Client-Side Implementation of PTP and NTP Clock Synchronization Using Servo-Clock Control

João Martins
MSc in Electrical and Computer Engineering
FEUP
Porto, Portugal
up202107275@up.pt

Lourenço Carvalho
MSc in Electrical and Computer Engineering
FEUP
Porto, Portugal
up202107707@up.pt

***Abstract*—In distributed systems, precise clock synchronization is essential for time-sensitive applications such as industrial automation, telecommunications and financial systems.**

This work implements user-space clients for PTP (IEEE 1588) and NTP using a servo-clock mechanism with a proportional-integral (PI) controller to correct both time offset and frequency drift continuously.

We compare both protocols against real public NTP servers and a local PTP grandmaster, measuring synchronization error under realistic conditions (drift, network jitter, query failures). Performance is evaluated by varying the synchronization interval (5-30 s) with fixed PI gains.

Results show PTP achieves lower steady-state jitter and better long-term stability than NTP in most cases, reaching sub-millisecond accuracy when the servo is properly tuned.

***Index Terms*—Clock Synchronization, Precision Time Protocol, PTP, Network Time Protocol, NTP, Servo Clock, PI Controller**

I. Introduction

Clock synchronization is a fundamental requirement in distributed systems, providing a coherent global time reference that is essential for event ordering, timestamping, resource coordination, and other time-critical operations.

Local clocks are typically implemented using quartz crystal oscillators, which are prone to frequency drift caused by manufacturing variations, temperature fluctuations, and aging. As a result, clocks gradually diverge over time, even after initial synchronization.

Two widely used protocols address this problem:

- Network Time Protocol (NTP) – provides millisecond-level accuracy over wide-area networks.
- Precision Time Protocol (PTP, IEEE 1588) – achieves microsecond or sub-microsecond accuracy, especially in local networks.

This project implements client-side synchronization for both protocols using a servo-clock, which is controlled by a proportional-integral (PI) feedback loop. Unlike basic offset correction approaches, the servo-clock continuously adjusts the clock frequency to compensate for both time offset and frequency drift.

II. Problem Definition

The objective of this project is to achieve high-accuracy clock synchronization within a local area network (LAN) using standard publicly available PTP and NTP servers. The synchronization is performed from the client side, without relying on specialized hardware.

PTP is specifically designed to provide sub-microsecond precision, particularly when hardware timestamping is available at both the master and client clocks. In contrast, NTP is robust, widely deployed, and sufficient for millisecond-level accuracy over the Internet, but it generally cannot match PTP's precision in a LAN environment.

Several key challenges must be addressed to achieve reliable synchronization:

- Handling network jitter and asymmetric delays, which can introduce temporary or systematic time offsets.
- Compensating for systematic frequency drift in local clocks, which can be significant (i.e. 500ppm)
- Managing large initial time offsets between the client and server clocks (i.e. 20s)
- Dealing with query timeouts, packet loss, and temporary server failures, which can disrupt continuous synchronization.

The proposed solution employs a servo-clock controlled by a proportional-integral (PI) controller, whose performance depends critically on three parameters: the proportional gain K_p, the integral gain K_i, and the synchronization interval between successive corrections.

This approach builds upon prior work in user-level PTP/NTP simulation [1], as well as hybrid strategies that combine PTP and NTP to leverage the strengths of both protocols [3].

III. Solution Description & Limitations

The proposed solution is implemented entirely in Python and organized into clear, modular components, allowing flexible configuration of time sources, clock models, and experiment parameters.

The workflow is divided into two main experiments: the first focuses on tuning the servo-clock PI controller, and the second compares NTP and PTP under realistic operating conditions.

A. Time Source Abstraction

An abstract `TimeSource` class provides a unified interface for querying both NTP and PTP servers. Key features include:

- timeout protection to prevent blocking on unresponsive servers,
- consecutive failure counting to trigger fallbacks or alerts,
- fallback to the last known good time in case of errors,
- optional Gaussian noise simulation to emulate realistic measurement jitter.

The NTP client queries a rotating pool of public servers using a standard SNTP-style four-timestamp offset calculation, while the PTP client reads the current master offset via the `pmc` tool from the `linuxptp` package.

B. Servo Clock with PI Control

The servo-clock maintains a virtual clock with:

- the current time,
- a configurable base drift (in parts per million),
- dynamic frequency correction computed by a proportional-integral (PI) controller.

The PI control law is expressed as:

$$u = K_p \cdot e + K_i \cdot \int e \, dt, \tag{1}$$

with basic anti-windup protection to prevent instability.

The effective clock rate of the servo becomes:

$$\text{Clock rate} = 1 + \text{drift} + \text{rate correction}, \tag{2}$$

which continuously adjusts the frequency to reduce both offset and long-term drift.

C. Experiment Framework

Experiments are run for 180–300 seconds, with the system state sampled every 0.1 seconds. Synchronization events occur at configurable intervals, and performance is evaluated against a free-running baseline clock that applies only the configured drift.

The first experiment (*Experiment 1*) evaluates servo tuning. Using either an NTP or PTP time source, different PI gains (K_p, K_i) are arbitrated at a fixed synchronization interval to observe the offset evolution of the servo-clock. Both the proportional and integral gains, as well as the synchronization period, are varied to identify stable and convergent configurations.

The second experiment (*Experiment 2*) directly compares NTP and PTP. Using the stable PI gains identified in Experiment 1, the synchronization error is measured for several fixed sync intervals (5–30 s) to evaluate each protocol's accuracy, jitter, and convergence behavior.

D. Important Limitations

Due to network driver limitations and hardware constraints, all experiments were performed on a single physical Linux machine.

Originally, the project baseline envisioned two separate machines on the same LAN: one acting as a dedicated PTP grandmaster and the other as a PTP client (slave).

However, due to the lack of suitable hardware, we ran the PTP grandmaster inside a virtual machine on the same host. This introduced additional jitter, delay asymmetry, and software-only timestamping on both master and client sides.

Consequently, this setup cannot fully replicate a dedicated PTP environment with hardware timestamping, and the observed results represent a realistic yet constrained scenario.

IV. Results

Multiple experiments were conducted to evaluate servo tuning behavior and protocol comparison.

A. Servo Tuning

The first experiment focused on tuning the servo-clock using NTP as a time source. Its main goal was to evaluate the impact of different proportional (K_p) and integral (K_i) gains, combined with various synchronization intervals, on the servo clock offset evolution.

The base configuration for all variations was: a sampling period of 0.1 s, a total duration of 180 s, a base drift of 500 ppm, an initial offset of 20 s, a start delay of 5 s, and an offset noise standard deviation of 200 μs.

Five servo variations were tested: three with a 10 s synchronization interval (overdamped, critically damped, underdamped), one fast sync at 5 s, and one slow sync at 20 s. Each variation was designed to highlight typical behaviors: overdamped response converges slowly, critically damped converges quickly without overshoot, and underdamped exhibits oscillations.

Figure 1 shows the offset evolution for all tested variations. It is clear that the critically damped combination (K_p=0.07, K_i=0.025, sync=10 s) provides a smooth and fast convergence, while aggressive gains show small oscillations and conservative gains converge more slowly.

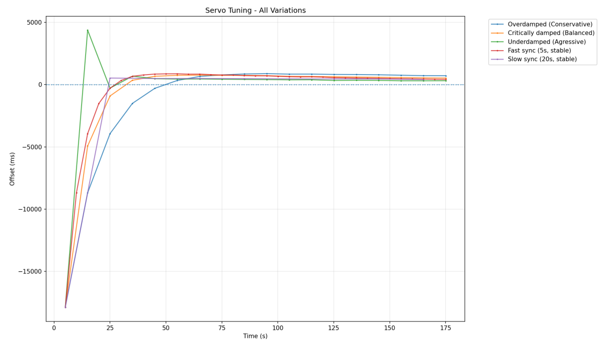

Fig. 1. Offset evolution for all servo tuning variations.

Figure 2 presents the time evolution for this critically damped variation. The red line represents the real time, the dashed blue line shows the servo clock without compensation (baseline drift only), and the green line depicts the servo clock after PI correction. The corrected servo closely tracks real time, demonstrating the effectiveness of the chosen PI parameters.

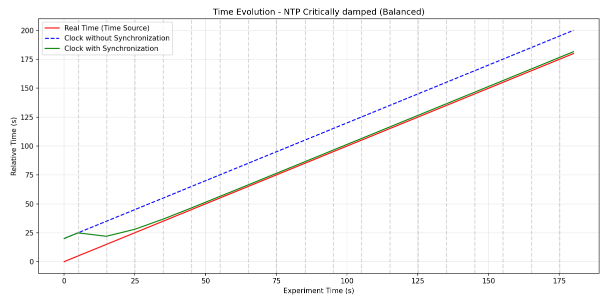

Fig. 2. Time evolution for the critically damped variation.

B. NTP vs PTP Comparison

The second experiment compared NTP and PTP synchronization using a fixed, stable PI configuration (K$_p$=0.05, K$_i$=0.015) for various synchronization intervals. Each variation was run for 300 s with a sampling period of 0.1 s, initial offset of 20 s, base drift of 500 ppm, start delay of 5 s, and offset noise standard deviation of 200 μs.

Figure 3 shows the offset evolution of NTP and PTP servo clocks for this configuration. The curves are almost identical, reflecting the stable PI parameters; visually, no difference is apparent. However, as reported in Tables I and II, PTP still achieves slightly lower RMS and jitter values numerically.

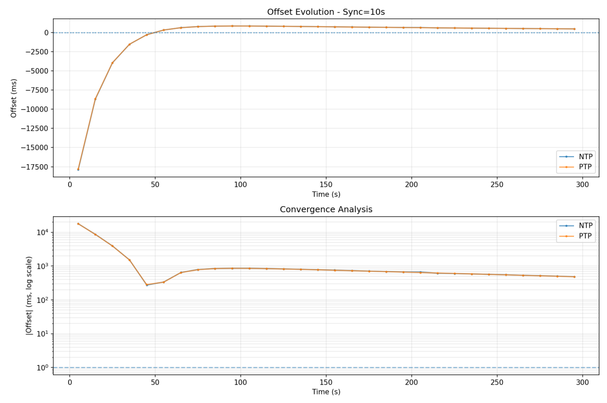

Fig. 3. Offset evolution of NTP and PTP servo clocks for a 10 s synchronization interval.

PTP generally shows better steady-state performance, especially in terms of jitter at shorter sync intervals. Large initial transients are caused by the 20 s initial offset combined with 500 ppm base drift and software-only timestamping. At longer intervals, both protocols exhibit similar RMS errors, but PTP maintains slightly lower jitter.

TABLE I
NTP STEADY-STATE PERFORMANCE SUMMARY

Sync (s)	SS-RMS (ms)	SS-Jitter (ms)	Max Offset (ms)	Settle (s)
5	2486.61	2470.84	17852.49	N/A
10	552.40	41.65	17889.58	45.10
20	377.33	25.14	17889.04	25.00
30	335.50	119.33	17889.79	125.10

TABLE II
PTP STEADY-STATE PERFORMANCE SUMMARY

Sync (s)	SS-RMS (ms)	SS-Jitter (ms)	Max Offset (ms)	Settle (s)
5	594.11	102.86	17789.39	175.10
10	552.28	45.20	17789.08	45.00
20	375.22	15.92	17789.84	25.00
30	335.56	111.95	17789.67	125.10

V. Critical Analysis & Conclusions

The results presented in Tables I and II clearly show that PTP consistently achieves better steady-state performance than NTP, especially at shorter synchronization intervals. For instance, at a 5 s sync period, PTP reduces the RMS offset from 2486.61 ms (NTP) to 594.11 ms, and jitter from 2470.84 ms to 102.86 ms. The unusually large NTP values at this interval are explained by server rotation: the NTP client occasionally switched between different public servers due to query failures, introducing additional offset and jitter.

Under ideal conditions – with a dedicated two-machine LAN, a proper PTP grandmaster, and hardware timestamping NICs – both protocols would be expected to perform significantly better. PTP could achieve sub-millisecond RMS offsets and minimal jitter, while NTP would benefit from a stable, local server without network-induced variability. Our single-machine, VM-based setup artificially increased jitter and limited the achievable precision.

The servo-clock mechanism substantially improves performance over simple offset corrections, continuously compensating for both initial offset and frequency drift. Its effectiveness is evident in the time evolution plots of Section **??**, where critically damped PI parameters allow the servo clock to closely track real time after the initial transient.

Key parameters influencing servo performance are:

- proportional and integral gains (K_p, K_i),
- synchronization request interval.

Despite these improvements, some limitations affected the results:

- experiments relied solely on software timestamping, without hardware support,
- the PTP grandmaster ran inside a VirtualBox VM, adding additional jitter and delay asymmetry,
- a single shared network interface was used (host + guest),
- the experiments were conducted on a single machine rather than a true multi-node LAN.

These constraints can explain some anomalies, such as the slightly higher jitter of PTP at 10 s intervals compared to NTP, and the prolonged settling time for the 30 s interval in both protocols. Nevertheless, PTP consistently outperforms NTP under most conditions, particularly at aggressive synchronization rates.

In conclusion, closed-loop servo control significantly enhances both protocols by reducing offsets and jitter, while PTP remains the preferred choice for applications requiring the highest timing accuracy. Future work should include hardware timestamping NICs, adaptive PI tuning, and experiments across multiple physical machines in a real LAN environment to achieve the expected performance predicted by theory and prior studies.

VI. Contributions

Approximate workload distribution:

- Lourenço Carvalho: 50% – core implementation, experiments
- João Martins: 50% – testing, data analysis, report writing

References

[1] F. Terra, M. Cardoso, and P. Castro, "User-level Implementation of Precision Time Protocol," Faculty of Engineering, University of Porto, 2023.

[2] A. Perez, "Synchronization IEEE 1588 Mechanism," in *Implementing IP and Ethernet on the 4G Mobile Network*, Elsevier, 2017, pp. 253–267.

[3] M. Lichvar, "Combining PTP with NTP to Get the Best of Both Worlds," Red Hat Blog, Jul. 2016. [Online]. Available: https://www.redhat.com/en/blog/combining-ptp-ntp-get-best-both-worlds