The American Journal of Engineering and Technology
ISSN 2689-0984 Volume 08 - 2026

Reliability and Recovery Design for OTA Software Updates in Automotive Embedded Systems

Srikanth Puram
General Motors Warren Michigan USA

Received: 22 Feb 2026 | Received Revised Version: 14 Mar 2026 | Accepted: 18 Apr 2026 | Published: 26 May 2026

Volume 08 Issue 05 2026 | Crossref DOI: 10.37547/tajet/Volume08Issue05-13

Abstract

Over-the-air (OTA) software updates are essential in modern software-defined vehicles, enabling continuous feature delivery, security patching, and system optimization [1], [3]. However, ensuring the reliability of OTA workflows in automotive embedded environments is challenging due to intermittent connectivity, power-state transitions, and stringent cybersecurity and safety requirements [5], [6], [10], [11]. This paper presents a reliability-focused design for OTA software update systems on Android-based automotive embedded platforms, emphasizing failure handling, recovery mechanisms, system integrity validation, and staged orchestration. The proposed architecture incorporates checkpoint-based progress tracking, modular update delivery, cryptographic verification, and deterministic recovery workflows. The design specifically addresses interruptions such as network disruptions, process restarts, and suspend-to-RAM (STR) events [3], [6], [7], [9], [12]. This framework provides a practical approach for constructing resilient OTA workflows that utilize both platform-level update support and application-layer recovery logic.

Keywords: Hybrid Cloud Security, Zero Trust Architecture, Cloud Security Evaluation, Data Protection, AI-Driven Threat Detection.

Cite This Article: Puram, S. (2026). Reliability and Recovery Design for OTA Software Updates in Automotive Embedded Systems. The American Journal of Engineering and Technology, 8(05), 136–139. https://doi.org/10.37547/tajet/Volume08Issue05-13

1. Introduction

The automotive industry is rapidly transitioning toward software-defined vehicles, with OTA updates serving as a cornerstone for continuous feature delivery, security maintenance, and system optimization [1], [3], [11], [16]. Unlike conventional mobile or enterprise deployments, automotive updates must function within constrained,

safety-critical environments, where failures can result in inconsistent software states or delayed restoration of functionality. Android and Android Automotive provide foundational components for OTA delivery, package installation, persistent work scheduling, and power-state management. Nevertheless,

production-grade update workflows demand additional orchestration beyond the platform layer [5], [7], [8], [9]. This paper proposes a

reliability-oriented OTA design for Android-based automotive embedded systems, emphasizing checkpoint-based recovery, modular update delivery, integrity validation, and deterministic resume logic for handling

interruptions such as network loss, reboot, and STR.

2. Problem Statement

OTA workflows in vehicles encounter four primary reliability challenges. First, connectivity is often intermittent and variable, especially when vehicles are in motion or parked in areas with weak signals. Second, the platform may transition through ignition cycles, garage-mode behavior, or suspend-to-RAM states, which can interrupt long-running processes [5], [6]. Third, embedded systems are constrained by limited storage and memory, as well as limited background execution capabilities [7], [8]. Fourth, software update systems must comply with increasing cybersecurity and regulatory requirements, including engineering risk management and governance

mandates under ISO/SAE 21434 and UNECE R156 [10], [11]. Previous research and standards on automotive FOTA and OTA security highlight that failed updates, rollback risks, and incomplete state transitions can undermine serviceability and erode trust in connected vehicles [13], [14], [15], [16].

3. Proposed Architecture

The proposed OTA system employs a staged update pipeline that includes metadata retrieval, package download, integrity verification, installation, and activation. This structure facilitates failure isolation, post-interruption state validation, and workflow resumption without restarting the entire update process. Each stage is checkpointed in persistent storage, enabling the system to resume from the last successful state rather than repeating the full workflow after an interruption. This approach aligns with Android's persistent background work model and installation APIs, while providing the application-specific state control necessary for production automotive deployments [7], [8], [9].

The architecture supports modular or split-package delivery, enabling updates to specific assets, configurations, or functional components as required. At the platform level, Android's OTA and A/B update documentation demonstrates the importance of maintaining a bootable or recoverable state during updates, while dynamic partition support enhances operational flexibility [2], [3], [4]. At the application level, this principle is extended through smaller update units, explicit validation points, and deterministic resume logic.

4. Recovery Mechanisms

Recovery is initiated whenever the OTA workflow restarts following network loss, process termination, reboot, or STR. Upon restart, the system accesses the

persisted checkpoint, revalidates the state of downloaded artifacts, and resumes only the incomplete stages. This process prevents redundant data transfers and minimizes the risk of partial or outdated installation states. In Android Automotive OS (AAOS), STR is especially significant because the system retains memory while suspending active execution, requiring reconciliation between the expected workflow state and the actual platform state upon resumption [5], [6].

To strengthen recovery, the design integrates three primary controls: artifact validation, staged rollback or cleanup, and reinitialization of lifecycle-dependent components. Prior to resuming installation, the system verifies package hashes and expected metadata to confirm that downloaded artifacts correspond to the recorded checkpoint. If temporary files do not align with the checkpoint, the recovery process removes or quarantines them instead of reusing them. The design also reinstates pending work, observers, and installer callbacks in a validated sequence using persistent background-execution mechanisms [7], [8], [9].

5. Security and Integrity Considerations

Reliable OTA systems require robust integrity and trust controls. Official Android OTA documentation and automotive security standards highlight the necessity of authenticity validation, rollback protection, and secure update verification [11], [12], [13], [14]. Signature verification, hash checks, and checkpoint validation are implemented to prevent interrupted or tampered update workflows from proceeding in an unsafe state. The system links the update state to trusted metadata and halts resumed operations if the artifact state does not match the recorded checkpoint. These checks help prevent unsafe continuation of interrupted or tampered update workflows.

6. Implementation Considerations for Android-Based Automotive Platforms

In Android-based automotive systems,

platform-native services are most effective when combined with application-level coordination.

WorkManager enables scheduling of background tasks that persist across application restarts and device reboots [7], [8], while PackageInstaller offers structured installation sessions and status callbacks for controlled application deployment [9]. In automotive contexts, developers must integrate these services with update-specific constraints, including power-state awareness, network availability, storage prechecks, and post-resume validation.

7. Evaluation

This paper qualitatively evaluates the design against interruption scenarios commonly encountered in field deployments, including interrupted downloads, process restarts, deferred installation windows, reboot events, and STR transitions. For each scenario, the evaluation assessed whether the workflow could resume from a validated checkpoint, whether temporary artifacts could be safely reused, and whether the installation state remained consistent after recovery.

In these scenarios, the checkpoint model minimized unnecessary rework, preserved validated artifacts when safe to reuse, and enhanced the predictability of resumed installation sequences. The design further decreased the likelihood of stale or partial states persisting after an interruption by mandating artifact revalidation and controlled cleanup before resuming execution [11], [15], [16].

8. Discussion

The primary architectural insight is that platform-level OTA support is necessary but not

sufficient for reliable application and feature delivery in connected vehicles. Production systems require state-aware orchestration that integrates update transport, artifact validation, installation semantics, and power management. State-aware orchestration is particularly important for software-defined vehicles, where the frequency and scope of updates are increasing and the consequences of failed field updates remain significant [11], [15], [16].

9. Conclusion

This paper introduced a reliability-oriented OTA design for automotive embedded systems, utilizing staged orchestration, checkpoint-based resume logic, modular delivery, and integrity validation. By integrating Android-native update primitives with application-layer recovery controls, the design enhances resilience to network disruptions, restart events, and suspend-related interruptions. Future research should validate this approach through controlled fault-injection scenarios, update-latency measurements, and broader evaluation across ECU and software-package categories. Given the qualitative nature of this evaluation, further work is required to quantify recovery latency, artifact reuse efficiency, and validation coverage under controlled fault conditions.

References

1. Android Open Source Project, “OTA updates,” source.android.com, accessed Apr. 7, 2026.
2. Android Open Source Project, “Implement OTA updates,” source.android.com, accessed Apr. 7, 2026.
3. Android Open Source Project, “A/B (seamless) system updates,” source.android.com, accessed Apr. 7, 2026.
4. Android Open Source Project, “OTA for A/B devices with dynamic partitions,” source.android.com, accessed Apr. 7, 2026.
5. Android Open Source Project, “Power management,” source.android.com/docs/automotive/power/power, accessed Apr. 7, 2026.
6. Android Open Source Project, “Power policy,” source.android.com/docs/automotive/power/power_p olicy, accessed Apr. 7, 2026.
7. Android Developers, “Getting started with WorkManager,” developer.android.com, accessed Apr. 7, 2026.
8. Android Developers, “Task scheduling | Background work,” developer.android.com, accessed Apr. 7, 2026.
9. Android Developers, “PackageInstaller | API reference,” developer.android.com, accessed Apr. 7, 2026.
10. ISO, “ISO/SAE 21434:2021 Road vehicles — Cybersecurity engineering,” iso.org, 2021.
11. UNECE, “UN Regulation No. 156 — Software update and software update management system,” unece.org, 2021.
12. IEEE-ISTO, “IEEE-ISTO 6100.1.0.0 Uptane Standard for Design and Implementation,” uptane.org, accessed Apr. 7, 2026.

13. T. K. Kuppusamy, L. A. DeLong, and J. Cappos, "Uptane: Securing Software Updates for Automobiles," ESCAR USA, 2016.
14. T. K. Kuppusamy, L. A. DeLong, and J. Cappos, "Uptane: Security and Customizability of Software Updates for Vehicles," IEEE Vehicular Technology Magazine, 2018, doi: 10.1109/MVT.2017.2778751.
15. M. Shavit, A. Gryc, and R. Miucic, "Firmware Update Over The Air (FOTA) for Automotive Industry," SAE Technical Paper 2007-01-3523, 2007, doi: 10.4271/2007-01-3523.
16. "Analysis of Software Update in Connected Vehicles," SAE Technical Paper 2014-01-0256, 2014, doi: 10.4271/2014-01-0256.