Development of a Production-Ready Embedded Linux System with A/B Partition OTA Updates

Technical Report

Abhishek Abhishek
M.Sc. Electrical Engineering and Embedded Systems
Hochschule Ravensburg-Weingarten

February 2026

Abstract

This technical report presents the design, implementation, and validation of a production-ready embedded Linux system featuring an A/B partition scheme for over-the-air (OTA) firmware updates. The system targets the Raspberry Pi 3 Model B platform and employs Buildroot as the build framework to create a minimal, secure, and reliable operating system with a footprint of 107 MB.

The implemented solution addresses critical challenges in embedded systems deployment including update reliability, system recoverability, and IoT communication. Key innovations include a read-only root filesystem for enhanced reliability, an automated health-check mechanism with rollback capability, and bidirectional MQTT communication with ESP32 microcontrollers for distributed sensor networks.

The OTA update mechanism, inspired by Android's seamless update architecture, enables atomic partition switching with automatic rollback on boot failure, ensuring zero-downtime updates with fail-safe operation. System validation demonstrates successful update cycles between firmware versions with automatic service recovery and sub-45-second boot times.

1. Introduction

1.1 Background and Motivation

Modern embedded systems face increasing demands for remote maintainability, security patching, and feature enhancement through firmware updates. Traditional update mechanisms suffer from critical vulnerabilities: power loss during updates can render devices inoperable, network failures may result in partially-written firmware images, and software bugs in new releases can permanently disable field-deployed units.

The consumer electronics and automotive industries have addressed these challenges through A/B partition schemes, as implemented in Android's seamless updates and Tesla's over-the-air vehicle firmware delivery. These systems maintain two complete copies of the operating system, enabling atomic updates where the new version is written to an inactive partition while the system continues running from the active partition.

1.2 Project Objectives

The primary objectives of this project were:

- Design and implement a minimal embedded Linux distribution optimized for IoT gateway applications
- Develop an A/B partition OTA update system with automatic rollback capability
- Integrate MQTT-based IoT communication with ESP32 microcontroller nodes
- Implement automated health-check mechanisms for update validation
- Ensure system reliability through read-only root filesystem architecture

1.3 Scope and Deliverables

This project encompasses the complete lifecycle of embedded Linux system development from requirements analysis through validation. Deliverables include a bootable embedded Linux image, OTA update infrastructure, MQTT gateway application, bidirectional ESP32 integration, and comprehensive technical documentation.

2. System Architecture

2.1 Hardware Platform

The system targets the Raspberry Pi 3 Model B, featuring a Broadcom BCM2837 system-on-chip with a quad-core ARM Cortex-A53 processor operating at 1.2 GHz, 1 GB LPDDR2 RAM, and integrated 802.11n WiFi (BCM43438 chipset). The platform provides sufficient computational resources for gateway operations while maintaining low power consumption suitable for edge IoT deployments.

Component	Specification
Processor	BCM2837 (ARM Cortex-A53 quad-core @ 1.2 GHz)
Memory	1 GB LPDDR2 SDRAM
Storage	MicroSD card (32 GB used)
WiFi	BCM43438 (802.11n, 2.4 GHz)
Power	5V @ 2.5A (12.5W typical)

Table 2.1: Hardware Platform Specifications

2.2 Storage Partition Layout

The storage architecture employs a four-partition scheme optimized for reliability and OTA functionality. Partition 1 (32 MB FAT16) contains the GPU bootloader, kernel image, and device tree blobs. Partitions 2 and 3 (120 MB and 250 MB ext4 respectively) serve as the A/B root filesystems, while partition 4 (29.2 GB ext4) provides persistent storage for configuration, logs, and application data.

Partition	Size	Type	Purpose
p1	32 MB	FAT16	Boot (bootloader, kernel)
p2	120 MB	ext4 (ro)	Root A (active/standby)
p3	250 MB	ext4 (ro)	Root B (active/standby)
p4	29.2 GB	ext4 (rw)	Data (persistent storage)

Table 2.2: Storage Partition Scheme

2.3 Software Stack

The software architecture consists of several integrated layers. The Linux kernel 6.1.61 provides hardware abstraction and device driver support, including the brcmfmac WiFi driver for the

BCM43438 chipset. BusyBox implements core userspace utilities and the init system. Network services include wpa_supplicant for WPA2 authentication with nl80211 driver interface, dhcpcd for DHCP client functionality, and OpenSSH 9.7 for remote access. The MQTT infrastructure employs Eclipse Mosquitto 2.0.19 as the broker with a custom C application using libmosquitto for sensor data publication.

3. Implementation Details

3.1 Buildroot Configuration

Buildroot 2024.02.9 serves as the embedded Linux build framework, configured to generate a minimal distribution targeting ARM Cortex-A53 with hardware floating-point (VFPv4-D16) support. Critical configuration decisions include dynamic device management via mdev (essential for WiFi hardware detection), selection of glibc as the C library for enhanced compatibility, and explicit enablement of the brcmfmac-sdio-firmware-rpi-wifi package for wireless functionality.

The root filesystem overlay mechanism enables persistent customization across builds. The overlay directory structure at board/raspberrypi3/overlay contains /etc/fstab for automatic /data partition mounting, /etc/init.d/S99custom for service orchestration, and /etc/mosquitto/mosquitto.conf for MQTT broker network configuration. This approach ensures that custom configurations survive build regeneration and are present in all generated root filesystem images.

3.2 Boot Process and Init System

The Raspberry Pi boot sequence differs from conventional x86 systems in that the GPU initializes before the ARM processor. The VideoCore GPU executes bootcode.bin from the boot partition, loads start.elf (GPU firmware), which reads config.txt and cmdline.txt, loads the zImage kernel and device tree blob into RAM, and finally transfers control to the ARM processor. The kernel parameter root=/dev/mmcblk0pX in cmdline.txt determines the active root partition, enabling atomic partition switching by modifying a single parameter.

The BusyBox init system executes scripts in /etc/init.d/ alphabetically. The S99custom script orchestrates service startup: it validates /data partition availability, sources /data/startup.sh for WiFi initialization and SSH key management, and launches /data/mqtt_service.sh for MQTT broker and publisher daemon initialization. This late-stage execution (S99) ensures all system services are operational before custom services start.

3.3 WiFi Driver Integration

WiFi functionality required careful integration of the Broadcom brcmfmac driver with proper firmware loading. The implementation necessitated enabling mdev-based dynamic device management in Buildroot, as static /dev population prevents the WiFi interface from appearing. The brcmfmac driver loads firmware from /lib/firmware/brcm/ during kernel initialization. WPA2 authentication employs wpa_supplicant configured with nl80211 driver interface, storing credentials in /data/wpa_supplicant.conf for persistence across firmware updates.

3.4 Read-Only Root Filesystem Architecture

The root partitions mount read-only to prevent filesystem corruption from unexpected power loss or software failures. This design necessitates careful separation of static system files from dynamic runtime data. Persistent data resides in /data (partition 4), which remains writable across reboots.

SSH host keys, authorized_keys, and wpa_supplicant configuration files store in /data, with symbolic links from read-only locations (/etc/ssh/, /root/.ssh/) pointing to their /data counterparts.

Temporary remounting as read-write enables runtime configuration changes when necessary. The startup script pattern involves: mount -o remount,rw /, perform modifications, then mount -o remount,ro / to restore protection. This approach balances reliability with operational flexibility.

3.5 MQTT Gateway Implementation

The MQTT gateway consists of Eclipse Mosquitto broker and a custom C application implementing the publisher. The broker configuration explicitly binds to 0.0.0.0:1883 to accept connections from network clients, with anonymous authentication enabled for simplified IoT device integration. The publisher application, compiled using the Buildroot cross-compilation toolchain, collects system metrics (CPU temperature from /sys/class/thermal/thermal_zone0/temp, memory usage via free, and uptime) at 5-second intervals, publishing to hierarchically-structured topics: rpi/sensor/cpu_temp, rpi/sensor/memory, and rpi/sensor/uptime.

The implementation employs libmosquitto for MQTT protocol handling, with QoS 0 (fire-and-forget) selected for telemetry data where occasional message loss is acceptable in exchange for reduced protocol overhead. The application runs as a daemon, launched by the init system and designed for continuous operation.

4. OTA Update System

4.1 A/B Partition Update Mechanism

The A/B partition scheme maintains two complete root filesystems, enabling atomic updates with zero downtime. The update process operates on the inactive partition while the system continues running from the active partition. The OTA script (ota_update.sh) implements this workflow: it determines the current active partition from /proc/cmdline, downloads the new root filesystem image via wget to /data/, writes the image to the inactive partition using dd with 4MB block size for optimal SD card performance, and modifies cmdline.txt in the boot partition to switch the root= parameter to the newly-updated partition.

Partition switching occurs through a single-parameter modification in the boot configuration. The sed command replaces root=/dev/mmcblk0pX with the target partition number, achieving atomic switchover without complex bootloader modifications. Upon reboot, the GPU firmware reads the updated cmdline.txt, and the kernel mounts the new partition as root, completing the update cycle.

4.2 Health Check and Automatic Rollback

The health check system validates system integrity following updates. The health_check.sh script executes 30 seconds post-boot (after services initialize) and tests three critical subsystems: WiFi connectivity (via ip addr show wlan0), SSH server operation (process presence check), and MQTT broker availability. The script maintains a boot attempt counter in /data/boot_count, incrementing on each failed validation.

If health validation fails and the boot counter reaches the threshold (3 attempts), the rollback.sh script triggers automatically. Rollback operates identically to the update mechanism but in reverse: it identifies the current partition, determines the previous partition, updates cmdline.txt accordingly, and initiates reboot. This fail-safe mechanism ensures that catastrophic firmware bugs cannot permanently disable the system, as it will automatically revert to the last known-good partition after three consecutive boot failures.

4.3 Cross-Compilation Workflow

Application development employs cross-compilation to avoid installing development tools on the minimal target system. The Buildroot-generated toolchain resides in output/host/bin/ and provides arm-buildroot-linux-gnueabihf-gcc for C compilation. The toolchain automatically handles ARM-specific code generation, linking against the correct sysroot in output/staging/, and producing binaries compatible with the target's glibc version and processor architecture.

This workflow enables rapid development iteration: source code modification on the development host, cross-compilation (2-3 seconds), binary transfer via SCP to /data/, and immediate testing on the target. This approach reduces development cycle time from 45+ minutes (full Buildroot rebuild and reflash) to under 5 minutes.

5. ESP32 Integration

5.1 Bidirectional MQTT Communication

The ESP32 microcontroller serves as an edge node in the distributed sensor network, implementing bidirectional MQTT communication with the Raspberry Pi gateway. The ESP32 firmware, developed using the Arduino framework with PubSubClient library, maintains persistent WiFi connectivity to the same network as the gateway and establishes an MQTT client connection to the broker at the gateway's IP address.

The ESP32 publishes DHT11 sensor data (temperature and humidity) to topics esp32/sensor/temperature and esp32/sensor/humidity at 5-second intervals. Simultaneously, it subscribes to Raspberry Pi telemetry topics (rpi/sensor/cpu_temp, rpi/sensor/memory, rpi/sensor/uptime), receiving real-time system metrics for potential local processing or display. This bidirectional architecture enables distributed sensor networks where edge nodes both contribute data and receive system status information.

5.2 Data Persistence and Logging

The enhanced MQTT publisher on the Raspberry Pi implements data logging for ESP32 sensor readings. The application subscribes to esp32/# topics, capturing all messages from ESP32 nodes. Received data appends to /data/logs/esp32_data.log with timestamp, topic, and payload, creating a persistent record of sensor measurements. This logging mechanism enables historical data analysis, trend identification, and system monitoring without requiring continuous network connectivity.

6. Testing and Validation

6.1 System Performance Metrics

System validation measured key performance indicators across multiple categories. Boot time from power-on to SSH availability averaged 42 seconds, with WiFi connection established within 8 seconds of boot. The system footprint demonstrates significant optimization: root filesystem size of 107 MB represents a 90% reduction compared to standard Raspbian Lite (1+ GB), while idle memory consumption of 28 MB leaves 972 MB available for applications.

Metric	Measured Value
Boot Time (power-on to SSH)	42 seconds ± 3s
WiFi Connection Time	8 seconds
Root Filesystem Size	107 MB
Idle Memory Usage	28 MB (972 MB free)
OTA Download Speed	4.05-5.42 MB/s (WiFi)
OTA Write Duration (120 MB)	~105 seconds
CPU Temperature (idle)	47-49°C
Power Consumption (idle)	1.5-2.0 W

Table 6.1: System Performance Metrics

6.2 OTA Update Validation

The OTA update system underwent comprehensive functional testing through multiple complete update cycles. Test scenarios included: successful update from version 1.0 to version 1.2 (partition 2 to partition 3), intentional health check failure triggering automatic rollback, and manual rollback invocation from version 1.2 to version 1.0. All test cases executed successfully, demonstrating robust partition switching, correct cmdline.txt modification, proper health check execution, and automatic service restoration post-update.

Update reliability testing included power-loss simulation during firmware download (system recovered, retained previous version) and mid-write interruption (old partition remained bootable, update reattempted successfully). These scenarios validate the fundamental advantage of A/B updates: the active partition remains untouched during the update process, ensuring system recoverability even in adverse conditions.

6.3 ESP32 Communication Validation

ESP32-Raspberry Pi MQTT communication underwent end-to-end testing across various network conditions. Bidirectional message delivery achieved 100% reliability under normal network

conditions, with message latency averaging below 100ms on the local network. The ESP32 successfully received and displayed RPi system metrics, while the RPi logged ESP32 sensor readings with proper timestamp formatting. Network resilience testing demonstrated automatic reconnection following WiFi interruption, with no message queue overflow during brief disconnections.

7. Discussion

7.1 Achievements and Contributions

This project successfully demonstrates the implementation of production-grade OTA update mechanisms in resource-constrained embedded Linux systems. The A/B partition architecture, combined with automated health validation and rollback, provides update reliability comparable to commercial consumer electronics while maintaining a minimal system footprint suitable for IoT edge deployment. The integration of Buildroot overlay system enables reproducible builds with persistent customization, addressing a common challenge in embedded Linux development where custom configurations must survive build regeneration.

The read-only root filesystem architecture enhances system reliability by preventing corruption from unexpected power loss or software faults. The careful separation of static system files from dynamic runtime data, implemented through symbolic links and the dedicated data partition, maintains this protection while enabling runtime configuration flexibility. This approach proves particularly valuable for field-deployed IoT devices where physical access for recovery may be impractical or costly.

7.2 Challenges Encountered

WiFi driver integration presented significant challenges due to the Broadcom BCM43438 chipset's requirements for proper firmware loading and device management. Initial attempts with static /dev population failed to expose the wireless interface. Resolution required switching to mdev-based dynamic device management and explicit enablement of nl80211 driver support in wpa_supplicant. This experience highlights the importance of understanding the complete driver initialization chain in embedded Linux systems.

SSH configuration in the read-only root context required multiple iterations to achieve correct operation. Initial attempts encountered authentication failures due to incorrect permission handling on authorized_keys files and improper sshd_config settings conflicting with the read-only mount. The final solution employing symbolic links from read-only locations to writable /data storage, combined with proper permission management in the startup script, demonstrates the complexity of security-sensitive service configuration in restricted filesystems.

The discovery that Buildroot overlay files must be configured before the first OTA update revealed a critical dependency: new firmware images must contain both OTA scripts and overlay configurations to support automatic service recovery post-update. This circular dependency required careful sequencing of development steps and highlights the importance of comprehensive system design before OTA deployment.

7.3 Comparison with Alternative Approaches

Alternative OTA implementations include single-partition atomic updates (employed by some IoT platforms) and differential updates (transmitting only changed blocks). Single-partition updates require complex filesystem-level atomic operations and lack the inherent safety of maintaining a

known-good bootable system. The A/B approach trades storage capacity (requiring space for two complete systems) for simplified implementation and guaranteed rollback capability.

Buildroot was selected over Yocto/OpenEmbedded for this project due to its simpler configuration model and faster build times (30-90 minutes versus 2-4 hours). While Yocto provides superior flexibility for multi-variant products and complex software stacks, Buildroot's menuconfig-based approach proved sufficient for this single-target IoT gateway application. The choice demonstrates appropriate tool selection based on project requirements rather than pursuing maximal flexibility at the cost of development velocity.

8. Conclusions and Future Work

8.1 Summary of Achievements

This project successfully developed a production-ready embedded Linux system incorporating modern OTA update mechanisms, demonstrating that enterprise-grade update reliability can be achieved on resource-constrained platforms. The implemented system achieves all stated objectives: minimal footprint (107 MB) enabling deployment on cost-optimized hardware, reliable A/B partition updates with automatic rollback, bidirectional IoT communication through MQTT, and comprehensive health validation ensuring system recoverability.

The system architecture demonstrates professional embedded Linux development practices including cross-compilation workflows, read-only filesystem design, persistent configuration management, and automated service orchestration. These techniques are directly applicable to commercial IoT product development and provide a foundation for scalable device management in distributed sensor networks.

8.2 Future Enhancements

Several extensions would enhance the system's capabilities for production deployment:

- TLS/SSL encryption for MQTT communication to secure sensor data transmission over untrusted networks
- Differential update system to transmit only changed blocks, reducing bandwidth consumption for limited-connectivity deployments
- Watchdog timer integration for automatic recovery from application hangs or kernel panics
- Multiple WiFi network failover with priority-based selection for enhanced connectivity reliability
- Web-based dashboard for real-time system monitoring and remote configuration
- Integration with container technology (Docker) for application-level updates independent of system firmware
- Signed firmware images with cryptographic verification to prevent unauthorized or corrupted updates
- Fleet management capabilities for coordinated updates across multiple deployed units

8.3 Applicability to Industry

The techniques demonstrated in this project have direct applicability to commercial IoT product development across multiple domains. Industrial automation systems benefit from the reliable remote update capability, enabling software maintenance without service interruption. Smart building management systems can employ the MQTT gateway architecture for distributed sensor networks. Edge computing applications leverage the minimal footprint and low power consumption for deployment in resource-constrained environments.

The A/B update mechanism's tolerance to power-loss and network interruption makes it particularly suitable for remote or unattended deployments where physical access for recovery is

impractical. The automatic rollback capability addresses a critical concern in field-deployed systems: ensuring that software updates cannot permanently disable devices, thereby reducing support costs and improving customer satisfaction.

References

1. Buildroot Project. (2024). The Buildroot user manual. Retrieved from https://buildroot.org/downloads/manual/manual.html
2. Eclipse Foundation. (2024). Eclipse Mosquitto: An open source MQTT broker. Retrieved from https://mosquitto.org/documentation/
3. Raspberry Pi Foundation. (2024). Raspberry Pi 3 Model B: Technical specifications and documentation. Retrieved from https://www.raspberrypi.org/documentation/
4. Google Inc. (2018). Implementing A/B system updates in Android. Android Open Source Project Documentation.
5. Yaghmour, K. (2008). Building embedded Linux systems (2nd ed.). O'Reilly Media.
6. Simmonds, C. (2021). Mastering embedded Linux programming (3rd ed.). Packt Publishing.
7. OASIS. (2019). MQTT Version 5.0: OASIS Standard. Retrieved from http://docs.oasis-open.org/mqtt/mqtt/v5.0/
8. Broadcom Corporation. (2016). BCM43438: Single-chip IEEE 802.11 b/g/n MAC/Baseband/Radio with Bluetooth 4.1. Product Brief.

Appendix A: System Specifications Summary

A.1 Software Component Versions

Component	Version
Buildroot	2024.02.9
Linux Kernel	6.1.61
glibc	2.38
BusyBox	1.36.1
OpenSSH	9.7
wpa_supplicant	2.10
dhcpcd	10.0.8
Mosquitto	2.0.19

Table A.1: Software Component Versions