An Android Application For Autonomous File Sharing In Delay Tolerant Network

Posted on September 17, 2018

Abstract

This project proposes an Android application for autonomous file sharing in a delay-tolerant network. The original design correctly recognizes that conventional communication depends too heavily on stable end-to-end connectivity and that mobile devices can exchange content through store-carry-forward routing when infrastructure is intermittent. A modern implementation should not rely on the discontinued Nearby Messages API or on Wi-Fi Direct alone. Android's current Nearby Connections API can discover, connect, and exchange payloads with nearby devices using Bluetooth, BLE, and Wi-Fi while abstracting some radio details. The application should combine that connectivity layer with explicit user consent, encrypted identity, signed metadata, safe payload storage, duplicate suppression, expiration, forwarding policy, and auditable delivery state. The system is best suited for opt-in emergency, fieldwork, campus, festival, and rural scenarios where devices intermittently meet and no continuous server connection can be assumed. (Google Developers, n.d.-a, n.d.-b; Fall, 2003; Vahdat & Becker, 2000)

Introduction

Most mobile applications assume that a device can reach a server whenever it needs to send or retrieve information. That assumption fails during disaster, infrastructure outage, remote fieldwork, crowded events, travel, or communication restrictions. A delay-tolerant network, or DTN, is designed for environments in which a complete source-to-destination path may not exist at one time. A node stores a message, carries it while moving, and forwards it when it encounters another useful node.

Android phones are attractive DTN nodes because they combine storage, computation, several radios, location capability, and user mobility. They also impose constraints involving battery, background execution, permissions, privacy, storage, and manufacturer-specific behavior. A successful application therefore needs more than a routing algorithm. It must align the networking concept with current Android platform rules and protect users from unauthorized discovery or file exchange.

Problem Statement

Users need to exchange selected files where mobile internet, Wi-Fi infrastructure, or continuous peer connectivity is unavailable. Direct transfer solves only the case in which sender and recipient are

nearby. The proposed system allows trusted intermediary devices to carry encrypted content and forward it later.

The application should not promise immediate delivery. It should present delay-tolerant semantics clearly: a file is queued, replicated according to policy, and delivered when an encounter path emerges. The user needs status such as pending, carried by peers, delivered, expired, failed validation, or manually canceled.

Objectives

1. Create a consent-based Android application for nearby discovery and file exchange.
2. Implement store-carry-forward routing without continuous infrastructure.
3. Protect content, identity, and metadata with strong security and data minimization.
4. Support useful delivery while controlling battery, storage, and replication overhead.
5. Measure delivery probability, latency, overhead, energy, and failure behavior.

Scope and Use Cases

The first version should target small groups that deliberately join the same sharing session. Example uses include distributing emergency notices in a shelter, exchanging field observations among researchers, sharing event materials at a congested venue, or moving documents between teams in remote areas.

The application should not operate as an invisible public relay for arbitrary strangers. Automatic participation without clear consent could expose users to surveillance, illegal content, harassment, and storage abuse. Session codes, QR invitations, organization-issued credentials, or preapproved group keys can limit participation.

Delay-Tolerant Networking Model

A DTN introduces a bundle layer above the transport available during an encounter. Each bundle contains a payload or payload reference and metadata such as identifier, source, intended recipient or group, creation time, expiration, priority, size, forwarding limit, integrity value, and delivery receipt policy.

Nodes maintain a durable queue. When two devices connect, they exchange compact summaries of known bundle identifiers and capabilities. Each side requests missing bundles that satisfy policy. Completed transfers are stored safely before being advertised to the next node.

Store-Carry-Forward

Store-carry-forward separates movement from connectivity. A device can receive a file at one location, remain offline while the user travels, and later encounter the destination or another relay. This mechanism extends communication range through mobility.

Durability is essential. An incomplete or unverified file should not be treated as stored. The application should write to temporary storage, verify length and cryptographic digest, then atomically commit metadata and payload state.

Routing Alternatives

Direct Delivery

Direct delivery stores the bundle until the destination appears. It has minimal overhead but poor performance when sender and recipient rarely meet.

Epidemic Routing

Epidemic routing replicates a bundle to every eligible peer that lacks it. It can produce high delivery probability in small networks but consumes bandwidth, battery, and storage. The original project’s default flooding approach is therefore unsuitable without limits.

Spray and Wait

Spray and Wait assigns a limited number of copies. Copies are distributed during encounters, and carriers then wait for the destination. It provides a practical balance for an undergraduate Android implementation.

Encounter-Based Routing

A device can estimate which peers have recently or frequently encountered a destination and forward accordingly. This may reduce overhead but stores social-contact information and requires careful privacy design.

Recommended Routing Policy

The application should begin with binary Spray and Wait. A source assigns a configurable copy

budget. When a carrier with more than one copy meets an eligible relay, it gives approximately half the remaining copies and retains the rest. A node with one copy forwards only to the destination.

Priority, expiration, file size, battery, available storage, and trust should influence transfer order. Emergency text bundles can precede large media files. Users and administrators need limits for maximum copies, maximum age, and maximum relayed storage.

Android Connectivity Layer

Nearby Connections is an appropriate current Google API for discovering and connecting nearby devices and exchanging byte, file, or stream payloads. Google documents that it uses Bluetooth, BLE, and Wi-Fi and can operate without internet access. It provides point-to-point, star, and cluster strategies. A cluster strategy may suit multiple peer relays, while point-to-point can provide higher-throughput individual transfers.

The application should not be designed around Nearby Messages because that service was deprecated and shut down. Wi-Fi Direct remains available for direct peer-to-peer connections, but recent Android releases require runtime permissions, including `NEARBY_WIFI_DEVICES` for relevant APIs. Building directly on several radio stacks increases complexity; a well-supported abstraction should be used where its limits match the project.

Permissions and Platform Behavior

Nearby Connections requires different Bluetooth and nearby-device permissions according to Android version. The app should request only permissions needed for the chosen feature, explain the user benefit, and handle denial gracefully. Location should not be requested merely because an older tutorial required it if the current API and operating-system version provide a narrower alternative.

Android limits background activity to protect battery and privacy. Continuous discovery should run only during an explicit sharing session, with a foreground service and visible notification where required. WorkManager can schedule deferred cleanup, receipt processing, or server synchronization, but it is not a substitute for a live nearby connection.

Application Architecture

A layered design improves testability and replaceability:

- Presentation layer: Compose or Views screens, accessibility, permissions, and status.
- Session layer: group joining, identity verification, and encounter lifecycle.
- Routing layer: copy budgets, eligibility, priority, and receipt handling.

- Transfer layer: Nearby Connections payload exchange, chunking, and retry.
- Repository layer: Room database and encrypted file storage.
- Security layer: keys, signatures, encryption, trust, and policy.

Interfaces should isolate Nearby Connections from routing logic so another transport can be added later without rewriting the complete application.

Data Model

The Room database can include the following entities:

- Peer: internal ID, public-key fingerprint, display alias, trust state, and last encounter.
- Bundle: UUID, source, destination or group, timestamps, priority, size, digest, status, and copy budget.
- Payload: local URI, MIME type, encrypted-file path, received bytes, and validation state.
- Encounter: peer, start, end, connection result, and summarized metrics.
- Transfer: bundle, peer, direction, bytes, timestamps, and result.
- Receipt: bundle ID, destination signature, receipt time, and propagation state.

Foreign keys and unique constraints prevent duplicate bundle records and orphaned transfers. The database should store file references rather than large file bytes.

Bundle Identification and Deduplication

Each bundle should use a random UUID plus a signed source identity. A content digest verifies the payload but should not be the sole identifier because two intentional messages may contain identical data. Peers exchange Bloom filters or compact ID summaries to avoid retransmitting known content.

Deduplication must account for partially received payloads. A completed bundle, an in-progress transfer, and an invalid bundle have different states. Resumption tokens can prevent restarting large files from zero after a brief disconnection.

Connection Protocol

1. A user starts or joins an approved session.
2. The device advertises a randomized endpoint identity and discovers compatible peers.
3. Both devices initiate a connection and display the authentication code provided by Nearby Connections.
4. Users or organizational policy verify the code before accepting.
5. Peers exchange signed capabilities and inventory summaries.

6. The routing engine selects eligible bundles.
7. Metadata is sent before file payloads.
8. Payloads are validated and committed.
9. Peers exchange signed acknowledgments and update copy counts.

Security Threat Model

Threats include an unauthorized node joining a group, interception, content alteration, spoofed identities, replayed bundles, malicious files, metadata inference, storage exhaustion, denial of service, and a compromised relay. The app should assume that an intermediary can be curious or malicious.

Nearby Connections encrypts communication links, but end-to-end protection is still necessary because relays store the payload. A destination's public key can encrypt a random content key, and the bundle can carry ciphertext that intermediaries cannot read. Group messages require carefully governed group keys or per-recipient encrypted keys.

Identity and Authentication

A device can generate a long-term signing key in Android Keystore. Contacts exchange public-key fingerprints through QR codes or an organization directory. Nearby endpoint names should be temporary and should not reveal phone numbers or real names.

Connection authentication tokens must be confirmed or verified through trusted policy. Google warns that unauthenticated connections are insecure. Automatic acceptance of any nearby endpoint would undermine the project's privacy and integrity.

End-to-End Encryption and Signatures

The sender encrypts the file with an authenticated-encryption algorithm, wraps the content key for authorized recipients, and signs the immutable metadata. The destination verifies the signature and digest before opening the file. Relays see only the routing metadata necessary for forwarding.

Metadata minimization is important. A relay may not need the sender's legal identity, filename, or exact destination name. Opaque recipient identifiers and coarse priority reduce exposure. Some leakage remains because file size, timing, and encounter patterns are visible.

File Safety

Received files should remain in app-private storage until the recipient deliberately exports or opens them. The app should validate declared MIME type, sanitize display names, limit size, and use Android content URIs rather than unrestricted file paths. Executable installation packages and dangerous formats can be blocked by default.

Malware scanning may be offered where a trusted engine exists, but absence of detection does not prove safety. The interface should identify the sender or trust group and warn users before opening active content.

Storage Management

Each user should set separate limits for owned and relayed files. The eviction policy can remove expired, invalid, low-priority, or well-replicated relayed bundles before removing unique high-priority bundles. User-created content should not be deleted without clear notice.

Expiration and hop or copy limits prevent indefinite propagation. Cleanup jobs should erase payloads securely enough for the device storage model and retain only minimal metrics according to policy.

Energy Management

Continuous scanning and advertising consume energy. Discovery can use bounded windows, screen-state awareness, charging preference, encounter learning, and user schedules. The app should expose battery mode rather than hide consumption.

Transfers should consider radio setup cost and remaining contact time. Small metadata and priority bundles can be transferred before large files. A critically low-battery device can refuse relay work while preserving emergency receiving.

User Interface

The home screen should show session status, nearby trusted peers, queued files, storage used, and battery mode. File sharing should require the user to choose recipients, priority, expiration, and relay permission. The app should explain that relay permission stores encrypted copies on other devices.

Status should avoid false certainty. “Forwarded to three relays” is different from “delivered to recipient.” A progress history can show accepted, replicated, delivered, expired, or rejected. Accessibility support includes screen-reader labels, large text, sufficient contrast, and alternatives to color-only status.

Delivery Receipts

The destination creates a signed receipt after verifying and committing the bundle. Receipts propagate through the network like small high-priority bundles. On receiving a valid receipt, nodes can delete or deprioritize the corresponding payload.

A receipt reveals that delivery occurred and can expose timing. Users should be able to disable receipts for sensitive use cases, with the tradeoff that relays may carry data until expiration.

Conflict and Version Handling

Files are immutable bundles in the initial version. A changed file is a new bundle referencing the previous version. This avoids difficult multiwriter conflict resolution.

A future collaborative feature could use operation-based synchronization, but it would require authorization, merge semantics, and substantially more testing. Autonomous file sharing should not be confused with a full distributed file system.

Server Assistance

The core transfer should work without internet. An optional server can register group public keys, distribute revocation lists, back up receipts, and synchronize when connectivity returns. The server must not be required for every encounter.

Server assistance improves manageability but introduces central trust and metadata. The system should document which functions remain offline and what breaks if the server is unavailable.

Implementation Plan

Phase One: Direct Trusted Transfer

Implement discovery, authenticated connection, encrypted single-hop file transfer, Room persistence, and status. Validate platform behavior across supported Android versions and several manufacturers.

Phase Two: Store-Carry-Forward

Add bundles, inventory exchange, Spray and Wait routing, expiration, duplicate suppression, and relay storage limits.

Phase Three: Receipts and Resumption

Add signed delivery receipts, partial-transfer resumption, copy cancellation, and metrics.

Phase Four: Policy and Administration

Add invitation methods, organization groups, key revocation, configurable routing, and privacy-preserving export of performance data.

Testing Strategy

Unit tests should cover routing copy counts, expiration, prioritization, state transitions, and cryptographic verification. Instrumentation tests should cover permissions, process death, storage pressure, and UI accessibility.

Network experiments require physical devices moving through repeatable encounter patterns. Cases should include interrupted transfer, simultaneous peers, duplicate advertisement, malicious metadata, low battery, clock changes, canceled sessions, and destination receipt propagation.

Evaluation Metrics

- Delivery ratio within an expiration period.
- Median and percentile delivery delay.
- Transmitted bytes divided by unique delivered bytes.
- Average copies per bundle.
- Battery consumed during active sessions.
- Storage occupancy and eviction count.
- Failed, interrupted, and resumed transfers.
- Unauthorized connection attempts rejected.

Comparison should include direct delivery, epidemic routing, and Spray and Wait under the same movement trace and workload.

Ethical and Legal Considerations

Autonomous relaying can move content through a person’s device without the person reading it. Participation must therefore be voluntary and limited by clear rules. Users need control over storage, data use, network groups, and withdrawal.

Emergency deployments should not imply that an application replaces official alerts or professional communication. Illegal or harmful content, child safety, export restrictions, privacy law, and organizational record retention require scenario-specific policy. The developer should not promise anonymity that metadata cannot provide.

Limitations

Android devices vary in radio behavior and background restrictions. Nearby Connections abstracts transport but does not guarantee discovery or throughput in every environment. User movement may be insufficient to produce a path. Encryption protects content but not all traffic analysis.

Store-carry-forward delivery is probabilistic. Highly partitioned networks may require fixed gateways, vehicles, or satellite connectivity. The proposed app is a complement to infrastructure, not a universal substitute.

Conclusion

An Android delay-tolerant file-sharing application can provide useful communication when stable end-to-end connectivity is unavailable. The design should use store-carry-forward bundles and a controlled replication policy rather than unrestricted flooding. Nearby Connections provides a current Android mechanism for encrypted local connections and payload exchange across Bluetooth, BLE, and Wi-Fi, while authentication must be verified before data are exchanged.

The difficult parts are not merely discovering peers and sending files. A trustworthy system requires end-to-end encryption, signatures, minimal metadata, permissions, safe storage, energy controls, routing state, receipts, expiration, and user-visible consent. A phased implementation using direct transfer followed by Spray and Wait routing is achievable and testable. Its success should be measured through delivery, delay, overhead, energy, security, and usability rather than through a demonstration in which two phones happen to remain connected.

References

Fall, K. (2003). A delay-tolerant network architecture for challenged internets. *Proceedings of ACM SIGCOMM 2003*.

Vahdat, A., & Becker, D. (2000). *Epidemic routing for partially connected ad hoc networks*. Duke University Technical Report CS-2000-06.

Google Developers. (n.d.-a). *Nearby Connections overview*.

Google Developers. (n.d.-b). *Nearby Connections strategy guide and Android API guidance*.

Android Developers. (n.d.). *Wi-Fi Direct (peer-to-peer) overview*.