ECE 635- Network Embedded System Design ~ Final Project

Time Synchronization via Sensing

https://github.com/poojithaumas/Network-Embedded-Final-Project-

Team Members:

Abishek Thirunavukkarasu (35191531)

Sanjay Kumar Palanivelu (35197855)

Deepikaveni Venkatanarayanan (34546183)

Poojitha Singh

Motivation:

Time synchronization is the basis for distributed embedded systems. From autonomous vehicles synchronizing actions to IoT smart homes cutting energy usage, synchronized clocks are required for reliable communication, event ordering, and control action.

Legacy techniques like NTP (Network Time Protocol) and PTP (Precision Time Protocol) get synchronized, but are:

- Too heavyweight for low-power IoT devices.
- Resource hungry, using extra packets, extra computation, and reliable network links.
- Vulnerable to drift in noisy wireless environments.

Concurrently, most IoT/edge devices already produce timestamped sensor readings (temperature, vibration, light, etc.) sent from time to time to gateways. We could thus forego loading dedicated synchronisation protocols on devices and instead use native sensor data streams to synchronize clocks across nodes.

This project aims to design and test a lightweight, sensing-based time synchronisation protocol for embedded IoT devices, based on a Raspberry Pi gateway.

Design Goals:

- **Protocol Design**: Design a synchronization technique based on periodic timestamped sensor data instead of using sync packets.
- **Edge-Centric Execution**: Ensure sync logic is executed completely on a Raspberry Pi gateway to minimize device-side complexity.
- **Drift & Delay Characterization**: Characterize network delay and relative clock drift between heterogeneous ESP32 devices.
- **Lightweight Implementation**: Demonstrate feasibility on resource-constrained hardware (ESP32).

- **Benchmarking**: Compare with NTP/PTP for performance to measure efficiency and accuracy trade-offs.
- **Scalability**: Make the protocol support more than two devices.
- **Future Extensions**: Explore event-based sync (e.g., pulses of light and vibration) and ML-augmented drift prediction for higher accuracy.

Deliverables:

- **ESP32 Firmware:** Sensor data logging + timestamp embedding + transmission via Wi-Fi/Bluetooth.
- **Raspberry Pi Protocol Stack:** Software for ingesting data, applying sync protocol, and computing drift adjustments.
- **Network Analysis:** Measurements of one-way delay, jitter, and packet loss impact on sync accuracy.
- **Drift Estimation Module:** Algorithms to calculate and correct clock offsets between devices.
- **Benchmark Suite:** Accuracy/error comparison against NTP/PTP under identical conditions.
- **Visualization Toolkit:** Graphs of drift, error trends, delay distributions.
- **Final Report + GitHub Repo:** Documentation, results, and implementation details.

System Blocks:

1. Sensor Nodes (ESP32)

- Collect periodic sensor readings (temperature/light/vibration).
- Attach local timestamps.
- Transmit data via Wi-Fi/Bluetooth to Raspberry Pi.

2. Communication Layer

- Provides link between ESP32 nodes and Raspberry Pi.
- Handles latency, jitter, and packet reordering.

3. Raspberry Pi Gateway

- Receives sensor streams.
- Runs synchronization protocol.
- Estimates relative drift and applies correction.

4. Drift & Delay Estimation Module

- Performs statistical analysis of inter-arrival times.
- Builds clock offset models.
- Optionally uses regression/ML to predict long-term drift.

5. Evaluation & Benchmarking

- Runs side-by-side with NTP/PTP for accuracy comparison.
- Generates plots and logs for quantitative analysis.

Hardware & Software Requirements:

i. Hardware

- **ESP32 Thing boards**: For sensor nodes.
- **Raspberry Pi 4**: Gateway node.
- **Sensors**: Light sensor, accelerometer/gyroscope(ICM-20948), vibration sensor.
- **Networking**: Wi-Fi/BLE.

ii. Software

- **Arduino IDE** (firmware development).
- **Python 3.x**(sync algorithm, analysis).
- **pandas, numpy, matplotlib** (data analysis + visualization).
- **NTP/PTP clients** (benchmarking).

Project Timeline:

➢ **Week 1–2 (Setup & Firmware):**
- Bring up ESP32 + Raspberry Pi environment.
- Implement basic timestamped data logging + transmission.

➢ **Week 3–4 (Protocol Implementation):**
- Prototype sync protocol on Raspberry Pi.
- Begin initial delay/jitter measurements.

➢ **Week 5–6 (Algorithm Development):**
- Implement drift estimation and correction logic.
- Explore adaptive methods (event-based, ML).

- **Week 7 (Benchmarking):**
  - Evaluate under varied network loads.
  - Collect experimental results.
- **Week 8 (Integration & Finalization):**
  - Polish GitHub repo.
  - Prepare graphs, plots, and a final report.
  - Rehearse demo + presentation.

Team Members & Roles:

Abishek Thirunavukkarasu – Setup + Networking

- Hardware bring-up (ESP32s + Raspberry Pi, sensors).
- Configure Wi-Fi/Bluetooth stack between ESP32s and Raspberry Pi.
- Measure and log network delay, jitter, and packet loss.
- Maintain connectivity tests and debugging scripts.

Poojitha Singh – Embedded Software + Data Analysis

- Develop ESP32 firmware for timestamped sensor data generation and transmission.
- Validate the integrity of transmitted sensor logs.
- Collect experimental data sets from multiple runs.
- Perform initial exploratory analysis.

Deepikaveni Venkatanarayanan – Algorithm Design + Protocol Development

- Design synchronization protocol logic on Raspberry Pi.
- Implement drift estimation and compensation algorithm.
- Explore adaptive correction as an extension.
- Benchmark protocol against standard implementations.

Sanjay Kumar Palanivelu – Research + Writing

- Review references
- Document design goals, system architecture, and evaluation methodology.
- Maintain GitHub repo (README, code structure, issue tracking).
- Compile final report and presentation materials.

https://github.com/poojithaumas/Network-Embedded-Final-Project-

References

[1] HAEST: Harvesting Ambient Events to Synchronize Time across Heterogeneous IoT Devices.

[2] Automated Synchronization of Driving Data Using Vibration and Steering Events.

[3] Exploiting Smartphone Peripherals for Precise Time Synchronization.