Bootstrapping IoT Provisioning with PoMA

Extended Abstract

Lucas Gutierrez
lucas.gutierrez@digital-stream.com.ar
Stream S.A.
La Plata, Argentina

Federico Balaguer
federico.balaguer@lifia.info.unlp.edu.ar
LIFIA Fac. Informática, Universidad Nacional de La Plata
La Plata, Argentina
Stream S.A.
La Plata, Argentina

Abstract

The proliferation of Internet of Things (IoT) devices across diverse environments demands a robust, user-friendly, and efficient deployment strategy. Conventional setup methods typically involve complex, multi-step procedures that require specialized tools or technical expertise, increasing deployment time, operational costs, and the likelihood of human error.

This paper introduces PoMA (Poor Man's API), a lightweight protocol and accompanying library designed to simplify and accelerate the on-site configuration and initial connection of IoT devices. PoMA relies on three core commands and can operate over localized temporary wireless links—such as Bluetooth Low Energy or temporary Wi-Fi access points—or through Short Messaging Service (SMS). Each communication channel can be configured to different security levels based on the deployment scenario. The proposed solution provides a standardized, simple, and secure foundation for rapid IoT field installation.

Keywords

IoT Deployment, Field Provisioning, Zero-Touch Configuration, Device Setup Protocol

ACM Reference Format:
Lucas Gutierrez and Federico Balaguer. 2026. Bootstrapping IoT Provisioning with PoMA: Extended Abstract. In *8th International Workshop on Software Engineering Research and Practices for the IoT (SERP4IoT '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3786159.3788476

1 Introduction

Deploying Internet of Things (IoT) devices remains challenging. The initial configuration of each device involves several interdependent parameters—such as cloud endpoints, connectivity settings, and functional modes—that must be correctly defined before the device can operate.

These deployment challenges differ from what is typically referred to as provisioning. Provisioning is the essential first step in securely connecting an IoT device, encompassing identity verification, authentication, and configuration [2, 5]. In contrast, deployment focuses on enabling the device to function in its intended environment. In this sense, deployment is the bootstrap stage of the provisioning process.

In our experience, deploying IoT systems in rural areas with limited or unreliable network infrastructure is hard. We often encountered installations where mobile infrastructure was scarce, mobile Internet was available for a few hours per day, and Short Message Service (SMS) was the only stable communication method. Provisioning devices under such conditions proved time-consuming and error-prone. To address these limitations, we developed PoMA (Poor Man's API), a lightweight and adaptable protocol for field deployment and configuration of IoT devices [7].

2 Background

Provisioning is the essential first step in securely connecting an Internet of Things (IoT) device. It encompasses identity verification, authentication, and configuration [2, 5]. The most common approach to identity management relies on X.509 digital certificates, which enable large-scale secure authentication through mutual TLS (mTLS) [3]. However, in many practical scenarios—particularly during initial installation—this level of security introduces unnecessary complexity.

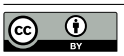
Beyond identity, provisioning typically uses specialized communication protocols such as MQTT (Message Queuing Telemetry Transport) [6] and CoAP (Constrained Application Protocol) [7]. While MQTT supports cloud-based communication, it requires an active network connection, and running an MQTT server within a constrained device is often impractical. CoAP, based on UDP, offers a RESTful interface for message exchange in formats such as JSON or XML, but its processing overhead increases memory usage and firmware complexity.

For more advanced devices requiring complete lifecycle management, Lightweight M2M (LwM2M) [1] extends CoAP to include configuration management, firmware updates, and monitoring capabilities. Similarly, local network setup can be simplified using standards such as the Wi-Fi Device Provisioning Protocol (DPP).

However, many existing provisioning approaches rely on embedded HTTP servers for local configuration. These methods are costly to maintain, increase firmware size, and significantly impact energy consumption—particularly for devices operating on battery power.

2.1 Problem Statement

Traditional HTTP-based provisioning systems pose several limitations that hinder practical IoT deployment, especially in resource-constrained or remote environments:



SERP4IoT '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2391-9/2026/04
https://doi.org/10.1145/3786159.3788476

- **Development and Maintenance:** Implementing and maintaining local web-based provisioning interfaces is time consuming and often outside the expertise of firmware developers.
- **Resource Consumption:** Embedded HTTP servers increase firmware size and drain limited battery resources.
- **User Experience:** Web forms built into device firmware are typically simplistic, outdated, and not user-friendly.
- **Limited Accessibility:** These interfaces can only be accessed locally, making remote reconfiguration impractical once devices are deployed.

These limitations underscore the necessity for a streamlined, remotely manageable configuration framework that prioritizes energy efficiency and architectural simplicity across diverse communication channels and deployment environments.

3 PoMA

3.1 Requirements

Four key requirements guided the design of PoMA:

- **Ease of Development and Maintenance:** The solution should be simple enough for junior firmware developers to implement and customize across different projects without extensive web development expertise.
- **Minimal Resource Footprint:** The protocol must occupy as little memory as possible and impose minimal impact on battery life.
- **Flexible User Interaction:** The mechanism should support both single-parameter configuration and bulk updates.
- **On- and Off-site Operation:** Devices must be reconfigurable not only during installation but also after deployment in the field—for instance, to recalibrate sensors or modify the data upload frequency.

Security considerations are intentionally excluded from PoMA's core design, as they fall under broader system-level mechanisms.

3.2 Base Design

PoMA is based on the idea that several firmware variables serve as runtime parameters. Usually, those variables are set at compilation time. Still, it is helpful to be able to change them at run time, such as sleep time, loading frequency, data endpoint, OTA endpoint, operation mode, etc. Moreover, the variables that comprise the configuration can change over time or between projects; therefore, it is essential to have a solution that can be customized quickly by any developer. PoMA considers that each available variable is a topic and that each variable can be used to get a value or to set a value. Listing 1 shows the definition of the structure of a Topic.

Listing 1: Topic's structure

```
typedef struct {
    char key[21];
    void (*getter)(char*, char*);
    void (*setter)(char*, char*);
    struct Topic *next;
} Topic;
```

PoMA maintains a linked list of Topic elements that are defined by a C structure with the following fields:

- **Key:** a fixed-length string
- **Getter:** a pointer to a getter function provided by the firmware developer. This function expects a socket file descriptor to write the response.
- **Setter:** a pointer to a setter function provided by the firmware developer. This function expects a socket file descriptor to write the response and a pointer to the arguments. It is the developer's responsibility to handle the arguments appropriately.
- **Next:** a pointer to the next Topic or Null

PoMA works based on messages of the form:

<command><topic><parameters>.

The available commands are:

- **Read:** it is a command that takes the form `? topic` and returns the assigned value or an ERROR if the topic is missing
- **Write:** it is a command that takes the form `= topic data` and returns an ACK or ERROR if the topic is missing
- **List:** it is a command that takes the form `*` and returns a list of available topics

Figure 1 shows a diagram of the publicly available initial design of PoMA. There are three main blocks of functionality: Parse Command, Find Topic, and Call Handler. Parse Command gets a message from a socket and validates if it is a well-formed command. It detects the command (setter, getter, or list) and the payload (topic and arguments). Find Topic gets the topic from the topic registry and takes the appropriate handler. The Call Handler calls the proper function, which, in turn, writes the operation's result back to the socket.

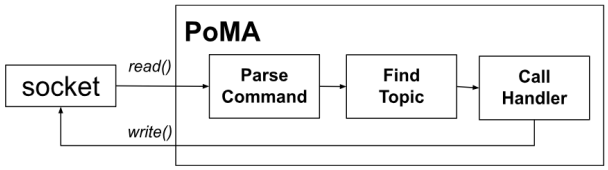


Figure 1: Overall Design of PoMA

The first design addresses the initial three requirements: ease of development/maintenance, resource strain, and user interaction.

PoMA was implemented using esp-idf, a port of FreeRTOS. PoMA runs in its own task after creating a Wi-Fi access point during a hard reset. The task that runs PoMA is sent to sleep after a given inactivity period. PoMA allows users to send single commands from a terminal or use a mobile application to send bulk commands.

The design and the available implementation were kept as simple as possible so that they could be adapted to scenarios not known at the time [8]. The WiFi access point handled the user authentication problem. Later, a secret key was added at compile time to each device; it was used to create the PoMA session.

One feature that proved to be invaluable was the capability to execute complex operations in conjunction with a read command.

In particular, we implemented a CommCheck topic with an associated write command that initiates communication with the cloud endpoint. This functionality allows installers to verify end-to-end device connectivity, ensuring that the device can successfully transmit data to the cloud and receive commands or firmware updates.

One problem that we faced was the need to reconfigure devices considering off-site scenarios. Every time there was a configuration issue with a deployed device, we needed someone nearby to reconfigure it.

3.3 Updated Design

Subsequent iterations separated the command source from the protocol core, introducing two independent connectors. Figure 2 shows the updated architecture of PoMA used internally at our company. Two module types are defined: Core and Connector. Core handles a received command; in this new version, Core does not run on its own thread or task. Connector implements the functionality required to handle a new command and to produce a reply. Connectors usually run in their own thread or task.

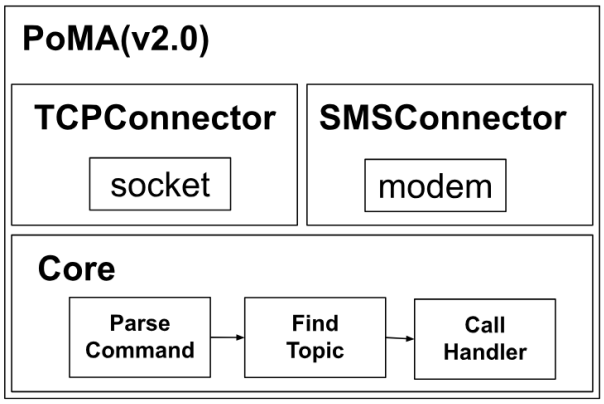


Figure 2: PoMA v2.0

TCPConnector manages the lifecycle of a socket; it implements the security mechanism in place for the firmware. It is a firmware responsibility to manage the lifecycle of the TCPConnector. Depending on the requirements, it can be activated after different events, such as a hard reset, RF-token proximity, or button press, among others.

SMSConnector manages the lifecycle of a cellphone modem; it implements the security mechanism in place for the firmware, such as a white list of cellphone numbers. Depending on the requirements, it can be activated after different events, such as a hard reset, data upload, or mobile connectivity failure.

The code implementing PoMA v2.0 is not open sourced yet. It was developed as part of an ESP-IDF-based firmware that runs on the ESP32 family of microcontrollers and multi-band cellular communication modules, such as Simcom's SIM700x and Quectel's BG96.

4 Conclusions and Future Work

PoMA was conceived to address practical challenges encountered during the deployment of IoT devices in rural and resource constrained environments. The protocol was designed with a clear focus on simplicity, low resource consumption, and ease of customization, allowing developers to configure devices efficiently across a wide range of field conditions.

From a software engineering perspective, PoMA embodies a minimalist design philosophy: it focuses on solving only the problems that are essential to deployment, avoiding unnecessary complexity or over-engineering. Rather than replacing existing protocols such as MQTT or CoAP, PoMA complements them by facilitating the initial configuration and bootstrapping phase before secure provisioning and cloud integration. It has also proven effective in machine-to-machine communication scenarios; Yahp! is a platform for prototyping haptic wearable devices [4], PoMa is used to communicate with mobile devices and wearable prototypes.

We are in the process of making the source code of the new architecture publicly available for ESP-IDF and Arduino.

Future work will extend PoMA's applicability to new communication contexts. We are currently developing a Bluetooth Connector and plan to introduce a satellite connector for ultra-low-bandwidth IoT networks, enabling devices to exchange configuration messages as small as 20 bytes per day. Additional directions include improving interoperability with standardized provisioning frameworks and exploring adaptive security mechanisms suitable for constrained devices.

References

[1] M. Alkwiefi. 2024. M2M Protocols: An Overview of LwM2M and XMPP Machine-to-Machine Protocols in IoT Context. In *2024 IEEE/ACIS 24th International Conference on Computer and Information Science (ICIS)*. IEEE, 209–215.
[2] I. Boškov, H. Yetgin, M. Vučnik, C. Fortuna, and M. Mohorčič. 2020. Time-to-provision evaluation of IoT devices using automated zero-touch provisioning. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*. Ieee, 1–7.
[3] Jakob Hallin. 2025. Evaluation of TLS and mTLS in Internet of Things systems.
[4] HapticsYahp. 2025. Yahp! Yet another haptic probe. https://github.com/hapticsYahp.
[5] S. Nastic, S. Sehic, D. H. Le, H. L. Truong, and S. Dustdar. 2014. Provisioning software-defined IoT cloud systems. In *2014 International Conference on Future Internet of Things and Cloud*. IEEE, 288–295.
[6] Akshatha PS, S. M. Dilip Kumar, and Venugopal KR. 2022. MQTT implementations, open issues, and challenges: A detailed comparison and survey. *International Journal of Sensors Wireless Communications and Control* 12, 8 (2022), 553–576.
[7] Victor Seoane, Carlos Garcia-Rubio, Florina Almenares, and Celeste Campo. 2021. Performance evaluation of CoAP and MQTT with security support for IoT environments. *Computer Networks* 197 (2021), 108338.
[8] Stream. 2020. PoMA. https://github.com/reugalabf/PoMA.