engineering proceedings

MDPI

Proceeding Paper

IOTAfy: An ESP32-Based OTA Firmware Management Platform for Scalable IoT Deployments †

Ioannis Chrysovalantis Panagou, Stylianos Katsoulis, Evangelos Nannos, Fotios Zantalis and Grigorios Koulouras *

TelSiP Research Laboratory, Department of Electrical and Electronic Engineering, School of Engineering, University of West Attica, Ancient Olive Grove Campus, 250 Thivon Str., GR-12241 Athens, Greece; ipanagou@uniwa.gr (I.C.P.); skatsoulis@uniwa.gr (S.K.); enannos@uniwa.gr (E.N.); fzantalis@uniwa.gr (F.Z.)

* Correspondence: gregkoul@uniwa.gr; Tel.: +30-210-53851647

† Presented at the 6th International Electronic Conference on Applied Sciences, 9–11 December 2025; Available online: https://sciforum.net/event/ASEC2025.

Abstract

Managing firmware updates in large-scale IoT deployments presents significant challenges regarding security, reliability, and operational cost. This paper presents the design and implementation of "IOTAfy", a new, open-source, comprehensive, Over-The-Air (OTA) firmware management platform tailored for ESP32-based IoT devices. "IOTAfy" distinguishes itself through its efficient database design and asynchronous update mechanism, enabling scalable deployments. The proposed system comprises a device-side solution for ESP32 microcontrollers, incorporating a custom bootloader, an application firmware boilerplate code, and an OTA library, alongside a web-based management interface developed using PHP, SQLite3, and Bootstrap. This integrated approach facilitates secure and reliable OTA application firmware updates. The platform enables centralized monitoring of device status, scheduled firmware updates, and robust version control for multiple IoT devices. OTA updates are implemented directly on the ESP32, eliminating the need for physical access or intervention. The web interface provides administrators with features for group upgrades, rollback capabilities, real-time update status monitoring and alerts. The system was tested in a controlled environment with 50 ESP32 devices, achieving a 98% success rate for OTA updates. The results demonstrate the potential for significant cost savings and reduced maintenance time compared to manual update processes. The architecture of the "IOTAfy" platform, leveraging an efficient database design and asynchronous update mechanism, facilitates scalability to hundreds or thousands of devices. This work offers a practical and scalable solution for managing firmware in large-scale IoT deployments, contributing to enhanced device security, improved reliability, and reduced operational expenditures.

Keywords: Internet of Things (IoT); platform; firmware; Over-The-Air (OTA); ESP32; security; scalability

Check for updates

Academic Editor: Alessandro Lo Schiavo

Published: 19 February 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

The widespread adoption of the Internet of Things (IoT) solutions has fundamentally transformed the landscape of industrial automation, smart agriculture, smart cities and consumer electronics [1]. As IoT networks evolve from small-scale pilot projects to massive deployments involving thousands of interconnected nodes, the operational lifecycle management of these devices becomes a critical burden [2]. Unlike traditional IT assets located

in regulated conditions server rooms, IoT devices are frequently deployed in geographically dispersed, harsh, or physically inaccessible environments.

In this context, the ability to remotely update device firmware, known as Over-The-Air (OTA) updates, is not only a feature but a fundamental requirement for system extended life-cycle and security. Firmware updates serve three critical functions: patching security vulnerabilities discovered post-deployment, fixing software bugs that degrade performance, and introducing new features [3].

In the absence of robust remote access and management, administrators must often rely on physical intervention to perform on-site updates via serial interfaces (UART/USB). This labor-intensive methodology is unfeasible for large-scale deployments, especially those situated in geographically distributed or logistically constrained environments. Consequently, Over-The-Air (OTA) update mechanisms have emerged as a fundamental requirement for modern IoT architectures, as noted in recent reviews of OTA security challenges [4].

1.1. The Challenge of Current Solutions

While the necessity of OTA is undisputed, the implementation strategies which are available to developers often present a dilemma between cost, complexity, and control. On one hand, major cloud providers offer mature IoT device management capabilities (e.g., AWS IoT, Azure IoT Hub). However, they introduce significant complexity and higher operational costs. As highlighted in a recent comprehensive evaluation by Panagou et al. [5], the selection of an IoT cloud platform is a multi-criteria decision problem where factors such as cost, vendor lock-in, and interoperability often weigh heavily against the adoption of these hyperscale solutions. Panagou et al. emphasize that the complexity of feature sets in major cloud platforms can hinder rapid deployment for specific use cases.

On the other hand, ad-hoc DIY solutions often rely on simple HTTP file hosting services without adequate management. These implementations frequently lack critical safety features such as atomic updates and automated rollbacks [6]. A potential failure during the update operations may leave the device in a critically corrupted state, often called “bricked”, effectively destroying its operational utility and requiring on-site technical support. Recent studies in 2025 have further emphasized the shift towards edge-based management to reduce latency and dependency on centralized cloud structures [7].

1.2. Contribution

This paper introduces IOTAfy, a purpose-built, open-source OTA management platform designed specifically for the ESP32 microcontroller ecosystem. IOTAfy aims to bridge the gap between cost-intensive enterprise solution and insecure custom-developed scripts. The key contributions of this study are: (1) a lightweight architecture utilizing a PHP/SQLite3 stack; (2) an asynchronous client library allowing non-blocking updates; and (3) experimental validation involving 50 physical devices and large binary payloads.

2. Materials and Methods

The IOTAfy platform is based on a classic client–server model, utilizing HTTP polling for command and control. The system is comprised of two primary components: the Central Management Server (CMS) and the Device-Side Firmware (DSF).

2.1. System Architecture

The architecture is designed to minimize server overhead while ensuring reliable delivery of binary payloads. Figure 1 illustrates the high-level interaction between these components.

https://doi.org/10.3390/engproc2026124040

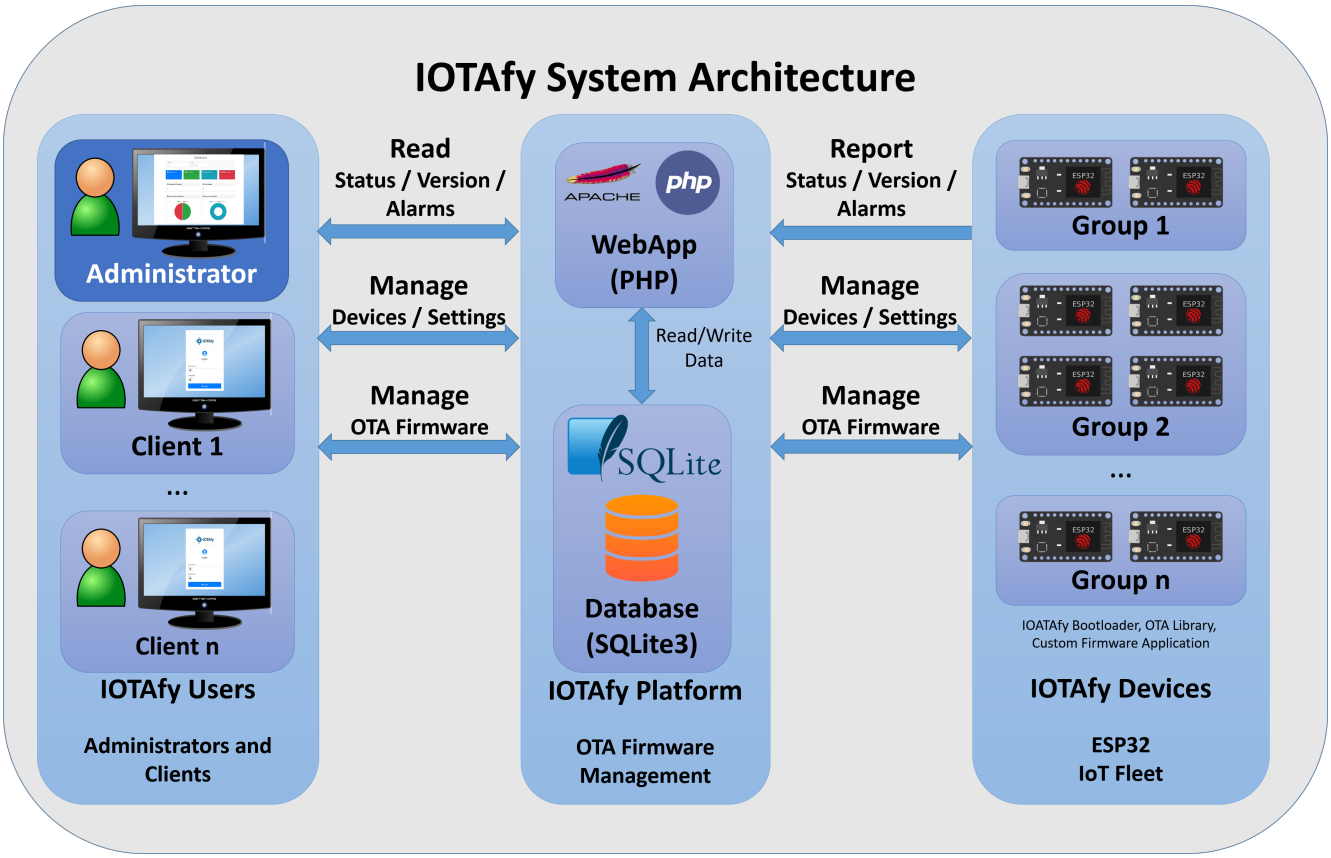


Figure 1. IOTAfy System Architecture. The diagram illustrates the interaction between Users (Administrators/Operators), the Central Platform (WebApp/SQLite), and the IoT devices (ESP32 Groups). The platform facilitates bidirectional communication for management and reporting, while managing OTA firmware distribution.

2.1.1. The Central Management Server

The server-side component is designed to be hosting-agnostic, capable of running on any standard LAMP stack.

- **Database Design:** To ensure portability, SQLite3 was selected [8]. The schema consists of three core tables: `devices` (storing identity and state), `firmware_repo` (storing binaries and checksums), and `groups` (orchestration for batch management).
- **Web Interface:** Developed using PHP [9] and Bootstrap [10], offering a dashboard for real-time heartbeat monitoring and version control.

2.1.2. Device-Side Implementation

Client devices runs on ESP32-S3, ESP32-C6 and ESP32-WROOM-32 modules [11]. The development of firmware updates utilizes the Espressif IoT Development Framework (ESP-IDF) [12].

2.1.3. Partition Table Design

A critical aspect of the IOTAfy development is the allocation of the ESP32 flash memory. In order to support automatic updates and rollback capabilities, a custom partition table is defined [12]. The memory map is divided into specific segments, as detailed in Table 1 and Figure 2.

Crucially, the Non-Volatile Storage (NVS) partition serves as a shared storage resource for both application slots (app0 and app1). To maintain system stability during the Over-The-Air (OTA) process, particularly in the event of a rollback to an older firmware version,

strictly backward-compatible data handling is enforced. The updated firmware is designed to read existing configuration keys without altering their data structure during the initial boot phase. To ensure backward compatibility, the shared NVS namespace is preserved across updates. New configuration parameters that require a different data structure are assigned unique, versioned keys (e.g., param_v2), leaving the legacy keys intact for use by the rollback firmware. This strategy ensures that if the new firmware fails to boot and the bootloader reverts to the previous version, the older firmware does not encounter corrupted or incompatible persistent data structures.

Table 1. Detailed memory specifications for the IOTAfy partition table (4 MB Flash).

Name	Type	SubType	Offset	Size	Description
nvs	data	nvs	0x009000	0x005000	Non-Volatile Storage (WiFi credentials, configuration)
otadata	data	ota	0x00E000	0x002000	OTA data (boot slot selection flags)
app0	app	ota_0	0x010000	0x1C0000	Application Slot A (Active/Passive)
app1	app	ota_1	0x1D0000	0x1C0000	Application Slot B (Active/Passive)
spiffs	data	spiffs	0x390000	0x070000	File System (Static Data/Assets)

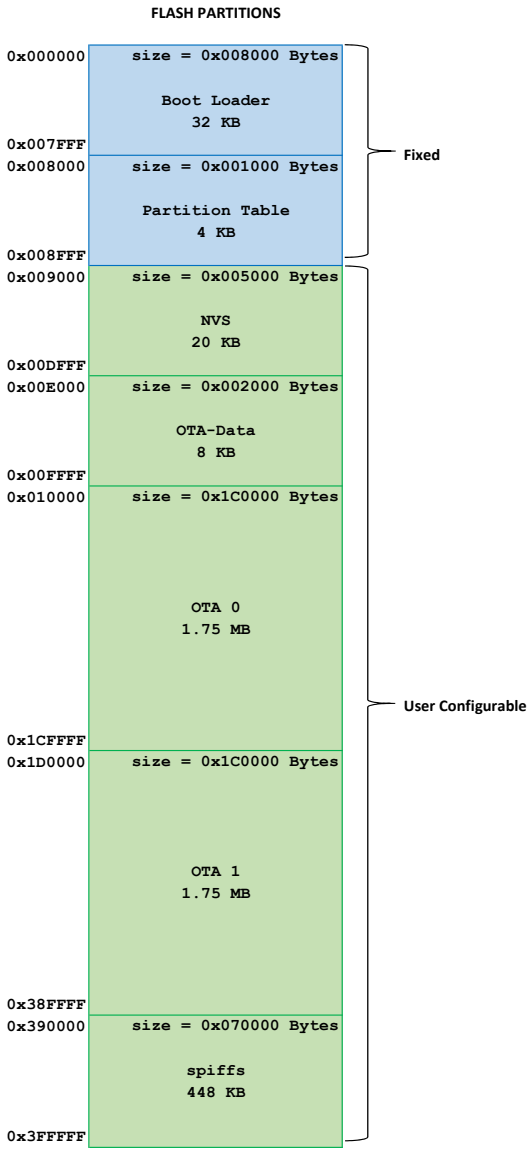


Figure 2. Visual representation of the IOTAfy partition memory allocation (4 MB Flash). The blue sections indicate fixed system partitions, whereas the green sections represent user-configurable partitions.

The firmware update workflow relies on a dual-partition architecture managed by the `otadata` partition. This partition acts as the key control mechanism, storing boot flags and instructing the bootloader on which application partition (e.g., `app0` or `app1`) to execute. As depicted in Figure 3, the process begins with the OTA data partition initially referencing OTA 0 as the active partition (Step 0). During the update procedure, the device identifies the currently running partition and targets the idle one. The passive partition (OTA 1) is first erased (Step 1) and subsequently populated with the new firmware image (Step 2). Following the final binary commit, the system initiates a validation check (typically an MD5 checksum). If the integrity check passes, the `otadata` partition is modified to point to OTA 1, flagging it as the active partition (Step 3). This ensures the new firmware loads on the next power cycle. To ensure system reliability, if the new application fails to boot—for example, due to a Watchdog Timer (WDT) trigger—the bootloader automatically instructs the `otadata` partition to revert changes and point back to the known working partition (`app0`). Figure 4 demonstrates the symmetrical process wherein OTA 1 serves as the active partition and OTA 0 functions as the target for the upgrade.

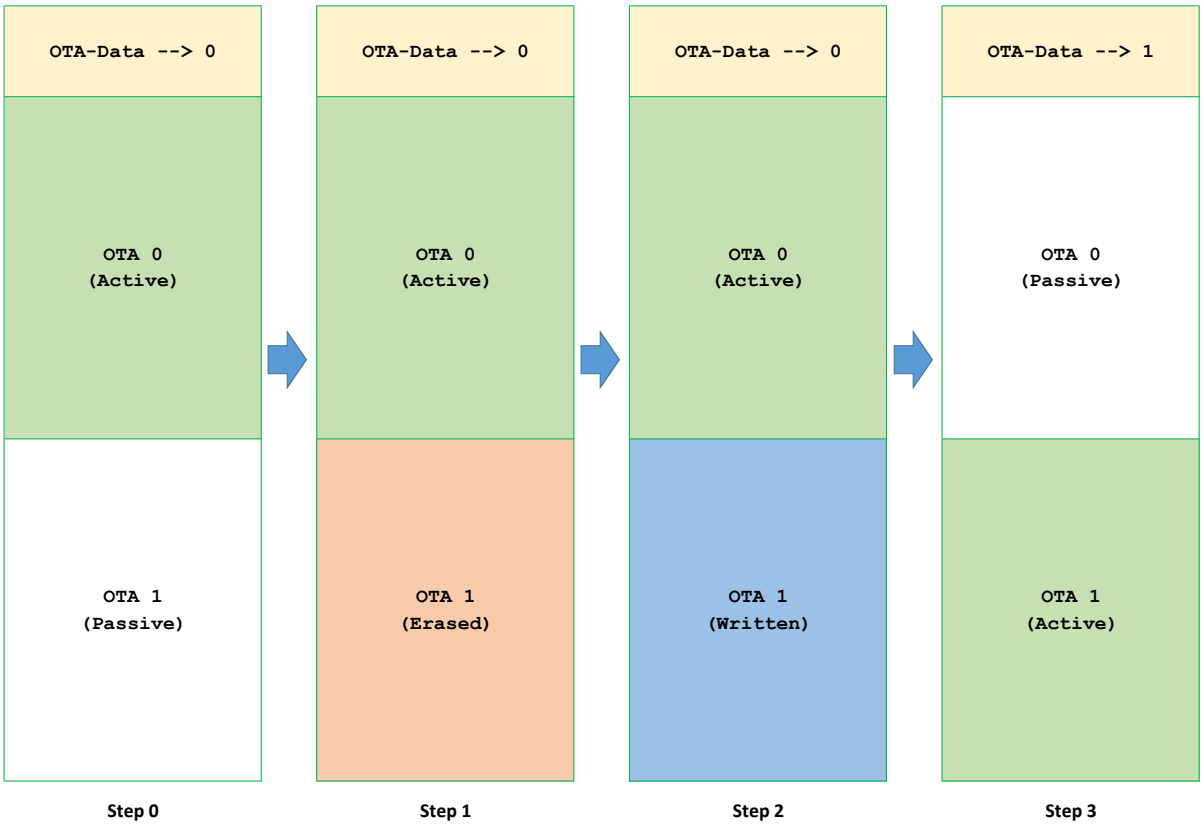


Figure 3. IOTAfy Firmware Upgrade Flow (OTA 0 → OTA 1). The colors indicate the current state of each partition: green represents the active and running firmware, white denotes the passive/inactive partition, orange indicates an erased partition, blue highlights a partition being written with new firmware, and yellow represents the OTA-Data partition that tracks the active boot index.

2.2. Asynchronous OTA Workflow

To prevent blocking the main execution loop and ensure the device remains responsive to sensor inputs, the OTA process is implemented as a non-blocking state machine. Figure 5 illustrates the detailed sequence of operations, identifying the interaction between the Running Application, the IOTAfy Server, and the Bootloader. The workflow operates through the following distinct states, utilizing HTTP over TLS (HTTPS) to ensure channel security:

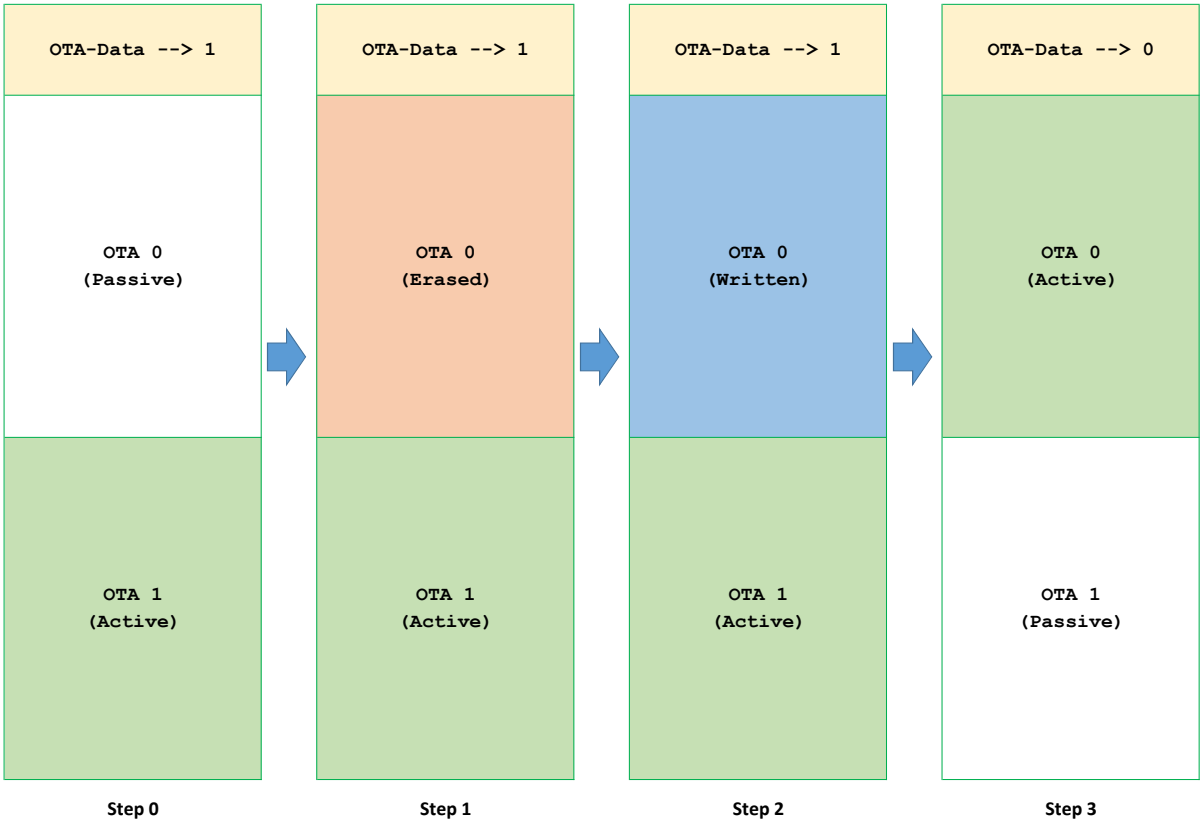


Figure 4. IOTAfy Firmware Upgrade Flow (OTA 1 → OTA 0). The colors indicate the current state of each partition: green represents the active and running firmware, white denotes the passive/inactive partition, orange indicates an erased partition, blue highlights a partition being written with new firmware, and yellow represents the OTA-Data partition that tracks the active boot index.

1. **Trigger / Polling:** The update process is initiated either by a periodic timer or a manual event triggered by the authorized user. By default, the device is configured to poll the IOTAfy Server every 60 min, though users can enable or disable this automatic procedure via the server's web interface. The system supports holistic remote device management, enabling targeted firmware control for individual devices or entire groups. Furthermore, the version control logic is agnostic to release chronology, supporting both upgrades and downgrades.
2. **Handshake:** The device sends a request containing its MAC address and current firmware version. This serves as a basic authentication mechanism, allowing the server to identify the device and determine if a specific update target exists in the database.
3. **Response:** The server verifies the device status. If an update is queued, it responds with the binary URL and an MD5 checksum for integrity verification.
4. **Downloading:** Upon receiving a valid update response, the application downloads the binary payload in chunks. This process is asynchronous; the device writes the data to the passive partition (e.g., `app1`) while continuing to execute critical tasks on the active partition (e.g., `app0`).
5. **Verification:** After the download completes, the firmware calculates the MD5 checksum of the written data and compares it with the signature provided by the server to ensure integrity [3].
6. **Reboot & Switch:** If validation succeeds, the `otadata` partition is updated to point to the new boot slot, and a system reboot is triggered.

7. **Confirmation:** Upon booting the new firmware, a self-test is performed. If successful, the firmware explicitly marks the new application as valid. If the WDT resets the device before this confirmation (due to a crash), the bootloader automatically reverts to the previous known-good firmware. Furthermore, in such an event, the system automatically disables the automatic update procedure to prevent recursive boot loops.

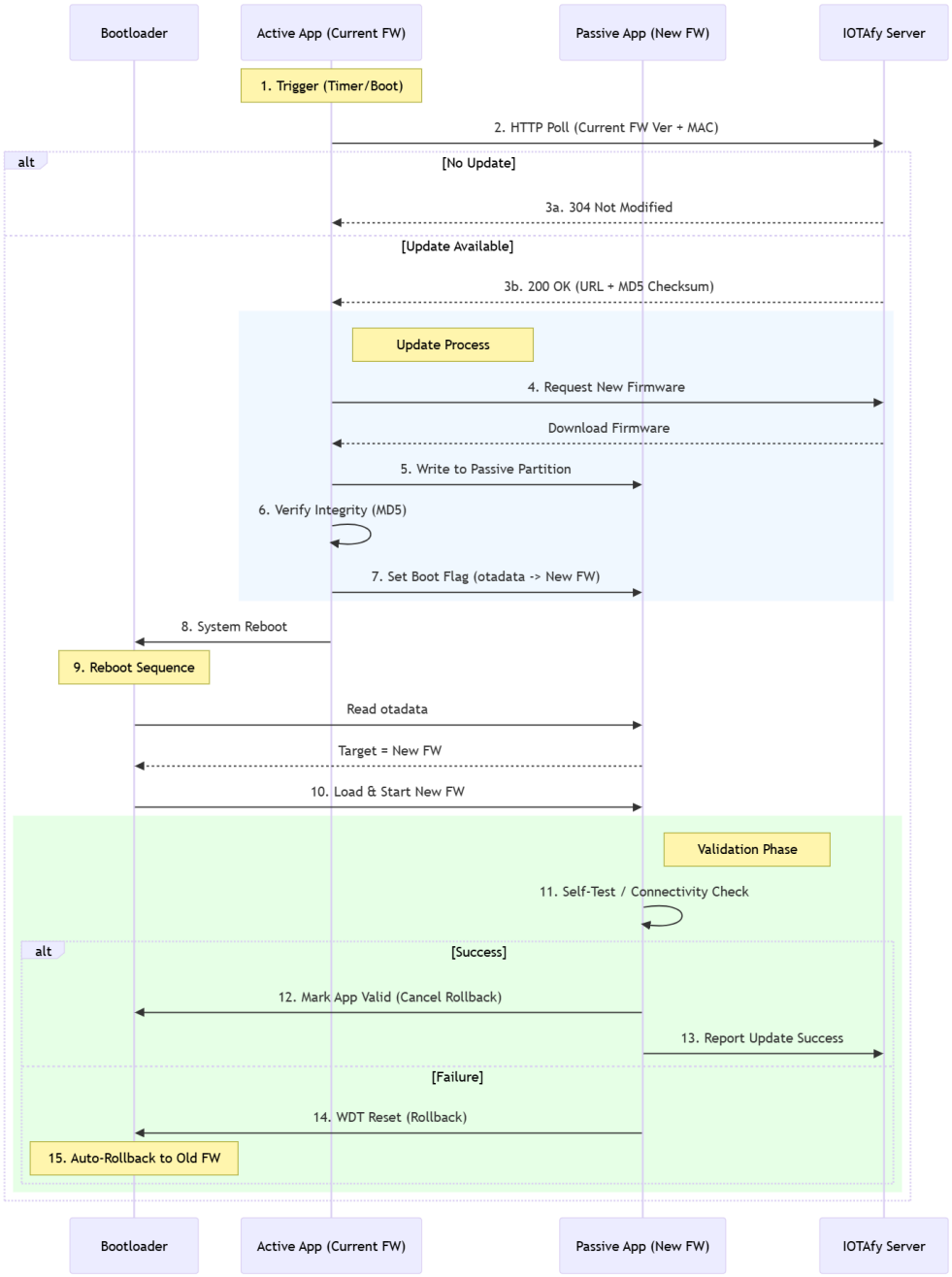


Figure 5. OTA Update Sequence Diagram. The diagram illustrates the interaction between the Server, the Application (Running Firmware), and the Bootloader. The background colors denote distinct operational phases: the light blue area groups the steps of the core Update Process (downloading and writing the new firmware), while the light green area highlights the Validation Phase (self-testing, confirmation, or rollback). Additionally, the yellow boxes emphasize key events, triggers, and phase labels.

https://doi.org/10.3390/engproc2026124040

3. Results

For evaluation purposes, a physical testbed was performed using 50 ESP32-S3, ESP32-C6 and ESP32-WROOM-32 devices connected to a standard 802.11 b/g/n Wi-Fi router.

3.1. Update Success Rate

A series of 5 full-fleet update cycles were executed (250 total attempts) to rigorously evaluate the reliability of the OTA mechanism. The system achieved an overall success rate of 98% (245/250). Table 2 presents a detailed breakdown of each update cycle, categorizing the specific causes of failure.

Table 2. Detailed breakdown of the five fleet-wide update cycles (250 total attempts), categorizing success rates and specific failure modes.

Cycle	Attempts	Successful	Failed	Failure Reason
1	50	50	0	-
2	50	49	1	Network Timeout
3	50	48	2	Power Loss (Simulated)
4	50	49	1	Network Timeout
5	50	49	1	Network Timeout

The recorded failures were identified as isolated incidents rather than systemic defects. Specifically, three failures were attributed to transient network timeouts, and two were the result of simulated power losses during the critical flash-writing phase. In all 5 failure cases, the rollback mechanism described in Section 2.1.3 functioned correctly, automatically reverting the devices to the previous stable firmware version and preventing any "bricked" states.

3.2. Performance Metrics and Dual Core Analysis

For this evaluation, we utilized a feature-rich, production-grade firmware binary with a fixed size of 1.2 MB. The file size was selected to resemble a complex IoT application, accounting for both TLS security libraries and extensive sensor support.

We measured the time required to update the fleet under typical network conditions and analyzed the CPU utilization of the ESP32's dual-core architecture during the process, as summarized in Table 3. The ESP32 utilizes Core 0 (IOTAfy Processes) for Wi-Fi/Bluetooth/IOTAfy stacks and Core 1 (Application Processes) for user code.

Table 3. Performance Metrics and Dual-Core Resource Usage.

Metric	Value
Firmware Binary Size	1.2 MB
Average Network Throughput	∼180 KB/s
Download Time (HTTP Stream)	6.8 s
Flash Writing & Validation Time	∼11.0 s
Total Update Time per Device	17.8 s
ESP32 Resource Utilization:	
Core 0 Load (Wi-Fi/IOTAfy Stack)	32% (Peak)
Core 1 Load (Application/Sensors)	12% (Average)
Server-Side Metrics:	
Server CPU Load (50 concurrent)	16% (Single vCPU)
Database Query Time (SQLite3)	<5 ms

https://doi.org/10.3390/engproc2026124040

Data Analysis

The duration of the update operation of the 1.2 MB firmware file was completed in 17.8 s. The breakdown of CPU usage confirms the efficiency of the asynchronous design:

- Core 0 bore the burden of the network overhead (TCP/IP stack), peaking at 32% usage.
- Core 1, where the user application resides, maintained a low average load of 12%. This demonstrates that the device remained responsive to sensor inputs and local control logic even during the critical firmware download phase.

The SQLite3 backend on the server successfully handled the concurrent polling with negligible latency.

3.3. Comparative Cost and Time Analysis

For quantifying the operational benefits, we compared IOTAfy against a traditional manual update process.

- **Manual Scenario:** A technician must travel so as to get to the device, open the enclosure, connect his equipment via USB, flash the firmware, and verify the boot sequence. We estimate that the update/verification process takes an average of 3 min per device, excluding travel time.
- **IOTAfy Scenario:** The administrator uploads the firmware file once. The devices are being updated autonomously and in parallel.

Calculations for a 50-device fleet (Equations (1) and (2)):

$$\text{Manual Time} = 50 \text{ devices} \times 3 \text{ min/device} = 150 \text{ min } (2.5 \text{ h}) \quad (1)$$

$$\text{IOTAfy Time} \approx 3 \text{ min (for full fleet propagation)} \quad (2)$$

This demonstrates an improvement in time efficiency of approximately 98%. If we take into consideration the travel time and expenses required for geographically distributed devices (e.g., smart meters across a city), the cost savings become exponential, effectively reducing the marginal cost of software maintenance to near zero.

4. Discussion

The implementation and performance of the IOTAfy demonstrate that a lightweight, open-source stack can effectively manage IoT fleets without the overhead of enterprise cloud solutions.

4.1. Scalability and Performance

The use of SQLite3 was validated as a suitable choice for the observed scale of 50 devices. While typical web architectures suggest MySQL/PostgreSQL for scalability, the specific workload of an OTA server is "read-heavy" (devices checking for updates) and "write-light". SQLite3 handles concurrent reads efficiently. However, for deployments exceeding 10,000 concurrent devices, migrating the backend to a connection-pooled MySQL instance would be required. Scaling deployments to support multiple concurrent devices introduces additional challenges regarding communication protocols and workload balancing.

4.2. Security Implications

The implementation of the rollback mechanism is critical for operational security. IOTAfy minimizes operational risk by implementing an automatic rollback mechanism, ensuring that failed updates do not result in "bricked" devices and the high costs of on-site recovery visits for remote deployments. While the current architecture utilizes HTTPS to secure the communication channel and authenticate the server, establishing strict client-side trust remains a challenge in lightweight IoT implementations. Currently, the system

identifies devices via MAC address protected by the TLS tunnel. However, to prevent unauthorized devices from querying the server in a zero-trust network environment, future revisions will introduce mutual TLS (mTLS). This would require devices to present a valid client certificate during the handshake, ensuring cryptographically verified identity before any firmware metadata is exchanged. Regarding data integrity, the system currently uses MD5 checksums. Although sufficient for detecting transmission errors, strictly secure deployments should transition to SHA-256 and Digital Code Signing (using ESP32 eFuse) to prevent malicious firmware injection, as recommended in recent security reviews [13].

4.3. Cost Analysis vs. Cloud Platforms

Referring back to the analysis by Panagou et al. [5], the decision between building or purchasing a solution is determined by hidden costs. Cloud platforms typically charge based on data transfer. Sending a 1.2 MB file to 1000 devices (1.2 GB total) repeatedly can incur substantial costs. IOTAfy, being self-hosted, incurs only the fixed cost of hosting the server, making the financial model predictable. This cost-efficiency is a key driver identified in the 2024 Eclipse IoT Developer Survey [2].

5. Conclusions

In this work, we presented the design, implementation, and evaluation of "IOTAfy", an open-source, end-to-end firmware management platform tailored for the ESP32 ecosystem. Addressing the critical challenges of security, scalability, and expenditure in large-scale IoT deployments, IOTAfy bridges the gap between rudimentary ad-hoc scripts and complex, capital-intensive commercial cloud suites.

The empirical evaluation, conducted on a physical testbed of 50 devices, validated the robustness of the proposed architecture. The system achieved a 98% update success rate under realistic network conditions, even when transmitting large, feature-rich binary payloads of 1.2 MB. Crucially, the implementation of the custom bootloader interaction and the A/B partitioning scheme proved resilient against catastrophic failures; in all induced power-loss scenarios, the devices successfully reverted to a functional state, eliminating the risk of "bricking" remote assets.

From an operational perspective, the comparative analysis demonstrated that IOTAfy offers a transformative improvement in maintenance efficiency. By reducing the time required for fleet-wide updates from hours to minutes and reducing the maintenance latency by 98%, the platform effectively removes the geographical constraints on IoT scalability. Furthermore, the proposed lightweight server architecture, built on SQLite3, demonstrates efficient performance under high-frequency polling workloads while maintaining low resource overhead and offers a viable alternative to the consumption-based pricing structures of enterprise cloud platforms [5].

Future Work

While IOTAfy provides a solid foundation for secure OTA management, future research and developments will focus on the following key areas:

1. **Enhanced Security Chain of Trust:** Integrating Secure Boot V2 and Flash Encryption to ensure that only digitally signed binaries from authorized administrators can be executed, thereby mitigating the risk of supply-chain attacks.
2. **Protocol Optimization:** Transitioning from the current HTTP-polling model to an MQTT-based "Push" architecture. This will significantly reduce bandwidth consumption and server load by eliminating unnecessary check-ins when no updates are pending.

https://doi.org/10.3390/engproc2026124040

3. **Hardware Agnosticism:** Abstracting the device-side library to support a wider range of microcontrollers, such as the STMicroelectronics STM32 series and the Raspberry Pi RP2040, enabling the management of heterogeneous IoT fleets within a single, unified user's interface.
4. **BLE-to-Cloud OTA Gateway:** Expanding connectivity options to support ultra-low-power Bluetooth Low Energy (BLE) devices. Following the architectural insights provided by [14], we aim to develop a specialized BLE gateway. This gateway will bridge resource-constrained BLE sensors to the IOTAfy platform, enabling secure OTA capabilities for devices that lack native Wi-Fi or cellular connectivity while maintaining the power-efficiency benefits of the BLE protocol.

In conclusion, IOTAfy provides the IoT domain a practical, scalable, and cost-effective framework, enabling developers and organizations to maintain secure and up-to-date sensor networks with greater assurance.

Author Contributions: Conceptualization, I.C.P., S.K., E.N., F.Z. and G.K.; methodology, I.C.P., S.K., E.N., F.Z. and G.K.; software, I.C.P., S.K., E.N., F.Z. and G.K.; validation, I.C.P., S.K., E.N., F.Z., and G.K.; formal analysis, I.C.P., S.K., E.N., F.Z. and G.K.; investigation, I.C.P., S.K., E.N., F.Z. and G.K.; resources, I.C.P., S.K., E.N., F.Z. and G.K.; data curation, I.C.P., S.K., E.N., F.Z. and G.K.; writing—original draft preparation, I.C.P., S.K., E.N., F.Z. and G.K.; writing—review and editing, I.C.P., S.K., E.N., F.Z. and G.K.; visualization, I.C.P., S.K., E.N., F.Z. and G.K.; supervision, G.K.; project administration, G.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The source code used to support the findings of this study is available at https://github.com/TelSiP-RLab/IOTAfy (accessed on 24 January 2026). The online web-based tool for evaluating IoT Cloud Platforms is available at https://iotafy.telsip.uniwa.gr (accessed on 24 January 2026). In this study, no new data were generated.

Acknowledgments: During manuscript preparation, the authors used the Gemini 3.0 assistant solely for basic editorial support. The tool was used to check grammar, sentence structure, spelling, and formatting consistency. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AWS	Amazon Web Services
BLE	Bluetooth Low Energy
CMS	Content Management System
CPU	Central Processing Unit
DIY	Do It Yourself
DSF	Context-dependent: often Distributed Service Framework
ESP-IDF	Espressif IoT Development Framework
HTTP	Hypertext Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IoT	Internet of Things
LAMP	Linux + Apache + MySQL + PHP
MAC	Media Access Control Address
MD5	Message-Digest Algorithm 5
MQTT	Message Queuing Telemetry Transport
MQTTS	Message Queuing Telemetry Transport Secured
mTLS	mutual Transport Layer Security

NVS	Non-Volatile Storage
OTA	Over-the-Air
TCP/IP	Transmission Control Protocol / Internet Protocol
TLS	Transport Layer Security
UART	Universal Asynchronous Receiver-Transmitter
URL	Uniform Resource Locator
USB	Universal Serial Bus
vCPU	Virtual Central Processing Unit
WDT	Watchdog Timer
Wi-Fi	Wireless Fidelity

References

1. Din, I.U.; Guizani, M.; Hassan, S.; Kim, B.S.; Khan, M.K.; Atiquzzaman, M.; Ahmed, S.H. The Internet of Things: A review of enabled technologies and future challenges. *IEEE Access* **2018**, *7*, 7606–7640. [CrossRef]
2. Eclipse Foundation. *2024 IoT & Embedded Developer Survey Report*; Technical Report; Eclipse Foundation: Brussels, Belgium, 2024.
3. Malumbres, V.; Saldana, J.; Berné, G.; Modrego, J. Firmware updates over the air via lora: Unicast and broadcast combination for boosting update speed. *Sensors* **2024**, *24*, 2104. [CrossRef] [PubMed]
4. El Jaouhari, S.; Bouvet, E. Secure firmware Over-The-Air updates for IoT: Survey, challenges, and discussions. *Internet Things* **2022**, *18*, 100508. [CrossRef]
5. Panagou, I.C.; Katsoulis, S.; Nannos, E.; Zantalis, F.; Koulouras, G. A comprehensive evaluation of IoT cloud platforms: A feature-driven review with a decision-making tool. *Sensors* **2025**, *25*, 5124. [CrossRef] [PubMed]
6. Park, C.Y.; Lee, S.J.; Lee, I.G. Secure and Lightweight Firmware Over-the-Air Update Mechanism for Internet of Things. *Electronics* **2025**, *14*, 1583. [CrossRef]
7. Chang, Y.H.; Wu, F.C.; Lin, H.W. Design and implementation of esp32-based edge computing for object detection. *Sensors* **2025**, *25*, 1656. [CrossRef]
8. SQLite. SQLite. 2026. Available online: https://www.sqlite.org (accessed on 20 January 2026).
9. PHP Group. PHP: Hypertext Preprocessor. 2026. Available online: http://www.php.net (accessed on 20 January 2026).
10. Bootstrap Team. Bootstrap: The Most Popular HTML, CSS, and JS Library in the World. 2026. Available online: https://getbootstrap.com (accessed on 20 January 2026).
11. Espressif Systems. ESP32 Technical Reference Manual (Version 5.6). 2025. Available online: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf (accessed on 20 January 2026).
12. Espressif Systems. ESP-IDF Programming Guide: Partition Tables. 2026. Available online: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/partition-tables.html (accessed on 20 January 2026).
13. Barker, E. *Recommendation for Key Management: Part 1–General*; Technical Report NIST Special Publication 800-57 Part 1, Revision 5; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2020. [CrossRef]
14. Koulouras, G.; Katsoulis, S.; Zantalis, F. Evolution of Bluetooth Technology: BLE in the IoT Ecosystem. *Sensors* **2025**, *25*, 996. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.