Leap-second considerations in distributed computer systems

Markus G. Kuhn

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

http://www.cl.cam.ac.uk/~mgk25/

ITU-R SRG 7A – Colloquium on the UTC timescale
Torino, 28–29 May 2003

Applications of synchronized computer clocks

accurate user display, distributed activity scheduling,
unique timestamp generation, mergable distributed logs,
causality checking, replacement for Lamport/vector clocks,
transmission rate control, performance monitoring, interval timing

Typical application needs

- $\longrightarrow$ simple and robust link to civilian time zones
- $\longrightarrow$ monotonicity & robust measurement of time intervals
- $\longrightarrow$ scalar representation
- $\longrightarrow$ compatibility & synchrony
- $\longrightarrow$ no rare events and difficult to test special cases

Computer-clock hazards and disruptions

- $\longrightarrow$ increasingly unpredictable instruction execution times

  preemptive scheduling, context switches, virtual memory, interrupts, power-saving modes, system bus arbitration, cache latency, pipelining, multiprocessing, hyperthreading

- $\longrightarrow$ lack of resolution

  Traditional filesystem timestamp resolution: 1 s

- $\longrightarrow$ lack of synchronization
- $\longrightarrow$ unpredictable latency in communication network
- $\longrightarrow$ crystal frequency error (10^{-4}) and instability (10^{-5})
- $\longrightarrow$ operator error
- $\longrightarrow$ UTC leap seconds

Standard computer clocks are not well-suited or even designed for precision time-interval measurements and are therefore rarely used for this purpose directly.

Scalar time

→ Computer clock = oscillator + counter

→ raw counter value C mapped to standardized scalar time scale T, in the simplest case by a (piecewise) linear relation:

$$T = \frac{1}{f} \cdot C + e$$

→ Traditional human clock-value notation is broken-down time:

YYYY-MM-DD hh:mm:ss.sss

→ Established conventions to map broken-down time to standardized scalar time scale, e.g. POSIX "Seconds since the Epoch".

“Seconds since the Epoch” – POSIX:1996

“A value to be interpreted as the number of seconds between a specified time and the Epoch.

A Coordinated Universal Time name (specified in terms of seconds (`tm_sec`), minutes (`tm_min`), hours (`tm_hour`), days since January 1 of the year (`tm_yday`), and calendar year minus 1900 (`tm_year`)) is related to a time represented as seconds since the Epoch, according to the expression below. [...]

```
tm_sec + tm_min*60 + tm_hour*3600 + tm_yday*86400 +
    (tm_year-70)*31536000 + ((tm_year-69)/4)*86400 ”
```

Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API). ISO/IEC 9945-1:1996 (IEEE Std 1003.1-1996). Same text in earlier versions.

The classic definition of Unix time was unaware of leap seconds. It effectively specified two contradicting values:

- ⟶ A count of SI seconds (leap-second days would count 86401 s)
- ⟶ An encoding of a YYYY-MM-DD hh:mm:ss time value as a single integer, in which both 23:59:60 and the immediately following 00:00:00 are represented by the same value

"Seconds since the Epoch" – POSIX:2001

"A value that approximates the number of seconds that have elapsed since the Epoch.

A Coordinated Universal Time name (specified in terms of seconds (`tm_sec`), minutes (`tm_min`), hours (`tm_hour`), days since January 1 of the year (`tm_yday`), and calendar year minus 1900 (`tm_year`)) is related to a time represented as seconds since the Epoch, according to the expression below. [...]

```
tm_sec + tm_min*60 + tm_hour*3600 + tm_yday*86400 +
    (tm_year-70)*31536000 + ((tm_year-69)/4)*86400 -
    ((tm_year-1)/100)*86400 + ((tm_year+299)/400)*86400
```

The relationship between the actual time of day and the current value for seconds since the Epoch is unspecified.

How any changes to the value of seconds since the Epoch are made to align to a desired relationship with the current actual time is implementation-defined. As represented in seconds since the Epoch, each and every day shall be accounted for by exactly 86400 seconds."

`http://www.opengroup.org/onlinepubs/007904975/basedefs/xbd_chap04.html#tag_04_14`

Handling leap seconds in scalar time

A `get_time()` implementation in an operating systems could

- → jump back 1 s at leap-second start: 23:59:59.1 = 23:59:60.1
- → jump back 1 s at leap-second end: 23:59:60.1 = 00:00:00.1
- → also return a leap-second-in-progress indicator bit
- → stop clock during leap second: 23:59:60.1 = 23:59:60.9
- → block access to clock during leap second
  (system call during leap second will return after 00:00:00.0)
- → suspend all processing during leap second
- → abort with error code
- → overflow minor component in struct: $2 \times 10^9 > \texttt{nsec} \geq 10^9$
- → change clock frequency before, around or after leap second

Existing practice

- ⟶ Berkeley Unix introduced the `adjtime()` system call. Applications can use it to adjust the system clock phase gently. It temporarily alters the system clock frequency by a small percentage to preserve clock monotonicity.

- ⟶ Early network time-synchronization systems used `adjtime()` or equivalents to control system clocks. Clocks would gradually converge after a leap second back to UTC within a few minutes, but not in a compatible way, because of different update times and rates.

- ⟶ David Mills' Unix kernel modification (Mills, 1994) receives a leap-second announcement from NTP software and automatically performs leap second correction at the end of the UTC day. It is today implemented in several widely-used POSIX operating systems.

- → A new special system call `ntp_gettime()` returns "Seconds since the Epoch" with leap-second-in-progress bit, enabling applications to output 23:59:60.

- → Most adoptions of Mills' kernel modification cause the output of the standard clock function to jump back at the start of 23:59:60 by one second, leading to non-monotonic timestamps and unlimited relative errors in time-interval measurements

Proposed solution

Define a standardized variant of UTC for use in computer applications, which adjusts the clock phase gradually by 1 s near a leap second, similar to `adjtime()`, but more exactly and carefully specified.

Smoothed Coordinated Universal Time (UTS)

- → UTS is a YYYY-MM-DD hh:mm:ss clock – like UTC – but seconds are limited to the range 00 to 59 (no leap seconds).
- → UTC can be converted into UTS if there is a warning of an approaching leap second at least 20 minutes in advance.
- → UTS equals UTC, except for a linearly increasing offset of up to 1 s during the last 20 minutes of a UTC leap-second day.
- → On a day with a positive UTC leap second, the last 1000 seconds of UTS are stretched to 1001 ms each, that is the rate of the UTS clock is reduced by 0.1% from 23:43:21 to 24:00:00.
- → On a day with a negative UTC leap second, the last 1000 seconds of UTS are reduced to 999 ms each, that is the rate of the UTS clock is increased by 0.1% from 23:43:19 to 24:00:00.
- → All UTS days account for 86400 seconds, but the last 1000 UTS seconds can differ from UTC/SI seconds by 0.1% in duration.

UTS near positive UTC leap second

```
    UTC                 UTS

23:43:20.000        23:43:20.000
23:43:21.000        23:43:21.000  <- end of UTS = UTC
23:43:22.000        23:43:21.999
23:43:23.000        23:43:22.998
23:43:24.000        23:43:23.997
  ... 995 seconds later ...
23:59:59.000        23:59:58.002
23:59:60.000        23:59:59.001  <- leapsec starts
00:00:00.000        00:00:00.000  <- UTS = UTC again
00:00:01.000        00:00:01.000
```

UTS near negative UTC leap second

```
     UTC                UTS

23:43:18.000      23:43:18.000
23:43:19.000      23:43:19.000  <- end of UTS = UTC
23:43:20.000      23:43:20.001
23:43:21.000      23:43:21.002
23:43:22.000      23:43:22.003
23:43:23.000      23:43:23.004
   ... 995 seconds later ...
23:59:58.000      23:59:58.999
00:00:00.000      00:00:00.000  <- UTS = UTC again
00:00:01.000      00:00:01.000
```

UTS design rationale

- → Linear interpolation avoids discontinuities, ensures monotonicity, and limits errors of time-interval measurements to 0.1%.
- → Linear interpolation is easier to describe, understand, and implement than interpolation with higher-degree polynomials.
  UTS is not intended to control movement of large masses, where more complex interpolation techniques (e.g., B-splines) that minimize control forces might be preferred.
- → Stretching the correction over 1000 seconds leads to nice decimal UTS display values at the start of UTC seconds.
  This would not be the case if correction interval lasted an integral number of minutes.
- → The 1000 seconds advance notice for an upcoming leap second leaves plenty of time for error checking and correction in a noisy time signal that starts to announce leap seconds 59 minutes before the end of the day (e.g., DCF77).
- → Where a UTS clock is used to control the output of audio signals (e.g., Internet telephony), the frequency change remains with 0.1% just below human perception limits (about 0.3%).

$\longrightarrow$ The temporary frequency error introduced by UTS is not worse than $10\times$ the error of the low-cost crystal oscillators commonly used in computers.

$\longrightarrow$ Centering the leap second in the compensation interval would have made it possible to maintain $|\mathrm{UTS} - \mathrm{UTC}| < 500$ ms.

This approach is deliberately not proposed here for two reasons:

- The leap-second warning in most UTC transmissions disappears immediately after the leap second occurs. Keeping the period in which UTS and UTC differ entirely **before** the end of the leap second enables receivers to convert UTC into UTS correctly even if switched on directly after a leap second.
- BBC beeps and many deadlines coincide with the start of a full hour, therefore it is convenient if UTS and UTC are identical there, and not 500 ms apart.

UTS application areas

UTS is intended to be used as the basis for defining the internal clock representation used in information systems that have problems handling UTC leap seconds.

UTS is **not** intended to be used

- → for radio time broadcast signals
- → for the definition of national and regional civilian time
- → inside the NTP protocol
- → in reference clock hardware

Conversion from UTC to UTS will typically be performed in the kernel clock driver of a computer operating system.

Good time signal receivers should be configurable to output any of UTC, UTS, TAI, UT1, etc.

Advantages of UTS over UTC

→ UTS will practically eliminate the potential of leap-second induced disruptions if it becomes the formally standardized, properly documented, commonly used and recommended form of Universal Time for use in distributed computer systems.

→ UTS allows application software, network protocols and programmers to remain completely ignorant of leap seconds.

→ Only few experts will need to be aware of leap seconds and test their designs for them, namely implementors and operators of

- time signal transmitters and receivers
- reference clocks
- operating system kernel clock drivers
- network time synchronization software
- certain distributed scientific instruments

UTS and the need to revise UTC

- → Discussions on redefining UTC are mainly motivated by concerns about hypothetical problems that might arise from the inability of established computer interfaces to handle 23:59:60.
- → Concerns were caused by a lack of formal standards that specify recommended best practice for handling leap seconds in application program interfaces and data formats.
- → Use of UTS as recommended practice will answer sufficiently the concerns of potential leap-second induced disruptions in distributed computer systems.
- → Adoption of UTS as a formal standard will eliminate the most significant motivation for redefining UTC.

Remarks on other reasons to revise UTC

→ Existing satellite navigation systems use both TAI and UTC internally, along with leap-second announcements. They can therefore easily convert between the two internally, thereby circumventing disruptions that could be caused by leap seconds.

Bugs in early GLONASS components that had led to leap-second glitches appear to have been fixed years ago.

See the GLONASS user advisory notes quoted in
`http://www.mail-archive.com/leapsecs@rom.usno.navy.mil/msg00086.html`.

With the help of leap-second announcements, state variables and PN-generator phases can be adjusted instantaneously with modest hardware/software measures, to handle even a signal phase locked to UTC instead of TAI.

→ A leap-second free time scale would certainly simplify the design and use of distributed geophysical and astronomical instruments.

- UTS would there be just as disruptive as leap seconds.
- TAI should be made available more widely, for the benefit of such specialist communities.
- A formally standardized notation for TAI timestamps that differs significantly from notations used with UTC/UTS will help to avoid confusion.
- The needs of small specialist communities can hardly justify the dissociation of civilian time from the rotation of the earth, thereby breaking a tradition that stretches over the entire recorded human history.

Data useful in time broadcast signals

- → UTC and TAI
- → list of all past and announced future TAI−UTC changes
- → UT1 and other earth-orientation parameters
- → list of all civilian time zones in reception range including:
  - DST change-over algorithm
  - history of UTC offset
  - geographic boundary polygons (progressive sampling)
  - names and acronyms (multilingual) for the time zone
  - names, acronyms, established codes (ISO 3166, etc.) and geographic coordinates for associated locations
- → transmitter id, location, service status announcements, etc.
- → regional emergency warning (type, geographic center, radius)
- → weather data (e.g., temperature forecast for heating systems)

LF time signal broadcast

Existing services (JJY, MSF, WWVB, HGB, DCF77, BPC, etc.) grow in popularity, in spite of global UHF satellite navigation signals:

- $\longrightarrow$ longwaves penetrate buildings and mountain ranges well
- $\longrightarrow$ low bitrate and simple code can be processed with the lowest-power microcontrollers (4-bit, 32 kHz, 1k ROM) commonly used in watches
- $\longrightarrow$ miniature ferrite core antennas provide sufficient sensitivity for 50 kW transmitter to cover large parts of a continent
- $\longrightarrow$ cheap AM decoder circuit with low-tolerance components

LF time signals are today used ubiquitously in low-cost mass-market products (wrist watches, alarm clocks, personal computers, etc.)

Problems: diverse frequencies (40–80 kHz), incompatible code formats, disseminated data not comprehensive, lack of global standard.

Wishlist for next-generation LF service

- $\longrightarrow$ single carrier frequency $\Rightarrow$ simple global antenna design
- $\longrightarrow$ 20–50 bit/s data rate, proper use of receiver bandwidth (QPSK)
- $\longrightarrow$ within 10 s: acquisition of crude UTC and TAI ($\ll 10$ ms)
- $\longrightarrow$ within 60 s: UT1, leap second announcement, good UTC/TAI ($\ll 10\ \mu$s) and crude 2D receiver position ($\ll 1$ km) with three transmitters in range
- $\longrightarrow$ within 1000 s: acquisition of full TAI-UTC history, earth-orientation parameters, transmitter status information, descriptions of civilian time zones, better position and time
- $\longrightarrow$ time-division multiple access (TDMA) to permit receiving several stations for location fixing even near one of the transmitters
- $\longrightarrow$ open international standard signal specification, royalty free

Conclusions

- ⟶ Leap seconds and current UTC definition pose no significant problem for distributed computer systems.
- ⟶ Only naïve implementations of leap seconds (23:59:60) in computers with externally synchronized clock cause disruption.
- ⟶ Standard convention (UTS) for a variant of UTC with short-term rate adjustment is an adequate and practical solution.
- ⟶ Timekeeping and timestamping considerations in distributed computers are not a significant reason for decoupling civilian time from earth orientation.
- ⟶ Dropping leap seconds from UTC would not cause noticeable costs in office, telecom and e-commerce applications, either.
  An exception are astronomy and space systems that assume UTC $\approx$ UT1.
- ⟶ For special applications, wider availability of TAI would be desirable (astronomy, geophysics, etc.).

Suggested ITU-R action

→ **No change** to basic principles of 1972 UTC definition.

→ **Standardize a smoothed variant of UTC** for use in applications with low-to-medium frequency accuracy requirements (application program interfaces, asynchronous communication protocols, etc.).

→ **Prepare comprehensive recommendation on time broadcast data** (TAI, leap second announcements, time zones). Recommendation to upgrade existing services to provide conforming data set in backwards compatible way.

→ **Standardize comprehensive time broadcast services** — both standalone and piggy-backed on other services (LF, GSM, WLAN, TV, phone dial tone, etc.). Where feasible also add positioning information.

Drop in ITU-T TF.583 the recommendation that new services should copy the code of an existing service and recommend a new state-of-the-art format instead.

References

- R.A. Nelson, D.D. McCarthy, et al.: *The leap second: its history and possible future.* Metrologia, Vol. 38, pp. 509–529, 2001.
- Dennis D. McCarthy: *Astronomical Time.* Proceedings of the IEEE, Vol. 79, No. 7, pp. 915–920, July 1991.
- David L. Mills: *Unix kernel modifications for precision time synchronization*. Electrical Engineering Department Report 94-10-1, University of Delaware, October 1994.
- Judah Levine, David Mills: *Using the Network Time Protocol (NTP) to transmit International Atomic Time (TAI)*. Proc. Precision Time and Time Interval (PTTI) Applications and Planning Meeting, Reston, VA, November 2000.
- Information technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language]. ISO/IEC 9945-1:1996 (IEEE Std 1003.1-1996).
- Information technology — Portable Operating System Interface (POSIX) — Part 1: Base Definitions. International Standard ISO/IEC 9945-1:2002 (IEEE Std 1003.1-2001). `http://www.unix-systems.org/online.html`
- *Enhanced View of Time Specification*. Version 1.1, Object Management Group, May 2002.
- Markus Kuhn: *Proposed new <time.h> for ISO C 200X*. 1998–2002. `http://www.cl.cam.ac.uk/~mgk25/c-time/`
- Markus Kuhn: *Proposal for a Smoothed Coordinated Universal Time (UTS).* 2000-10-23. `http://www.cl.cam.ac.uk/~mgk25/uts.txt`