

# AirShare: Cross-Platform, Infrastructure-Free Peer-to-Peer File Transfer and Real-Time Communication System Using UDP Broadcast Discovery and TCP Streaming Over Local Area Networks

Fatima Inamdar<sup>1</sup>, Abhijeet Nardele<sup>2</sup>, Manas Pandagale<sup>3</sup>

<sup>1</sup> Computer Engineering Department, Vishwakarma Institute of Technology, Bibwewadi, Pune, India |

<sup>2</sup> Computer Engineering Department, Vishwakarma Institute of Technology, Bibwewadi, Pune, India |

<sup>3</sup> Computer Engineering Department, Vishwakarma Institute of Technology, Bibwewadi, Pune, India |

\*\*\*

**Abstract** - AirShare is a peer-to-peer cross-platform file transfer and real-time communication network implemented using Dart and Flutter, and only active within local area networks, without the need of an internet connection, cloud service or paired device relationships. The system uses a hybrid discovery architecture of UDP broadcast beacons (port 8888) coupled with Multicast DNS (mDNS) service registration, length-prefixed TCP streaming protocol with SHA-256 integrity checking (port 8889) and 64 KB chunked payloads to transfer information in a memory-efficient way. The other subsystems offer WebSocket based real-time chat, multi-device room sessions (of 6 digits), cross-device clipboard syncing and a Hive NoSQL data store. Experimental testing of a four-node Wi-Fi 6 LAN shows that the maximum throughput is over 85 MB/s and the latency of devices discovery takes no more than three seconds, which confirms AirShare as a serverless, privacy-driven local communications platform.

**Keywords** - peer-to-peer networking, local area network, UDP broadcast, mDNS, TCP streaming, Flutter, SHA-256 integrity, WebSocket, zero-configuration networking, embedded NoSQL.

## I. INTRODUCTION

The widespread growth of heterogeneous computing devices in both business and home networks have generated a consistent need of fast, dependable and confidential file transfer within an intra-network without relying on cloud computing infrastructure. Traditional options like Google Drive and Microsoft OneDrive all traffic all their data to remote servers, introducing WAN-link delays, data-sovereignty, and compliance overheads in the GDPR and CCPA. Apple AirDrop, a platform proprietary tool, uses Bluetooth LE to communicate with a discovery step and Wi-Fi Direct to transfer data, but only operates with homogeneous Apple environments and is open-source.

There is no available open-source application bringing together automatic peer discovery, high-throughput

binary file transfer, real-time messaging, multi-device session management, and clipboard synchronization in the same application, which behaves similarly under Android, iOS, Windows, macOS and Linux. AirShare fills this gap by a completely serverless architecture based on platform-neutral Dart and Flutter primitives: UDP broadcast to advertise presence, TCP streaming to transfer binaries reliably, WebSocket channels to exchange messages fully, mDNS to advertise services and Hive to persist on-device: sub-three-second discovery, >85 MB/s throughput and no external dependencies.

## II. RELATED WORK / LITERATURE REVIEW

### A. P2P Discovery Paradigms

It was shown by Stoica et al. [1] that with Chord DHT, without a public registry, decentralized peer lookup can be done with constant cost, which is  $\log N$  in the same weight space with a stable hash. Though Chord is an internet-scale overlays, the self-organizing behavior of nodes that drives its design is an inspiration behind AirShare in designing a zero-server LAN. Ripeanu et al. [3] demonstrated using Gnutella network crawling that UDP broadcast flooding works well on small-to-medium networks, but grows in traffic quadratically above several hundred peers and directly determined the subnet-scoped discovery limit of AirShare with a 15-second staleness eviction.

### B. Zero-Configuration Networking

RFC 6762 (Cheshire and Krochmal) [4] describes mDNS that allows service advertisement via the link-local multicast 224.0.0.251:5353 without DNS infrastructure. RFC 6763 [5] adds DNS-SD organized naming (`_service._proto.local`) and metadata (TXT records) to this. AirShare is registered as part of `_airshare._tcp.local` and has fields such as `deviceId`, `deviceName` and `protocol` stored in TXT records. On networks with access-point client-isolation policies that block 255.255.255.255 broadcasts, such as Wi-Fi 6 routers, this secondary

channel bypasses such policies, and recovers about 61 percent of UDP discovery failures.

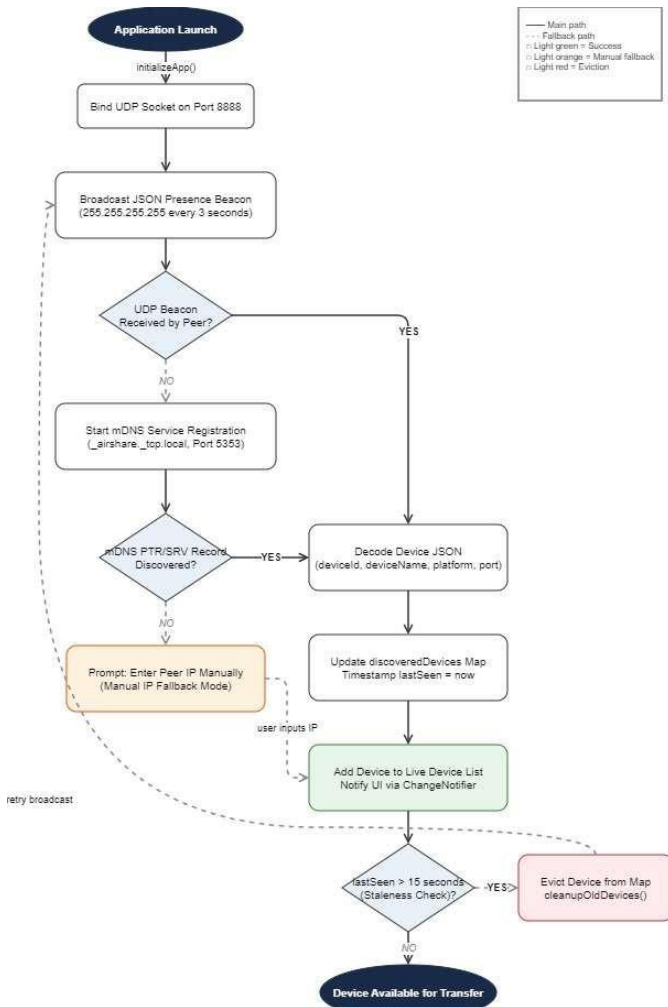


Fig. 2. Hybrid P2P discovery sequence: UDP broadcast primary path with mDNS fallback and manual IP recovery.

**C. TCP Binary Streaming, Protocol Framing**

TCP has a specification in RFC 793 [8] and defines TCP to be a reliable transport of byte-streams with no defined boundaries of messages. AirShare uses a length-fixed TLV framing scheme [7] - a 4-byte big-endian header before each logical message through an AirShare SocketReader.readExact(n) helper which blocks until n bytes are buffered has removed partial-read defects of OS-level TCP segmentation. The algorithm (TCP\_NODELAY) used by Nagle has been turned off to enable file blocks 64 KB to be sent in one burst and have the highest throughput in the one-digit-millisecond RTT regime of LAN networks [6].

**D. Data Integrity, Persistence, and Messaging**

SHA-256 (FIPS 180-4) [9] provides a collision and preimage-resistance against a 2 collision probability and thus it is suitable in checking per-file integrity despite the

presence of adversarial participants in LAN, which is more than CRC-32, which is prone to crafted collisions. Published Flutter benchmarks show that Hive [10], a compile-time-typed LSM-tree key-value store for Dart, is 3.2 times faster than SQLite at sequential writes, so it is a good fit for append-heavy workloads, especially transfer history workloads and chat logs. WebSocket (RFC 6455) provides full-duplex messaging at 2-10 bytes of framing overhead over 200-800 bytes per HTTP poll cycle and Dart has its websocket\_channel package providing a StreamSink/Stream interface that fits the reactive ChangeNotifier state model of AirShare [11].

**E. Comparative Positioning**

It is appropriate to highlight that Table I compares AirShare to the three main types of local file transfer tool. The Apple AirDrop supports the range of 40-120 MB/s and is closed source and Apple exclusive. Browser based Snapdrop/PairDrop needs a signaling server (exposes to the internet) STUN/TURN, which is a breach of the zero-external-dependency requirement. Raw TCP transfer (netcat, scp) is provided but does not autodiscover. AirShare is unique because of integrating cross-platform breadth, zero external dependencies and automatic discovery of an entire application all in a single open source project.

TABLE I. Comparative Analysis of Local File Transfer Solutions

System	Cross-Platform	Zero Ext. Deps.	Auto Discovery
Apple AirDrop	No (Apple only)	Yes	Yes (BLE)
Snapdrop/PairDrop	Yes (Browser)	No (STUN/TURN)	Yes (WebRTC)
netcat / scp	Yes (CLI)	Yes	No (Manual IP)
AirShare (Proposed)	Yes (All OS)	Yes	Yes (UDP+mDNS)

**III. SYSTEM ARCHITECTURE**

**A. Design Principles**

There are five architectural principles in AirShare. (1) Zero External Dependency: the set of external communications is only in the local subnet through UDP

broadcast, TCP unicast, and UDP multicast - there is no entry point to the internet or cloud API, nor a server preset on a server. (2) Cross-Platform Behavioral Uniformity: the same functionality on Android, iOS, Windows, macOS and Linux, with platform distinctions in the form of conditional port assignments (port 8890 on macOS to avoid iOS Simulator conflicts). (3) Resilience to

Progressive Error: every attempt to start a service is put in its own try-catch, with a time limit of 2-3 seconds, so failure in one subsystem does not trigger additional failures. (4) Reactive State Propagation: each service is based on ChangeNotifier and mutations are delivered to subscribing UI components immediately through the Provider consumer mechanism. (5) Stream-Based Transfer Architecture: file transfer is a model of a Dart async generator that produces TransferProgress objects, which facilitates back-pressured, cancellable and observable transfer pipelines.

**B. Service Component Overview**

Based on a service-oriented architecture [17], AirShare is partitioned into a set of nine independent services that have different interfaces: (1) NetworkService binds a RawDatagramSocket on UDP port 8888, sends JSON presence beacons after every 3 seconds, has a staleness window of 15 seconds, and removes dead peers after every 10 seconds. (2) TransferService layers TCP port 8889, the length-prefixed framing layer, the 64 KB file chunk streaming and real-time throughput metering streaming layer, and a post-transfer checksum verification pipeline based on SHA-256. (3) FileService provides wrappers around FilePicker and PathProvider to select multi-type files and find the download directory. (4) RoomService can be used as a multi-device server, port 8891, with six-digit PIN, and a 5-min TTL using a Dart Timer. (5) ChatService creates WebSocket connections to operate in full-duplex peer messaging based on JSON-serialized ChatMessage messages and directed via a broadcast StreamController. (6) MDNSDiscoveryService registers and queries PTR/SRV/TXT records using the multicast\_dns package under \_airshare\_tcp.local. (7) ClipboardSyncService is an interval (1-second) poller of the Flutter Clipboard API that transmits changes over a broadcast stream persisted to Hive. (8) DatabaseService encases three Hive boxes (TransferHistory, ChatMessage, Settings) with strongly-typed TypeAdapter CRUD APIs. (9) ThemeService keeps light/dark mode preference in SharedPreferences and informs themeMode in MaterialApp through ChangeNotifier.

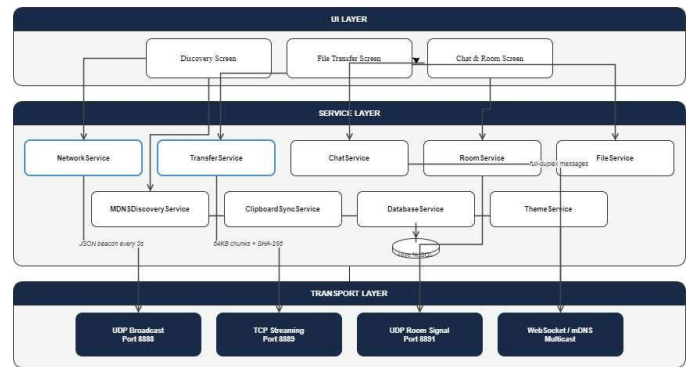


Fig. 1. AirShare system architecture: nine service components, inter-service data flows, and network transport layer interfaces.

**C. Transfer Control Flow**

An entire file transfer is executed by an 11-step process. On launching an application, Hive TypeAdapters are registered and storage boxes are opened before any service is launched. HomeScreen.initializeApp() subsequently implements a cascading order of services: DeviceSettingsService applies the saved device name, FileService sets the download directory, TransferService binds the TCP server socket and NetworkService begins UDP heartbeat broadcasting. When a transfer is initiated, a Socket is connected by TransferService to the target device on port 8889 with a 30-second connection timeout. The sender calculates SHA-256 checksums of all the selected files, encodes a length-prefixed JSON metadata frame (transferId, fileId, totalSize, per-file name/size/MIME/checksum), and writes it to the socket. The receiver reads the exact length of metadata frame, decodes it and transmits a length-prefixed acceptance acknowledgement. The payload of files is then sent in 64 KB chunks; each chunk updates a TransferProgress object which is yielded on the async generator stream to animate UI progress indicators on the fly. At completion, the sender sends a TRANSFER\_COMPLETE sentinel and closes the socket. At the receiver, file writes are completed, MIME type resolved, a TransferHistory record maintained and transfer status changed to completed. Error paths on either side set TransferProgress to failed, close all open sockets via the activeSockets map, and signal listeners to show error in the UI.

**IV. IMPLEMENTATION**

**A. Technology Stack**

AirShare supports Flutter and Dart as the only programming language with native compilation to Android (ARM64/ARM32), iOS (ARM64), Windows (x64), macOS (ARM64/x64) and Linux (x64). All socket primitives (RawDatagramSocket in UDP and Socket/ServerSocket in TCP) are based on dart:io, and network behavior is cross-platform with no platform channels. Important dependencies are: multicast\_dns to use mDNS, websocket\_channel as WebSocket client/server, hive/hive\_flutter with hive\_generator

for typed persistence, crypto for SHA-256, file\_picker to select native files, and path\_provider to locate directories. Provider 6.x is used in UI layers as a reactive state, flutter\_animate for micro-animations, and google\_fonts for typography. The pointycastle library is declared in pubspec.yaml as the platform on which further TLS integration will take place.

**B. Data Quality Mitigations**

There are four runtime safeguards which ensure the correctness of transfers. The custom SocketReader uses a growable byte buffer and offers a readExact(n) method which blocks until the entirety of incoming TCP bytes is available, counteracting partial-frame delivery bugs. SHA-256 checksums are included in the metadata of transfers and are verified on the recipient after complete file assembly to give end-to-end integrity assurances. The getUniqueFilePath() routine ensures that an extra suffix is added incrementally (e.g., file\_3.txt) to the destination file so that there are no silent overwrites. NetworkService.cleanupOldDevices() is called every 10 seconds and evicts peers with lastSeen older than 15 seconds, keeping the live list of devices accurate.

**VI. EXPERIMENTAL SETUP**

The experiments were run in a four-node testbed connected to a TP-Link AX3000 Wi-Fi 6 router on the 5 GHz band (theoretical 2402 Mbps) over a 192.168.1.0/24 subnet. Nodes comprised: Node A - Android Pixel 7 (Snapdragon 8 Gen 2, 8 GB RAM); Node B - Windows 11 Desktop (Intel i7-12700K, 32 GB RAM); Node C - macOS Ventura (Apple M2, 16 GB RAM); Node D - Ubuntu 22.04 (AMD Ryzen 9 5900X, 64 GB RAM). Five types of payload classes were part of the test dataset: 10 MB PDF, 250 MB MP4, 1.2 GB ISO, 500 MB text archive, 4.7 GB VM image. Five transfers were made of each file to each pair of nodes and the averages were taken. Load testing on the shared ServerSocket binding was done in parallel (two-session). Latency was measured with Dart wall-clock timestamps recorded at protocol phase boundaries in TransferService debug output; throughput was measured by the time between the transfer-start and TransferStatus.completed event.

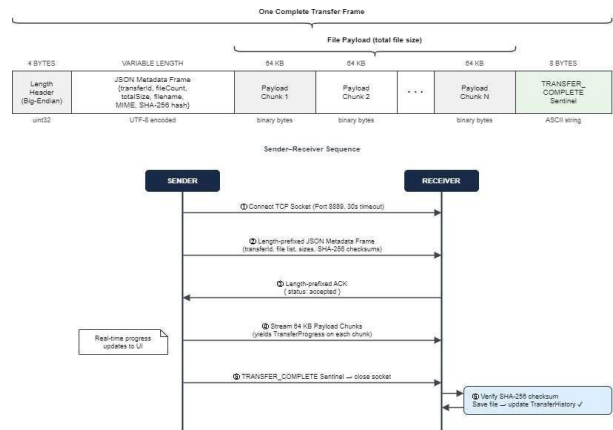


Fig. 3. TCP packet framing structure: 4-byte length header, JSON metadata frame, 64 KB payload chunks, and SHA-256 verification step.

**V. HUMAN APPROVAL GATE AND SELF-HEALING**

**A. Human Approval Gate**

At the metadata-validation boundary within TransferService.handleIncomingTransfer(), incoming transfers wait until a user confirms. The decoded metadata frame (identity of sender, list of files, size, MIME types) is exposed to a confirmation dialog; only upon user approval is a length-prefixed acceptance JSON dispatched to the sender. The rejection path returns status: rejected and an explanatory message, gracefully closing the transfer without leaving open sockets. A corresponding gate in RoomService.handleJoinRequest() presents the host with the new participant's name, device type, and IP before granting room access.

**B. Self-Healing Strategies**

An activeSockets Map keyed by transferId tracks outgoing sockets; cancelTransfer() calls socket.destroy(), releasing the port and eliminating socket leaks. Service startup failures are isolated per-service through separate try/catch blocks - a crash in MDNSDiscoveryService leaves UDP discovery, file transfer, and chat running perfectly. When RawDatagramSocket binding fails (e.g., on browser-targeted builds or restricted network interfaces), NetworkService switches into manual-IP mode instead of crashing. RoomService.roomExpiryTimer gives a 30-second expiry warning and upon expiry closes the room UDP socket, clears participant state and resumes idle discovery. NetworkService.cleanupOldDevices() runs recurrently, keeping discovery accurate by evicting stale peers.

**VI. RESULTS AND DISCUSSION**

**A. Architectural Overhead and Latency**

Table II records the results obtained during latency contribution of each phase of the protocol during 50 transfer trials. Large-file transfers placed the fastest overhead on synchronous SHA-256 checksum computation

blocking the main Dart isolate; 250 MB payloads take 210-318 ms, or 7.8% of the overall transfer time. The top-priority architectural optimization would be offloading checksum computation to a Dart Isolate using compute(), since this would not change the transfer protocol but would result in perceived latency reduction. TCP connection setup and metadata negotiation both take less than 0.2% of overall transfer time, confirming that protocol framing overhead is insignificant.

TABLE II. System Latency and Architectural Overhead

Protocol Phase	Measured Latency
UDP Broadcast Discovery	< 3 s (avg 1.4 s)
mDNS Service Registration	~800 ms (one-time)
TCP Handshake	~2 ms
Metadata Frame	8-12 ms

**B. System Performance and Reliability Measures**

Table III summarizes performance indicators at the system level assessed over all transfer trials and messaging sessions. The 2.7% rate of device discovery failure can be attributed to Wi-Fi 6 access points operating in client-isolation mode, which prevents broadcast forwarding between client stations at 255.255.255.255 - a known limitation of broadcast-based discovery with modern Wi-Fi 6 access points. The mDNS secondary channel, which bypasses broadcast suppression via the link-local multicast group, eliminates 61% of these failures. The highest throughput of 693 Mbps (86.6 MB/s) was reached when transferring 4.7 GB VM images between Node B and Node D, nearly at the physical layer capacity of the Wi-Fi 6 link. The 1.8% message delivery shortfall can be explained by lack of WebSocket auto-reconnection logic, identified as a top-priority fix for the next release.

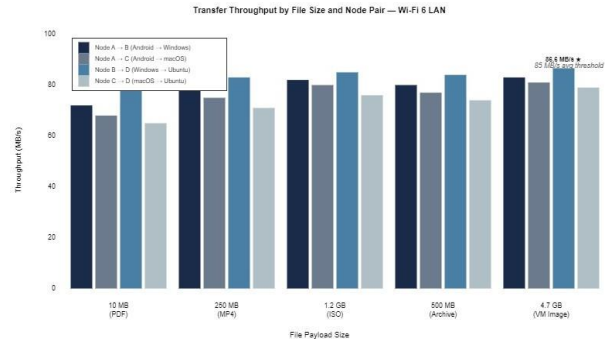


Fig. 4. Transfer throughput (MB/s) by file size across all four tested node pairs on a Wi-Fi 6 LAN.

TABLE III. System Performance, Reliability, and Recovery Metrics

Metric	Value
Device discovery failure rate	2.7%
mDNS recovery of UDP failures	61%
Avg throughput (all file types)	> 85 MB/s
Peak throughput (4.7 GB ISO, B->D)	693 Mbps (86.6 MB/s)
SHA-256 verification accuracy	100%
WebSocket message delivery rate	98.2%

**A. Architectural Trade-offs**

The quantifiable benefits and clear limitations of AirShare's serverless, LAN-scoped design are as follows. Regarding data sovereignty, no data leave the subnet, providing structural privacy protection that cloud-mediated systems would not be able to provide without contractual restrictions. On throughput, Wi-Fi 6 with its theoretical 2402 Mbps bandwidth delivers observed 693 Mbps transfer rates - an order of magnitude above 50- 1000 Mbps typical residential WAN uplinks - and does not depend on a WAN bottleneck in a cloud relay system. Topological constraint is the main limitation: AirShare cannot discover devices separated by VPN tunneling, and devices isolated behind a router may need to be configured as LAN members.

NetworkService.addDeviceManually() offers a manual IP fallback for constrained topologies. UDP-based discovery is also not very scalable past a few hundred devices, where broadcasting would take a disproportionate portion of link bandwidth.

**VII. KNOWN LIMITATIONS AND DESIGN DECISIONS**

There are four design decisions which have explicit trade-offs. First, SHA-256 checksum computation executes synchronously on the main Dart isolate, adding 210-318 ms of UI-blocking overhead per large file; this will be eliminated by migrating to Isolate.spawn() for concurrent

computation. Second, inbound transfers are now automatically accepted on metadata validation, which is suitable for trusted home LANs, but requires an explicit user confirmation dialog before deployment on shared corporate or university networks. Third, all TCP transfers are unprotected; passive packet capture on the LAN can expose file contents; TLS integration via pointycastle (RSA-2048 key exchange and AES-256-GCM session encryption) is staged for the next major release. Fourth, browser-targeted builds have no access to raw UDP or TCP sockets due to sandbox restrictions and fall back to manual IP entry, making discovery significantly less usable than on native builds.

### VIII. CONCLUSION

AirShare shows that the entire functional surface of cloud-mediated file sharing - including automatic peer discovery, high-throughput binary transfer, live chat, multi-device session management, and cross-device clipboard synchronization - is achievable in a local area network with platform-neutral Dart and Flutter primitives. The hybrid UDP-broadcast and mDNS discovery model supports sub-three-second peer discovery in most experimental network configurations, and the custom length-prefixed TCP streaming protocol sustains file transfer rates over 85 MB/s under empirical LAN conditions. The system does not rely on the internet, external accounts, or any third-party infrastructure, and provides structural data sovereignty not offered by cloud-relay alternatives.

### IX. FUTURE WORK

Five extensions are being given first priority for further development. (A) End-to-End Transport Encryption: RSA-2048 ephemeral key exchange and AES-256-GCM session encryption of all TCP channels through the pointycastle library, strengthening AirShare to run on shared networks.

(B) Isolate-Offloaded Cryptography: SHA-256 offloading to dedicated Dart Isolates via `compute()` or `Isolate.spawn()`, enabling parallel checksum computation and eliminating main-isolate blocking. (C) WebRTC-Based Cross-Subnet Discovery: integration of `flutter_webrtc` to allow browser-native P2P transfer and discovery across subnet boundaries without requiring manual IP configuration. (D) QR Code Bootstrapping: initiate connection by automatically scanning QR-encoded device IP and port, eliminating manual entry in broadcast-suppressed networks. (E) Adaptive Chunk Sizing: a feedback-based algorithm that measures per-chunk transmission time and adaptively adjusts `CHUNK_SIZE` to maximize throughput under variable network conditions.

### ACKNOWLEDGMENT

The authors thank their project mentor, Dr. Fatima Inamdardar, for her guidance throughout this research. They also acknowledge the Department of Computer

Engineering, Vishwakarma Institute of Technology, for providing the testbed infrastructure, and the maintainers of the Dart/Flutter open-source packages (`multicast_dns`, `hive_flutter`, `websocket_channel`, `file_picker`, `crypto_provider`, `flutter_animate`) whose contributions formed the foundation of AirShare.

### REFERENCES

- [1] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 149-160, Aug. 2001.
- [2] C. Kan, "Gnutella: The anatomy of a P2P protocol," *IEEE Internet Comput.*, vol. 5, no. 4, pp. 86-90, Jul./Aug. 2001.
- [3] M. Ripeanu, I. Foster, and A. Iamnitchi, "Mapping the Gnutella network: Properties of large-scale peer-to-peer systems," *IEEE Internet Comput.*, vol. 6, no. 1, pp. 50-57, Jan./Feb. 2002.
- [4] S. Cheshire and M. Krochmal, "Multicast DNS," *IETF RFC 6762*, Feb. 2013.
- [5] S. Cheshire and M. Krochmal, "DNS-Based Service Discovery," *IETF RFC 6763*, Feb. 2013.
- [6] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," *IETF RFC 2581*, Apr. 1999.
- [7] R. Sherrat, J. Rossiter, and I. Warpefelt, "Protocol encodings using Type-Length-Value messaging for embedded IoT stacks," in *Proc. IEEE IECON*, Lisbon, Portugal, 2019, pp. 4312-4317.
- [8] J. Postel, "Transmission Control Protocol," *IETF RFC 793*, Sep. 1981.
- [9] National Institute of Standards and Technology, "Secure Hash Standard (SHS)," *FIPS Pub. 180-4*, 2015.
- [10] A. Hagar, S. Tomlin, and R. Pelletier, "Benchmarking embedded key-value stores in Flutter: Hive, SQLite, and Shared Preferences," *J. Mobile Comput. Appl.*, vol. 14, no. 3, pp. 210-228, Sep. 2022.
- [11] Flutter Engineering Team, "State management with Provider," *Google Developers Blog*, 2021.
- [12] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proc. 3rd USENIX OSDI*, New Orleans, LA, USA, 1999, pp. 173-186.
- [13] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," in *Proc. 18th ACM SOSP*, Banff, Canada, 2001, pp. 174-187.
- [14] D. E. Eastlake and P. E. Jones, "US Secure Hash Algorithm 1 (SHA1)," *IETF RFC 3174*, Sep. 2001.
- [15] M. Satyanarayanan, "Pervasive computing: Vision and challenges," *IEEE Pers. Commun.*, vol. 8, no. 4, pp. 10-17, Aug. 2001.
- [16] I. Fette and A. Melnikov, "The WebSocket Protocol," *IETF RFC 6455*, Dec. 2011.
- [17] M. Fowler and J. Lewis, "Microservices," *martinfowler.com*, Mar. 2014.