Raspberry Pi-Based Systems as Precise Time Servers

1. Introduction

"A man with a clock knows what time it is – a man with two clocks is never sure." –Segal's Law

Clock synchronization is a critical component in financial trading, where precise timing ensures the integrity and efficiency of trading operations. Historically, trading firms used mechanical time-stamping devices to record transaction times, allowing for market pattern analysis, efficiency evaluations, and the prevention of fraudulent activities such as frontrunning. With the evolution from mechanical to electronic, distributed systems, the need for precise timestamps has grown exponentially. Microsecond misalignment between clocks in electronic trading systems can lead to erroneous order sequences and substantial financial losses, as most exchanges prioritize orders based on price and then time. Such discrepancies can compromise a firm's financial stability and credibility, and invite regulatory scrutiny.

The trend towards ultra-low latency (ULL), high-frequency trading (HFT) has further amplified the need for accurate clock synchronization. Regulatory bodies worldwide have established stringent standards to ensure synchronization precision. For instance, the European Union's MiFID II regulations mandate market participants to synchronize business clocks to within 100 microseconds of Coordinated Universal Time (UTC). In the United States, the Financial Industry Regulatory Authority (FINRA) and the Securities and Exchange Commission (SEC) have set regulations requiring synchronization within 50 milliseconds of the official US time provided by the National Institute of Standards and Technology (NIST). These regulations aim to ensure fairness and transparency in the markets by maintaining a unified time frame for all participants.

Fault tolerance and traceability are also vital for clock synchronization for financial trading. Timekeeping software like Network Time Protocol (NTP) and Precision Time Protocol (PTP) can under-report errors or fail silently, which can have severe consequences in a trading environment. Financial institutions are advised to use multiple independent clock sources to mitigate the risk of relying on a single point of failure. A minimum of tricyclic redundancy from different physical locations or technologies (i.e. GPS, terrestrial signals), enhances the reliability and stability of the timekeeping infrastructure. Ensuring all time sources are traceable back to an official standard, such as UTC or NIST, is also mandated by regulatory bodies. A universally accepted alternative traceable time source is satellite time, such as that from GPS, due to it's accuracy and reliability. Satellite time and lab-based time are typically within a few nanoseconds of each other and national laboratories provide official validation of satellite time's accuracy, making it traceable to official standards like UTC and NIST.

The evolution of timestamping technology from mechanical devices to sophisticated electronic systems reflects the advancing needs of the financial trading industry. The adoption of stringent regulations, comprehensive record-keeping practices, fault-tolerant systems, and the establishment of traceable time sources are all testament to the critical role that accurate clock synchronization plays in maintaining the fairness, efficiency, and integrity of financial markets. As electronic trading continues to evolve and expand into new asset classes, the demands on timekeeping accuracy and reliability will only continue to increase, underscoring the ongoing importance of this foundational element of financial trading infrastructure.

The adaptation of small-scale, cost-effective solutions like Raspberry Pi-based systems as time servers represents a significant evolution in the approach to clock synchronization. These systems, equipped with GPS receivers and network capabilities, provide a practical and affordable method for achieving synchronization to a highly precise standard. The Raspberry Pi's ability to run full-fledged NTP or PTP services could allow even small trading firms or individual traders to meet the stringent timing requirements imposed by regulatory bodies, without the need for expensive proprietary time servers.
In this project, we evaluate the use of Raspberry Pi-based servers as PTP GPS Grandmaster Clocks. The Raspberry Pi, a cost-effective single board computer, presents a flexible and affordable solution for precise time synchronization. By configuring the Raspberry Pi with GPS modules and PTP software, we aim to achieve high precision and reliability in timekeeping. This project begins by exploring the potential of Raspberry Pi-based systems to serve as primary or backup time servers in financial trading environments.

2. Background

The following sections define and elaborate on crucial concepts of this project.

2.1 Timing Fundamentals

Historically, a second was defined by subdividing the day into 24 hours, 60 minutes per hour, and 60 seconds per minute. However, this method was subject to the Earth's varying rotational cycle. Today, the definition of one second is far more precise—based on 9,192,631,770 oscillations of a caesium-133 atom, a standard established due to the irregularity in Earth's rotation and the necessity for more consistent timekeeping, especially for technological and scientific applications.

2.1.1 Clocks and Clock Error A clock is any device that measures and displays time. Theoretically, a perfectly accurate clock, displaying the wall-clock time, does not exist in practice due to inherent inaccuracies termed as clock error or skew. Clock error consists of two main components: - **Drift**: A predictable error that changes over time with factors like temperature, voltage, and age. - **Jitter**: The unpredictable, rapid variance in clock error, contributing to its randomness.

2.1.2 Clock Synchronization

- **NTP** operates over UDP and can synchronize time with a precision of within a few milliseconds over the public internet, and slightly better over local networks.
- **PTP** is designed to provide very high precision time synchronization, typically within a sub-microsecond range, over a local area network (LAN). It achieves higher accuracy by using a master-slave architecture where one or more master clocks distribute time to slave clocks.

Reason for using PTP primarily:

1. Unlike NTP, PTP devices actually timestamp the amount of time that synchronization messages spend in each device, accounting for device latency.
2. NTP networks have extra latency and less accuracy simply because they're software-based, and all timestamp requests have to wait for the local operating system. PTP provides the level of time synchronization precision needed for high-frequency trading firms, especially for latency arbitrage.

Since we want to mimic "trading at the speed of light", high time precision is crucial to the low-latency nature of our operations. Therefore, we have chosen to use PTP primarily for our project.

2.1.3 Time Standards and GPS Global timekeeping standards such as UTC (Coordinated Universal Time) integrate leap seconds to align with Earth's rotation. Meanwhile, GPS (Global Positioning System) provides a practical application of advanced timekeeping, where each satellite carries multiple atomic clocks. These satellites broadcast time signals with precise timestamps, enabling global synchronization and positioning.

2.1.4 Oscillators in Timekeeping Oscillators are crucial in timekeeping, providing the fundamental frequency from which time intervals are measured. There are several types of oscillators used in timekeeping devices:

- **Quartz Oscillators**: Widely used in consumer electronics due to their stability and low cost. Quartz oscillators exploit the piezoelectric properties of quartz crystals, which oscillate at a stable frequency when an electric field is applied. There are several types of quartz oscillators, tailored for different stability and environmental requirements:
  - **Simple XO (Crystal Oscillator)**: The most basic form of quartz oscillators, providing a stable output frequency under constant conditions but susceptible to temperature changes and environmental factors.
  - **TCXO (Temperature Compensated Crystal Oscillator)**: Enhances the basic crystal oscillator by including a temperature-sensitive component that adjusts the oscillator's output to minimize frequency variations due to temperature changes.
  - **VCXO (Voltage Controlled Crystal Oscillator)**: Allows the frequency of the oscillator to be adjusted by varying a control voltage, useful in applications where frequency modulation or phase-locked loop (PLL) adjustments are needed.
  - **OCXO (Oven Controlled Crystal Oscillator)**: Provides superior frequency stability by housing the crystal in a temperature-controlled chamber, or "oven," which maintains a constant temperature regardless of external conditions. This type is ideal for telecommunications and precision equipment where high stability is critical.

- **Atomic Clocks**: Represent the pinnacle of precision in timekeeping, used primarily in global navigation satellite systems like GPS and in scientific research. Atomic clocks count the oscillations of atoms, such as caesium or rubidium, under specific conditions, offering stability of about one second in millions of years.

2.1.5 Time in Computers

Time is managed through several interconnected systems:

- **System Clock**: The primary timekeeping device on a motherboard, typically a quartz crystal oscillator, which regulates the timing of all operations within the computer, including the processor speeds and data bus timing. The frequency of this clock determines the speed at which the CPU operates.
- **Real-Time Clock (RTC)**: A battery-powered clock, separate from the main system clock, which keeps track of time even when the computer is powered off. It is crucial for maintaining system time across reboots and power failures.
- **Time Synchronization in Networks**: Computers often use protocols like NTP to synchronize their clocks to a more accurate external time source, typically an NTP server linked to an atomic clock. This is essential in environments where precise timing is crucial for data integrity and coordination, such as in financial transactions or data centers.

The combination of these systems ensures that computers not only maintain accurate time during operation but also across restarts and shutdowns, adapting to the requirements of both general and specialized applications.

2.2 Hardware

2.2.1 Raspberry Pi 4 Model B

The Raspberry Pi 4 Model B represents a significant advancement in single-board computing, offering enhanced performance and versatility in a compact package. The Raspberry Pi 4 is ideal for multimedia and IoT. Its affordability and extensive connectivity options make it a popular choice for hobbyists, educators, and professionals alike.

2.2.2 Raspberry Pi Compute Module 4

The Raspberry Pi Compute Module 4 marks a significant evolution in embedded computing, offering enhanced performance and flexibility in a compact, industrial-grade form factor. Its modular design, with options for different memory configurations and onboard eMMC storage, makes it well-suited for custom-built embedded systems and IoT projects.

2.2.3 Raspberry Pi 5

Raspberry Pi 5 features the Broadcom BCM2712 quad-core Arm Cortex A76 processor @ 2.4GHz, making it up to three times faster than the previous generation. With RAM variants up to 8GB, this is the fastest, smoothest Raspberry Pi experience yet.

2.2.4 Comparison of Raspberry Pi-Based Platforms

Feature	Raspberry Pi 4	Compute Module 4	Raspberry Pi 5
SOC	Broadcom BCM2711	Broadcom BCM2711	Broadcom BCM2712
CPU	Quad core Cortex-A72 (ARM v8) 64-bit @ 1.8 GHz	Quad core Cortex-A72 (ARM v8) 64-bit @ 1.8 GHz	Quad-Core Cortex-A76 (ARM v8) 64-bit @ 2.4 GHz
Networking	Gigabit Ethernet Power-over-Ethernet	Gigabit Ethernet	Gigabit Ethernet Power-over-Ethernet (new version)
Clock	N/A	Real time clock (Supports hardware timestamping)	Real time clock (Supports hardware timestamping)
Geekbench Single-Core	~259	~317	~789
Geekbench Multi-Core	~671	~683	~1650

2.2.5 u-Blox 5 and u-Blox 9 The u-blox LEA-5T only supports GPS (United States), whereas the EVK-F9T supports multiple Global Navigation Satellite Systems (GNSS), including GPS, GLONASS (Russia), Galileo (European Union), and BeiDou (China). This makes the EVK-F9T much more ideal for global deployment. Additionally, the EVK-F9T is unaffected by ionospheric errors. The LEA-5T uses the GPS L1 frequency and only supports single-frequency GPS. In contrast, the EVK-F9T supports L1/L2 for GPS and GLONASS, E1/E5b for Galileo, and B1I/B2I for BeiDou. Furthermore, the EVK-F9T supports high-precision positioning with GNSS corrections and can track multiple constellations simultaneously, thereby improving positioning accuracy and reliability .

The LEA-5T has 16 channels and an accuracy of around 2.5 meters, while the EVK-F9T has 184 channels and offers astonishing accuracy of 0.01 meters + 1 ppm when using RTK positioning. The LEA-5T offers a time pulse accuracy of less than 30 nanoseconds, with a configurable time pulse frequency of up to 10 Hz. In comparison, the EVK-F9T provides a significantly improved time pulse accuracy of less than 5 nanoseconds and a configurable time pulse frequency of up to 25 Hz. Additionally, the EVK-F9T features a holdover specification of less than 1 microsecond over 24 hours, ensuring precise timing even during signal interruptions.

The more advanced EVK-F9T supports both multi-band RTK, which offers centimeter-level accuracy, and PPP, allowing for enhanced precision in positioning applications. The LEA-5T lacks both of these features. In terms of data rates, the LEA-5T supports a maximum update rate of 5 Hz, whereas the EVK-F9T significantly outperforms this with a maximum update rate of 20 Hz, making it suitable for applications that require high-frequency data acquisition, such as “trading at the speed of light.”

2.3 System Optimizations

2.3.1 Busy Poll Detection Switching from interrupt-driven to busy poll-driven detection of the PPS (Pulse-Per-Second) output presents a viable optimization strategy for enhancing the precision and responsiveness of the GPS grandmaster system. In this approach, instead of relying on interrupts to detect the rising edge of the PPS signal, a dedicated core continuously monitors specific memory locations where the PPS signal is mapped. By employing a busy poll-driven mechanism, the system eliminates the latency associated with interrupt handling and context switching, allowing for near-instantaneous detection of the PPS signal’s rising edge. This method ensures that the system captures the precise moment when the PPS signal transitions from low to high, indicating the start of a new second. As a result, the system achieves higher accuracy in timestamping events and synchronizing time across networked devices, crucial for applications demanding stringent timing requirements, such as telecommunications and high-frequency trading. Additionally, by dedicating a core solely to this task, the system can operate with minimal interference from other processes, further enhancing its reliability and consistency in timekeeping operations. Overall, by leveraging busy poll-driven detection, the system can achieve superior performance in maintaining precise time synchronization, thereby enhancing the overall effectiveness of the GPS grandmaster solution.

2.3.2 Recompiling: Cache Fitting Examining the recompilation of libraries and daemons to adjust parameters such as code size versus speed performance offers a nuanced approach to optimizing the GPS grandmaster system. By modifying compilation settings and potentially leveraging assembly optimizations, developers can tailor the performance characteristics of critical components to better suit the specific requirements of the application. In scenarios where maintaining consistent and predictable timing is prioritized over sheer computational speed, optimizing for smaller code size becomes crucial. Smaller code footprints increase the likelihood of instructions fitting within processor caches, thereby reducing memory access latency and enhancing overall system responsiveness. By recompiling libraries and daemons with an emphasis on minimizing code size, developers can mitigate the risk of cache misses and ensure more reliable execution within tight timing constraints. By strategically adjusting compilation parameters to prioritize code size over raw speed, developers can achieve a more finely tuned balance between performance and predictability, ultimately enhancing the effectiveness and reliability of the GPS grandmaster solution.

2.3.3 Recompiling: Remove Logging Optimizing code to improve performance involves a meticulous examination and refinement of the software’s logic and structure. By identifying and eliminating inefficiencies such as unnecessary logging or less efficient functions, developers can significantly enhance the overall performance and responsiveness of the GPS grandmaster system. One aspect of optimization involves scrutinizing the logging mechanisms within the codebase and removing any extraneous or verbose logging statements that do not contribute essential information for system operation or debugging purposes. Streamlining the logging process helps reduce the overhead associated with I/O operations, thereby improving the system's throughput and responsiveness, especially in high-demand scenarios.

2.3.4 Recompiling: Targeting Raspberry Pi Architecture Ensuring compilation targets the exact chip being used, rather than a generic ARM core, is essential for maximizing the performance and functionality of the GPS grandmaster system, particularly on hardware platforms like the Raspberry Pi. While ARM processors share common architecture features, different chip variants may incorporate additional instructions, optimizations, or hardware accelerators unique to their specific implementations. By targeting the exact chip model in the compilation process, developers can take full advantage of the hardware capabilities provided by the Raspberry Pi's ARM processor. This includes optimizing code to utilize specialized instructions, leveraging hardware-accelerated features such as NEON SIMD instructions for parallel processing, and accessing peripherals or hardware components specific to the Raspberry Pi's chipset.

2.3.4 Parameter Tuning The adjustment of parameters such as frequency governor and thermal controls on the Raspberry Pi plays a crucial role in managing the variability in runtime. Modern CPUs are designed with dynamic mechanisms to regulate their clock speeds based on factors such as temperature and real-time usage. By fine-tuning the frequency governor settings, we can control how the CPU adjusts its clock speed in response to workload demands. Similarly, optimizing thermal controls ensures that the CPU operates within an optimal temperature range, preventing thermal throttling that can significantly affect performance. These adjustments not only enhance the stability and efficiency of the GPS-based PTP server but also contribute to minimizing variations in runtime, thereby improving overall system reliability and performance consistency.

2.3.5 Thermal Regulation Utilizing custom busy spin code executed on non-critical cores of the Raspberry Pi aids the thermal regulation of the surrounding environment. This method is to generate controlled amounts of heat, contributing to the stabilization of the surrounding thermal environment. By strategically deploying this code, we can actively manage temperature fluctuations around critical hardware components, such as the Raspberry Pi and GPS receiver, promoting a more thermally stable operating environment. An unstable thermal environment can induce fluctuations in the performance of crystal oscillators, which are critical components for generating stable clock signals in electronic devices, including those used in GPS grandmaster systems. Crystal oscillators rely on the piezoelectric properties of quartz crystals to maintain precise frequency stability. However, variations in temperature can affect the crystal's resonant frequency, leading to fluctuations in the output frequency of the oscillator. When exposed to temperature changes, the quartz crystal expands or contracts, causing shifts in its resonant frequency. This phenomenon, known as temperature-induced frequency drift, can result in deviations from the desired clock frequency, impacting the accuracy of timekeeping in the GPS grandmaster system.

2.3.6 Dedicated CPU Core for PPS Signal Polling By isolating a CPU core exclusively for PPS signal polling, interruptions from other processes can be minimized, ensuring precise timing for critical operations. Utilizing the isolcpus kernel parameter enables the isolation of a CPU core dedicated solely to polling the PPS signal. By isolating a specific CPU core, we effectively minimize interruptions from other processes that may be running concurrently on the system. This dedicated core ensures that critical timing operations, such as processing GPS data and maintaining precise time synchronization, receive uninterrupted attention. As a result, the system can operate with enhanced accuracy and reliability, as it is shielded from the variability introduced by competing processes vying for CPU resources.

2.4 Tools for Synchronization

GPSd:

GPSd serves as a fundamental bridge between GPS receivers and the synchronization process. The input to GPSd consists of data streams from GPS receivers, typically transmitted using communication protocols such as NMEA. Acting as an interface layer, GPSd takes the raw data input from GPS receivers, which includes satellite position, time, and other relevant information, and processes it into a standardized format that can be easily utilized for time synchronization purposes. GPSd also provides APIs for client applications to retrieve GPS data in a structured and accessible way.

Linux-PTP:

Linux PTP (Precision Time Protocol) provides the necessary tools and protocols for achieving precise time synchronization across networked devices. It enables the distribution of highly accurate time information over Ethernet networks. The input to Linux PTP typically consists of timestamped PPS (Pulse-Per-Second) signals from GPS receivers, which serve as the primary reference for time synchronization. Linux PTP includes tools like `ptp4l` and `phc2sys`, which help in synchronizing system clocks to a master clock over a network using the IEEE 1588 PTP standard.

chrony:

Chrony provides robust time synchronization capabilities essential for maintaining accurate time across networked devices. It utilizes a combination of techniques, including NTP (Network Time Protocol) and PTP (Precision Time Protocol), to achieve precise timekeeping. Chrony continuously evaluates and adjusts the system clock based on the input received, such as GPS receivers, NTP servers, and reference clocks, to ensure synchronization with the most accurate time source available. It is particularly effective in environments with intermittent connectivity or variable latency.

ethtool:

Ethtool is a utility found in Unix-like operating systems that provides the ability to control and query Ethernet-based network interface controllers and drivers. In our project, ethtool plays a vital role in ensuring that the network interface is optimally configured and functioning correctly.

ubxtool:

Ubxtool is a utility for configuring and managing u-blox GPS receivers. It allows users to send configuration commands, switch between different operating modes, and query the status and settings of the GPS device. Ubxtool can be used to enable or disable specific features, adjust navigation parameters, and save configuration settings to the GPS receiver's non-volatile memory. It is essential for customizing the behavior of u-blox GPS modules to fit specific applications.

tcpdump:

TCPDump is a powerful command-line packet analyzer tool that provides the capability to intercept and display the TCP/IP and other packets being transmitted or received over a network to which the computer is attached . In our project, the primary purpose of using TCPDump is to monitor and analyze the Precision Time Protocol (PTP) traffic, which is essential for achieving precise clock synchronization across network devices. PTP operates primarily on UDP ports 319 and 320, which are crucial for the timestamping necessary for clock sync techniques.

Example usage: sudo tcpdump -i eth0 -n udp port 319 or udp port 320 -vv Port 319: Used for PTP event messages (such as timestamps). Port 320: Used for PTP general messages.

3. Methods

3.1 Local Systems

The Local Set Up utilized a Raspberry Pi 5 configured to serve precise time synchronization. The system employed a u-blox LEA-5T-0-003 GPS module and a 28 dB GPS Active Antenna for NMEA and PPS signals. This was interfaced with the Raspberry Pi 5 serial connection for ubx Binary data and GPIO pin 4 for PPS. This grandmaster clock was directly connected to the client Raspberry Pi 5 via a static ethernet connection.

`ubxtool` was used to disable NMEA messages and enable UBX's proprietary Binary messages. `gpsd` was then used to manage the GPS data which was referenced by `chrony` to set system time. LinuxPTP's `phc2sys` was used to adjust the Real-Time Clock based on system time which was then transmitted as a PTPv2 packet by `ptp4l.`

The grandmaster was set to masterOnly, while the client device was set to slaveOnly. On the client device, the ptpv2 packets were received via the network stack by `ptp4l.` This was used to set the hardware clock. `phc2sys` referenced the hardware clock to adjust system time.

3.2 Lab Systems

The Lab Set Up employed a Raspberry Pi 4 Model B configured as a grandmaster clock to provide precise time synchronization. A u-blox EVK-F9T-00b GNSS module is connected to the Raspberry Pi via USB and GPIO for GPS data and PPS signals, respectively. A 28 dB GPS Active Antenna was also utilized. GNSS data was transmitted via USB and PPS signals through GPIO. As with the local set up, UBX Binary messages were utilized over NMEA. All GPS data was managed by `gspd` and provided to `chrony` for system time synchronization. `ptp4l` in software timestamping mode was used to distribute ptpv2 time packets to the client, the Raspberry Pi Compute Module 4 (CM4).

Both devices were connected via Ethernet, facilitated by a TP-Link TL-SG105E switch. The CM4 received these packets using `ptp4l`. The CM4 then used `phc2sys` to adjust its system time based on the hardware clock (/dev/ptp0).

This setup ensures precise time synchronization between the devices, leveraging both software and hardware timestamping. The PTP messages are exchanged over a TP-Link TL-SG105E switch, facilitating PTP over Ethernet communication.

3.3 Data Collection

We used a custom Bash scripts running on both the grandmaster and the client devices in both set ups. The tools we used include:

PTP Utilities: The `ptp4l` and `phc2sys` utilities are the most important part of the project for collecting PTP synchronization data. These tools allowed us to monitor and log the synchronization performance, including offsets, frequency corrections, and path delays, which were critical for assessing the effectiveness of our synchronization optimizations.

vcgencmd: This is used to measure system temperature. Monitoring temperature is essential as it can impact the performance and stability of hardware during intensive synchronization tasks.

htop: To monitor CPU usage, we used the top command, which provides a dynamic view of a running system. It was configured to output CPU utilization, helping us understand how much processing power is consumed during the synchronization processes.

systemctl: This is used to restart services, ensuring a clean state and consistency.

journalctl: To capture logs specifically related to ptp4l, we used journalctl, which helps in filtering and extracting relevant log entries based on timestamps. This ensures that we collect only the pertinent data within the specified duration of our tests.

3.3.1 Local Set Up

The `v4_metrics_ptp.sh` script was used to collect extended PTP data, including path delays, basic system metrics such as temperature and CPU utilization, and synchronization details for both, the grandmaster and the client.

Before data collection, all relevant system services were restarted to determine how long the devices took to synchronize. Data collection on both devices occurred simultaneously and was collected for 3 hours per trial. Latency, mean path delay, and offset information was collected continuously, while system metrics were sampled every 5 seconds.

The following trials were conducted:

Test #	Description
1	Baseline without any optimization
2	On grandmaster: pinned GPSD to CPU core 1, Chrony to core 2, PTP4l and phc2sys to core 3. On client: pinned LinuxPTP to CPU core 2, phc2sys to core 3
3	Based on test 2, steered PPS interrupt to the GPSD core (core 1) on grandmaster
4	Based on test 3, added active cooler thermal control for both grandmaster and client
5	Based on test 3, steered PPS interrupt to the Chrony core (core 2) on grandmaster
6	Ran test 2 again

Test #	Description
7	Assigned a dedicated CPU core for PPS interrupts and added interrupt steerings other than PPS (UART and Ethernet) On grandmaster: assigned CPU core 1 for PPS interrupts; GPSD and Chrony pinned to core 2, UART interrupts to core 2 PTP4l and phc2sys pinned to core 3, Ethernet interrupts steered to core 3, all other interrupts to core 0
8	Based on test 7, used P2P Layer 2 to connect the grandmaster and the client
9	Based on test 8, added Clock Servo to compensate PTP clock error
10	Based on test 9, added Idle CPU Busy Spin to generate heat for thermal stability
11	Based on test 7, added Idle CPU Busy Spin to generate heat for thermal stability

3.3.2 Lab Set Up The `v5_metrics_ptp.sh` script was used to collect extended PTP data, including path delays, basic system metrics such as temperature and CPU utilization, and synchronization details for both the grandmaster and the client.

Before data collection, all relevant system services were restarted to determine how long the devices took to synchronize. Data collection on both devices occurred simultaneously and lasted for 3 hours per trial. Latency, mean path delay, and offset information were collected continuously, while system metrics were sampled every 5 seconds. Additionally, the script gathered NTP metrics on the grandmaster, including offset, jitter, root delay, and root dispersion, providing a comprehensive dataset for analyzing the time synchronization performance of the setup.

The following trials were conducted:

Test #	Description
1	Baseline without any optimization
2	Second baseline without any optimization
3	On grandmaster: no optimization. On client: isolated CPU cores using `isolcpus`
4	Isolated CPU cores on both grandmaster and client. On grandmaster: pinned GPSD and Chrony to core 2, PTP4l to core 3. On client: pinned ptp4l to core 2, phc2sys to core 3
5	Similar to trial 4, but steered all IRQs to CPU 0, and steered eth0 IRQ to CPU 1
6	On grandmaster: pinned Chrony to core 1, GPSD to core 2, PTP4l to core 3; steered all possible IRQs to core 0 except PPS to core 1. On client: steered eth0 IRQ to core 1, pinned ptp4l to core 1, phc2sys to core 3; steered all other IRQs to core 0

3.4 Data analysis

Initially, a paired sample t-test was considered to compare the changes in offsets pre- and post-optimization. This test is generally used under the assumption that the differences between paired observations are normally distributed. However, the Shapiro-Wilk test conducted on our dataset indicated that the differences do not follow a Gaussian distribution, which is a critical assumption for the validity of the paired t-test. Therefore, we decided to use Wilcoxon's signed-rank test and the Mann-Whitney U test as alternatives. They are both non-parametric statistical tools used to compare two groups, which do not require the assumption of normal distribution and are robust against outliers and skewed data.

Wilcoxon Signed-Rank Test: This test is used instead of the paired sample t-test when the distribution of differences between paired observations (e.g. offsets before and after optimization) is not normally distributed. It is ideal for our needs as it compares two related samples or repeated measurements on a single sample.

Mann-Whitney U Test: This test is used for comparing two independent samples, which is suitable for us because our optimizations were applied independently at different times, ensuring the observations are independent.

By using these tests, we can confidently assess whether the observed improvements in synchronization accuracy are statistically significant, thereby validating the effectiveness of our optimizations in a rigorous manner. If the p-value from the tests is less than the significance level (set at 0.05), we reject the null hypothesis that there is no difference before and after the optimization, leading to the acceptance of the optimization.

3.5 Optimizations

For our systems, we implemented the following optimization techniques:

Dedicated CPU Core for PPS Signal Polling

We implemented the dedicated CPU optimization by utilizing the `isolcpus` parameter in `/boot/firmware/cmdline.txt`. The config file will run on each reboot of the raspberry PI and all user process will only be assigned to CPUs that are not in the isolated CPU list. For our implementation, we selected CPU core 1, 2, and 3 as the isolated CPU cores and only CPU 0 is not an isolated CPU. All irrelevant process will now be assigned to CPU 0, leaving process on the dedicated CPU for the Grandmaster Server or Client uninterrupted.

```
isolcpus=1-3
```

To run a process on a dedicated CPU, we use the `taskset` command. This command allows user to launch a new command with a given CPU affinity by modifying the service file for each daemon by including CPUAffinity.

Parameter Tuning: CPU Fan Thermal Control

We implemented the CPU fan thermal control by setting the Raspberry Pi's Fan control parameters. The `dtparam` in `/boot/firmware/config.txt` is a set of parameters controlling different hardware setting of a raspberry pi and we utilized the `fan_temp` parameter family: `fan_temp` is the temperature threshold of the fan speed activation; `fan_temp_hyst` is the temperature hysteresis, which defines the fan deactivation threshold by `fan_temp - fan_temp_hyst`; `fan_temp_speed` is the fan speed on the given temperature threshold. Our parameter tuning controls the temperature to about 45-50 celcius.

Parameter Tuning: CPU Clock Control

Similar to CPU Fan Thermal Control we control the CPU clock in `/boot/firmware/config.txt`. By default Raspberry Pi 5 runs on a CPU clock of 240 MHz. By setting the `arm_freq` and `over_voltage_delta` parameter, we can raise the CPU clock to 300 MHz and increase the voltage by 50 mV.

Thermal regulation: Busy Spin on Unused CPU

We implemented thermal regulation with PWM and PI control.

Pulse Width Modulation (PWM) emerges as a versatile technique pivotal in modern electronics for efficiently controlling analog circuits using digital signals. By swiftly varying the width of pulse signals, PWM enables precise manipulation of power delivered to devices such as, in our case, CPU heat generator.

The Proportional-Integral (PI) controller is a feedback controller that dynamically adjusts its output based on the error between the desired setpoint and the measured process variable. By incorporating both proportional and integral terms, this controller not only responds to current errors but also considers past deviations, offering improved stability and robustness.

To increase the temperature with busy spin code, we run a integer multiplication as the payload. Then we incorporate PWM. For every time interval of 0.1 second, we can control the percentage of time (duty cycle) that the CPU is fully commited to run the payload and the percentage of time that the CPU is idle. By control the duty cycle, we can control the CPU usage rate to generate heat.

We then control the CPU usage rate with PI control. By setting the target temperature to 50 celcius, the CPU usage rate is the temperature difference between the current temperature and the target temperature, plus a discrete integral of all temperature difference. Thus we have the logic presented as the pseudo code below.

```
error = targetTemp - cpuTemp
# PI control

cumulativeError += error * ki
if error > 0:
    error = error * kp + cumulativeError
    dutycycle = min(error / 10, 1.0)
```

`kp` and `ki` are parameters manually set to decide the performance of the PI controller. For our project, it was set as `kp = 1` and `ki = 0.1`.

Recompiling: Targeting Raspberry Pi architecture | Remove logging | Cache fitting

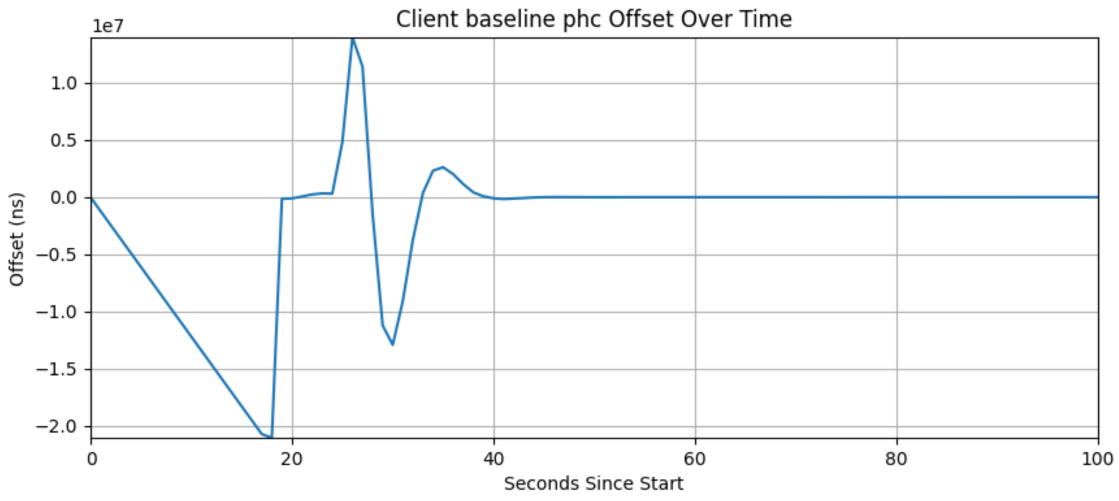
For this optimization we recompiled the GPSd and Linux-PTP for a minimal compilation on Raspberry Pi. Since we are directly compiling on Raspberry Pi with its compilers, we can assume the compilation is targeting to the specific architecture.

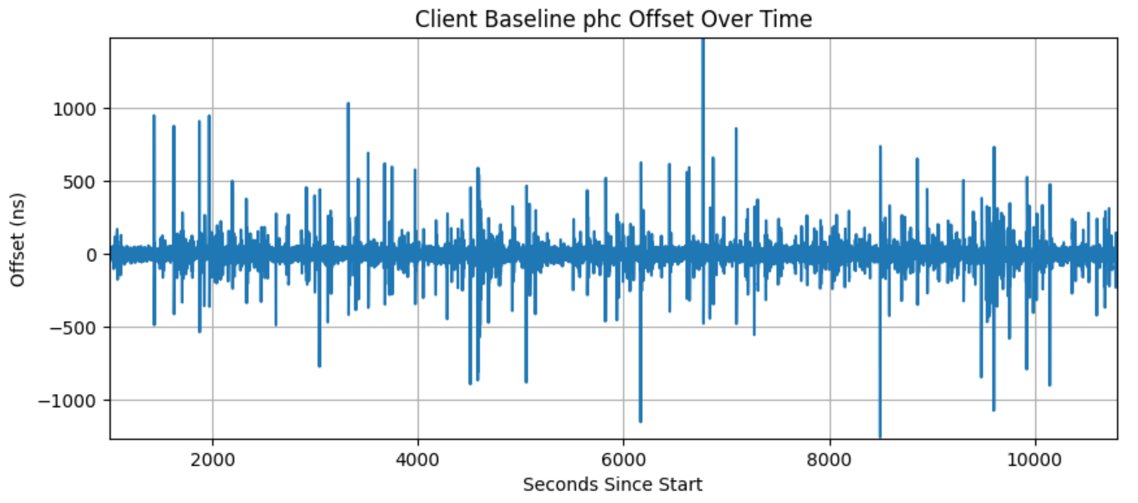
For GPSd, a configuration parameter for minimal compilation is provided. By changing the CXXFLAGS to `-march=native` and `-Os`, we can compile GPSd with only PPS enabled and any other unnecessary module disabled. The `-march=native` sets the targeting compilation platform to the current device and `-Os` optimize the compilation prioritizing code size.

For LinuxPTP, we removed logging by setting the `use_syslog` variable to 0 and compiled on Raspberry Pi. Thus we acquire the binary version of Linux PTP with logging removed and compiled targeting the architecture of Raspberry Pi.

4. Results

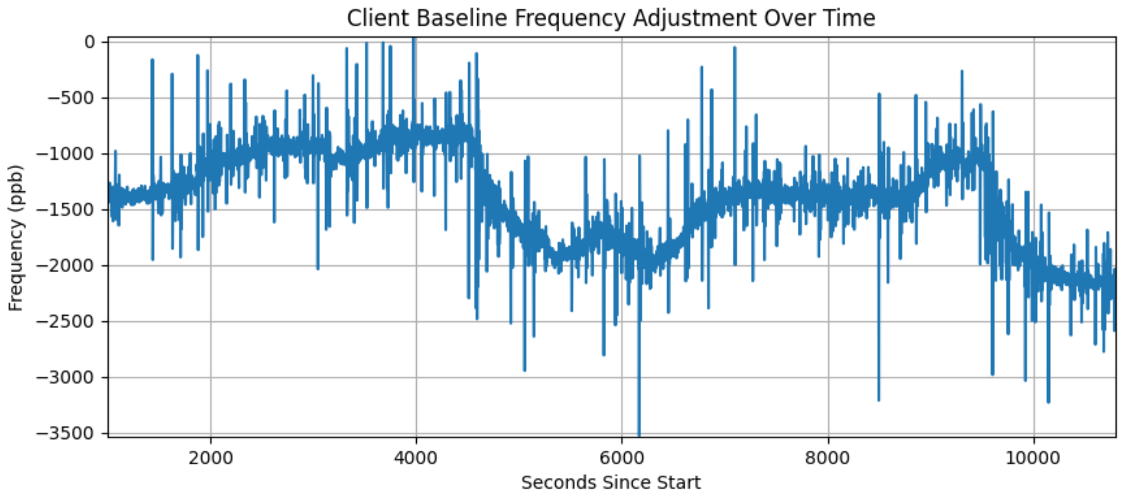
From the data, it is evident that the clock takes approximately forty seconds to stabilize from a cold start. To ensure accurate analysis, data collected after 100 seconds from the start was used. Once stabilized, the time series of offset demonstrated consistent patterns for both the grandmaster PHC offset and client PTP offset.

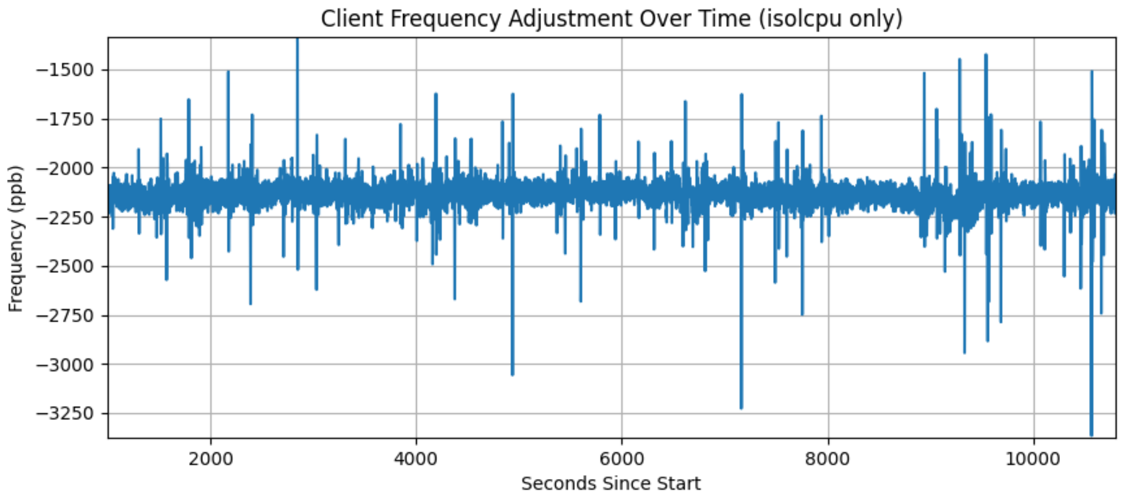




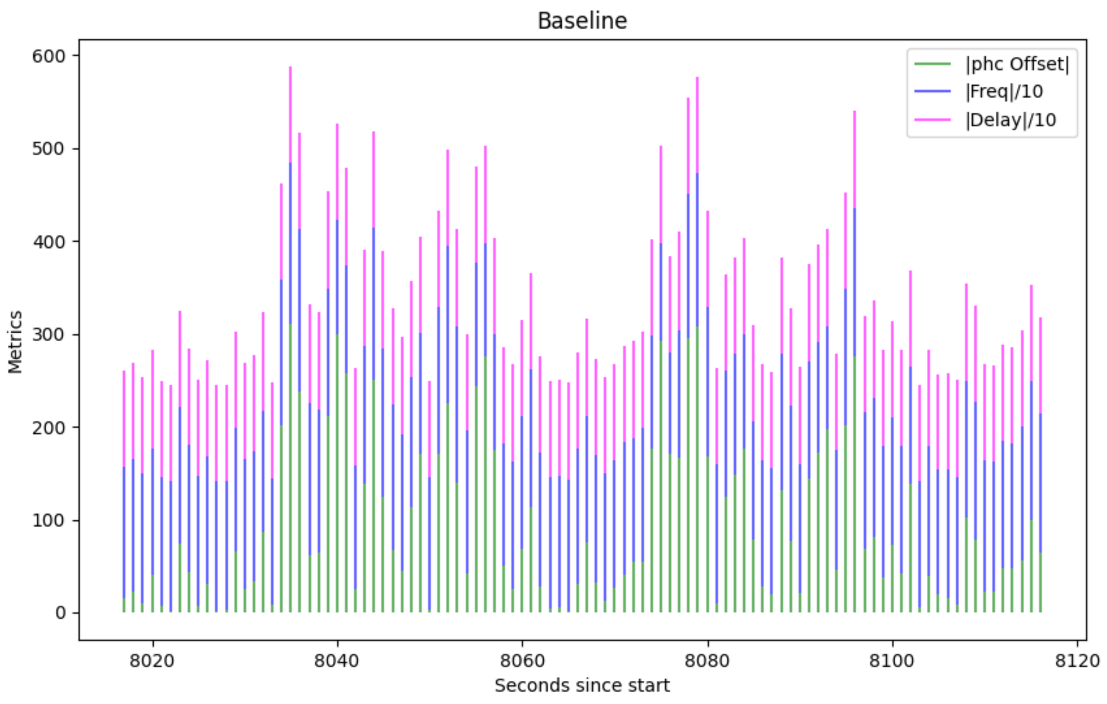
4.1 Isolcpu Application

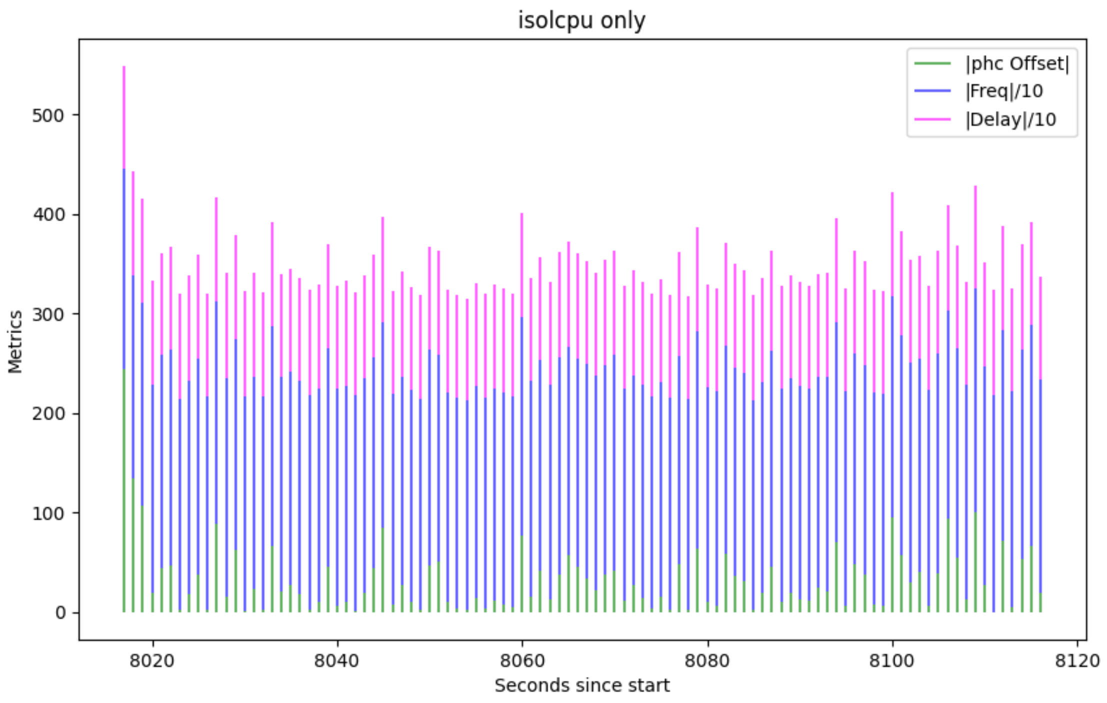
After applying `isolcpu`, a significant improvement was observed in both decreased offset and stability in frequency adjustment.



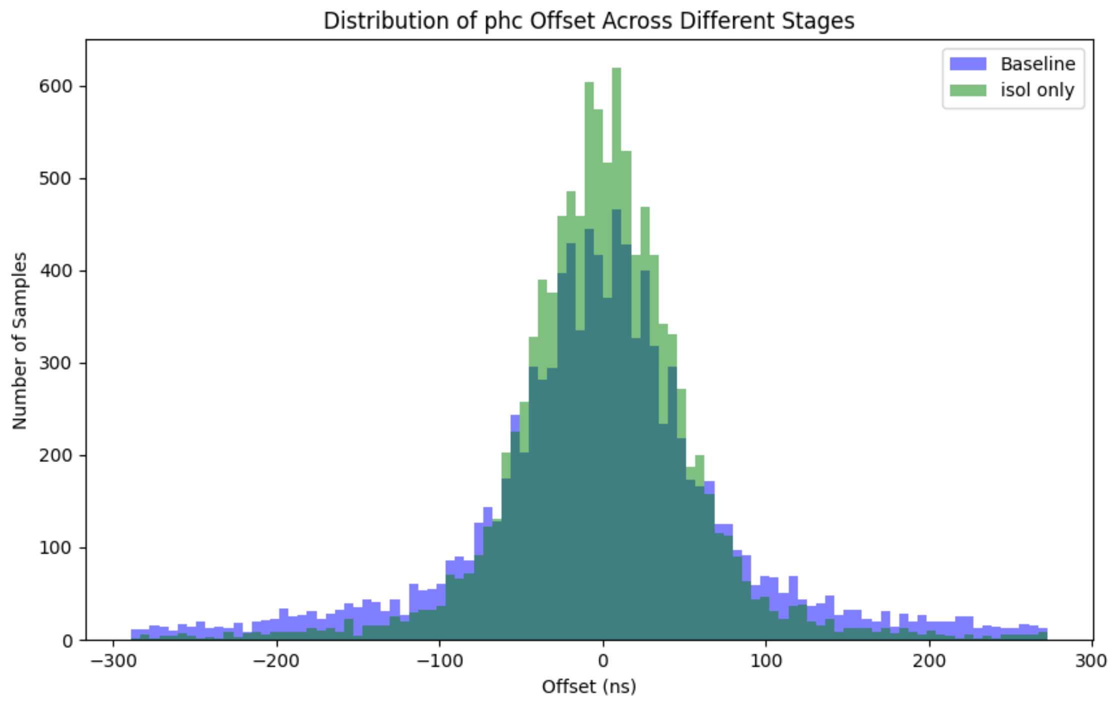


Similar to the data analysis presented here, we plotted time series with multiple metrics, including offset, frequency adjustment, and delay. For better visualization, we used the absolute values and scaled the frequency adjustment and delay by dividing the magnitude by 10.





The results clearly indicate that applying `isolcpu` led to a smaller and more stable offset. To further analyze the distribution of offsets, we referenced a study here. The optimization successfully reduced the offset (closer to 0) and decreased its variance.



Statistical significance was confirmed using Wilcoxon's signed-rank test and the Mann-Whitney test (‘$p < 0.05$‘).

4.2 CPU Affinity

Assigning CPU affinity to the grandmaster resulted in a marginally smaller average offset (2ns smaller), but both statistical tests (Wilcoxon's signed-rank test and the Mann-Whitney test) indicated that this difference was not significant (p=0.41 for the client and p=0.87 for the grandmaster).

4.3 PPS IRQ Steering to GPSD

We added PPS IRQ steering to the grandmaster and client as follows: **Grandmaster:** 1 - GPSD, 2 - Chrony, 3 - PTP + PHC PPS IRQ 184 to GPSD
Client: 2 - PTP, 3 - PHC

However, the statistical tests indicated no significant difference (p=0.87 for the client and p=0.99 for the grandmaster).

4.4 Thermal Control

Applying normal thermal control did not yield favorable results. The frequency adjustment and delay remained largely unchanged, but the offset nearly doubled, making it worse than the baseline. Consequently, normal thermal control was removed from further optimizations.

4.5 PPS IRQ Steering to Two Cores

Steering PPS IRQ to two cores proved less effective than previous optimizations. The change was statistically significant, but it degraded performance.

4.5 Steering All Unrelated IRQ to CPU 0

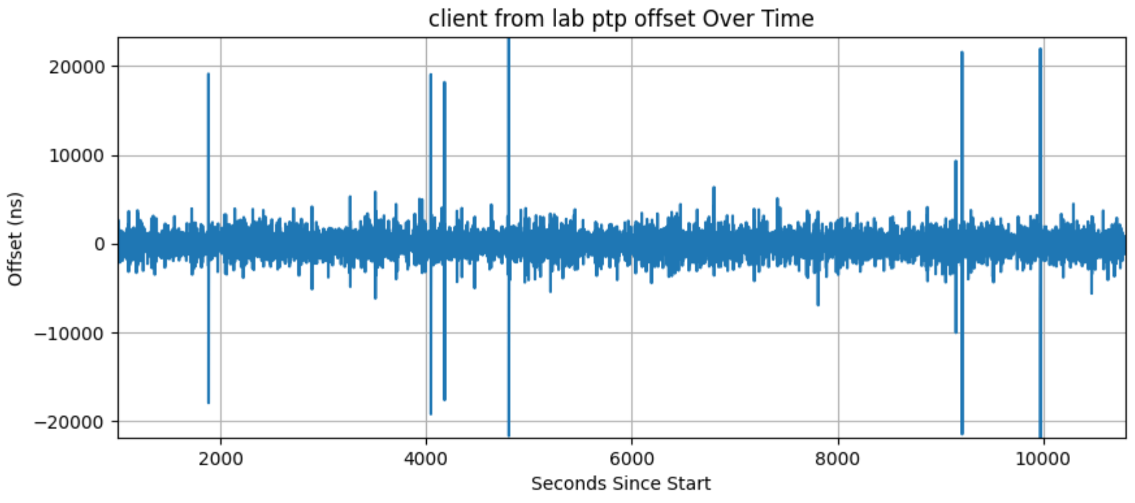
Steering all unrelated IRQ to CPU 0 yielded the best results, with the smallest offset and variance for delay and frequency. This improvement was statistically significant compared to previous optimizations. We decided to keep this setting for further testing and optimizations.

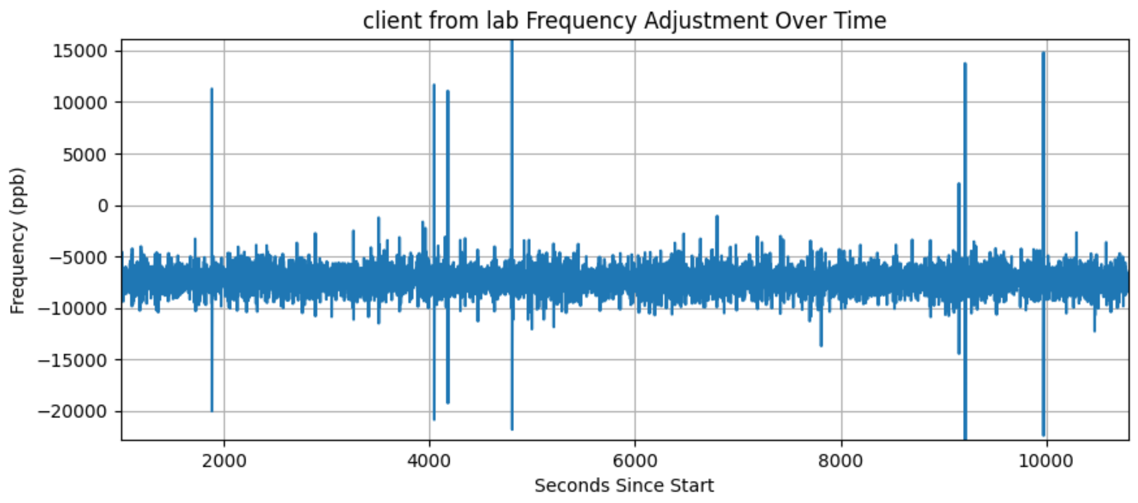
Additional Optimizations Applying P2P/L2 and clockServo on top of the optimized setup degraded the results. Advanced heat control based on this setup showed no significant change.

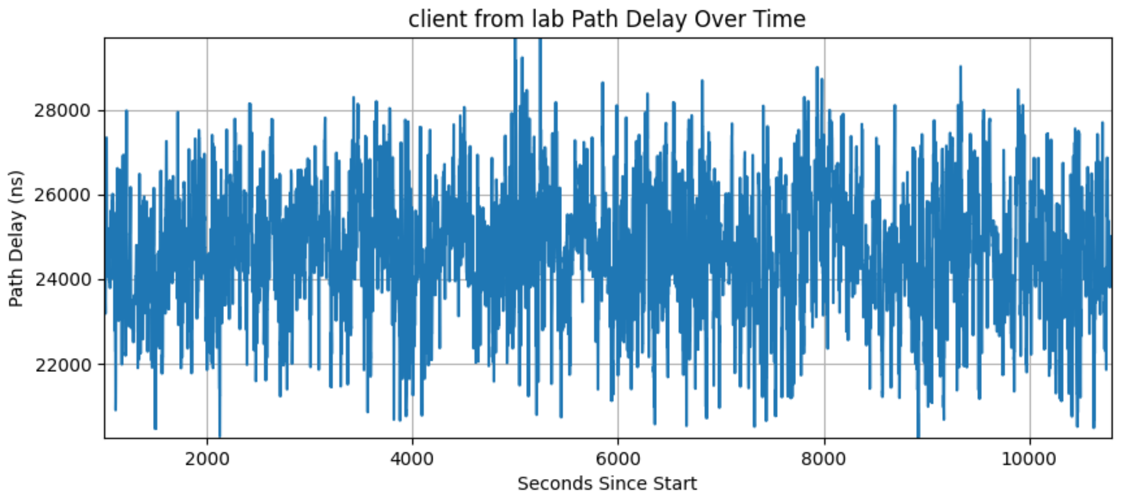
Summary of Results Below is a summary table with the mean and maximum offset for each optimization and the baseline:

Trial	Client Offset (phc avg)	Client Offset (phc max)	Client Offset (ptp avg)	Client Offset (ptp max)	Master Offset (phc avg)	Master Offset (phc max)
0510_1719	84.362	1444	62.094	1427	52.852	1477
0510_2128	46.541	977	34.169	816	31.012	812
0511_0228	44.003	661	31.932	667	29.969	807
0511_1120	46.141	633	33.333	594	30.859	546
0511_1945	93.849	1360	70.442	1318	56.989	1344
0511_2356	50.781	679	37.217	507	33.484	521
0512_0831	52.949	995	38.991	956	34.329	1172
0512_1417	39.558	464	28.900	436	27.275	522
0512_1948	42.159	881	30.622	978	28.987	1251
0512_2344	56.001	707	38.486	522	33.761	569
0513_0825	41.626	646	28.684	524	25.979	561
0513_1625	41.600	684	28.788	577	27.021	471

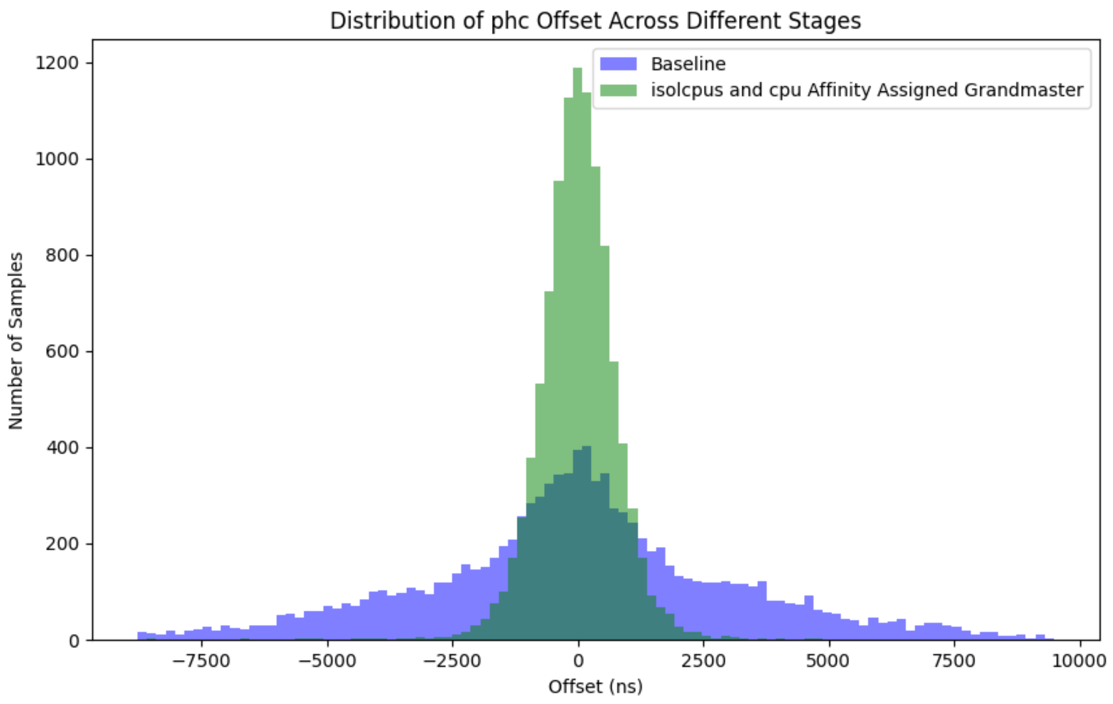
Lab Results Here are the time series plots for the client. Since the lab devices are not directly connected and one Raspberry Pi 4 does not support hardware timestamping, the offset is much larger than the local offset.







Similar observations were made for the lab data. Applying `isolcpu` on the client led to improvements in offset, and further significant improvements were observed after assigning CPU affinity to the grandmaster.



5. Discussion

5.1 Local Systems Observations

In our local setup, isolating CPU cores and pinning specific processes to these cores yielded a noticeable reduction in clock offset and improved the stability of frequency adjustments. However, while pinning processes to dedicated cores did reduce offsets and delays, the marginal benefits indicated that the tasks were already being efficiently managed by the isolated CPU cores. The added complexity of process pinning did not result in significant performance gains.

By steering critical time synchronization processes and interrupt requests (IRQs) to specific cores, we ensured that each core handled distinct tasks without interference. This segregation led to the most significant improvements in synchronization accuracy and stability. Steering IRQs to dedicated cores optimized the handling of time-sensitive signals, further enhancing the precision of our setup.

On the other hand, implementing active heat control measures did not improve overall performance. This outcome is likely due to the environmental conditions of our setup, which did not impose significant thermal stress on the hardware. The existing thermal management was adequate for maintaining stable operation without additional cooling interventions.

5.2 Lab Systems Observations

Our lab setup requires further optimization to collect more comprehensive data. Implementing and testing more advanced techniques will be crucial to fully understand and improve the system's performance. The initial results are promising, but additional refinements and data collection are necessary to achieve optimal synchronization precision.

One possible source of problems in the lab setup was the lack of a separate communication channel for interacting with the grandmaster and client. This limitation potentially introduced additional network traffic and contention, affecting synchronization accuracy. Additionally, the reliance on software timestamping with the Raspberry Pi 4 posed significant challenges. The absence of hardware timestamping capabilities led to increased variability and reduced precision in time measurements, underscoring the need for more robust hardware solutions in future implementations.

5.3 Future Work

Future work could focus on more low-level optimizations, such as kernel modifications to further reduce latency between receiving GPS signals and setting system time. Additionally, verifying PPS signals using external sources, such as GPIO-generated PPS signals on both Raspberry Pis, could enhance reliability. Exploring these areas could provide even more precise and robust time synchronization solutions.

6. Conclusion

This project successfully demonstrated the significant impact of various optimization techniques on the precision and stability of time synchronization in a Raspberry Pi-based setup. By systematically applying and evaluating optimizations such as CPU core pinning, PPS interrupt steering, thermal management, and network optimization, we achieved substantial improvements in timing accuracy.

The iterative and experimental approach adopted during this project not only addressed the technical challenges but also provided valuable insights into the intricate workings of time synchronization systems. The lessons learned and the techniques developed can serve as a foundation for future enhancements and applications in similar embedded systems projects.

In conclusion, this project highlighted the critical role of precise time synchronization in networked systems and the effectiveness of targeted optimizations in achieving high-precision results. Continued exploration and refinement of these techniques will contribute to the advancement of time synchronization technologies in embedded systems.

7. References

1. **IEEE** (2017). Precision Time Protocol Version 2 (PTPv2) - RFC 8173. Retrieved from RFC 8173
2. **Mills, D.** (2000). Pulse Per Second (PPS) API - RFC 2783. Retrieved from RFC 2783
3. **Meinberg Global** (2019). Structure of PTP Announce Message. Retrieved from Meinberg Blog
4. **Hydrogen18**. How to give a single process its own CPU core in Linux. Retrieved from Hydrogen18 Blog
5. **Red Hat**. Performance Tuning Guide. Retrieved from Red Hat Documentation
6. **FreeDesktop.org**. Understanding systemd. Retrieved from Systemd Exec
7. **Computer Hope**. What is Southbridge (RE: RP1). Retrieved from Computer Hope
8. **Austin's Nerdy Things**. Millisecond Accuracy with Chrony and NTP. Retrieved from Austin's Blog
9. **Gentoo Wiki**. Chrony with Hardware Timestamping. Retrieved from Gentoo Wiki
10. **Red Hat**. Configuring PTP Using ptp4l. Retrieved from Red Hat Documentation
11. **GSI Wiki**. Running Ordinary Boundary Clocks Using Linuxptp. Retrieved from GSI Wiki
12. **TSN Documentation**. System Time Isn't Synchronized with PHC. Retrieved from TSN Docs
13. **GPSD Documentation**. GPSD Performance Considerations. Retrieved from GPSD Docs
14. **Red Hat**. Combining PTP and NTP to Get the Best of Both Worlds. Retrieved from Red Hat Blog
15. **Red Hat**. Resource Management Guide: Configuring Time Synchronization. Retrieved from Red Hat Documentation
16. **Linux Man Pages**. phc2sys. Retrieved from Ubuntu Manpages
17. **Linux Man Pages**. pmc. Retrieved from Ubuntu Manpages
18. **Linux PTP Documentation**. phc2sys Documentation. Retrieved from Linux PTP Docs
19. **Red Hat** (2015). Performance Tuning Guide: Low Latency Performance Tuning for Red Hat Enterprise Linux 7. Retrieved from Red Hat Documentation
20. **Mlichvar, M.** (n.d.). Does Chrony support PTP? In FAQ - Chrony. Retrieved from Chrony FAQ
21. **FSMLabs** (n.d.). Introduction to Clock Sync in Finance and Beyond. Retrieved from FSMLabs
22. **European Commission** (2016). Commission Delegated Regulation (EU) 2016/958 of 9 March 2016 Supplementing Directive 2014/65/EU of the European Parliament and of the Council with Regard to Regulatory Technical Standards for the Level of Accuracy of Business Clocks. Retrieved from EU Commission
23. **Financial Industry Regulatory Authority (FINRA)** (n.d.). FINRA Rules 6820. Retrieved from FINRA
24. **Wireshark** (2020). Protocols/ptp. Retrieved from Wireshark
25. **Tcpdump public repository** (2024). tcpdump(1) - Linux man page. Retrieved from Tcpdump

26. **u-blox** (2019). ZED-F9T-00B Data Sheet. Retrieved from u-blox
27. **u-blox** (2011). LEA-5T Data Sheet. Retrieved from Datasheets.com
28. **Masterclock** (n.d.). Network timing technology: NTP vs. PTP. Retrieved from Masterclock
29. **Raspberry Pi Foundation** (2023, October). Raspberry Pi 5 product brief. Retrieved from Raspberry Pi 5 Product Brief
30. **Raspberry Pi Foundation** (2023, November 24). Compute Module 4 datasheet. Retrieved from Compute Module 4 Datasheet
31. **Raspberry Pi Foundation** (2023, December). Raspberry Pi 4 product brief. Retrieved from Raspberry Pi 4 Product Brief
32. **Red Hat** (n.d.). Ethtool. Retrieved from Red Hat Documentation

8. Reflections

8.1 Laxmi "Lux" Vijayan

1. What did you specifically do individually for this project?

I was the team lead for this project. As such, I was responsible for strategic planning and coordination, hardware procurement and setup, leading the implementation of optimizations, structuring data collection, and drafting the final report. In other words, I did everything from writing our initial proposal, organizing team members into respective roles, assigning tasks, buying and setting up the local systems for us to work with while we waited for the lab set up to be functional, implement optimizations, collect data, and write the report!

2. What did you learn as a result of doing your project?

Wow, so much. The more obvious answer would be that I learned about time synchronization, different protocols that optimize for global synchronization, the use of GPS in precise timing, and the importance of and challenges associated with precise timing. This was also a great learning experience for project management and leading a team on a technical project. There are increased challenges when you're working with a cross-functional team and one such challenge I encountered was knowing how to leverage everyone's diverse backgrounds effectively to meet our shared goals. Lastly, I gained an incredible level of comfort with Linux and the Raspberry Pi platform. One problem working with open-source platforms is that the internet is saturated with different solutions for the bug you're facing and it's hard to evaluate which solution will resolve the causes of your problem. It's much like diagnosing a disease, come to think of it; maybe it's because of my biomedical background, but I found this continuous cycle of implement-debug rather enjoyable. It gave me more insight into how these systems function and ideas for further development on this project.

3. If you had a time machine and could go back to the beginning, what would you have done differently?

This is a tough one. I think I would have started with the Raspberry Pi 4, instead of the Raspberry Pi 5. We chose to work with the newer Raspberry Pi 5 platform because of it's ability to hardware timestamp. This was both great and a challenging choice; we're some of the first people to attempt using the Pi 5 for precise time serving, but as a result, we did face a lot of problems we couldn't easily find solutions to. I think if we'd had a local Pi 4 set up, we'd have gotten our feet without the hassle of charting new territory, and then it would have been relatively simple work to switch to the Pi 5.

4. If you were to continue working on this project, what would you continue to do to improve it, how, and why?

I definitely want to continue working on this project! I want to work even more low-level optimizations than we managed. For example, I'd like to learn how kernel modifications can help reduce latency between receiving GPS signals and setting system time with chrony. I would also like to see if we can verify the PPS using external sources by setting up one of the GPIO pins on both Raspberry Pis to generate PPS signals. There's still so much left to do for this project that I don't quite feel done with it yet.

5. What advice do you offer to future students taking this course and working on their semester-long project (besides "start earlier"... everyone ALWAYS says that). Providing detailed thoughtful advice to future students will be weighed heavily in evaluating your responses.

I have two pieces of advice to give to future students; the first pertains to their success in this course, and the second is more broad based on my experience working with these tools this semester.

- Regardless of your project, experiment! Play! Break things! I find I learned a lot more about all the tools I was working with, the concept of time synchronization, Linux, and computer architectures when I was debugging than when things went well. The process of figuring out what every little thing did, why it was necessary (or not), and more was more fun as a result.
- Document your code well. It is incredibly frustrating to go to a man page or some other documentation, only to find that it entirely fails to answer your question about a configuration or setting. We hit so many walls because some of the tools were so sparsely documented. Often we had to rely on third-party solutions. Well-documented code might be a hassle to do while you're writing it, but it takes your work from shoddy to exceptional if you just spend a few extra minutes.

8.2 Yiqing Huang

1. What did you specifically do individually for this project?

I was responsible for drafting the scripts for collecting the metrics, analyzing all the data, plotting graphs for visualization of data, and giving advice on how to proceed with optimizations along the data collection process.

2. What did you learn as a result of doing your project?

I became more familiar with writing bash scripts. I felt more comfortable working in the Linux environment. I was not familiar with time synchronization concepts like ntp, ptp, pps before, and now I find it a very interesting topic to delve into. I also learned how to use tools like chronyc, ptp4l, phy2sys, how to get temperature and cpu usage, how to filter and log the data according to our needs.

3. If you had a time machine and could go back to the beginning, what would you have done differently?

I would like to have more exposure to setting up the hardware and implementing the optimizations. I have very dependable teammates who are more specialized in those, and if I had the chance, I would have learned those aspects as well and as soon as possible to have a more well-rounded understanding of this project.

4. If you were to continue working on this project, what would you continue to do to improve it, how, and why?

I would look closely into my teammates' work, and try to build more advanced optimizations based on the current ones. I would like to see how much the offset etc can be further decreased due to our optimizations.

5. What advice do you offer to future students taking this course and working on their semester-long project (besides "start earlier"... everyone ALWAYS says that). Providing detailed thoughtful advice to future students will be weighed heavily in evaluating your responses

We originally set a timeline, but we encountered a series of problems and underwent a frustrating period of debugging the setup that made it impossible to keep the original schedule. Therefore, I would recommend setting up the deadlines for each stage earlier than you plan to. Because there will definitely be obstacles that postpones your progress. I think in addition to weekly meetings, a mid-week check-up would be of great help. It doesn't have to be a meeting and it doesn't have to be long, but a mid-week check-up can encourage teammates to work during the week instead of working one day before the meeting.

8.3 Haoyuan you

1. What did you specifically do individually for this project?

My main assigned tasks were implementing optimization scripts and setups for our GPS grandmaster performance. Here's a complete list of all the optimizations: isolating CPU and pinning process, thermal control with the active cooler, CPU overclock, advanced thermal control with idle CPU busy spinning, and recompilation for Linux-PTP, GPSD, and Chrony and their corresponding part for the documentation and reports.

2. What did you learn as a result of doing your project?

The concept of different time synchronization methods like GPS based (with PPS), PTP, and NTP. The implementation of interrupt steering and CPU cores isolations.

3. If you had a time machine and could go back to the beginning, what would you have done differently?

I would have tested the ublox with an oscilloscope the first time I failed to set it up and immediately realized that the ublox module was broken, instead of spending hours after hours trying to figure out if there were mistakes in the GPS receiver environment setup.

4. If you were to continue working on this project, what would you continue to do to improve it, how, and why?

There are still a few other optimizations to be implemented and they should be able to stabilize the system even more. Some of the optimizations were not refined as well like the 2 thermal control scripts. The parameters could use some fine tuning or even measuring to estimate a model. I wrote the PI controller parameter purely based on experiences and it's amazing it worked.

5. What advice do you offer to future students taking this course and working on their semester-long project (besides "start earlier"... everyone ALWAYS says that). Providing detailed thoughtful advice to future students will be weighed heavily in evaluating your responses.

I think some of the things we did right were 1. Weekly meetings and occasional working meetings. These are helpful to keep the project on schedule even if a key component went catastrophic wrong and were struck for weeks. 2. Don't hesitate to talk to the professor if things are not as expected. Sometimes the direction is wrong but you don't even see it.

8.4 Victor Li

1. What did you specifically do individually for this project?

I ensured the network infrastructure is robust, scalable, and optimized for precise time synchronization(debugging), implementing necessary adjustments to minimize latency from the outset. And I was in charge of implementing optimizations and collecting data in the lab setup. I refined network configurations and protocols for the intermediate and advanced optimizations.

2. What did you learn as a result of doing your project?

I learned lots and lots of linux commands by implementing the optimization scripts, and debugging the network and time synchronization in the lab. Since I did a lot of debugging, I think I also learned when some problems arise, what system status we should look into, and what processes we should stop and restart them.

I was a complete noob for computer networking and time synchronization. By doing this project, I read some of the documentation for phc2sys, ptp4l, tcpdump, ethtool and gained a lot of experience working with computer networks.

3. If you had a time machine and could go back to the beginning, what would you have done differently?

I would try to understand clearly what we are trying to do. In the first few weeks, I have no clear idea of what the goal of this project is supposed to be. I should have known it clearly on day 1.

And I think we probably could have communicated and collaborated better. There were problems that we couldn't resolve on both our local and lab setup for sometime, and for that time the project basically got stuck. We cannot do optimization on lab or local -> don't know what optimizations to do next (and don't know if the current optimization is effective) -> cannot based on the result write new optimization scripts (since we cannot collect data at all) -> cannot do data analysis. I think we could discuss current priorities during meetings and even that's supposed to be one person's work, since if that problem doesn't get solved other people get stuck too (The stuck person's priority is the team's priority, the only way to move forward), every member of the team should just focus on that problem.

4. If you were to continue working on this project, what would you continue to do to improve it, how, and why?

I would implement the optimizations that we didn't have enough time to do and conduct a detailed analysis comparing the performance of setup A versus setup B. For example, what happens if we run another script in the background while collecting data? Would it worsen the performance, and if so, by how much? Does a more resource-intensive local background script that uses CPU, memory, and generates more heat cause the machine to lose more performance compared to a lighter script that uses eth0? All else being equal, how much performance improvement does using the RPi 5 provide over the CM4?

5. What advice do you offer to future students taking this course and working on their semester-long project (besides “start earlier”... everyone ALWAYS says that). Providing detailed thoughtful advice to future students will be weighed heavily in evaluating your responses.

Advice 1. As a project-heavy class, it’s easy to delay tasks more than originally planned. I think the scheduled weekly meetings certainly help. As concrete advice, I recommend setting automatic reminders or scheduled auto-emails to remind your team daily about what everyone should do, and perhaps how to do it. For example, an email could be sent out every day at 1 PM detailing the tasks each person in your group should do, and a reminder tone could pop up on your phone and laptop every day at 7 PM.

Advice 2. Watch some YouTube videos and read the GitHub repositories of people who have done part of what you are trying to do. Document your process and what you did at each step. If you encounter a hardware-related bug, try restarting all the relevant processes, and if that doesn’t solve the problem, reboot the system.