offset ceases to be negative, we continue along normally.

The expected error in the time can be determined and adjusted as the system communicates with external time sources. If the system "loses contact" with these external time sources the expected error value would slowly but steadily increase.[†] The expected error would reduce upon the re-establishment of communication with appropriate external time sources.

TIMESCALES

For a timestamp to be generally useful, one must know the timescale in which it was taken. Terrestrially, these include the SI-based timescales UTC, TAI, GPS, and Loran. Also included would be a number of other timescales, including ones based on Earth-angle time and "length of day". Steve Allen has compiled a good list of these.[‡]

---

[*] http://www.iers.org/IERS/EN/Publications/Bulletins/bulletins.html
[†] In NTP, this is PHI, Φ, 15 parts per million per second.
[‡] http://www.ucolick.org/~sla/leapsecs/timescales.html

A current case for extra-terrestrial time is Martian Standard Time. NASA scientists like a 24-hour day, and the length of a day on Mars is pretty close to the length of a day on Earth. A Martian day is just under 40 minutes' time longer than a day on Earth. The timestamps for events that happen on Mars must be correlated with terrestrial timestamps.

As far as Network Time Protocol (NTP) goes, it pretty much doesn't care about timescales. If you feed it uncompensated GPS time, it will serve that. If you feed it Martian Standard Time on a system with a clock that is capable of ticking to the Martian second, it will keep and serve Martian Standard Time.

Precision Time Protocol (PTP) is another case. It works with different profiles. The default profile uses the TAI timescale, while the power profile uses an alternate timescale that defines a time zone so the local environment can use local time.

NTP and PTP software occupy similar environments, and one of the things Network Time Foundation is doing is finding a way for these two protocols to work together. For this and other common use cases to work well, we need a way to convert timestamps between different timescales. An upcoming revision of NTP will likely need to include the timescale in its packets.

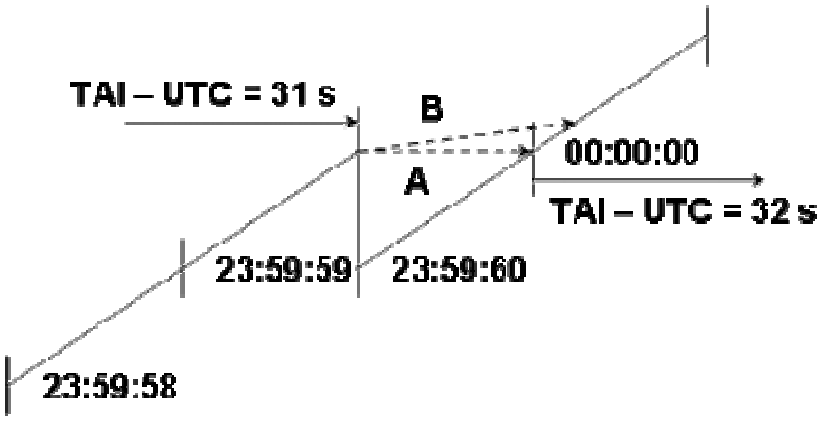

Figure 1. NTP Offset in the Vicinity of a Leap Second.

LEAP SECOND INSERTION WITH THE PROPOSED NEW TIMESTAMP

Professor Mills suggests that NTP can implement leap seconds by "freezing" the reported time for one second around the leap:*

> [T]he routine that actually reads the clock is constrained never to step backwards, unless the step is significantly larger than one second, which might occur due to explicit operator direction. In this design time stands still during the leap second, but is correct commencing with the next second (Figure 1).
>
> […] Unlike the POSIX conventions, the NTP clock is frozen and does not advanced during the leap second, so there is no need to set it back one second at the end of the leap second. The chronometric correspondence between the UTC and NTP timescales contin-

* http://www.eecis.udel.edu/~mills/leap.html

ues, but NTP has forgotten about all past leap insertions. Thus, determination of UTC time intervals spanning leap seconds will be in error, unless the exact times of insertion are known from the NIST table and its successors.

Immediately after the leap second insertion, both timescales resume ticking the seconds as if the leap had never happened. The clock reading is constrained to always increase, so every reading during the leap second increases the NTP clock by at least one microsecond for older kernels and one nanosecond for newer ones.

In case A the clock was not read during the leap second, so appears to stand still. In case B the clock was read one or more times during the leap second, so the value increments beyond the last reading. This will persist until after the leap second the stepped-back clock catches up to this value.

Table 1 shows how it might look using the proposed new timestamp structure, which is displaying {System Time, Offset to Correct Time}. The NTP Windows implementation of leap seconds applies the leap second over two seconds, and the example below is only one way that leap might be represented with the new timestamp structure.

Table 1. Example Representations of Leap Second.

UTC	NTP-DLM	NTP-Windows	Smear24H
00:00:00.00	{00:00:00.00, 0}	{00:00:00.00, 0}	{00:00:00.00, 0}
06:00:00.00	{06:00:00.00, 0}	{06:00:00.00, 0}	{05:59:59.75, .25}
12:00:00.00	{12:00:00.00, 0}	{12:00:00.00, 0}	{11:59:59.50, .50}
18:00:00.00	{18:00:00.00, 0}	{18:00:00.00, 0}	{17:59:59.25, .75}
23:59:59.00	{23:59:59.00, 0}	{23:59:59.00, 0}	{23:59:58.00, 1}
23:59:59.50	{23:59:59.50, 0}	{23:59:59.50, 0}	{23:59:58.50, .5}
23:59:59.99	{23:59:59.99, 0}	{23:59:59.99, 0}	{23:59:58.99, .0}
23.59.60.00	{23:59:59.99, .0}	{23:59:59.99, .0}	{23:59:59.00, .0}
23.59.60.50	{23:59:59.99, .5}	{23:59:59.99, .25}	{23:59:59.50, .0}
23.59.60.99	{23:59:59.99, 1}	{23:59:59.99, .50}	{23:59:59.99, .0}
00:00:00.00	{00:00:00.00, 0}	{00:00:00.00, -.50}	{00:00:00.00, 0}
00:00:00.50	{00:00:00.50, 0}	{00:00:00.50, -.25}	{00:00:00.50, 0}
00:00:01.00	{00:00:01.00, 0}	{00:00:01.00, 0}	{00:00:01.00, 0}

API LIBRARY OPERATIONS

So it's time (if you'll excuse the expression) for a general and portable timestamp library and API. This library should handle timestamp arithmetic, comparison, and conversion. As mentioned above, NTF has a student working on this project for GSoC 2013.

Timestamp arithmetic includes addition and subtraction of two timestamps, in possibly different timescales. Comparing two timestamps is an obvious operation, and will have to handle timestamps that are in different timescales. Timestamps will also need to be converted to other timescales.

CONCLUSIONS

Events happen. It's generally useful to know when they happened. Timestamps are useful for this. There is no useful BCP about what a timestamp should contain. Current timestamps do not, in my opinion, contain enough information to be generally useful. Timestamps should at least contain the system time, the expected offset to true time, an indication of the error bounds for the system timestamp, and the timescale being used. A portable library to operate on these timestamps is needed. This library should handle basic arithmetic on timestamps, comparison of timestamps, and conversion of timestamps from one timescale to another.

APPENDIX: ABOUT NETWORK TIME FOUNDATION

I first began working with NTP in the early 1990s, and eventually, with the encouragement and blessing from Dave Mills, fell in to the roles of autoconfiscator,[*] patch integrator, developer, release engineer, and project manager. Over the years, the rest of the volunteer crew and I noticed our workloads steadily increase. We also noticed the demands of Real Life would encroach on the time we had available to volunteer to work on NTP, and the lack of any legal entity for NTP made it difficult or impossible to accept and receive funds or equipment. This also meant there was no way to pay qualified people to focus and work on NTP long-term.

The clear next step was to create an organization to overcome these limitations. In the process of "defining the mission", I thought about the Old West and stagecoaches. The stagecoach lines generally considered themselves to be in the "stagecoach" business, instead of being in the "transportation" business. Those companies are now long gone. I remembered how, back in the day, there were mainframes, mini-computers, and those small, slow PCs. And Moore's law. I had also put a lot of thought into where NTP was positioned in the network protocol stack and where PTP was in that stack, and just knew that there would soon be places where NTP and PTP would each want to exchange time with the other.

With these thoughts in mind, the nascent NTP Foundation became Network Time Foundation, and was incorporated as a California Public Benefit Corporation in June of 2011. NTF's purpose is to provide direct services and support to improve the state of accurate computer network time-keeping in the general community.

The first project to be supported by NTF was the NTP Project. Shortly thereafter I was chatting with George Neville-Neil, the lead developer of the PTPd Project. George decided that there would be benefits to PTPd joining the party. I had also started communicating with Eric S. Raymond of the GPSD project and Julien Ridoux of the RADclock project, and by the fall of 2011 these projects were on-board as well. Finally, I began discussions with Richard Cochran of the Linux PTP Project, and after some clarifying discussions it joined the coalition as well.

It has been said I have a plan for world domination. Following in the footsteps of Professor Chaos[†] and the Astronomer in Samuel Johnson's "The History of Rasselas, Prince of Abissinia",[‡] I'm pretty sure the reason George, Eric, Julien, and Richard agreed to work with me and NTF even after I told them that some people have said this about me is because of the philosophy I've

---

[*] I converted the NTP codebase to use GNU Autoconf and Automake; the original code required manual editing of a configuration header and possibly some `Makefile`s and only built in the source directory.

[†] http://southpark.wikia.com/wiki/Professor_Chaos_(character)

[‡] http://www.ourcivilisation.com/smartboard/shop/johnsons/prince/chap42.htm

embraced from my late friend and teacher Ron Kurtz,[*] who used to say "I'm in charge, you can do what you want."

Please help NTF help you—support the efforts of Network Time Foundation.[†]

[*] http://www.goodtherapy.org/famous-psychologists/ron-kurtz.html
[†] http://nwtime.org