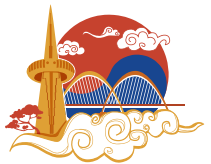
What's the Time?

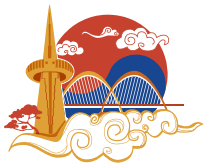
Geoff Huston

APNIC



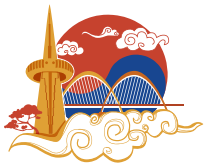
Background

- All computers run with some kind of internal oscillator (called a ‘clock’)
  - This clock manages the internal state changes at each cycle of the central processing unit
  - Clock ‘ticks’ are fed to a digital counter
  - From this counter the computer can maintain a conventional clock and maintain the current time

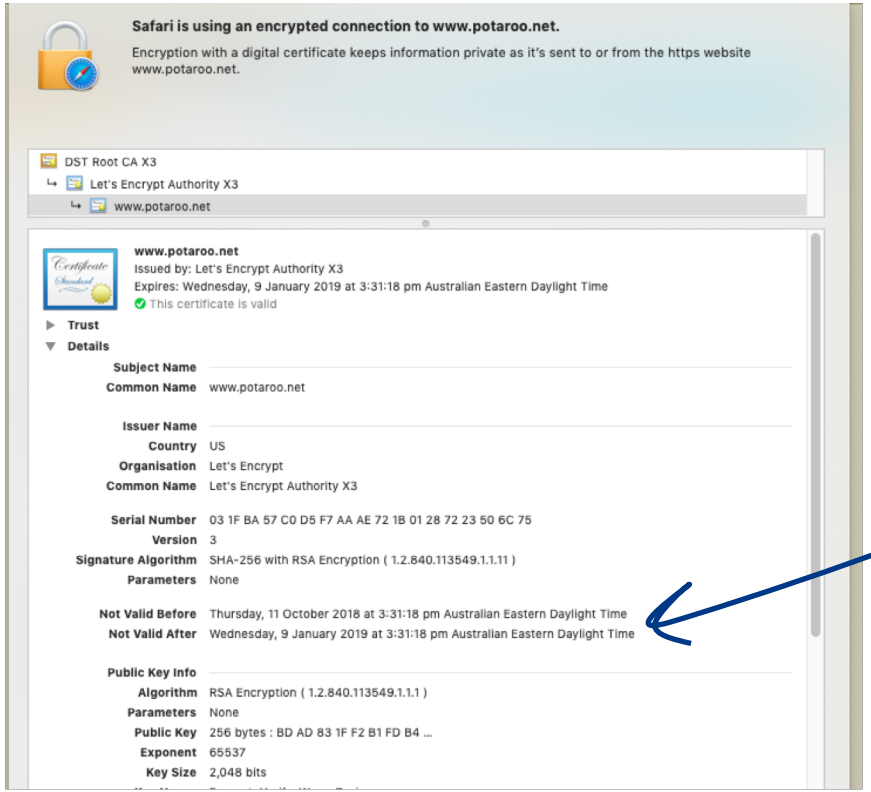


Why is Time useful for a computer?

- To understand when things happen
  - Crontab and event scheduling to ensure that a computer performs certain tasks at precise times
- To understand the relative age of things
  - For example, with NFS file systems its vital to understand which file is more recent
- To understand when things are valid



Security Certificates and Time

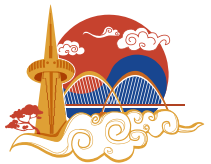


These security credentials are only usable in a defined window of time

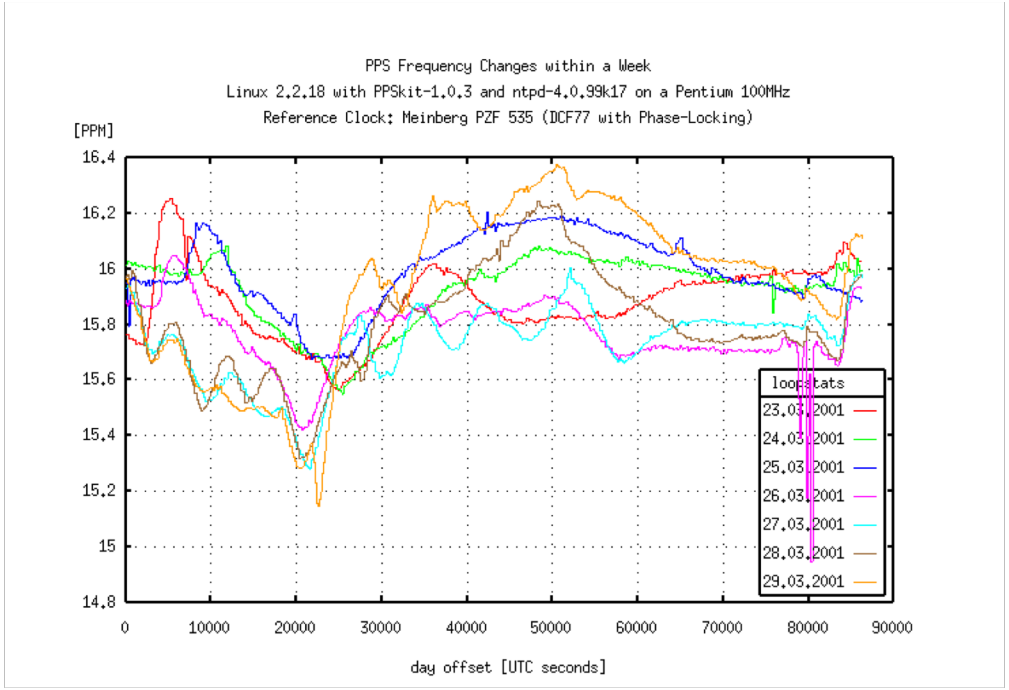
The computer's local clock is compared to these dates to determine whether to trust this certificate or not

So we need to keep "time"

- But this can be challenging
- Computer clocks are based on quartz crystal oscillation
  - Quartz crystal oscillation is only stable if the temperature and excitation voltage are kept stable. Changes in temperature or voltage will cause oscillation changes
- Computer time of day clocks rely on counting ticks in a register
  - Which is performed by software running in the processor at an elevated interrupt level
  - If the processor runs for extended times at an even higher interrupt level then clock ticks can be ‘lost’



Example of Computer Clock Stability

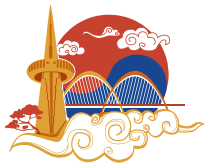


Dave Mill's 2001 experiment on looking at clock stability over a one week period using a Linux PC

https://www.eecis.udel.edu/~ntp/ntpfaq/NTP-s-sw-clocks.htm

So we need to keep "time"

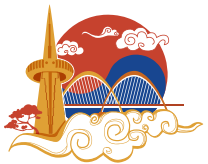
- We actually want to keep **accurate** and **stable** time
  - **Accurate** in that every reference timekeeper keeps the same time
    (modulo the spacetime stretch factors of relativity)
  - **Stable** in that the duration of each measured interval is exactly the same



So we need to keep "time"

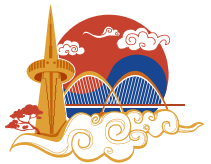
What is “time”?

- We all know that time is divided into days, where a ‘day’ is defined as the duration between successive events when the sun is at precisely the same elevation in the sky
  - But we don’t do this any more because the earth and the sun are poor timekeepers
- We turned to distant quasars as the reference point
  - But we don’t do this any more because we needed even greater precision



So we need to keep "time"

- We turned to nuclear physics:
  - Time is defined using Système International (SI) seconds, defined as the duration of 9,192,631,770 periods of the radiation emitted by a caesium-133 atom in the transition between the two hyperfine levels of its ground state at a temperature of 0K



Keeping Accurate Time

- Not everyone can afford to run their own caesium clock

5071A

Cesium Primary Frequency Standard

Overview Resources Applications Parametric Search Ordering Support

Unsurpassed accuracy, stability and reliability for demanding laboratory and field applications

Accurate, stable, and capable of delivering extremely predictable time and phase for long periods, the 5071A meets the needs of leading-edge metrology and calibration labs. Used with GPS systems, the 5071A provides master clock integration for telecommunication, satellite communication, and navigation systems.
Microsemi's 5071A primary frequency standard delivers exceptional frequency stability. Thanks to a stability specification for 30-day averaging time, the 5071A can keep extremely predictable time and phase for long periods. Combined with a GPS timing receiver the 5071A primary frequency standard can produce a highly robust, inexpensive, and redundant frequency and time system.

Key Features

Frequency accuracy to 5 x 10^{-13}

The intrinsic accuracy of the cesium beam tube assures that any 5071A Option 001 will power up to within 5 x 10^{-13} of the accepted standard for frequency. This is achieved under full environmental conditions in 30 minutes or less- and without the need for any adjustments or alignments.

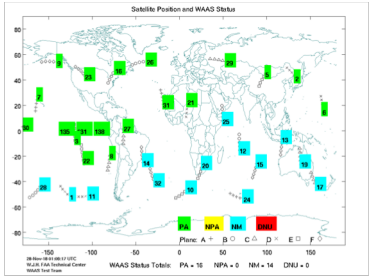
Long-term stability better than 1 x 10^{-14}

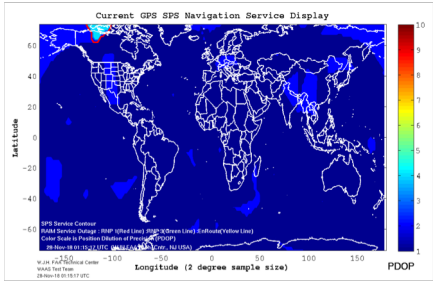
The 5071A Option 001 high-performance cesium beam tube guarantees stability to be better than 1 part in 10^{-14} for averaging times of five days or greater. The 5071A is the first cesium standard to specify stability for averaging times longer than 100,000 seconds (approximately one day).

Keeping Accurate Time

Not everyone can launch their own GPS network

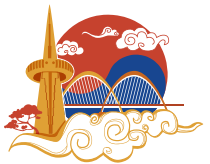
- The GPS satellite constellation is a set of 31 active earth-orbiting spacecraft operated by the US Air Force
- These spacecraft are equipped with Caesium-133 reference clocks that broadcast time signals
- GPS receivers can use triangulation from multiple satellites and delay measurement to determine the receiver's position and provide an accurate reference time





Distributing Accurate Time

- Not every computer runs their own Cesium Clock or runs a GPS receiver to maintain accurate time
  - But some folk do, and that's a good thing!
- So what we would like is a way to take this set of highly accurate reference time sources and provide a mechanism for others to synchronize their local clock against a reference source
- On the Internet we use the Network Time Protocol (NTP) to perform this time synchronization function



NTP Operation

- Time sources are classified by their accuracy
  - A Stratum 0 server is a reference clock (GPS or cesium)
  - A Stratum 1 server is directly connected to a reference clock source
  - A Stratum 2 server receives its time from a Stratum 1 server, and so on
- NTP is a simple clock exchange UDP protocol

Client Offset = ½ ((T2-T1) + (T3-T4))

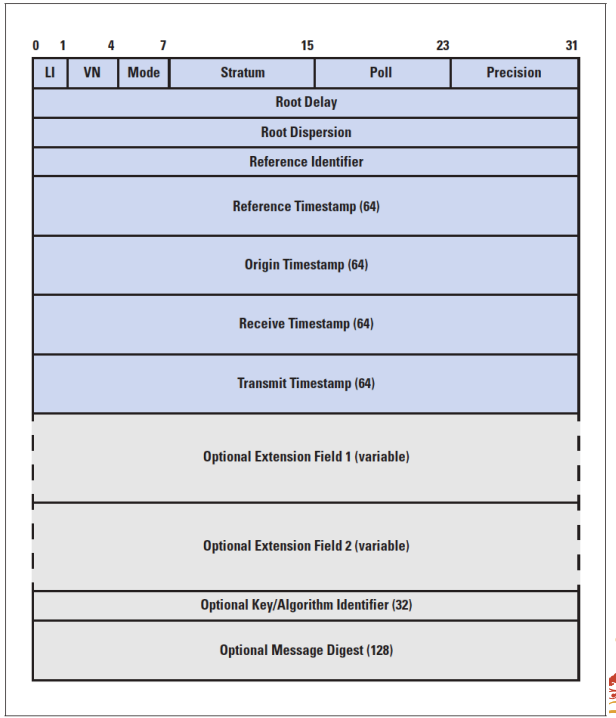
NTP UDP Packets

A 48 byte UDP packet is passed between the client and server

The fields in the packet are:

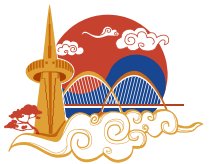
- The header section contains leap seconds, NTP version, NTP mode, Stratum level, polling interval and clock precision
- Server's round trip delay to its reference source and dispersion
- Identification of reference source
- 64 bit reference date (seconds and fractions from 1 January 1900 00:00:00 UTC)
- Time the request left the client
- Time the request arrived at the server
- Time the response left the server

NTP runs UTC – remember this!



NTP Operation

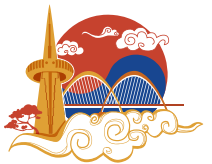
- In steady state the UDP clock packet exchange happens every 16 seconds
  - Faster clock exchanges happen when the client clock has lost synchronisation with the server, and it will burst 8 packets evenly spaced across a 16 second interval
- If the local clock needs to be adjusted the client time application will use adjtime() to slew the local clock. Clock correction is slow – 0.5ms per second
  - Jumping the clock can fatally confuse applications, so this gentle slew is far kinder
- NTP can normally maintain a client clock within a few hundredths of second of the server reference clock



So we all agree on the time?

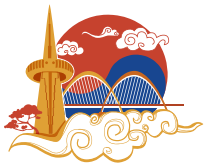
- If everything supports NTP, and there is a well structured mesh of NTP reference clock servers then every connected Internet device that runs a clock should have the same value of time
  - "same" is within a tenth of a second or less

- But does the Internet agree on the time?

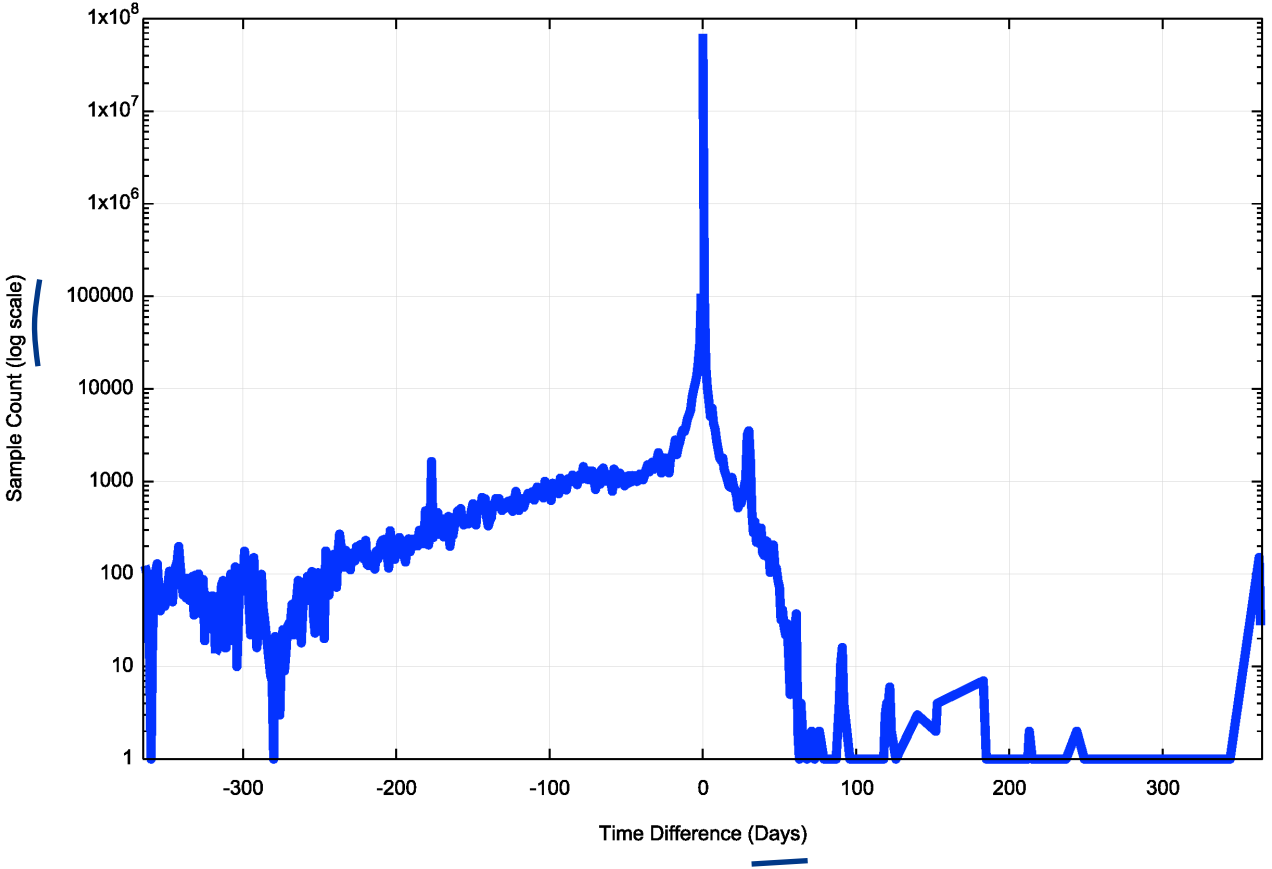


The Experiment

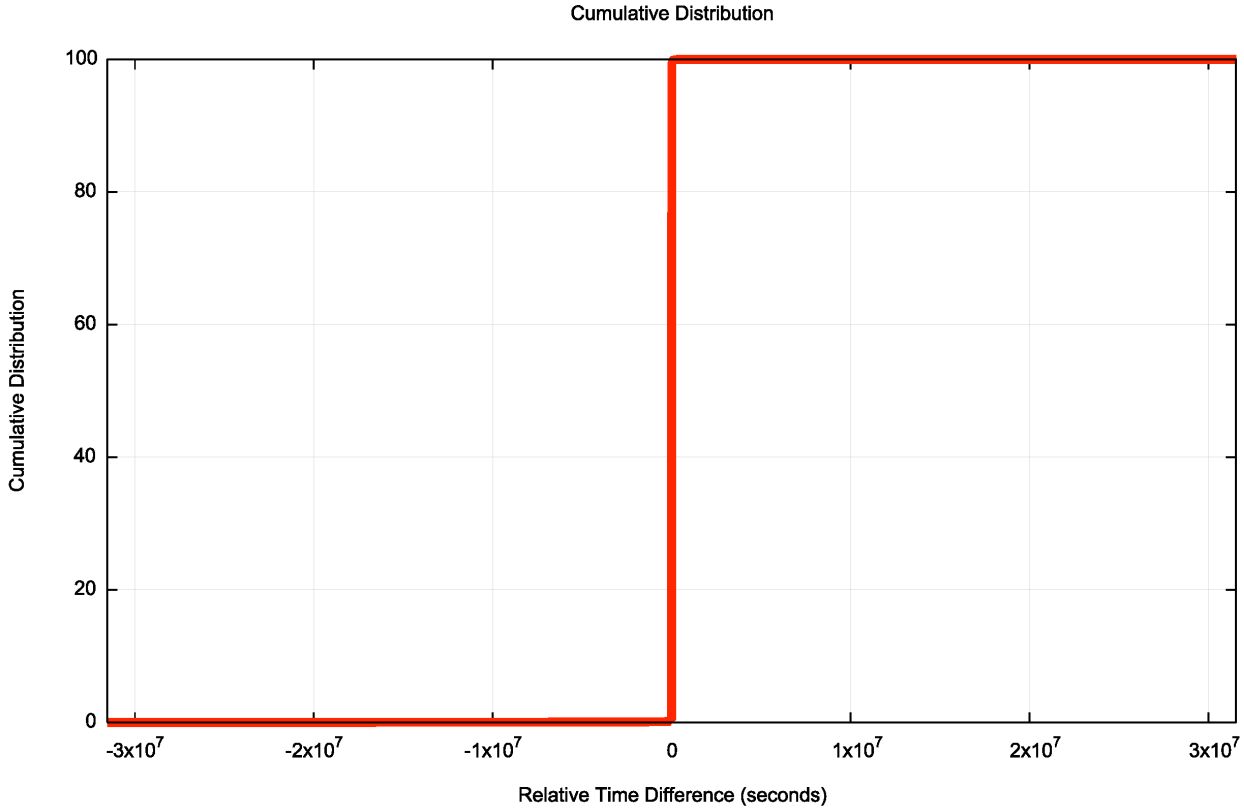
- Use a scripted online ad to direct a client to report back the value of the client's clock
  - Use the Javascript getTime() method to get the local UTC clock value
  - Pass this value to the server as an argument to a URL fetch operation
- Use NTP-managed clock on the server to maintain a stable reference clock
- Record the distribution of differences
  - Ignore the fine-grained differences due to local processing and network propagation time
  - Which means that we are looking at measurements of time within +/- 1 second as being equivalent



Results


Sample Count (log scale)
1x10^8
1x10^7
1x10^6
100000
10000
1000
100
10
1
-300
-200
-100
0
100
200
300
Time Difference (Days)

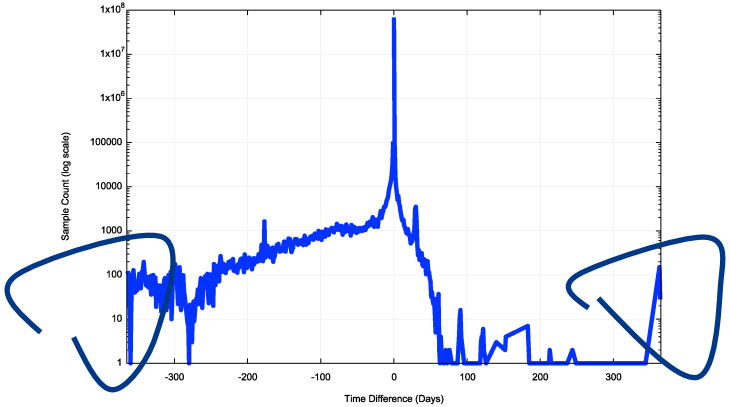
Log Scales can be misleading

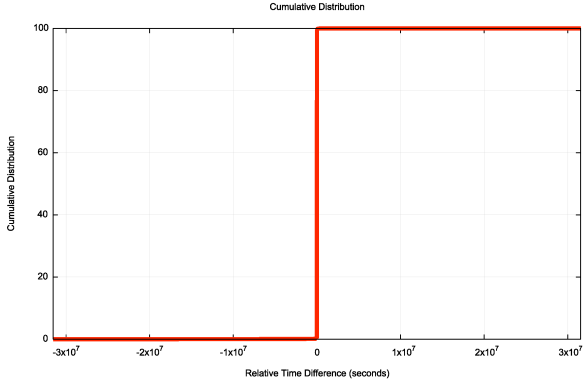


Results

We tested the clock of 202,460,921 clients over a 80 day period:

- 11% of clocks are more than 1 second fast
- 57% of clients are more than 1 second slow
- We observed clock slew values of up to 1 year both fast and slow
- 92% of clients are within 120 seconds of the reference clock



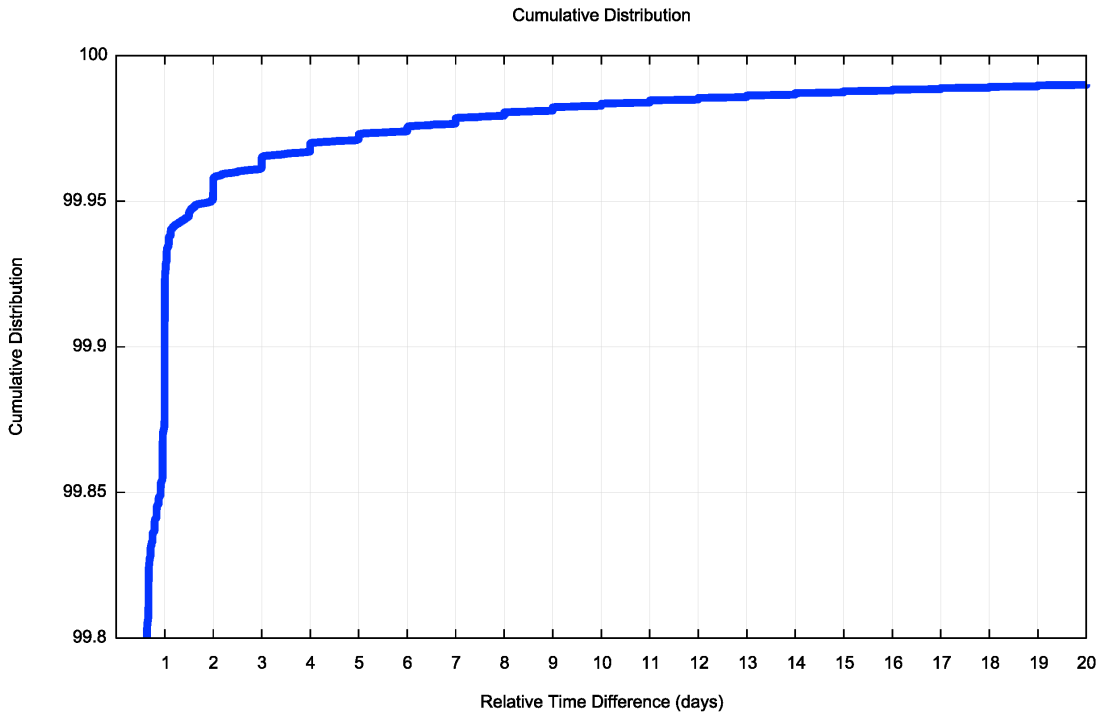


Fast Clocks

0.05% of all clocks are ahead by more than 2 days

There is a clear step function in this distribution that is aligned quite precisely to whole days

How can a client clock maintain a stable per-second clock, yet report a time value that is off by a number of whole days?

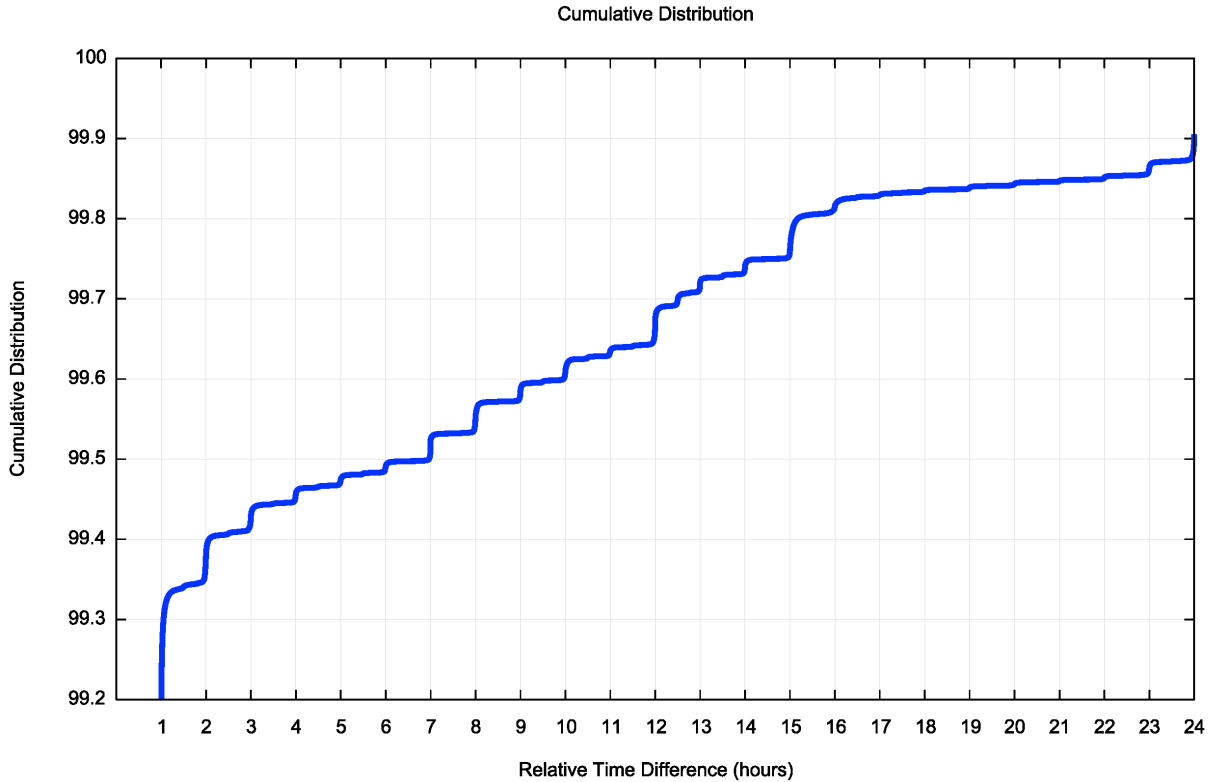


Fast Clocks

0.7% of all clocks are ahead by more than 1 hour

As with the day distribution, there is a marked clustering of the clock offsets into units of hours, and a slightly smaller clustering into half-hours

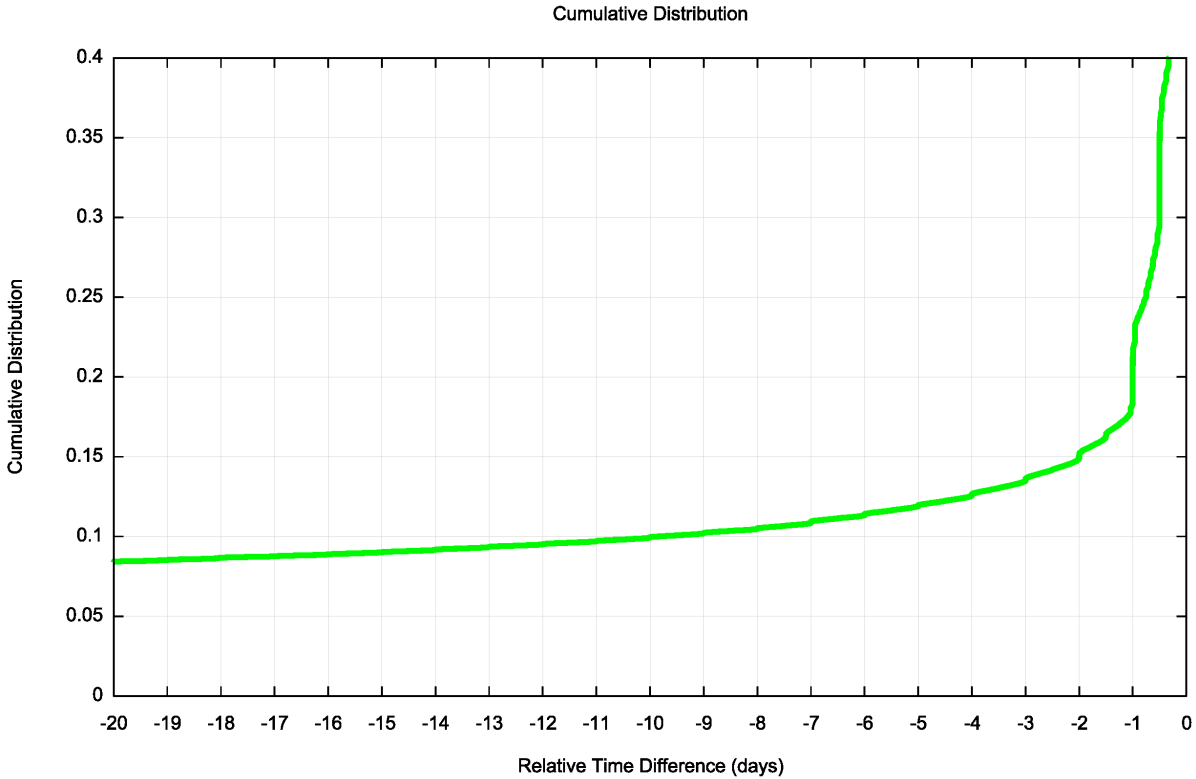
Similar question: How can a client clock maintain a stable per-second clock, yet report a time value that is off by a number of whole hours?



Slow Clock

0.15% of all clocks lag by more than 2 days (3 x the number of fast clocks)

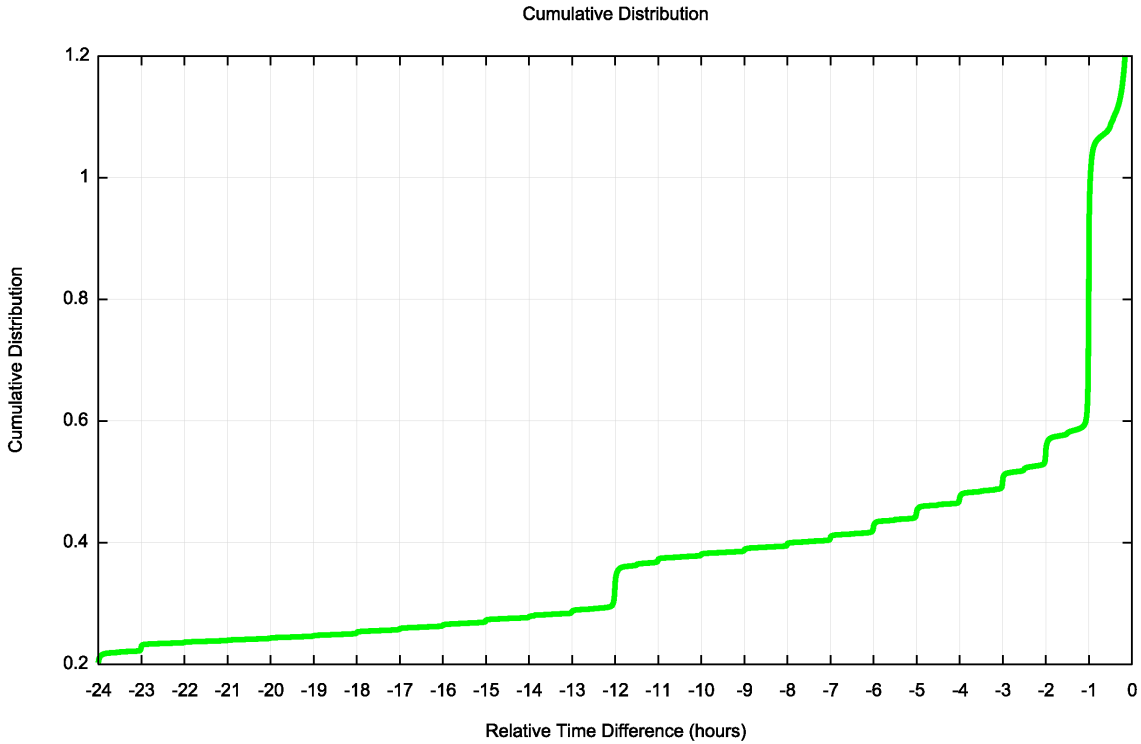
The per-day clustering is not so clear for slow clocks with a lag of greater than 2 days.



Slow Clocks

1.05% of all clocks lag by 1 hour or more

Here there is a marked clustering of the clock offsets into units of hours



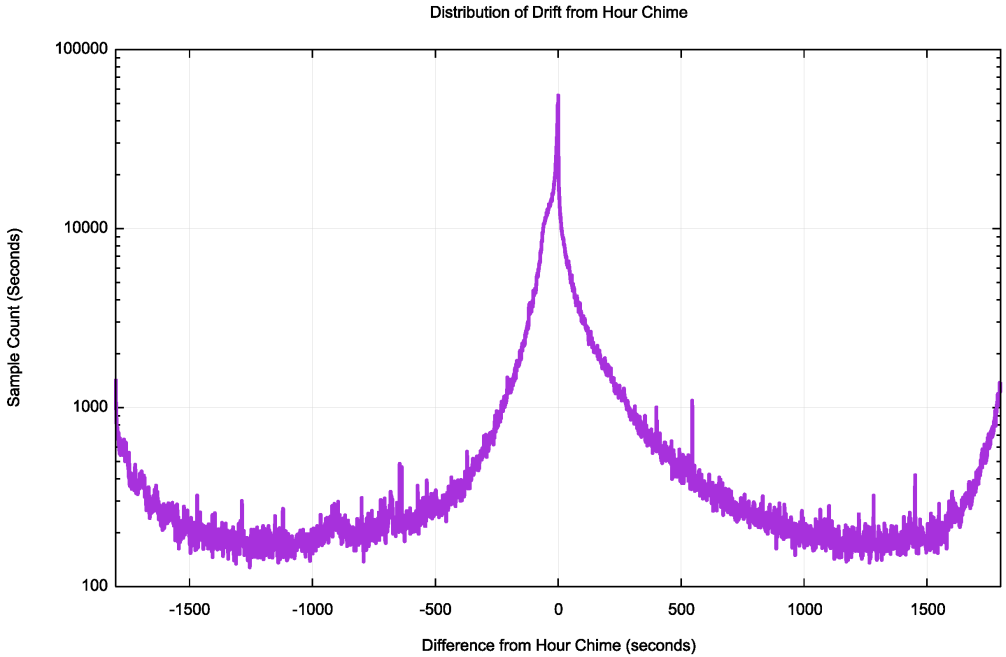
Clustering of Clock Slew Values

This is a distribution of the clock slew values when the whole hours are removed

There is a very strong signal that when a clock has slewed from UTC time it does so in units of hours (and less so in units of half-hours)

NTP does not stabilize a local clock into a slew value of a whole number of hours, so this distribution is not an artefact of NTP.

What is going on here?



Clustering of Clock Slew Values

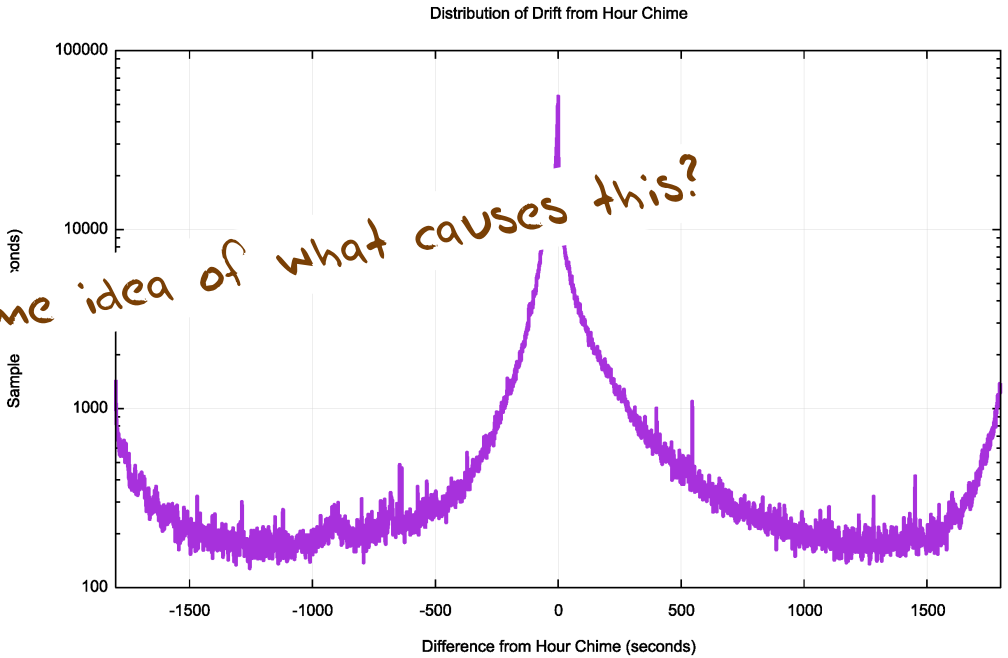
This is a distribution of the clock slew values when the whole hours are removed

There is a very strong signal that when a clock has slewed from UTC time it does so in units of hours (and less so in units of half-hours)

NTP does not adjust the clock into a slew that is a whole number of hours, so this distribution is not an artefact of NTP.

What is going on here?

Does anyone here have some idea of what causes this?

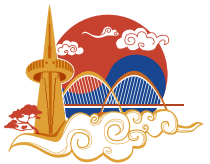


A Possible Theory

Localtime and UTC time are getting confused

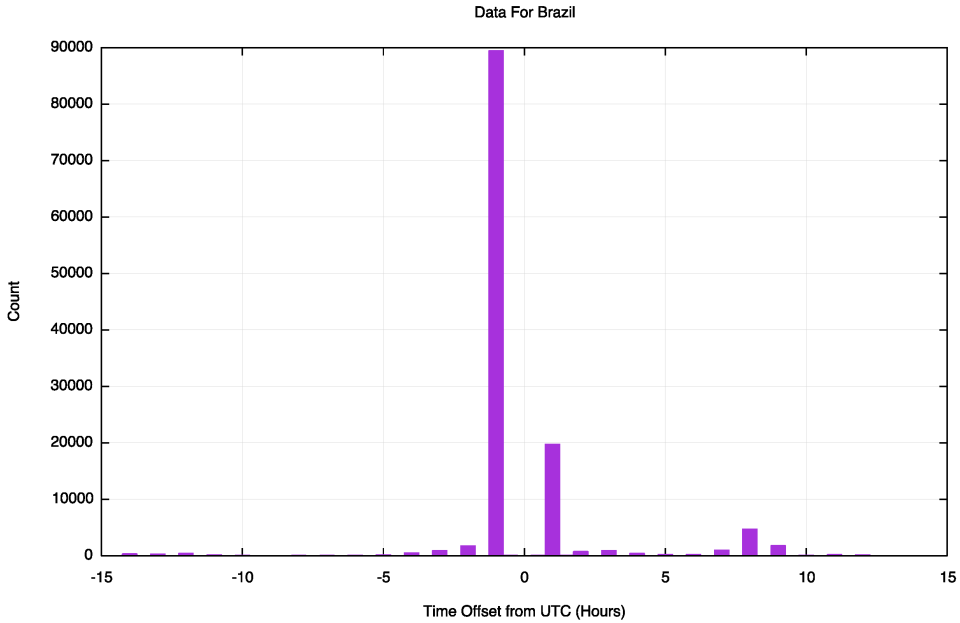
We can test this theory with some additional data

Lets look at 3 countries with a large user population



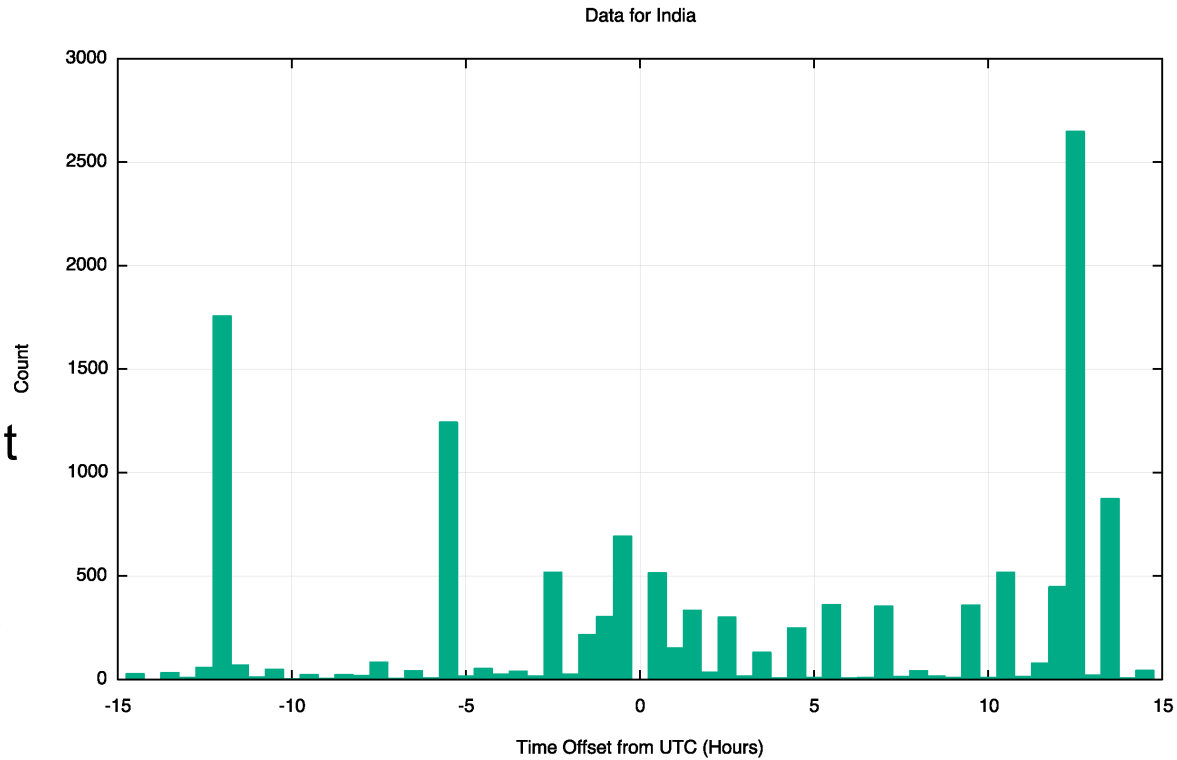
Brazil

- LocalTime is UTC – 2, UTC-3, UTC -4, UTC-5
- DST is variously applied in Brazil
- So we should expect localtime at UTC -1 through UTC-5



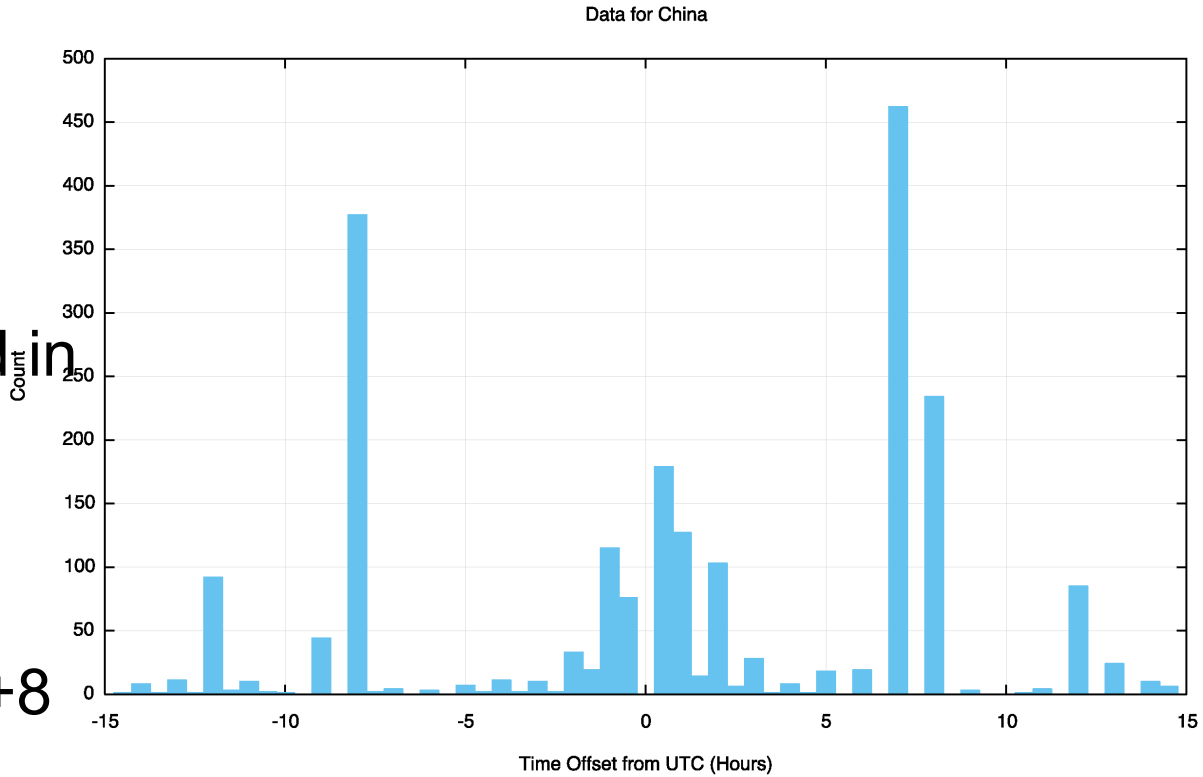
India

- LocalTime is UTC +5:30
- DST is not applied in India
- So we should expect localtime at UTC +5:30
- This is not clearly evident in the data
- There is a strong bias to 30 minute offsets, but no pronounced peak at +5:30



China

- LocalTime is UTC +8
- DST is not applied in China
- So we should expect a peak of localtime at UTC +8

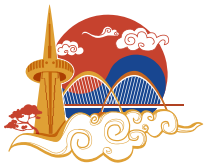


A Possible Theory

Localtime and UTC time getting confused

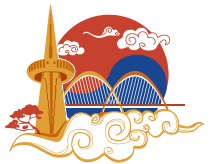
This occurs between 10% to 20% of the time

For the remainder of cases this is not simple clock drift. Some time source is syncing the local hosts UTC clock to the right second, but the hour value of the sync source is incorrect



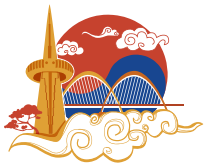
A view of Whole of Internet Time

- Only 58% of visible clients run their clock with 2 seconds of UTC time
- 92% of visible clients run a clock that is within 60 seconds of UTC time
- 98% of clients are within 1 hour of UTC time



If your application's behavior relies on a consistent view of UTC time …

- Its probably a poor idea to assume that all local clocks are tracking UTC time to within 1 hour
- Its probably more robust to work in periods of days rather than seconds, minutes or even hours



Thanks!