Credit card size computer Stratum 1 NTP Server

L P Damaceno, R P Mascarín, S T Müller and D V Magalhães
University of São Paulo, São Carlos, SP, Brazil
luiz.damaceno@aluno.ifsp.edu.br

Abstract. Considering the increasing number of IoT devices, internet-connected computers, smartphone devices, and connected industries, time is an important point to be taken into account. To confidently correlate data from all these devices, time synchronization with a common and global reference (TAI and UTC) is crucial. To provide to internet users our timescale and the benefits of having a traceable cesium frequency standard to BIPM, we assembled a Stratum 1 NTP server with a credit card size computer.

1. A Time Syncing Problem
It is well known how interesting is to have devices synchronized to a common and reliable reference [1], but how one can do this? One way is to have an NTP (Network Time Protocol) time server that keeps tracking of TAI (International Atomic Time) in the master clocking [2]. If we only have an NTP server running in a computer that is previously synced to another NTP server, or manually to UTC (Universal Time Coordinate) or TAI [3], his master clock will drift at different rates [4].

UTC and TAI are the most important timescales in the world, since they reference all computers and devices connected to the internet and are referenced to atomic sources. Computers, smartphones, IoT devices and even high performance hardware, sometimes do not have a good source of clock signal [5]. As a consequence, correlated data, cryptographic rules that are based in timestamping methods and many other things will not work properly [6].

To illustrate this, figure 1 shows a simple schematic and graphic comparison of an NTP client that has the system clock synchronized to the time server and then led to run freely (no NTP clock syncing) [7]. As expected, the time difference increases over time, and only after the NTP automatic synchronization is turned on again and the difference stays near to zero for a while.

2. Raspberry Pi tests
When we started to consider a low-cost a server synced to TAI or UTC the concern was about to remain reliable, with an affordable embedded hardware. The Raspberry Pi (RPi) model showed to be an interesting option, since it is easy to find and has a good processor and hardware to deliver a high number of NTP requisitions per second. Some characteristics of this hardware are (Raspberry Pi 3 B):

- ARM Cortex A53 1.2 GHz Processor;
- 1 GB of RAM;
- Embedded GPU (that is not important for our work);
- Accessible Master Frequency Reference (19.2 MHz crystal oscillator) that can be replaced by an external clock signal;
- GPIOs that can be configured to generate clock signal and receive 1 Hz for synchronization [8].

![Image](image1.png)

**Figure 1.** (a) NTP Server (1) and NTP Client (2) connected through internet to compute the offsets of the free-running clock. The same structure is valid for the client synchronizing the clock, only changing the operating mode; (b) Phase between NTP client unsynced and NTP server until 9,400 data points. Then the NTP client service is turned on and started to sync time every 8 s. The data was acquired with an interval of 6 s. In a period of 56,400 s free-running the client clock delayed 8 ms.

The first tests include generating 1 Hz signal in one of the Raspberry Pi GPIO pins. This frequency should be based on the Linux Kernel timestamp (that is monotonic as the time should be and must have a good resolution (few ns)). The first trial to generate this clock signal was done using a normal process running on the system. We could observe a lot of interference caused by another process concurring for CPU Time. In order to minimize this we installed a Real Time Kernel (Linux RT) and isolated one of the four CPU cores to run only the pulse generation process. This made the pulse generation very deterministic and mitigated the random Interrupt Requests (IRQs) to the CPU. Figure 2 shows the phase comparison scheme we used to evaluate the pulse generation. The Time interval counter used for the characterization is a Stanford SR620 model, with an external timebase from the UTC(LRTE).

The time difference drifting is shown in Figure 3, meaning that the time reference is not yet stable. After 1,400 data points we enabled the NTP service to synchronize again with 1 s per sample. The offset of ~750 μs is the delay computed from network. The default embedded systems and computer timing basis are not stable and will drift with time. Is not a good idea to make an NTP server without any kernel or hardware clock discipline. Considering this, the work described as follow aims to improve the clock reference in the RPi in order to obtain a reliable server.
Figure 2. Schematic used for 1 PPS comparisons.

Figure 3. Raspberry Pi generating 1 PPS after an initial clock synchronization and then led in free-running mode to compare with our stable UTC(LRTE) 1 PPS signal. After some time we enabled synchronization again with an NTP server (another Raspberry Pi with external clock and 1 PPS disciplined input) with 1 s intervals. The acquisition time is 10 s per sample.

3. Raspberry Pi modified clocking
In accordance with the test (RPi free-running), a modified Allan standard deviation plot (Figure 4) shows the clock stability of the system. The clock may vary with temperature, electromagnetic noises, mechanical vibrations, power supply oscillations, hardware load and many other things.

The frequency stability decreases with the sample size, but due to some environmental changes it drifts, decreasing again for samples bigger than 1800 s. This classifies the onboard clock source as a bad reference to be a stratum 1 time server [9] [10] (the clock can’t come from the system time base oscillator because this internal clock varies a lot).
In order to improve the reference source we removed the onboard crystal and provided the RPi with a synthesized signal synchronized to our local timescale UTC(LRTE). A 10 MHz signal from our timescale feeds a synchronization input of a Stanford Research Systems Synthesizer (SR345) that generates 19.2 MHz (the nominal base clock of the RPi) as shown in Figure 5.

![Modified Allan standard deviation plot for the RPi model running with Linux RT.](image)

**Figure 4**. The Modified Allan standard deviation plot for the RPi model running with Linux RT.

![Schematic representation of the structure feeding the time server with UTC(LRTE).](image)

**Figure 5**. Schematic representation of the structure feeding the time server with UTC(LRTE). The 1 PPS signal is connected to GPIO pin 4 of RPi board.

To evaluate if the NTP server is keeping UTC(LRTE) with reasonable stability we run the same program from the first test, generating 1 PPS signal at RPi Time server GPIO output and comparing it with the 1 PPS signal from UTC(LRTE) that synchronizes the system in the initialization [7], as can be seen in Figure 6.

The process of removing the onboard crystal oscillator is very important to feed the system base clock with the same reference that synchronizes the kernel’s 1 PPS. One should be attentive, principally with the temperature of the board (to avoid any damage to other components). The position of the crystal on the board is shown in Figure 7.
Figure 6. The system under test with the RPi referenced to UTC(LRTE) through a 19.2 MHz synthesizer.

Figure 7. In red, the crystal oscillator [11] that should be removed to insert the synthesized signal. The point “1” should be the signal connection and point “2” should be the ground connection. The signal amplitude used is 580 mVRms. One should be aware of values higher than 650 mVRms.

With the system in operation we started to contribute with the “Pool NTP Project” with a stratum 1 time server. The following tests were carried out with heavy I/O and Network load to see the real system limits. We isolated 1 of the 4 RPi’s cores to run only the 1 PPS generation process, while the 3 others deal with the usual IRQs and 3 instances of “RSNTP” server. The “RSNTP” program is a high performance NTP server that allows more than 1 core usage. So we can deliver more time requisitions than running just one instance. Also, the pps-gpio process is running to deliver to “chrony” (the NTP primary server that is running to deliver all time information to “RSNTP”) and synchronizing the kernel timer. A screenshot with the system status can be viewed in Figure 8.

Figure 8. The “chrony” process that is synchronizing the linux kernel time with pps-gpio time. Other servers are for monitoring. The “*” indicates that chrony is synchronized with the LRTE 1 PPS timing.
The phase data for this comparison is shown in Figure 9. Due to the high load on the server and numerous ethernet IRQs on the system we can see some peaks near to 90 μs at the 1 PPS comparisons. These spikes are filtered by the NTP algorithm due the number of samples the system make before any synchronization. The modified Allan Standard Deviation can be seen in Figure 10.

![Figure 9](image1.png)

**Figure 9.** The phase data between UTC(LRTE) Atomic reference 1 PPS and RPi time server 1 PPS.

![Figure 10](image2.png)

**Figure 10.** The modified Allan Standard Deviation of the NTP server if compared with the UTC(LRTE) scale.

With these satisfactory results we are able to distribute time to a large number of clients requesting time synchronization several times per second. The measured bandwidth has peaks of 40,000 NTP packets per second, as seen in Figure 11. The service is provided through “pool.ntp.org” worldwide. The public hostname is “lrte.ntp.ifsc.usp.br” and anyone can connect directly to our time server. With the “pool” the client will be redirected to the loadless and nearby time server. Doing so we have an NTP timeserver with low cost and high performance can be satisfied with this embedded board (RPi).
Figure 11. Our NTP Pool performance graph. Here we can see the high score of our server. In practical world this means our server is extremely stable and reliable.

4. Conclusions
We implemented a low cost and reliable NTP server, using a Raspberry Pi credit card size computer, directly synchronized to our local timescale, UTC(LRTE). We provided a step-by-step evaluation to validate each change made in the board and to develop the whole system. We hope these results can be used to implement many NTP servers and improve time distribution through internet, promoting better timestamps and correlations between different processes.

References
[1] Mills, D. L. Internet Time Synchronization: The Network Time Protocol - Global States and Time in Distributed Systems, 1994.
[2] Mills, D. L. NTP Architecture, Protocol and Algorithms – 2007.
[3] https://www.masterclock.com/company/masterclock-inc-blog/network-synchronization-internet-ntp-servers-vs-gps-ntp-servers (Last Access: June 2019)
[4] Oshima, S. Nakashima. Implementation of the Performance Evaluation System for the NTP Server.
[5] Bulut, H. Implementing a NTP-Based Time Service within a Distributed Middleware System. Proceedings of the 3rd International Symposium on Principles and Practice of Programming in Java, 2004.
[6] Durairajan R. Time's Forgotten: Using NTP to understand Internet Latency - Proceedings of the 14th ACM Workshop on Hot Topics in Networks - HotNets-XIV, 2015.
[7] Deeths, David B. Using NTP to Control and Synchronize System Clocks - Part I: Introduction to NTP. Sun BluePrints, July 2001.
[8] https://www.eecis.udel.edu/~mills/ntp/html/extern.html (Last Access: June 2019)
[9] https://www.worldtimesolutions.com/support/ntp/What_is_NTP_server_stratum.html (Last Access: June 2019)
[10] https://timetoolsltd.com/ntp/what-is-a-stratum-1-time-server/ (Last Access: June 2019)
[11] https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/rpi_SCH_3b_1p2_reduced.pdf (Last Access: June 2019)