Time Synchronization Via Sensing

SHRUTI PARAB

Time synchronization is critical for networked embedded systems used in applications such as smart homes, automated agriculture, and autonomous vehicles. Traditional synchronization methods are inefficient for resource-constrained edge devices due to their computational and communication overhead. This project proposes a lightweight synchronization protocol that leverages time-stamped sensor data from ESP32 edge devices and a Raspberry Pi gateway. The protocol addresses the challenges of network delay and clock drift to maintain synchronization accuracy. By characterizing delays and estimating drift, the approach ensures efficient and accurate time synchronization while minimizing resource consumption. This method offers a practical solution for maintaining synchronization in distributed embedded systems, enabling reliable operation in real-world deployments.

Additional Key Words and Phrases: Time Synchronization, Embedded Systems, Network Delay, Clock Drift, ESP32, Raspberry Pi, Resource-Constrained Devices, IoT, Distributed Systems, Edge Computing.

ACM Reference Format:
Shruti Parab. 2018. Time Synchronization Via Sensing. 1, 1 (December 2018), 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Synchronization of time is of great significance in ensuring the correct functionality of other distributed applications in networked embedded systems including smart homes, automated agricultural practices and even self-driving vehicles. These applications depend on synchronization of functions, data analysis, and the execution of commands in accordance with a precise time schedules. Although time synchronization has been implemented successfully there is a number of challenges that still persist especially when implemented in edge devices that are likely to have limited computational capabilities, limited memory and energy efficiency.

Typical time synchronization approaches, including NTP or PTP, can introduce prohibitive overhead to an embedded system. Furthermore, some of these devices work at the edge where connectivity at times can be latency or delay based. This makes conventional synchronization services ineffective for use in such environments.

In order to surmount these challenges, this project presents an innovative method of time synchronization that builds upon existing time-stamped sensor data gathered from the embedded devices. In particular, the system comprises ESP32 edge devices which periodically send the data of the sensors to Raspberry Pi gateway. The system also includes an **LM393 sound sensor** which takes sound data in every 5 seconds and sends data in the case that it detects sound. The synchronization protocol running on the Raspberry Pi records and measures network delays and then estimates clock skew to provide accurate network synchronization of the devices involved.

This approach reduces overhead costs in terms of resource utilization since it leverages on data that exists and is being transmitted by the devices typically in constrained settings. The objective of efficient time synchronization achieved by the system includes integration of activities, increased reliability of information, and increased effectiveness of distributed embedded applications.

The subsequent sections of the paper shall detail out the methodology by which the synchronization has been proposed, the hardware configuration that has been recommended, the output to be elicited from the it and the further enhancements to be incorporated into this specific synchronization protocol.

Author's Contact Information: Shruti Parab.

1.1 Literature Survey

Synchronization of time in embedded and distributed systems has emerged as an important research subject because of the substantial applications it serves. There exist several synchronization protocols and methodologies and all of them function from completely different angles and in addition differentiate in terms of precision, consumption of resources, and scalability. This survey has focused on papers that are related to time synchronization for low powered devices and the methods used.

1. Network Time Protocol (NTP) : The Network Time Protocol (NTP) is used to regulate time and Date in every system that is connected in the internet. The time distribution in NTP is based on the hierarchical set of time servers. It would be rather useful in many uses though; the computation and communication overhead incurred from employing NTP discriminates it for low power embedded gadgets. Additionally, NTP is highly dependent on the delay of the networks in this case sacrificing accuracy in unreliable or congested networks.

2. Precision Time Protocol (PTP): In the case of networks, the IEEE 1588 also known as the Precision Time Protocol is more accurate the NTP since it factors in network delays. PTP is typically applied to Industrial Automation and Telecommunications. However, applying PTP on resource-scarce gadgets poses a task because of its inefficiency and highly demanding on the onboard hardware time-stamping capacity.

3. Low Overhead Time Synchronization Protocols : A considerable effort has been done in designing lightweight protocols in sensors mainly for the application in the embedded and WSNs. Other protocols like the Tiny-Sync and the Flooding Time Synchronization Protocol (FTSP) fulfilling the objective with very low overhead by employing easier algorithms and utilizing one-way messaging just. These protocols work by trying to find out the offset and delay on the different clocks involved but can still suffer based on dynamic or lossy systems.

4. Sensor-Based Synchronization Methods: Based on the recent contributions, the idea of utilizing sensor data for synchronization is investigated in more detail and applied to IoT and edge computing.

For instance, HAEST (Harvesting Ambient Events to Synchronize Time) uses environmental events that aid in synchronizing clocks in devices with variant platforms. This method reduces communication overhead by using data already captured by sensors that it is ideal for constrained systems.

5. Vibratory and event synchronized system : It has been suggested that physical synchronization should be attained via some vibration or motion, like in some experiments. For example, Automated Synchronization of Driving Data Using Vibration and Steering Events attempts to align vehicle sensor data on identified physical event basis. These methods are effective, but they presuppose the availability of often permanently clearly defining phenomena for appropriate synchronization.

6. Facility Available on smartphone peripheral for synchronization : The Exploiting Smartphone Peripherals for Precise Time Synchronization as a research states that audio, motion, light sensors are valuable for synchronizing time. Both of these methods utilize already existing peripherals meaning that they are relatively cheap as far as widespread use is concerned.

1.2 Design Overview

Focusing on the ESP32 edge devices as well as an additional Raspberry Pi gateway, the time synchronization system proposed in this paper is specifically designed to be effective while requiring a low amount of resources for embedded systems. The LM393 sound sensor with an ESP32 chip for every device used in this work records sound occurrences after set intervals of 5 seconds. In this study, an audio clip, composed from YouTube, was set to play at 12BPM without interruption throughout 1 hour. In this study, the LM393 sound sensor is fo und to be used in detecting the existence of sound hence intimating the ESP32 to create a timestamp and send the data to the Raspberry Pi via Bluetooth.

The Raspberry Pi is the major switch and implements the synchronization management. This one takes the timestamped data which has been transmitted by ESP32 devices through Bluetooth and logs the reception timestamp. Whenever an ESP32 sends a timestamp to the Raspberry Pi, the received data are then saved in a JSON file for processing and analysis. When comparing the transmission timestamps to the reception times, the Raspberry Pi is able to quantify the networking delay that occurred during data transfer.

Apart from network delay characterization, the system identifies the drift of the clocks of the ESP32 devices with respect to Raspberry Pi by comparing the time stamps across time. As a result of it, the Raspberry Pi sends drift correction messages at fixed intervals requesting the desire ESP32 devices' clock be corrected. This approach makes sure that all the clocks in all the devices are synchronized at any one time even with discrepancies arising from network delay and chip clocks.

The design aims at optimality in all aspects by optimizing the computational work load, and reducing round trips on every message. The complexity and overhead of NTP or PTP synchronization is avoided in the system, while only reusing existing sensor data and using lightweight Bluetooth communication. Further, the architecture is extensible to accommodate more ESP32 devices than what is shown in this paper with little changes.

Specifically, this approach solves the problem of network delay and clock drift, and it is efficient and feasible to achieve time synchronization in a distributed embedded system. Bluetooth communication, detection of event just by sound and JSON based data storage is easy, accurate and more effective to implement.

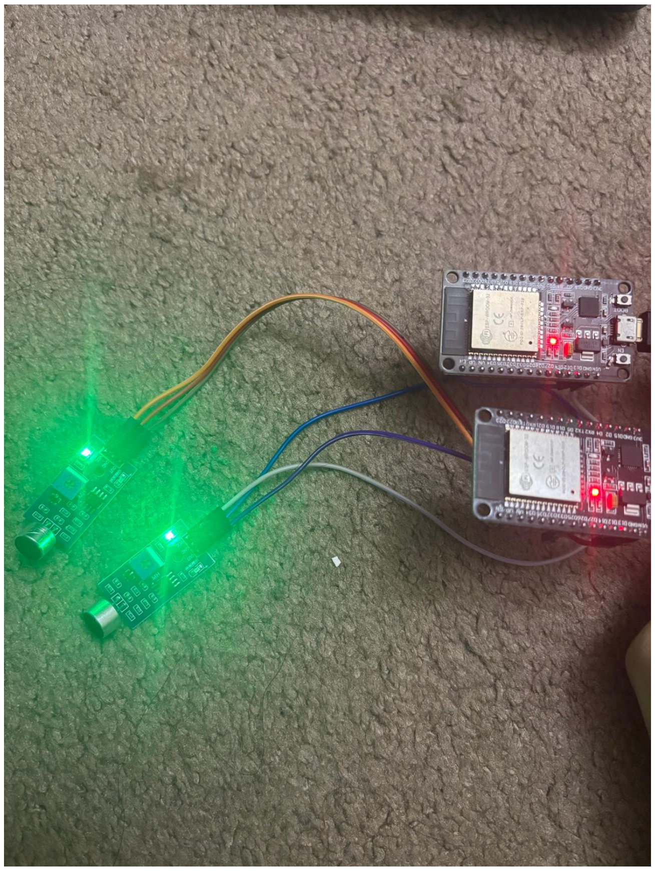

Fig. 1. ESP32 Microcontrollers with LM393 Sound Sensors for Time Synchronization Experiment

2 Methodology

2.1 Components

The time synchronization system performs synchronization among the embedded devices by using necessary hardware devices. These are the ESP32 microcontroller which is a powerful low cost low power microcontroller with in-built Wi-Fi and Bluetooth modules. ESP32 has a 32-bit dual core processor and multiple general-purpose input/output pins for connecting to sensors as well as for wireless data transfer. Each ESP32 has a LM393 sound sensor to capture the sound events happening in the environment. LM393 outputs a digital signal at the alarm pin when the sound event has been recognized by the device and this leads to a timestamp generation by the ESP32. The Raspberry Pi works as a hub since it acquires data from different ESP32s over Bluetooth, where every data is marked with time stamps and then computes the synchronization calculations. Bluetooth connection is employed so that there is no significant delay or loss of data between the ESP32 devices and the Raspberry Pi. The collected timestamps, network delays, and drift corrections are saved in a JSON file to facilitate a consistent method of analysis of the results and debugging.

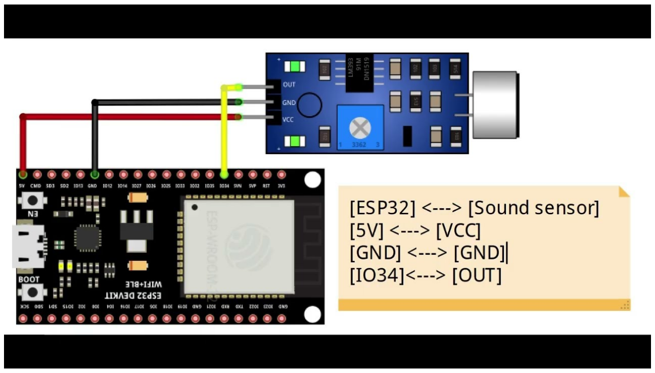

Fig. 2. Connection between ESP32 and LM393

2.2 Working

The working starts with two ESP32 microcontrollerswhere each circuit is integrated with an LM393sound sensor for sound event detection. The first ESP32 detect a sound event and labels it with an information time t1, to show the exact time in one ESP32. Likewise, when the second ESP32 captures a sound event it creates a timestamp T1 to indicate time at which the event happened in ESP2. It is worth to note that these timestamps are expressed in t1 and T1 and are transmitted by Bluetooth to Raspberry Pi.

When these timestamps are received, the Raspberry Pi records the data and stores t1 and T1 into a JSON file. The Raspberry Pi then deduces the difference between t1 and T1: This can be calculated as the difference between the time from the ESP1 and the time from the ESP2. In this way the offset is helpful in finding the time difference between two values of ESP32 system. The offset calculation is important for identifying the synchronization modification required to support the clocks of ESP1 and ESP2.

During the delay estimation phase, Raspberry Pi is used to determine the amount of network delay by assessing its time on Bluetooth transmission. The two appear in conjunction with the offset between t1 and T1 to enable the synchronization of the ESP32 devices. The system then goes into the drift estimation where the Raspberry Pi can know the difference in the time stamps as time moves on to find out if the time difference between ESP1 and ESP2 has drifted or not. This drift happens as a result of small disparities in the system oscillators of the ESP32 devices which, if not regulated, causes desynchronization.

In the synchronization phase the Raspberry Pi works out the amount of correction required for ESP1 and ESP2 system clocks. In its turn, the Raspberry Pi sends these drift correction messages to the ESP32 devices through Bluetooth. The ESP32 devices then apply the corrections to the clocks for both of the devices so that they operate in harmony. The different stages of delay estimation, offset calculation, drift correction, and synchronization are iteratively carried out to refresh the synchronization.

The sensor based event detection, Bluetooth control and chord, systematic delay and drift correction mechanism actually guarantees an effective and accurate inter-ESP32 synchronization. Instead of the console, which is often used in similar cases, logging into JSON files is useful to store the results of synchronizations as well as debug the system when needed.

3 Results

3.1 Offset Analysis for Ground Truth Experiment

The experiment tracks the end-to-end communication process, including sending timestamps, receiving acknowledgments, and calculating offsets for two ESP32 device to determine the synchronization accuracy.This experiment depicts ground truth of time synchronization providing an ideal data for comparison.The graph plots the calculated offsets for each device over a series of measurements, helping to visualize the synchronization performance.

Process Flow

1. Timestamps Sent by ESP32 Devices:
   - ESP1 generates a timestamp t1 when a sound event is detected.
   - ESP2 generates a timestamp T1 for a similar sound event.
2. Timestamps Received at Raspberry Pi:
   - The Raspberry Pi receives the timestamp from ESP1 at time t2.
   - The Raspberry Pi receives the timestamp from ESP2 at time T2.
3. Acknowledgment Sent Back to ESP32 Devices:
   - The Raspberry Pi sends an acknowledgment back to ESP1 at time t3.
   - The Raspberry Pi sends an acknowledgment back to ESP2 at time T3.
4. Acknowledgment Received by ESP32 Devices:
   - ESP1 receives the acknowledgment at time t4.
   - ESP2 receives the acknowledgment at time T4.

Offset Calculation: The offset for each ESP32 device is calculated using the formula:

$$\text{OFFSET} = \frac{(T2 - T1) - (T4 - T3)}{2}$$

Graph Analysis

- Yellow Plot (Offset for ESP1): The yellow plot represents the calculated offsets for ESP1 over a series of measurements. The offset values fluctuate around the zero baseline, typically within the range of -0.0015 seconds to 0.0015 seconds. These fluctuations indicate minor clock drift and variations in network delay during communication.
- Orange Plot (Offset for ESP2): The orange plot represents the calculated offsets for ESP2. The offset values show a similar pattern, fluctuating within the same range as ESP1. The close alignment between the yellow and orange plots suggests that both ESP32 devices experience similar delays and drift patterns.

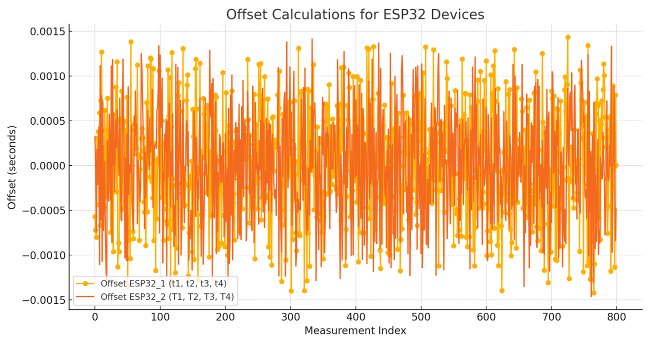

Fig. 3. Graph of ground truth

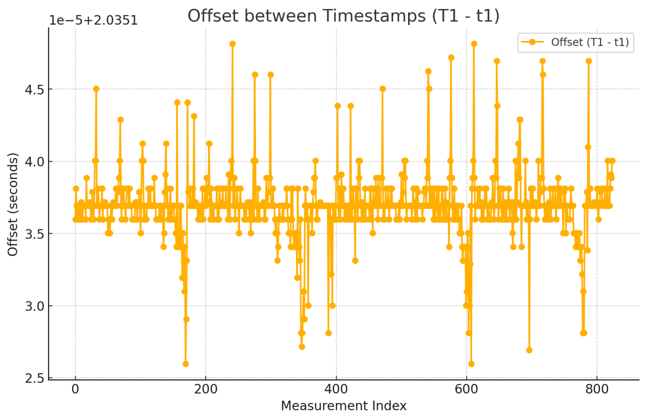

Fig. 4. Offset graph of experiment

Observations

1. Offset Range: The offsets as calculated for both ESP32 devices are within 0.0015 seconds. This small range clearly suggests that the synchronization system is good as it helps to maintain directly proportionality between the ESP32 clocks and the Raspberry Pi.

2. Periodic Fluctuations: The fluctuations of the offsets on high frequency indicate that even though the devices are married there is a slight clock drift. These fluctuations are observed due to natural imperfection of the ESP32 internal clock and delays which occur in the network.

3. Symmetry in Delays: ESP1 and ESP2 have relative delay offset witheach other , meaning that the two devices suffer the similar delay while communicating to the Raspberry Pi from their respective places. It can be seen that keeping track of time enables synchronizing it and this symmetry is useful in accuracy of synchronization.

4. Ground Truth Validation: The graph remains the best standard of measure for the experiment since it represents the synchronization accuracy obtained. The consecutive—that is, repeating—patterns in the offsets bear witness to the fact that the calculated time and clock drift are being quantified and then eliminated with some measure of success.

3.2 Offset Analysis for Timestamp Synchronization Experiment

We conducted another experiment where a speaker is placed close to two LM393 sound sensors, each connected to an ESP32 microcontroller. The speaker plays a sound at a rate of 12 beats per minute (BPM). Whenever the LM393 sound sensors detect the sound, they trigger the ESP32 devices to generate and send timestamps to Raspberry Pi.

Timestamp Generation:

- `t1` is the timestamp generated by ESP1 when the LM393 sensor connected to it detects a sound.
- `T1` is the timestamp generated by ESP1 when the LM393 sensor connected to it detects a sound.

This experiment is conducted over a duration of 1 hour, during which the timestamps t1 and T1 are sent to a Raspberry Pi (RPI). The Raspberry Pi logs the timestamps in a JSON file for further analysis.

Offset Calculation: The offset between the two timestamps is calculated by taking the difference between T1 and t1.

$$\text{Offset} = T1 - t1$$

Graph Plotting: This graph visualizes the calculated offsets over a brief period of time. The plot shows fluctuations in the offset, ranging typically between 3.5 and 4.5 seconds, with periodic spikes indicating variations caused by network delay, clock drift, and detection timing differences.

3.3 Comparison between ground truth and experiment

The offset range in the ground truth shows values within a very small range of ±0.0015 seconds, indicating precise synchronization with minimal drift. In contrast, the experiment shows a much larger offset range of 3.5 to 4.5 seconds, reflecting significant variability likely due to network delays and timing discrepancies.

In terms of precision, the ground truth results demonstrate high accuracy by accounting for round-trip delays and drift corrections. This ensures that the offsets remain tightly controlled. On the other hand, the experiment uses a more straightforward calculation of the difference between timestamps, without compensating for round-trip delays. This leads to larger and less precise offsets.

When considering variability, the ground truth shows close alignment between the two ESP32 devices, suggesting consistent and predictable drift patterns. In contrast, the experiment exhibits larger variability, which indicates inconsistencies in sound detection and network transmission processes.

This comparison highlights the importance of compensating for transmission delays and clock drift to achieve precise and consistent synchronization in embedded systems.

4 Conclusions

This report presented a comprehensive approach to time synchronization for embedded systems using ESP32 microcontrollers, LM393 sound sensors, and a Raspberry Pi. The experiment leveraged timestamped data generated by sound event detection and transmitted via Bluetooth for analysis. The collected data provided insights into synchronization accuracy, network delays, and clock drift between the ESP32 devices.

The experimental results demonstrated significant variability in offsets, with a range of 3.5 to 4.5 seconds. These variations were primarily due to network delays and timing discrepancies inherent in the simple timestamp difference calculation. In contrast, the ground truth analysis showed far more precise synchronization, with offsets confined to a narrow range of ±0.0015 seconds, achieved by compensating for round-trip delays and clock drift.

The comparison of the two methods highlighted the importance of incorporating delay and drift corrections to achieve accurate synchronization. The straightforward offset calculation used in the experiment provided an initial measure of synchronization discrepancies, while the refined ground truth calculation validated the system's potential for high-precision synchronization.

In conclusion, the time synchronization system effectively demonstrated the challenges and solutions in synchronizing resource-constrained embedded devices. The results underscore the necessity of periodic drift corrections and delay compensation for maintaining accurate synchronization. Future work could focus on optimizing the synchronization protocol, reducing network delays, and expanding the system to handle more devices in dynamic environments.

5 References

- Nasrullah, Adeel, and Fatima M. Anwar. "HAEST: Harvesting Ambient Events to Synchronize Time across Heterogeneous IoT Devices." 2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS). IEEE, 2024.
- Fridman, Lex, et al. "Automated synchronization of driving data using vibration and steering events." Pattern Recognition Letters 75 (2016): 9-15.
- Sandha, Sandeep Singh, et al. "Exploiting smartphone peripherals for precise time synchronization." 2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP). IEEE, 2019.