

An Android Application For Autonomous File Sharing In Delay Tolerant Network

Posted on September 17, 2018

View this essay online:

<https://academic-master.com/an-android-application-for-autonomous-file-sharing-in-delay-tolerant-network/>

Abstract

This project proposes an Android application for autonomous file sharing in a delay-tolerant network (DTN). The system is designed for situations in which devices cannot rely on continuous internet access or a stable end-to-end path. Instead of requiring immediate connectivity between sender and recipient, the application stores a file, carries it while the user moves, and forwards it when another trusted device becomes available. The design follows the store-carry-forward principle used in DTN architectures while adapting it to current Android constraints, privacy requirements, and nearby-device APIs.

A practical implementation can use Google Nearby Connections for local discovery and transfer because it supports offline peer-to-peer communication using combinations of Bluetooth, BLE, and Wi-Fi. The DTN layer should remain separate from the transport layer so that routing, storage, security, and delivery logic are not tied permanently to one Android API. The recommended first implementation uses limited-copy Spray and Wait routing, encrypted payloads, signed metadata, delivery receipts, storage quotas, and explicit user consent. The objective is not to create an anonymous public relay network, but a controlled file-sharing system for groups operating in emergency, fieldwork, campus, event, or rural environments where connectivity is intermittent.

System Design

Delay-tolerant networking is designed for environments where communication paths are disrupted, delayed, intermittent, or highly variable. RFC 9171 describes the Bundle Protocol as a store-carry-forward overlay for stressed networks. A device can accept a message, preserve it locally, move physically while disconnected, and forward it during a later encounter. This model differs fundamentally from ordinary client-server applications, which normally assume that the destination or server can be reached when the user initiates an action (Burleigh et al., 2022).

The Android application should represent each shared item as a bundle containing the encrypted payload and enough metadata to route and manage it. Useful fields include a unique identifier, source identity, destination or group identifier, creation time, expiration time, priority, file size, cryptographic

digest, routing copy count, and delivery status. Files should remain immutable after transmission begins. A revised file should be created as a new bundle so that version conflicts do not silently alter content already moving through the network.

Routing should begin with a limited-copy Spray and Wait strategy rather than unrestricted flooding. Epidemic routing can improve delivery probability by replicating content to every eligible node, but it can consume storage, battery, and radio capacity very quickly. Under binary Spray and Wait, the sender creates a fixed copy budget. A carrier with more than one copy may give roughly half of its remaining copies to another eligible relay, while a node holding only one copy waits for the destination. This provides a useful balance between delivery opportunity and resource consumption.

Each node should maintain a durable queue. When two trusted devices meet, they exchange compact summaries of known bundle identifiers before transmitting full files. Duplicate suppression prevents the same payload from being sent repeatedly. Priority and expiration should influence transfer order so that a small urgent text file can move before a large video. Storage limits should distinguish between files owned by the user and encrypted relay files carried for other participants.

Component	Main Role
Session manager	Creates or joins trusted sharing groups and controls discovery
Routing engine	Selects bundles, manages copy budgets, priorities, and expiration
Transfer layer	Uses Nearby Connections or another local transport
Repository	Stores bundle metadata, transfer state, and file references
Security layer	Manages keys, encryption, signatures, and trust
User interface	Shows peers, queue state, delivery progress, and storage use

Android Layer

Nearby Connections is suitable for the first implementation because it supports fully offline device discovery, connection establishment, and payload exchange. Google documents byte, file, and stream payloads and notes that Nearby Connections uses Bluetooth, BLE, and Wi-Fi underneath while abstracting much of the radio-specific complexity (Google Developers, 2026a). This allows the project to focus on DTN behavior instead of building every transport mechanism separately.

Nearby Connections should still be isolated behind an interface. The routing engine should not know whether a bundle travels through Nearby Connections, Wi-Fi Direct, a local hotspot, or a future transport. Separating these responsibilities makes the application easier to test and allows the transport layer to evolve without rewriting the DTN logic.

Current Android permissions must be handled according to OS version and target SDK. Android 13 and later use the `NEARBY_WIFI_DEVICES` runtime permission for relevant Wi-Fi operations, while newer Android versions introduce additional local-network restrictions. Google's current Nearby Connections documentation also identifies `ACCESS_LOCAL_NETWORK` for Android 17-targeting applications using local Wi-Fi transport. The app should therefore request only permissions required by the supported Android versions and explain why each permission is needed (Android Developers, 2026; Google Developers, 2026b).

Discovery should occur only during an explicit sharing session. Continuous invisible scanning would create unnecessary battery and privacy costs. Background work must also comply with modern Android execution limits. Long-running discovery or transfer may require a foreground service with visible user notification, while scheduled cleanup or deferred synchronization can use `WorkManager`. The application should remain functional when the internet is unavailable because internet connectivity is not the core transport assumption.

Nearby Connections provides authentication tokens that both users can compare before accepting a connection. Google specifically warns that unauthenticated connections are insecure. The application should therefore require token confirmation, QR-based group enrollment, organization-issued credentials, or another explicit trust mechanism rather than automatically accepting every nearby endpoint (Google Developers, 2026c).

Transfer state must also be handled correctly. Receiving the first byte of a file does not mean the entire payload has arrived; Google's API reports completion separately. The application should write incomplete files to temporary storage, wait for successful transfer completion, verify the expected digest, and only then commit the bundle as valid. Current Payload APIs also support offsets for resuming file or stream transfers, which can reduce wasted bandwidth after a short disconnection (Google Developers, 2026d).

Security Model

A relay device should not need to read the files it carries. Link encryption between nearby devices protects a transfer during one encounter, but store-carry-forward creates an additional problem because payloads remain at rest on intermediary devices. End-to-end encryption is therefore required. The sender can encrypt a file with a random content key and then encrypt that key for the intended recipient or approved group. Relays store ciphertext only.

The security model is conceptually consistent with the DTN security principles formalized in Bundle

Protocol Security. RFC 9172 defines integrity and confidentiality services for data that may travel through untrusted store-carry-forward nodes and emphasizes that transport protection alone may be insufficient in a DTN environment (Birrane & McKeever, 2022). This project does not need to implement the full Bundle Protocol to apply the same principle.

Each device can maintain a signing key in Android Keystore. Bundle metadata should be signed so recipients can verify who created the bundle and detect modification. Content digests verify file integrity after transfer. Endpoint aliases should avoid exposing phone numbers or full legal names during discovery. Metadata should also be minimized because file names, recipient identifiers, timing, and file size can reveal information even when the payload is encrypted.

The application should assume that a relay can be malicious. Threats include storage exhaustion, replay, identity spoofing, tampered metadata, malicious files, denial of service, and attempts to join private sessions. Copy budgets, size limits, expiration, signed metadata, authenticated group membership, and per-user storage quotas reduce these risks. Received files should remain in application-private storage until the final recipient deliberately opens or exports them.

File safety also matters. The application should sanitize display names, validate MIME types, block executable installation packages by default, and use Android content URIs rather than unrestricted file paths. A user should always know who or which trusted group supplied the file before opening it. Malware scanning can be added where practical, but a clean scan should not be presented as proof of safety.

Testing Plan

The project should be evaluated on physical Android devices rather than emulator-only testing because radio behavior, battery management, permissions, and vendor-specific background restrictions affect real performance. The test environment should include several Android versions and manufacturers where possible.

Unit tests should verify routing copy counts, duplicate detection, expiration, priority order, signature validation, receipt handling, and state transitions. Instrumentation tests should cover denied permissions, process termination, low storage, device rotation, interrupted transfer, and accessibility. Security testing should include an unauthorized node, modified metadata, replayed bundles, excessive payload size, and invalid signatures.

Network evaluation should compare at least three routing strategies under the same movement pattern: direct delivery, epidemic routing, and Spray and Wait. Useful outcome measures are delivery ratio, median delivery delay, transmitted bytes per successful delivery, average copies per bundle, battery consumption, storage occupancy, interrupted transfers, resumed transfers, and rejected unauthorized connection attempts.

Delivery receipts can be implemented as small signed bundles. After the destination successfully verifies and stores a file, it creates a receipt that propagates back through the network. Relays receiving a valid receipt can delete or deprioritize copies of the delivered payload. The user interface should distinguish clearly between “forwarded to a relay” and “delivered to recipient” so that the system does not create false certainty.

The first development phase should support authenticated direct transfer, encrypted storage, Room persistence, and clear status. The second phase can add Spray and Wait forwarding. A third phase can add resumption, signed receipts, and smarter storage management. More advanced features such as encounter-history routing or collaborative version synchronization should be postponed until the basic relay model has been tested thoroughly.

Limitations

The application cannot guarantee delivery because DTN performance depends on physical encounters. If no path emerges before the bundle expires, the file remains undelivered. Nearby-device discovery can also vary by hardware, radio conditions, OS behavior, and battery restrictions. A DTN application should therefore communicate probabilistic delivery honestly instead of presenting itself as a replacement for reliable internet messaging.

Privacy remains imperfect even with encrypted files. Relays can observe that a transfer occurred, its approximate size, timing, and encounter pattern. Large public relay networks could also be abused for unwanted or illegal content. For that reason, the proposed implementation should remain opt-in and group-based, with explicit control over relay participation and storage.

The project is best understood as a practical Android demonstration of store-carry-forward communication rather than a full standards-compliant Bundle Protocol implementation. Its value lies in combining current Android connectivity with DTN principles in a way that is secure, resource-aware, and understandable to users. A successful prototype would demonstrate that mobile devices can exchange files beyond direct radio range through trusted physical mobility even when no continuous internet path exists.

References

Android Developers. (2026). *Nearby Wi-Fi Devices and Local Network Permissions*.

Birrane, E., & McKeever, K. (2022). *Bundle Protocol Security (BPSec)*. RFC 9172.

Burleigh, S., Fall, K., Birrane, E., et al. (2022). *Bundle Protocol Version 7*. RFC 9171.

Google Developers. (2026a). *Nearby Connections Overview*.

Google Developers. (2026b). *Nearby Connections for Android: Get Started*.

Google Developers. (2026c). *Nearby Connections: Manage Connections*.

Google Developers. (2026d). *Nearby Connections: Exchange Data*.