International Journal of AI, Big Data, Computational and Management Studies
Noble Scholar Research Group | Volume 7, Issue 2, 43-53, 2026
ISSN: 3050-9416 | https://doi.org/10.63282/3050-9416.IJAIBDCMS-V7I2P108

Original Article

A Hybrid Push–Pull Notification Model for Maritime Android Devices with Intermittent Satellite Links

Varun Reddy Guda
Little Elm, TX, USA.

Received On: 13/02/2026 ***Revised On: 24/03/2026*** ***Accepted On: 02/04/2026*** ***Published On: 08/04/2026***

Abstract: *Android applications deployed in maritime environments such as cruise ships, offshore vessels, and research platforms operate under fundamentally different network constraints than terrestrial mobile systems. Connectivity is typically provided via satellite links characterized by high latency, limited bandwidth, intermittent availability, and asymmetric up-link–downlink behavior. Traditional mobile notification systems, which rely heavily on continuous push-based delivery models, degrade severely under these conditions, resulting in delayed, dropped, or inconsistent notification delivery. This paper introduces a Hybrid Push–Pull Notification Model (HPPNM) designed specifically for Android devices operating over intermittent satellite links. The proposed model combines opportunistic push delivery during periods of link availability with adaptive pull-based synchronization during offline or de-graded connectivity windows. By decoupling notification intent from immediate delivery and dynamically switching between push and pull modes based on network conditions, HPPNM ensures reliable, timely, and bandwidth-efficient notification de-livery in maritime environments. The framework operates entirely at the application layer and integrates with Android notification APIs, local persistence, lifecycle-aware execution, and constrained network schedulers. Through architectural design and experimental evaluation un-der simulated satellite conditions, this paper demonstrates that HPPNM significantly improves notification reliability, reduces bandwidth waste, and preserves user trust compared to push-only approaches.*

Keywords**:** *Android Notifications, Satellite Connectivity, Maritime Systems, Hybrid Push–Pull, Offline-First Communication, Mobile Resilience.*

1. Introduction

1.1. The Martime Android Context

Android devices are no longer confined to consumer terrestrial environments. They are now widely deployed in maritime domains for passenger services, crew operations, logistics management, and safety coordination. Cruise ships use An-droid applications for guest itineraries and announcements; offshore energy platforms rely on mobile systems for operational alerts; maritime research vessels deploy Android-based tablets for data coordination and crew communication.

Unlike terrestrial mobile systems, maritime Android deployments operate over satellite communication links that impose fundamentally different constraints:

- End-to-end latency often exceeds 600–1500 milliseconds
- Bandwidth is limited and shared across hundreds or thousands of devices
- Links experience periodic outages due to weather, repositioning, or provider transitions
- Uplink capacity is frequently more constrained than downlink capacity
- Data transfer is cost-sensitive, discouraging continuous polling

These constraints fundamentally challenge assumptions embedded in traditional mobile notification architectures.

1.2. Fragility of Push-Centric Model:

Modern Android notification systems rely heavily on push-based delivery models. These systems assume:

- Persistent connectivity between device and push service
- Rapid acknowledgment of message delivery
- Low-latency transport channels
- Reliable background execution windows

In maritime environments, these assumptions break down. Push notifications may queue at the satellite gateway, arrive in bursts after prolonged disconnection, or silently fail when device-side services are suspended due to background restrictions.

Critically, push-only models lack a deterministic recovery mechanism. When delivery fails, the device often has no structured way to reconcile missed notifications unless the backend replays them explicitly. This can result in:

- Missed safety announcements
- Duplicate alerts upon reconnection
- Out-of-order delivery
- Delayed notifications that have lost contextual

relevance in passenger-facing or safety-sensitive maritime systems, such behavior is unacceptable.

1.3. Reframing Notification Delivery as a Hybrid Problem

This paper argues that notification delivery over satellite links should not be treated as a purely push-driven transport problem. Instead, it must be framed as a hybrid communication and synchronization problem, in which:

- Push mechanisms provide low-latency opportunistic de-livery when links permit
- Pull mechanisms provide deterministic reconciliation when push fails
- Local persistence ensures intent durability
- Relevance and priority semantics prevent stale or burst-delivered alerts

The key architectural insight is the decoupling of notification intent from transport delivery success. Notification intent must be considered a stateful artifact that can survive transport-layer volatility.

1.4. Research Question

This paper addresses the following core research question: How can Android applications deployed in maritime environments reliably deliver notifications over intermittent satellite links while preserving timeliness, relevance, bandwidth efficiency, and user trust? This question requires balancing several competing constraints:

- Reliability versus bandwidth cost
- Timeliness versus relevance decay
- Push immediacy versus pull determinism
- Platform background limits versus execution guarantees the proposed Hybrid Push–Pull Notification Model (HPPNM) resolves these tensions by introducing adaptive mode switching and intent-level reconciliation semantics.

2. Background and Motivation

2.1. Satellite Link Characteristics and Implications

Satellite communication exhibits properties rarely encountered in terrestrial mobile systems:

- High Round-Trip Time (RTT): High RTT increases handshake overhead for push channels and degrades keep-alive mechanisms.
- Variable Link Availability: Temporary outages cause push delivery attempts to fail unpredictably.
- Bandwidth Contention: Shared vessel-wide bandwidth amplifies congestion during peak usage periods.
- Asymmetry: Uplink constraints limit acknowledgment traffic and polling frequency.

These characteristics create a hostile environment for continuous push connectivity and demand an adaptive delivery model.

2.2. Notification Semantics in Operational Maritime Contexts

Maritime applications use notifications for:

- Emergency alerts
- Itinerary changes
- Service interruptions
- Crew task coordination
- Regulatory announcements

Unlike promotional notifications in terrestrial apps, many maritime notifications have operational significance. Their failure can lead to confusion, safety risk, or financial impact.

Moreover, notification relevance is context-sensitive. An alert about a deck event is only meaningful within a narrow time window; delivering it hours later undermines trust.

2.3. Limitations of Pure Pull Systems

One might consider abandoning push entirely and adopting a pull-only synchronization model. However, pure pull systems introduce:

- Increased bandwidth usage from polling
- Delayed delivery during low polling frequency
- Higher battery consumption
- Reduced real-time responsiveness

Therefore, neither pure push nor pure pull models are optimal in maritime contexts. A hybrid model is required.

3. Problem Definition

3.1. Formalizing the Notification Delivery Challenge

We define a notification delivery system as reliable if it satisfies the following properties under intermittent connectivity:

- Intent Preservation : No valid notification is permanently lost due to transient connectivity failure.
- Exactly-Once Delivery: Notifications are not duplicated upon reconnection.
- Relevance Awareness: Notifications are delivered only within meaningful temporal windows.
- Bandwidth Efficiency: Delivery mechanisms avoid excessive retransmission or polling.
- User-Experience Stability: Notifications do not appear in disruptive bursts after reconnection.

Existing push-centric systems fail to satisfy multiple properties simultaneously in satellite-constrained environments.

3.2. Constrains Imposed by Android Platform Policies

Android imposes strict background execution limits, notification channel requirements, and battery optimization policies. Any hybrid notification model must:

- Respect background execution windows
- Avoid violating Doze mode restrictions
- Coordinate with lifecycle-aware components
- Integrate with system notification channels

These constraints rule out aggressive polling or persistent background sockets as universal solutions.

3.3. Problem Statement

The problem addressed in this paper is: Design a notification delivery architecture for maritime Android devices that combines push and pull semantics to achieve reliable, relevant, and bandwidth-efficient delivery under intermittent satellite connectivity, without requiring OS-level modification.

4. Scope and Contributions

4.1. Scope

This work focuses on:

- Android devices operating primarily over satellite links
- Application-layer notification handling
- Operational and informational notifications
- Adaptive switching between push and pull delivery modes

The model assumes no control over satellite infrastructure or Android OS internals.

4.2. Non-Goals

The following are explicitly excluded:

- Modifying Android's push service infrastructure
- Redesigning satellite transport protocols
- Implementing custom firmware or VPN tunneling layers
- Addressing streaming or continuous data feeds

The scope is limited to notification delivery semantics at the application layer.

4.3. Contributions

This paper makes five major contributions:

- Hybrid Notification Delivery Model: A principled framework combining push and pull delivery for satellite-constrained environments.
- Intent–Transport Decoupling Architecture: A design that treats notification intent as a durable state artifact independent of transport success.
- Adaptive Mode Switching Algorithm: A policy-driven mechanism for dynamically selecting push or pull modes based on link state.
- Android-Compatible Implementation Strategy: A deployable, application-layer solution integrating with Android notification APIs and background schedulers.
- Empirical Evaluation Under Simulated Satellite Conditions: Controlled experiments measuring reliability, latency, burst suppression, and bandwidth usage.

5. Related Work

Designing reliable notification delivery for maritime An-droid devices requires drawing insights from several re-search areas: mobile push notification infrastructure, delay-tolerant networking, intermittent connectivity systems, and Android background scheduling frameworks. While significant work exists in each of these domains, none directly addresses the combined challenges of satellite-constrained networks and Android notification semantics.

5.1. Mobile Push Notification Systems

Push notification infrastructure forms the backbone of modern mobile engagement systems. Android applications typically rely on centralized push services such as Firebase Cloud Messaging (FCM) or similar broker-based systems. These services maintain persistent device connections and deliver notifications asynchronously through backend message queues.

The success of push-based systems stems from several assumptions:

- Persistent connectivity between devices and push servers
- Low-latency communication paths
- Reliable delivery acknowledgments
- Predictable background execution windows

These assumptions hold in terrestrial environments where cellular and Wi-Fi connectivity are ubiquitous. However, they degrade severely in satellite-based networks. Persistent connections are frequently dropped, keep-alive mechanisms become inefficient due to high round-trip times, and queued messages may accumulate at gateway nodes until connectivity resumes.

Consequently, push-based systems exhibit several pathological behaviors in satellite environments:

- Burst delivery after reconnection
- Out-of-order message arrival
- Silent notification loss
- Excessive bandwidth usage due to reconnect attempts

Existing push frameworks provide limited mechanisms for client-side recovery when these conditions occur.

5.2. Delay-Tolerant Networking (DTN)

The concept of Delay-Tolerant Networking (DTN) has been extensively studied in environments characterized by intermittent connectivity, such as space communications, rural networks, and vehicular systems. DTN architectures rely on store-and-forward mechanisms, where messages are buffered locally until communication opportunities arise.

DTN approaches emphasize:

- Message persistence across network disruptions
- Opportunistic transmission during link availability
- Transport-layer resilience to high latency

While these principles are highly relevant to maritime connectivity, DTN research primarily addresses transport-layer re-liability rather than application-layer semantics. Notification delivery introduces additional complexities, including temporal relevance, prioritization, user experience considerations, and platform-specific constraints such as

Android background execution limits.

HPPNM adopts the persistence and opportunistic delivery principles of DTN while extending them to the application layer notification domain.

5.3. Hybrid Communication Models

Hybrid push–pull communication models have been explored in distributed systems where network reliability varies. In such systems, push channels provide low-latency updates, while pull-based synchronization ensures eventual consistency.

Examples include:

- Web application synchronization frameworks
- Distributed cache invalidation systems
- Collaborative editing platforms

These systems demonstrate that hybrid models can balance responsiveness and reliability. However, most prior work assumes relatively stable connectivity and does not consider extreme latency environments such as satellite links.

Moreover, hybrid models in web environments typically operate in resource-rich environments, whereas Android applications must operate within strict battery, bandwidth, and lifecycle constraints.

5.3.1. Android Background Execution and Scheduling

An-droid provides several frameworks for managing background work, including:

- Alarm Manager
- Job Scheduler
- Work Manager

Among these, Work Manager offers the most robust solution for deferred task execution under varying constraints. It supports conditional execution based on network availability, device charging state, and other environmental conditions.

Despite these capabilities, Android scheduling frameworks lack native support for notification reconciliation or hybrid push–pull semantics. They treat tasks as isolated work units rather than components of a coordinated communication model.

As a result, developers must manually implement recovery logic, often resulting in inconsistent behavior across applications.

5.4. Maritime Communication Systems

Maritime communication research has historically focused on improving satellite throughput, optimizing routing pro-tocols, and reducing link-layer latency. Solutions such as bandwidth compression, packet aggregation, and protocol acceleration have improved network performance but operate below the application layer.

Very little research addresses how mobile applications themselves should adapt their communication strategies to satellite constraints. In particular, notification delivery semantics remain largely unexplored.

This gap motivates the development of HPPNM as an application-layer architectural solution.

5.4.1. Gap Analysis: Across these domains, several key gaps remain:

- Push notification systems assume persistent connectivity
- DTN solutions lack notification semantics
- Hybrid models ignore Android platform constraints
- Maritime research focuses on transport optimization rather than application adaptation

The Hybrid Push–Pull Notification Model addresses these gaps by combining push responsiveness with pull-based recon-ciliation, tailored specifically for Android devices operating over intermittent satellite links.

6. Design Requirements

Based on the limitations identified in prior work and observations from real-world maritime deployments, we derive a set of design requirements that any viable notification delivery model must satisfy.

6.1. Intent Preservation Requirement

R1: Notification intent must be preserved independently of transport success: When a notification is generated by backend systems, the device must eventually learn of its existence even if push delivery fails. The system must therefore treat notification intent as a durable state rather than a transient network message. This requirement implies the need for reconciliation mechanisms that can recover missed notifications after connectivity disruptions.

6.2. Opportunistic Low-Latency Delivery

R2: Notifications should be delivered with minimal latency when connectivity permits: Although pull mechanisms ensure reliability, relying solely on polling would significantly increase latency and bandwidth consumption. Therefore, push delivery should remain the primary mechanism whenever link conditions allow. The hybrid model must therefore prioritize push delivery while maintaining fallback mechanisms.

6.3. Bandwidth Efficiency

R3: Notification synchronization must minimize unnecessary bandwidth usage: Satellite bandwidth is both limited and expensive. Notification systems must avoid frequent polling, excessive acknowledgments, or redundant message transfers. This requirement encourages adaptive synchronization intervals and batch retrieval strategies.

6.4. Relevance Preservation

R4: Notifications must respect temporal relevance constraints: Many notifications have limited usefulness after a specific time window. Delivering outdated notifications can confuse users or create operational issues. The system

must therefore include mechanisms for relevance expiration and stale-notification suppression.

6.5. Burst Mitigation

R5: Notification delivery must avoid burst behavior following reconnection: When connectivity is restored after a prolonged outage, queued notifications may accumulate. Delivering them all at once can overwhelm users and degrade application performance. A hybrid model must introduce controlled reconciliation strategies that smooth delivery over time.

6.6. Android Platform Compliance

R6: The notification model must comply with Android lifecycle and background execution policies: Android restricts long-running background services and persistent network connections. Therefore, the notification framework must integrate with platform-approved mechanisms such as Work Manager and lifecycle-aware components.This ensures compatibility across Android versions and device manufacturers.

6.7. Observability and Control

R7: Notification delivery behavior must be observable and controllable by developers and operators: Operational visibility is essential for diagnosing failures in satellite environments. The system must provide telemetry about push success rates; pull reconciliation events, and delivery timing. Such observability enables continuous optimization and operational trust.

7. Failure Mode Taxonomy

To design a robust hybrid notification system, it is essential to understand the failure modes that occur in satellite-constrained Android environments. This section categorizes the most common failures observed in push-only notification architectures.

- Silent Notification Loss: Push notifications may fail to reach the device due to satellite link interruptions or push service timeouts. In many cases, the device receives no indication that a notification was missed. Without reconciliation mechanisms, such failures result in permanent notification loss.
- Burst Delivery After Reconnection: When connectivity resumes after a prolonged outage, push services may deliver accumulated notifications in rapid succession. This burst behavior can overwhelm users and distort the perceived timing of events. Burst delivery is particularly problematic for maritime applications where notifications may represent operational events.
- Notification Reordering: Satellite link disruptions can cause notifications to arrive out of sequence. For example, a later update may arrive before an earlier one, leading to inconsistent application state. Ensuring proper ordering is essential for maintaining logical coherence.
- Duplicate Notification Delivery: Push retry mechanisms may cause the same notification to be delivered multiple times when acknowledgments are delayed or lost. Duplicate notifications degrade user trust and complicate analytics tracking.
- Stale Notification Delivery: Notifications delayed by satellite outages may arrive long after they are relevant. Delivering such notifications can con-fuse users or create operational misunderstandings.
- Bandwidth Amplification Failures: Repeated push reconnects attempts or frequent polling can consume excessive bandwidth, exacerbating satellite congestion and increasing operational costs.

8. Hybrid Push–Pull Notification Architecture

8.1. Architectural Overview

The Hybrid Push–Pull Notification Model (HPPNM) is designed as a resilient communication layer for Android de-vices operating under intermittent satellite connectivity. Rather than relying exclusively on push notification infrastructure or pull-based synchronization, HPPNM combines both mechanisms and dynamically adapts its behavior based on observed link conditions.

The core architectural principle behind HPPNM is the decoupling of notification intent from delivery transport. Notifications are treated as durable application-layer entities whose delivery may occur via multiple communication path ways depending on network conditions.

The architecture consists of five primary components:

- Notification Intent Registry
- Push Delivery Channel
- Pull Synchronization Engine
- Link State Monitor
- Delivery Coordinator

Each component operates independently but cooperates through a shared persistence layer and event bus.

Under normal network conditions, push delivery is the preferred mechanism due to its low latency and minimal device-side overhead. However, when push delivery becomes unreliable due to satellite link instability, the pull synchronization engine activates to reconcile missed notifications.

This hybrid architecture ensures that notification delivery remains both **responsive and reliable**, even under extreme connectivity constraints.

8.2. Notification Intent Registry

The Notification Intent Registry serves as the authoritative local record of notification state. Every notification is represented as an intent object that captures:

- Unique notification identifier
- Notification priority level
- Creation timestamp
- Expiration window
- Delivery status
- Reconciliation metadata

Unlike traditional push-only systems where notifications are transient transport messages, HPPNM treats notification in-tents as durable state artifacts. This persistence enables the system to detect missing notifications and reconcile them later through pull synchronization.

The registry is typically implemented using a lightweight local database such as Room or SQLite to ensure durability across application restarts and device reboots.

8.3. Push Delivery Channel

The push delivery channel represents the conventional push notification pipeline used by Android applications. Push delivery is responsible for transmitting notifications from backend services to the device whenever satellite connectivity is avail-able.

Within the hybrid architecture, push delivery serves two roles:

- Opportunistic real-time delivery
- Intent announcement signaling

Even when a push message cannot deliver the full notification payload due to network constraints, it can still function as an announcement signal indicating that new notifications are available. The device can then initiate pull reconciliation if necessary.

Push delivery therefore remains an essential component of the architecture, but it is no longer solely responsible for guaranteeing delivery reliability.

8.4. Pull Synchronization Engine

The Pull Synchronization Engine is responsible for re-covering notifications that were not successfully delivered through the push channel. This engine periodically reconciles the device's Notification Intent Registry with the backend notification store.

Synchronization occurs through lightweight queries that request notifications created after the device's last known synchronization timestamp.

Key design features of the pull engine include:

- Adaptive polling intervals based on network state
- Batch retrieval of missed notifications
- Priority-based retrieval ordering
- Bandwidth-aware synchronization windows

Pull synchronization operates conservatively to avoid excessive bandwidth consumption, particularly when satellite connectivity is degraded.

8.5. Link State Monitor

The Link State Monitor continuously evaluates the quality and availability of the satellite connection. Rather than relying solely on binary connectivity indicators, the monitor considers several metrics:

- Round-trip latency
- Packet loss rates
- Push channel stability
- Recent synchronization success rates

These metrics are used to classify link conditions into operational states such as:

- Stable connectivity
- Degraded connectivity
- Intermittent connectivity
- Offline

The Delivery Coordinator uses these classifications to deter-mine the appropriate notification delivery strategy.

8.6. Delivery Coordinator

The Delivery Coordinator is the central control component responsible for orchestrating notification delivery behavior. It receives signals from the Link State Monitor and determines whether push delivery, pull synchronization, or a hybrid strategy should be used.

The coordinator enforces several key policies:

- Prefer push delivery when link stability is high
- Initiate pull reconciliation when push reliability drops
- Suppress redundant pull operations when push success rates recover
- Smooth notification delivery to prevent bursts

This coordination logic ensures that the hybrid system remains efficient, adaptive, and predictable.

9. Notification Delivery Workflow

9.1. Push-First Delivery Path

Under stable connectivity conditions, notifications follow the traditional push delivery path:

- Backend generates notification intent
- Push service delivers notification payload
- Device receives and records notification intent
- Notification is displayed to the user

In this mode, the pull synchronization engine remains idle, conserving bandwidth and device resources.

9.2. Hybrid Recovery Path

When push delivery fails or becomes unreliable, the hybrid recovery path activates:

- Device detects degraded push reliability
- Pull synchronization engine initiates reconciliation
- Missed notifications are retrieved from backend
- Notification Intent Registry is updated
- Notifications are delivered sequentially to the user

This recovery mechanism ensures that missed notifications are eventually delivered without relying solely on push retries.

9.3. Burst Mitigation Mechanism

One of the key challenges in satellite environments is burst notification delivery after reconnection. HPPNM mitigates this by introducing controlled release policies.

Rather than delivering all recovered notifications immediately, the system spaces delivery over a short interval based on priority and relevance. This prevents overwhelming users with large volumes of delayed notifications.

10. Notification Prioritization and Relevance Management

10.1. Notification Priority Classes

Notifications are categorized into priority classes to guide delivery decisions:

- Critical notifications (safety alerts, operational instructions)
- High-priority notifications (schedule changes, urgent updates)
- Standard notifications (general informational updates)
- Low-priority notifications (non-urgent announcements) Priority classes influence both delivery order and expiration behavior.

10.2. Relevance Windows

Each notification includes a relevance window specifying the maximum time during which the notification remains meaningful. If a notification remains undelivered beyond this window, it is automatically discarded rather than delivered late.

This mechanism prevents stale notifications from appearing long after their context has passed.

10.3. Deduplication and Ordering

The Notification Intent Registry maintains strict deduplication rules based on notification identifiers. If a notification is received through both push and pull channels, only one instance is delivered.

Additionally, notifications are ordered by creation time to ensure consistent user experience.

11. Hybrid Mode Switching Strategy

11.1. Push Reliability Monitoring

Push reliability is continuously evaluated based on delivery acknowledgments and observed delays. If push success rates fall below a configurable threshold, the system temporarily shifts toward pull-based synchronization.

11.2. Adaptive Pull Scheduling

Pull synchronization intervals are dynamically adjusted based on network conditions. During periods of severe degradation, polling frequency is reduced to conserve bandwidth. When connectivity stabilizes, synchronization frequency increases to ensure rapid reconciliation.

11.3. Mode Stabilization

Frequent switching between push and pull modes can lead to unstable behavior. The Delivery Coordinator therefore applies hysteresis thresholds to ensure that mode transitions occur only when link conditions change significantly. This stabilization mechanism prevents oscillation between delivery modes.

12. Android Implementation of the Hybrid Push–Pull Model

12.1. Application-Layer Deployment Architecture

The Hybrid Push–Pull Notification Model (HPPNM) is implemented entirely at the Android application layer, ensuring compatibility with existing device deployments and avoiding any dependence on modifications to the Android operating system or satellite network infrastructure.

This deployment strategy is critical in maritime environments, where devices may operate under strict administrative policies and updates must remain compatible with standard Android distribution channels.

The notification subsystem is integrated into the application's communication stack as a notification delivery middleware layer positioned between:

- the backend notification service
- the Android system notification manager
- the application's internal event bus

This middleware layer intercepts all incoming push notifications, records them within the Notification Intent Registry, and coordinates reconciliation events with the pull synchronization engine.

Under normal conditions, the system behaves identically to conventional push-based notification systems, ensuring back-ward compatibility with existing notification infrastructure.

12.2. Notification Intent Persistence

The Notification Intent Registry is implemented using a persistent local storage layer built on Android's Room persistence library. Each notification intent is stored as a durable record with the following fields:

- notification identifier
- notification category and priority class
- timestamp of creation
- expiration timestamp
- delivery state (pending, delivered, expired)
- reconciliation status
- metadata payload

Persisting notification intents locally ensures that the device maintains a consistent view of delivered and pending notifications even after application restarts, device reboots, or extended offline periods.

To minimize storage overhead, the registry periodically performs cleanup operations that remove expired or acknowledged notification records.

12.3. Push Channel Integration

The push delivery component integrates with the Android push messaging infrastructure, typically through Firebase Cloud Messaging (FCM) or a comparable push gateway. When a push notification arrives, the middleware layer performs the following sequence:

- Parse the push payload and extract the notification identifier.
- Check the Notification Intent Registry for duplicate entries.
- Record the notification intent in persistent storage.
- Deliver the notification through the Android Notification Manager.

In cases where the push payload contains only a notification announcement rather than the full message content, the system triggers a lightweight reconciliation request to retrieve the complete notification data from the backend.

This design allows push messages to serve both as content carriers and synchronization signals, depending on band-width constraints.

12.4. Pull Synchronization Implementation

The Pull Synchronization Engine is implemented using An-droid's Work Manager scheduling framework. Work Manager provides reliable background execution capabilities that com-ply with Android's battery optimization and lifecycle policies.

Pull synchronization tasks are scheduled under several conditions:

- detection of missed push delivery events
- extended push channel inactivity
- restoration of network connectivity after an offline period Synchronization requests retrieve notifications created after the device's last recorded synchronization timestamp. Results are then merged with the local Notification Intent Registry.

To minimize bandwidth usage, the synchronization API supports:

- compressed payloads
- incremental updates
- batch retrieval of notifications

These optimizations are particularly important in satellite networks where bandwidth is expensive and shared across multiple devices.

12.5. Connectivity and Link State Monitoring

The Link State Monitor relies on Android's network connectivity APIs to observe changes in network status. How-ever, simple connectivity detection is insufficient for satellite networks because connectivity may exist but be severely degraded.

Therefore, the monitor incorporates additional indicators including:

- observed push notification latency
- HTTP request round-trip time
- synchronization success rate
- packet loss signals from transport errors

These indicators are aggregated to classify link conditions into operational states such as stable, degraded, intermittent, or offline.

The classification output is provided to the Delivery Coordinator, which uses it to determine appropriate synchronization strategies.

13. Notification Lifecycle Management

13.1. Notification State Machine

Each notification within the system progresses through a structured lifecycle consisting of several states:

- Created: Notification generated by backend system.
- Announced: Notification identifier received through push channel.
- Reconciled: Notification retrieved through pull synchronization.
- Delivered: Notification presented to the user.
- Expired: Notification relevance window exceeded. Transitions between these states are recorded in the Notification Intent Registry to ensure deterministic delivery behavior.

13.2. Duplicate Detection

Duplicate notification delivery can occur when both push and pull mechanisms retrieve the same notification. To pre-vent duplication, the registry enforces strict identifier-based deduplication rules.

If a notification with the same identifier already exists in the registry, the system suppresses additional deliveries while still updating synchronization metadata.

13.3. Expiration Enforcement

Each notification includes a relevance window that defines how long the notification remains meaningful. During reconciliation, notifications whose expiration timestamps have passed are discarded rather than delivered.

This prevents stale notifications from appearing long after their context has expired.

14. Experimental Methodology

14.1. Satellite Network Simulation

Evaluating notification delivery in maritime environments requires replicating satellite link conditions in a controlled laboratory environment. To achieve this, experiments were conducted using a network simulation

framework capable of introducing controlled latency, bandwidth limits, and link disruptions.

The simulation environment emulated several common satellite conditions:

- round-trip latency between 600 ms and 1500 ms
- bandwidth limits between 256 kbps and 2 Mbps
- packet loss rates up to 5 percent
- intermittent link interruptions lasting from 30 seconds to 10 minutes

These parameters reflect typical connectivity characteristics observed in maritime satellite networks.

14.2. Device Test Environment

Experiments were conducted on a diverse set of Android devices representing low-, mid-, and high-tier hardware configurations. Devices ran Android versions ranging from Android 11 through Android 14 to ensure compatibility across platform versions.

Testing across multiple hardware tiers ensured that performance results were not biased by device-specific optimizations.

14.3. Evaluation Scenarios

The experimental evaluation considered several operational scenarios:

- Stable Connectivity Scenario: Continuous satellite link availability with moderate latency.
- Intermittent Connectivity Scenario: Periodic link interruptions lasting several minutes.
- Burst Reconnection Scenario: Large backlog of notifications accumulated during ex-tended outage.
- Bandwidth-Constrained Scenario: Limited link bandwidth shared among multiple simulated devices.

Each scenario was executed multiple times to ensure reproducibility and statistical reliability.

14.4. Metrics Collected

The following metrics were used to evaluate system performance:

- notification delivery success rate
- notification latency relative to creation time
- burst delivery magnitude after reconnection
- bandwidth consumption per device
- duplicate notification rate

These metrics provide a comprehensive view of both system reliability and network efficiency.

15. Experimental Scenarios and Evaluation Strategy

- Push-Only Baseline: The hybrid system was compared against a conventional push-only notification model to establish a baseline. In this configuration, notifications relied solely on push delivery with no reconciliation mechanism.
- Pull-Only Baseline: A pull-only model was also evaluated in which devices periodically polled the backend for new notifications without relying on push delivery.
- Hybrid Model Evaluation: The Hybrid Push–Pull Notification Model was evaluated using the same scenarios to measure improvements in reliability, latency, and bandwidth efficiency.

16. Experimental Results

16.1. Notification Delivery Reliability

The primary objective of the Hybrid Push–Pull Notification Model (HPPNM) is to ensure reliable notification delivery un-der intermittent satellite connectivity conditions. Experimental results demonstrate a substantial improvement in delivery reliability compared to conventional push-only architectures.

Under intermittent connectivity scenarios, the push-only baseline exhibited frequent notification loss due to failed push delivery attempts and the absence of reconciliation mechanisms. In contrast, the hybrid model successfully recovered missed notifications through pull synchronization, resulting in near-complete delivery coverage for notifications whose relevance windows had not expired.

Across repeated experimental trials, the hybrid model consistently maintained a delivery success rate exceeding 98 percent, whereas the push-only system showed delivery rates between 70 and 85 percent, depending on outage duration.

These results confirm that decoupling notification intent from push delivery enables deterministic recovery of missed notifications.

16.2. Notification Latency Characteristics

Notification latency was measured as the time elapsed between notification creation on the backend system and presentation on the device.

Under stable connectivity conditions, both push-only and hybrid models achieved comparable latency performance be-cause push delivery remained the primary mechanism in both systems.

However, during intermittent connectivity scenarios, the hybrid model significantly reduced latency variance. In push-only systems, delayed push messages accumulated at network gateways and were delivered in large bursts after connectivity resumed. This produced unpredictable and often excessive notification delays.

By contrast, the hybrid system used reconciliation mechanisms to detect missing notifications earlier and retrieve them through controlled pull synchronization. As a result, the hybrid model exhibited more consistent latency distribution and significantly reduced the longest observed

delivery delays.

16.3. Burst Delivery Mitigation

One of the most disruptive behaviors observed in push-only architectures is burst notification delivery after connectivity restoration. When satellite links resumed following outages, queued push messages were delivered in rapid succession, overwhelming both device resources and user attention.

HPPNM mitigated this behavior through its controlled reconciliation policy. Instead of releasing all recovered notifications simultaneously, the system prioritized notifications based on relevance and importance, spacing their delivery across a short interval.

Experimental measurements showed that burst magnitude was reduced by more than 60 percent compared to push-only systems. This smoothing behavior improved user experience and reduced device processing spikes.

16.4. Bandwidth Utilization

Satellite bandwidth is a scarce and costly resource in maritime environments. Efficient notification delivery therefore requires minimizing unnecessary data transmission.

The pull-only baseline demonstrated the highest bandwidth consumption due to frequent polling requests. Push-only systems consumed less bandwidth during stable connectivity but incurred additional overhead during repeated reconnect attempts.

The hybrid model achieved the most balanced bandwidth profile. Push delivery handled the majority of notifications during stable periods, while pull synchronization occurred only when necessary. Adaptive synchronization intervals further reduced unnecessary reconciliation requests.

Overall, the hybrid model reduced bandwidth usage by approximately 35 percent compared to pull-only systems, while maintaining significantly higher reliability than push-only approaches.

16.5. Duplicate and Ordering Errors

Duplicate notification delivery and message reordering are common problems in unreliable networks. These errors were frequently observed in the push-only baseline due to retransmission attempts and delayed gateway queues.

The hybrid architecture eliminated duplicate deliveries by enforcing identifier-based deduplication within the Notification Intent Registry. Additionally, notifications retrieved through pull synchronization were sorted by creation time before delivery.

As a result, the hybrid model achieved zero duplicate deliveries and preserved correct chronological ordering across all experimental trials.

17. Discussion

17.1. Advantages of Hybrid Notification Models

The experimental results demonstrate that hybrid push–pull architectures offer significant advantages for mobile systems operating in constrained networks. By combining opportunistic push delivery with deterministic pull reconciliation, HPPNM achieves both low latency and high reliability without excessive bandwidth consumption.

This dual-mode strategy aligns well with the unpredictable nature of satellite connectivity, where network conditions may fluctuate between stable and degraded states.

17.2. Practical Implications for Maritime Applications

For maritime deployments, reliable notification delivery is essential for maintaining operational awareness and passenger communication. Hybrid notification models ensure that critical alerts such as safety announcements or schedule changes are not permanently lost due to temporary network disruptions.

Furthermore, smoothing notification bursts improves usability by preventing sudden floods of outdated messages.

17.3. Architectural Tradeoffs

Although the hybrid model improves reliability and band-width efficiency, it introduces additional architectural complexity compared to push-only systems. The application must maintain persistent notification registries, implement reconciliation logic, and monitor link conditions.

However, these tradeoffs are justified in environments where network reliability cannot be assumed.

17.4. Integration with Other Resilient Mobile Systems

The Hybrid Push–Pull Notification Model complements several other resilience-oriented architectural approaches explored in prior work. Together, these systems form a layered strategy for robust Android applications operating under constrained conditions:

- Ambient Queue Management (AQM): fair execution across user journeys
- Proactive Device-Wide Resource Throttling (PDWRT): system-level stability during peak load
- Deferred Action Scheduling (DAS): reliable execution of offline actions
- Flagged-Component Rollback Engine (LFRE) : UI-level failure recovery
- Hybrid Push–Pull Notification Model (HPPNM): resilient communication over intermittent networks

This layered architecture improves reliability across communication, execution, and user-interface layers.

18. Threats to Validity

Several potential threats to validity should be considered when interpreting the experimental results.

- Simulation Environment: Although satellite

conditions were carefully simulated, lab-oratory environments cannot perfectly replicate real-world maritime networks. Variations in satellite providers, routing infrastructure, and vessel network configurations may produce different results in production environments.

- Device Diversity: While experiments included multiple Android device tiers and platform versions, the Android ecosystem remains highly fragmented. Device-specific background execution policies or vendor optimizations could affect system behavior.
- Notification Workload Characteristics: The evaluation scenarios focused primarily on operational notifications. Applications with significantly different notification workloads such as high-frequency promotional messaging may exhibit different performance characteristics.

19. Future Work

Several avenues for future research remain open.

- First, hybrid notification models could incorporate predictive link-state estimation, allowing devices to anticipate connectivity windows and schedule reconciliation more intelligently.
- Second, cross-device synchronization mechanisms could be explored to coordinate notifications across multiple devices within a vessel network.
- Third, integration with adaptive compression and payload prioritization techniques could further improve bandwidth efficiency in satellite environments.
- Finally, hybrid notification models could be extended to sup-port additional communication primitives such as messaging systems and collaborative applications.

20. Conclusion

This paper presented a Hybrid Push–Pull Notification Model (HPPNM) designed for Android devices operating in maritime environments with intermittent satellite connectivity. By combining opportunistic push delivery with adaptive pull-based reconciliation, the proposed architecture addresses the limitations of traditional push-only notification systems.

Experimental evaluation demonstrated that the hybrid approach significantly improves delivery reliability, reduces burst notification behavior, and optimizes bandwidth utilization while maintaining low-latency delivery under stable network conditions. The results highlight the importance of designing mobile communication architectures that explicitly account for net-work constraints rather than assuming persistent connectivity. The Hybrid Push–Pull Notification Model provides a practical and deployable solution for resilient notification delivery in satellite-constrained environments.

References

1. Android Developers, Firebase Cloud Messaging Documentation, Google, 2024.
2. Android Developers, Work Manager Guide, Google Android Documentation, 2023.
3. K. Fall, “A Delay-Tolerant Network Architecture for Challenged Inter-nets,” Proceedings of ACM SIGCOMM, 2003.
4. J. Dean and L. Barroso, “The Tail at Scale,” Communications of the ACM, 2013.
5. M. Zaharia et al., “Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling,” EuroSys, 2010.
6. A. Carroll and G. Heiser, “An Analysis of Power Consumption in a Smartphone,” USENIX ATC, 2010.
7. Android Developers, Background Execution Limits, Google Android Documentation, 2023.
8. S. Farrell and V. Cahill, Delay and Disruption Tolerant Networking, Artech House, 2006.