**Vietnam National University Ho Chi Minh City**
**University of Technology**
**Faculty of Computer Science and Engineering**

SPECIALIZED PROJECT
REPORT

# ENCRYPTED PEER-TO-PEER MESSAGING APP OVER BLUETOOTH MESH NETWORKS

Major: Computer Science

**THESIS COMMITTEE:** HCMUT Computer Science Council
**SUPERVISOR:** Assoc. Prof. Trương Tuấn Anh, PhD
B.Eng. Dương Huỳnh Anh Đức
—**o0o**—
**STUDENT 1:** Trần Trung Vĩnh (2252914)
**STUDENT 2:** Nguyễn Trần Huy Việt (2252906)
**STUDENT 3:** Nguyễn Anh Quân (2252678)
**STUDENT 4:** Lê Phan Nhật Minh (2252478)

Ho Chi Minh City, October 2025

# PROTESTATION

The group hereby certifies that this specialized project is the result of our own research and work, conducted under the supervision of Dr. Truong Tuan Anh and Mr. Duong Huynh Anh Đuc. All research content and results presented in this project are truthful, objective, and have not been previously submitted to any educational institution for the purpose of obtaining an academic degree. All referenced materials are fully and properly cited, including the authors' names, titles of the works, dates, and places of publication. The group also certifies compliance with all regulations regarding the protection of intellectual property rights. Should any violations be discovered, the group accepts full responsibility before the Faculty Board and the Board of Rectors of Ho Chi Minh City University of Technology for the content of this declaration.

# ACKNOWLEDGEMENTS

# Abstract

This project presents the design and development of a decentralized messaging and file-sharing application using a Bluetooth mesh network. The system enables communication in environments where conventional internet or cellular networks are unavailable, such as during natural disasters, protests, large public gatherings, or remote outdoor activities. The application supports peer-to-peer encrypted messaging and secure file transfers, ensuring user privacy and data protection. To enhance reliability and scalability, it utilizes multiple data transmission techniques—including relay, hopping, and flooding—allowing messages to propagate efficiently across devices within the mesh network. This solution demonstrates how local wireless connectivity can be leveraged to create a resilient, infrastructure-independent communication platform for emergency and off-grid scenarios.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

## 1.1 Motivation

Communication has always been a fundamental human need. The desire to exchange information quickly, spontaneously, and across any distance has driven innovation for centuries, from carrier pigeons and handwritten letters to the invention of the telephone, and later, computers that revolutionized global connectivity. Today, modern technology and infrastructure allow us to communicate with anyone around the world in a matter of seconds.

However, there are situations where communication becomes critical but traditional infrastructure is unavailable or unreliable, such as natural disasters where internet and phone lines are destroyed or disrupted by floods, earthquakes, or storms; protests in which governments may deliberately restrict or shut down communication networks to limit coordination; large gatherings such as concerts or festivals where overloaded networks struggle to handle the surge of users; and remote activities like hiking or picnicking in isolated areas without mobile coverage.

In these scenarios, when only smartphones are available and no centralized service exists to connect them, alternative communication methods are essential. Fortunately, modern smartphones support technologies that enable direct device-to-device connections. One such technology is Bluetooth Low Energy (BLE), which provides a foundation for resilient peer-to-peer (P2P) communication.

This project, *"Encrypted Peer-to-Peer Messaging App over Bluetooth Mesh Networks"*, is motivated by the need for a secure, privacy-preserving, and infrastructure-independent communication system. By leveraging BLE and integrating end-to-end encryption, the proposed application enables users to communicate directly, reliably, and privately even in the absence of mobile networks or Wi-Fi. The system is designed to ensure resilience in challenging environments, empowering individuals to stay connected when traditional infrastructure fails.

## 1.2 Objectives

The primary objectives of this project are as follows:

- **To design and develop** a decentralized P2P messaging application using Bluetooth Mesh communication.

- **To ensure security and privacy** through end-to-end encryption, authentication, and integrity protection.

- **To support offline communication** in environments where internet or cellular networks are unavailable or unreliable.

- **To optimize performance and battery efficiency** by using Bluetooth Low Energy (BLE) for background operation.

- **To provide a user-friendly interface** that allows non-technical users to communicate easily without complex setup.

- **To evaluate system scalability and reliability** across different real-world use cases such as disaster zones, remote communities, and crowded public events.

## 1.3  Scope

The scope of this project includes the design, implementation, and testing of a Bluetooth Mesh-based communication system that enables short- to medium-range P2P messaging. The project focuses on local device-to-device networking without reliance on internet or mobile infrastructure.

Key features within the project scope include:

- Bluetooth Mesh message routing, node discovery, and relay functionality.

- End-to-end encryption for all transmitted data.

- Text and small media (e.g., image) message exchange.

- Android-based implementation using Kotlin and BLE APIs.

Out of scope for this project:

- Integration with cloud or server-based systems.

- Long-range or cross-network message relaying via the internet.

- Support for iOS or other operating systems in the current version (planned for future expansion).

The project will thus deliver a fully functional Android prototype demonstrating secure, decentralized, and offline communication over Bluetooth Mesh networks.

## 1.4 Significance of the Project

### 1.4.1 Significance from the Practical Perspectives

From a practical standpoint, the success of this project will deliver:

- **Alternative communication solutions:** Offering users in challenging environments such as disaster zones, remote areas, or crowded events, a reliable means of communication when conventional infrastructure is unavailable.

- **A secure messaging platform:** Through end-to-end encryption, authentication, integrity protection, and identity management, users can be assured that their conversations remain private, trustworthy, and verifiable.

- **Scalable connectivity:** The system is designed to expand seamlessly as the number of users grows. Increased participation will not compromise reliability, ensuring that messages are consistently delivered without loss or delay.

- **Cost-effective deployment:** By leveraging existing smartphones, the solution reduces reliance on expensive satellite phones or specialized equipment, making secure communication more accessible.

### 1.4.2 Significance from the Scientific Perspectives

From a scientific perspective, the success of this project will contribute to:

- **Exploring alternative communication paradigms:** While current systems depend heavily on the internet, radio signals, phone lines, cellular networks, and satellite communication, this project demonstrates the feasibility of infrastructure-independent approaches.

- **Evaluating security and networking protocols:** Serving as a detailed case study, the project will generate experimental results and insights into the effectiveness of various cryptographic and communication protocols in decentralized environments.

- **Advancing trust and identity management research:** Investigating innovative methods for establishing and maintaining trust among peers in distributed networks, with implications for broader applications in secure systems.

- **Informing potential standardization:** The findings could contribute to the development of future standards for secure, decentralized communication protocols, shaping the next generation of resilient communication technologies.

## 1.5    Report structure

This report is organized into nine chapters, structured as follows:

- **Chapter 1: Introduction** outlines the motivation, objectives, scope, and significance of the project.

- **Chapter 2: Background Knowledge** provides the theoretical foundation required to understand the system, including concepts of Bluetooth Mesh networking and P2P communication.

- **Chapter 3: Technology Stack** details the software and tools used to develop the application, including the programming language, development environment, and key libraries.

- **Chapter 4: Related Works** analyzes existing solutions and discusses how this project overcomes their limitations.

- **Chapter 5: Requirements Elicitation** defines the functional and non-functional requirements of the system, identifying the needs of the users and the constraints of the environment.

- **Chapter 6: System Analysis** presents the detailed analysis of the system using Use Case, Activity, and Sequence diagrams to model the system's behavior.

- **Chapter 7: System Design** describes the architectural design, including the 5-layer system architecture and the assignment of software components to hardware.

- **Chapter 8: System Implementation** demonstrates the actual development results, including the user interface and the realization of key features.

- **Chapter 9: Conclusion** summarizes the project's achievements, discusses current limitations, and proposes directions for future improvements.

# 2 Background Knowledge

## 2.1 Decentralized Communication Systems

### 2.1.1 Peer-to-Peer Networks

A P2P network is a decentralized system in which two or more computers are directly connected to exchange data. Unlike traditional client–server models, each computer in a P2P network can function as a client, a server, or both simultaneously. When a device joins the network, it can request data from other peers while also providing data to them, depending on the application in use.

The applications of P2P networking are diverse and far-reaching. Early examples include file-sharing systems such as Napster and BitTorrent, which revolutionized digital content distribution. Today, P2P principles underpin critical technologies such as cryptocurrencies and blockchain platforms. [1]

### 2.1.2 Comparison with Centralized Client–Server Architecture

Both centralized and decentralized architectures present distinct advantages and disadvantages when applied to messaging applications.

**Centralized Client–Server Systems**

**Advantages:**

- *Ease of governance:* Administrators can manage users, enforce policies, and moderate content through a central authority.

- *Simplified implementation:* Features such as chat history synchronization, group management, and data backup are straightforward to implement.

- *Performance optimization:* Central servers can be optimized to provide consistent latency and throughput.

- *Security control:* Uniform security policies are easier to enforce when all traffic passes through a single point.

**Disadvantages:**

- *Single point of failure:* Server outages, attacks, or maintenance can disrupt the entire system.

- *Limited scalability:* Rapid growth in users may cause congestion and degraded performance.

- *Data ownership concerns:* Without end-to-end encryption, service providers may access user data.

- *Censorship and surveillance risks:* Centralized control enables external intervention.

- *Infrastructure dependency:* Requires stable internet connectivity and server infrastructure.

## Decentralized Topologies

### Advantages:

- *Scalability potential:* Each new peer contributes resources to the network.

- *High availability:* The absence of a central server reduces the risk of total system failure.

- *User data ownership:* End-to-end encryption preserves privacy without centralized control.

- *Resilience in restrictive environments:* Difficult to censor or shut down.

- *Cost efficiency:* No requirement for expensive centralized infrastructure.

### Disadvantages:

- *Implementation complexity:* Requires advanced routing, synchronization, and consistency algorithms.

- *Data integrity challenges:* Ensuring reliability across distributed peers is difficult.

- *Performance variability:* Latency and throughput depend on peer availability.

- *Security risks:* Malicious peers may disrupt communication without strong identity management.

- *Resource consumption:* Peer discovery and message relaying increase battery and CPU usage.

## 2.2 Bluetooth Low Energy

Bluetooth Low Energy (BLE) was introduced in 2010 as an alternative to Bluetooth BR/EDR, specifically optimized for low power consumption [2]. This efficiency makes BLE well-suited for battery-constrained devices such as smartphones and IoT systems. On Android, BLE is supported through official APIs that enable device discovery, connection management, and data exchange.

However, when devices communicate via BLE, transmitted data may be accessible to applications running on the same device, introducing potential security risks. To address this concern, the proposed system integrates strong cryptographic mechanisms to ensure that all exchanged data remains confidential and protected against unauthorized access or interception. [3]

## 2.3 Mesh Topology

Mesh networks allow devices to be organized in various configurations depending on system requirements. Commonly used topologies include:

### 2.3.1 Ring Topology

In a ring topology, devices are connected in a circular structure, as illustrated in Figure 1. This topology is effective when devices are physically close and is often used for load balancing and fault tolerance.
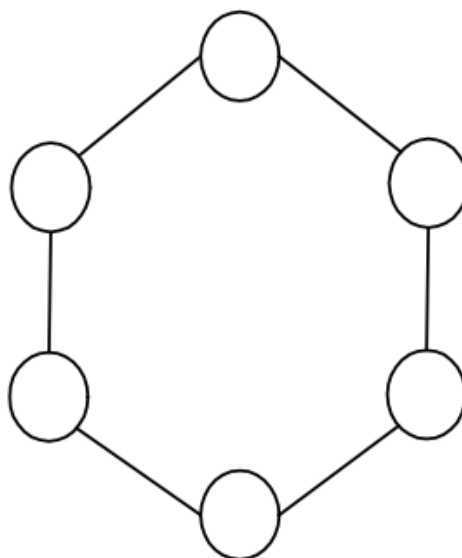


**Figure 1:** *Ring Topology*

### 2.3.2   Hierarchical Topology

Hierarchical topology organizes nodes into a tree-like structure, as shown in Figure 2. This approach is widely used in systems requiring governance and delegation, such as the Domain Name System (DNS).
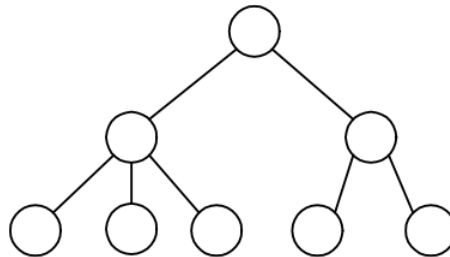


**Figure 2:** *Hierarchical Topology*

### 2.3.3   Decentralized Topology

In a decentralized topology, any machine can connect directly to any other machine, as illustrated in Figure 3. To join the network, a peer typically begins by contacting and advertising itself to a neighboring peer — a process known as bootstrapping. Through bootstrapping, the peer may also receive additional network information depending on the application's implementation. A well-known example of this topology is the early versions of the Gnutella file-sharing system, which relied on peers discovering and connecting to one another without centralized coordination.
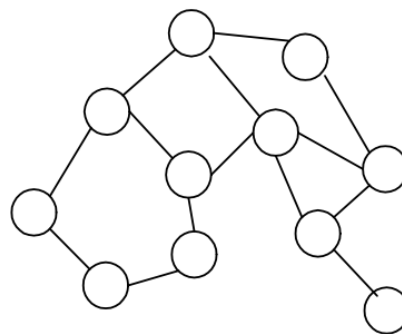


**Figure 3:** *Decentralized Topology*

### 2.3.4   Hybrid Centralized–Decentralized Topology

The hybrid topology introduces a special type of peer known as a super peer. Super peers possess greater processing power and network capacity compared to regular peers. They primarily communicate with other super peers when transferring data, while ordinary peers connect to a super peer to access services. The super peer then handles

processing and data transmission on their behalf. This topology, shown in Figure number, is exemplified by Internet email systems. Mail clients maintain decentralized relationships with specific mail servers, while the servers themselves exchange emails in a decentralized fashion similar to super nodes. [4]
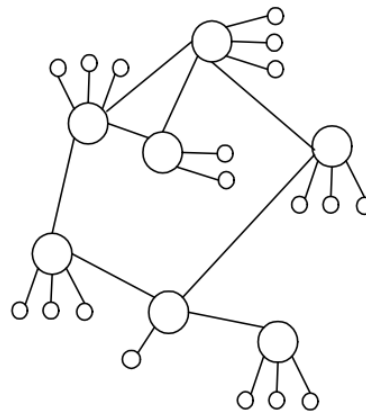


**Figure 4:** *Hybrid Centralized–Decentralized Topology*

## 2.4  Cryptography and Security

### 2.4.1  End-to-End Encryption

End-to-end encryption (E2EE) is a secure communication technique in which a message is encrypted at the sender's device and remains encrypted while traversing the network, including any intermediate nodes. Only the intended recipient, possessing the correct decryption key, can decrypt and read the message. This ensures that no third party — including service providers, network operators, or potential attackers — can access the content of the communication [5]. Over the years, several protocols have been developed to support end-to-end encryption, including:

- **Signal Protocol** Widely adopted in modern messaging applications such as Signal, WhatsApp, and Facebook Messenger (secret chats). It provides strong security guarantees through the Double Ratchet algorithm, forward secrecy, and authentication.

- **TLS in E2EE mode** While Transport Layer Security (TLS) is commonly used to secure client-server communication, it can also be configured to provide end-to-end encryption between communicating parties.

- **Matrix Protocol (Olm/Megolm)** Used in decentralized messaging platforms like Element, supporting both one-to-one and group messaging with end-to-end

encryption.

### 2.4.2   Hash Functions

Hash functions convert arbitrary-length input into fixed-size outputs, enabling integrity verification. Common hash algorithms include SHA-2, SHA-3, and legacy algorithms such as MD5 and SHA-1, which are now considered insecure.

### 2.4.3   Symmetric-Key Cryptography

Symmetric-key algorithms use a single shared key for encryption and decryption. Popular examples include AES, DES (obsolete), and ChaCha20.

### 2.4.4   Asymmetric-Key Cryptography

Asymmetric-key cryptography employs public and private key pairs, enabling secure communication without prior key exchange. Widely used algorithms include RSA, ECC, and DSA.

### 2.4.5   Key Exchange Mechanisms

Key exchange protocols allow secure establishment of shared secrets over insecure channels. Notable mechanisms include Diffie–Hellman (DH), Elliptic Curve Diffie–Hellman (ECDH), and Internet Key Exchange (IKE).

# 3   Related Work

This section reviews existing research and applications that address similar use cases to the proposed system. The surveyed works include applications that utilize peer-to-peer communication, Bluetooth Low Energy (BLE), and security mechanisms in distributed environments.

## 3.1   Bitchat

Bitchat [6], developed by Jack Dorsey, is a decentralized P2P messaging application that employs a dual transport architecture. It supports local Bluetooth mesh networks for offline communication and the Internet-based Nostr protocol for global reach. This design enables operation in both connected environments and scenarios where internet access is unavailable, such as disasters or remote areas.

Notable features of Bitchat include:

- **End-to-End Encryption:** Uses the Noise Protocol Framework for authentication, key exchange, and forward secrecy within the Bluetooth mesh.

- **Local Communication:** Direct P2P messaging within Bluetooth range.

- **Multi-hop Relay:** Supports message forwarding across nearby devices, up to seven hops.

- **Offline Capability:** Functions without internet connectivity.

- **Binary Protocol:** Employs a compact packet format optimized for BLE constraints.

- **Automatic Discovery:** Built-in peer discovery and connection management.

- **Adaptive Power Management:** Battery-optimized duty cycling to reduce energy consumption.

**Advantages**

- *Dual communication modes:* Supports both offline Bluetooth mesh and online communication.

- *Strong security:* End-to-end encryption prevents unauthorized packet inspection.

- *Energy efficiency:* Designed for low power consumption on mobile devices.

**Disadvantages**

- *No peer presence detection:* Lacks mechanisms to determine peer availability beyond discovery.

- *Flooding vulnerability:* Susceptible to packet injection and denial-of-service attacks.

- *Limited identity management:* No robust peer verification in local communication.

- *No file sharing support:* Restricted to text-based messaging.

## 3.2    Briar

Briar [7] is an open-source, P2P messaging application designed for secure and resilient communication. Unlike conventional messaging platforms, Briar does not rely on centralized servers. Instead, messages are synchronized directly between users' devices via Bluetooth, Wi-Fi, Tor over the internet, or removable storage such as USB drives.

Key features of Briar include:

- **End-to-End Encryption:** All communications are encrypted by default.

- **Offline Communication:** Supports Bluetooth and Wi-Fi messaging without internet access.

- **Tor Integration:** Routes traffic through the Tor network to provide anonymity.

- **Decentralized Synchronization:** Data is stored in encrypted form on participating devices.

- **Resilient Transport Options:** Supports data exchange via removable media.

- **Open Source:** Released under the GPL-3.0 license.

**Advantages**

- *Strong privacy protections:* Encryption and Tor mitigate surveillance and censorship.

- *Offline capability:* Operates effectively in disaster or restricted environments.

- *Decentralized design:* Eliminates single points of failure.

- *Cross-platform support:* Available on Android with desktop versions for major operating systems.

- *Activist-oriented design:* Tailored for journalists and high-risk users.

**Disadvantages**

- *Limited discoverability:* Communication is restricted to pre-established contacts.

- *No peer online detection:* Lacks explicit presence indicators.

- *Feature limitations:* No support for voice or video communication.

- *Platform restriction:* Full functionality primarily available on Android.

- *Performance constraints:* Synchronization may be slower than centralized systems.

## 3.3 Peer-to-Peer Networks for Content Sharing

P2P content sharing systems provide foundational insights into decentralized network behavior. One of the earliest and most influential examples is Gnutella, which pioneered large-scale decentralized file sharing.

### 3.3.1 Gnutella v0.4: Early Development

In its early versions, Gnutella v0.4 implemented a fully decentralized mesh network [8]. Key characteristics included:

- **Bootstrapping:** New nodes joined the network by connecting to known peers.

- **Ping and Pong messages:** Used as heartbeat packets to track peer liveness.

- **Flooding and adaptive routing:** Queries were broadcast across the network to locate content.

**Advantages**

- *Guaranteed content discovery:* Flooding ensured that available content could eventually be found.

**Disadvantages**

- *Bottlenecks:* Weak peers could act as cut vertices, fragmenting the network.

- *Network overload:* Excessive control traffic reduced scalability and throughput.

### 3.3.2 Gnutella v0.6: Later Development

To address scalability limitations, Gnutella v0.6 [9] introduced a hybrid decentralized–centralized topology:

- **Ultra nodes:** High-capacity peers responsible for most query processing.

- **Normal peers:** Connected only to ultra nodes for data exchange.

- **Peer-to-peer among ultra nodes:** Reduced flooding and improved routing efficiency.

This hybrid approach preserved decentralization while significantly improving scalability, reducing network overhead, and mitigating bottleneck risks.

## 3.4 Research Summary

### 3.4.1 Overall Insights

Based on the analysis of existing applications and systems, several key insights emerge:

- Few applications provide reliable messaging over Bluetooth-based P2P networks.

- P2P innovation has shifted from early file-sharing systems toward blockchain, decentralized finance, decentralized social platforms, and secure messaging.

### 3.4.2 Opportunities and Challenges

**Opportunities**

- *Dual communication modes:* Combining offline Bluetooth mesh and online communication fills a notable gap.

- *Shift in P2P innovation:* Advances in encryption and identity management can be leveraged.

- *Resilience in restricted environments:* Suitable for disaster recovery and censorship-heavy regions.

- *Privacy and security demand:* Increasing user concern over surveillance creates strong demand.

- *Niche market potential:* Activists, journalists, and low-connectivity communities form a viable user base.

## Challenges

- *Unreliable connectivity:* Bluetooth suffers from limited range, bandwidth, and interference.

- *Limited adoption:* User trust and acceptance may be difficult to achieve.

- *Scalability issues:* Flooding and routing overhead remain significant challenges.

- *Security and trust management:* Identity verification and attack mitigation are non-trivial.

- *Feature limitations:* Rich media, file sharing, and presence detection are harder to implement.

- *Innovation gap:* Fewer active research communities focus on Bluetooth-based messaging systems.

# 4 Requirements Elicitation

## 4.1 Functional requirements

**User Requirements**

U1. **Bootstrapping:** Upon launching the application, the system shall automatically initiate device discovery and implicitly integrate nearby compatible devices into the mesh network without requiring manual configuration.

U2. **Offline Chat:** Users shall be able to chat without an Internet connection.

U3. **Peer Selection:** Users shall be able to choose who to chat with when the app detects nearby peers.

U4. **User Profile:** Users shall be able to choose and change their nickname and profile picture.

U5. **Favorites and Blocking:** Users shall be able to mark another person as a favorite or block them.

U6. **Broadcast Messaging:** Users shall be able to send broadcast messages to nearby users (e.g., emergency alerts or announcements).

U7. **Reply and Quote:** Users shall be able to reply to or quote specific messages.

U8. **Chat History:** The system shall support chat history backup and export.

U9. **Notifications:** The system shall provide chat notifications.

U10. **Mute Options:** Users shall be able to mute group or peer notifications.

U11. **Tutorial/Onboarding:** The system shall provide a tutorial or onboarding flow for first-time users.

U12. **Appearance:** The system shall support dark mode and accessibility options.

U13. **Relay Functionality:** The user's device must help forward messages to other peers if it is not the destination.

U14. **Peer Detection:** The system shall continuously monitor and maintain awareness of all active devices within the mesh network.

U15. **Battery Saving:** The system shall provide a battery-saving mode.

U16. **File Sharing:** The system shall support file sharing between users.

**Group Requirements (Optional for this semester)**

**G1.** Users shall be able to create a group chat with multiple peers.

**G2.** The group chat creator is a **Leader** and can appoint $n-2$ more **Co-Leaders**, where $n$ is the total number of members.

**G3.** Leaders and Co-Leaders can swap roles with members who have lower roles.

**G4.** Leaders and Co-Leaders can set group chat status as:

- **Open** – anyone can join
- **Restricted** – users must request to join
- **Closed** – only whitelisted users can join

**G5.** All peers can join or request to join groups and can also leave groups.

**G6.** Join/Leave actions shall be displayed in the group.

**G7.** Leaders and Co-Leaders can:

- Add or remove members
- Change the group's name and profile picture

**G8.** The system shall support polls or voting for group decisions.

**G9.** The system shall support group event scheduling (e.g., meetups in remote areas).

## 4.2   Non-Functional Requirements

**Security**

**S1.** All messages and data must be encrypted end-to-end using a secure protocol.

**S2.** The system must ensure message and data traceability by reliably associating each transmission with its originating sender.

**S3.** The system shall guarantee data integrity by preventing unauthorized modification or tampering.

**S4.** The system shall support self-destructing messages or message expiration after 24 hours for sensitive communication.

### Scalability

**SC1.** The system must be able to handle a network of 100–500 nodes.

### Reliability

**R1.** The system must detect whether data has been successfully transferred.

**R2.** The system must synchronize data between peers once both are online again.

### Performance

**P1.** Peer discovery should complete within 5 seconds under typical mesh conditions.

**P2.** Message delivery latency should be under 1 second for nearby peers.

### Archivability

**A1.** The system shall persistently store user-associated messages on the user's device until explicitly deleted or modified by user-defined retention settings.

**A2.** Messages not directly associated with the user shall be temporarily cached for up to 24 hours and automatically forwarded upon detection of the recipient's availability. Once delivered, cached messages shall be deleted automatically.

### Efficiency

**E1.** The app must minimize battery usage during idle mesh scanning.

**E2.** Each message shall be limited to 1024 characters.

**E3.** Bluetooth transmissions should be optimized to reduce unnecessary chatter.

# 5 System Analysis

## 5.1 Use Case Diagrams

### 5.1.1 Whole system

The Bluetooth Mesh Chat App allows users to communicate with each other through Bluetooth connections without relying on the internet. The system consists of two main actors: the User (Actor) and the Device. The user interacts directly with the application to perform actions such as scanning for nearby devices, selecting who to chat with, and sending messages. Meanwhile, devices in the network act as peers or relays, helping transmit messages across multiple nodes to maintain connectivity in the mesh network.
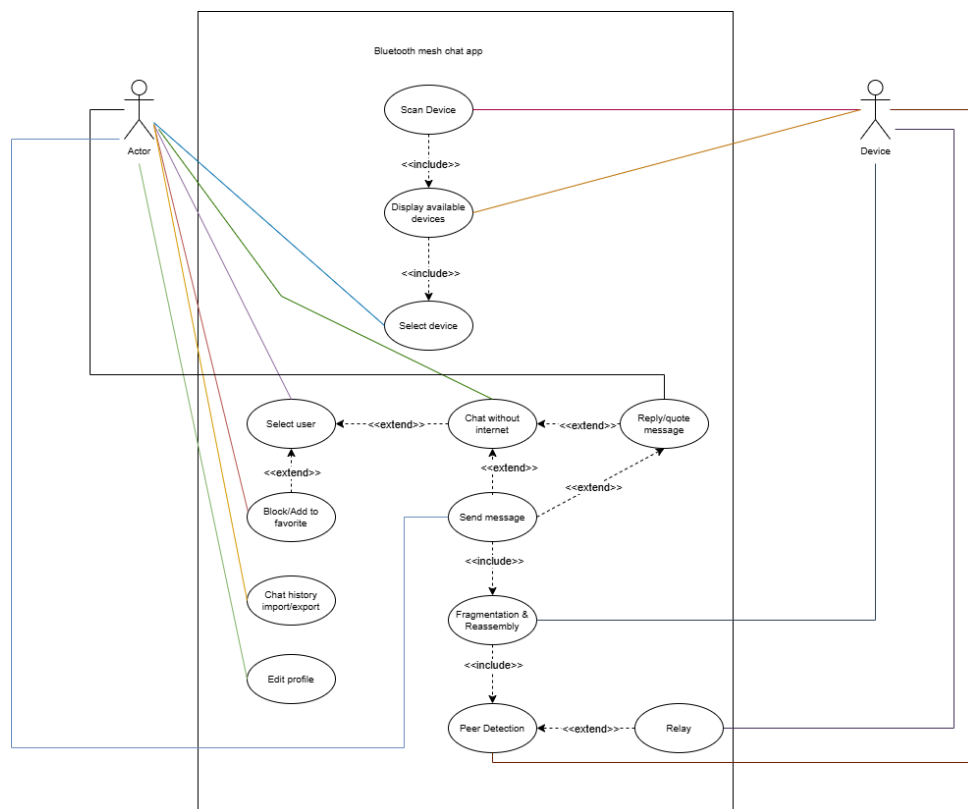


**Figure 5:** *Usecase diagram: Whole system*

The process usually begins when the user scans for nearby devices. This use case includes displaying a list of available devices and selecting one to connect with. Once connected, the user can initiate communication through the Chat without internet feature, which is the core function of the system. This use case extends other related functions such as sending messages, replying to or quoting messages, and selecting users. It represents the ability to exchange messages directly between devices without requiring mobile data or

Wi-Fi.

When the user chooses to send a message, the system ensures efficient and reliable delivery through two important processes: Fragmentation and Reassembly, and Peer Detection. Fragmentation and reassembly handle the breaking down of large messages into smaller packets suitable for Bluetooth transmission and reassembling them correctly on the receiving device. Peer detection identifies available mesh network nodes that can receive or forward messages. If a direct connection is not possible, the system uses the Relay function, where another device helps forward the message to its destination, maintaining the mesh network's multi-hop communication feature.

Beyond chatting, the user can perform several additional actions to enhance their experience. The Select user use case allows choosing specific users to chat with and can extend to features such as Block/Add to favorites, enabling users to manage their contact preferences. The app also supports Reply/quote message, which allows referencing previous messages in a conversation for clarity. Furthermore, users can edit their profiles, modifying information such as their name or visibility settings, and they can import or export chat history to save important conversations or transfer data between devices.

### 5.1.2 Bootstrapping
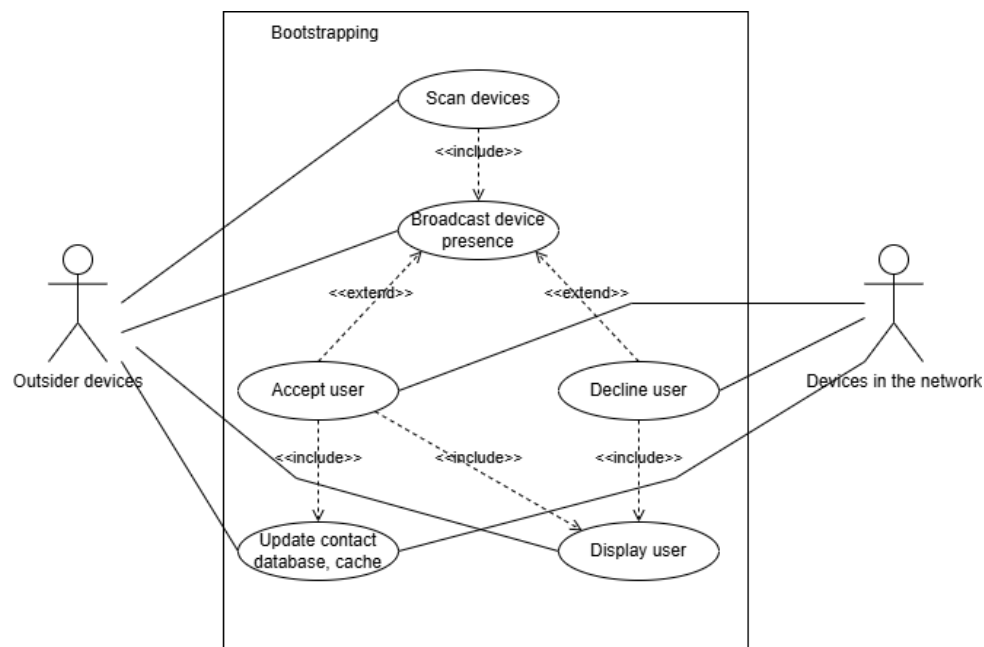


**Figure 6:** *Usecase diagram: Bootstrapping*

The Bootstrapping system begins when an outsider device initiates a scan for devices to find other nodes in its surroundings. This scanning process includes the Broadcast device presence use case, where the outsider device announces its availability to nearby

devices. This broadcast allows devices already in the network to recognize and potentially connect with the outsider device. The relationship here is mutual — while the outsider device scans and broadcasts its presence, the devices in the network also listen for new broadcasts to maintain awareness of potential peers.

Once the broadcast is received, the devices in the network can respond in one of two ways: Accept user or Decline user. These two actions represent the decision-making step that determines whether the outsider device will be integrated into the mesh network. Both of these use cases extend from Broadcast device presence, meaning they are optional outcomes triggered depending on the system's or user's decision.

If the device is accepted, the system proceeds to update the contact database and cache. This use case ensures that the newly accepted device is stored in the network's local memory so that future communication can occur smoothly without re-scanning or re-broadcasting. On the other hand, if the device is declined, the system still executes display user, which allows existing users or devices to view the detected outsider device without integrating it into the network. Both "accept" and "decline" include these supportive use cases to ensure proper record-keeping and user visibility.

### 5.1.3 Open single communication channel
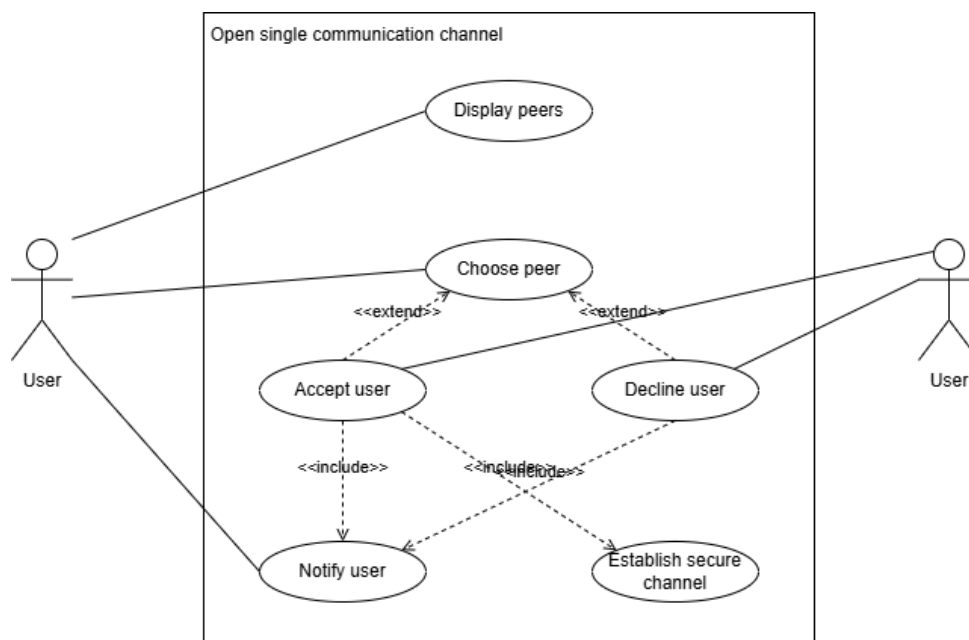


**Figure 7:** *Usecase diagram: Open single communication channel*

When a user performs the Display peers use case, the process begins. This step allows the initiating user to view nearby or available peers within Bluetooth range or reachable via

the mesh network. Once peers are displayed, the user proceeds to choose a peer, selecting a specific device or person they wish to communicate with. This marks the start of an attempt to open a single, direct communication channel between the two devices.

After choosing a peer, the system transitions to a decision phase involving two possible actions: Accept user or Decline user. These two use cases extend from Choose peer, as they are conditional outcomes that depend on how the receiving user responds to the connection request. If the receiving user accepts the request, the system proceeds with the Establish secure channel use case, which ensures that the communication link is encrypted and authenticated to prevent unauthorized access or data interception. This step is essential in maintaining privacy and trust between users in a Bluetooth-based mesh network.

Whether the request is accepted or declined, the system also includes the Notify user use case. This function ensures that the initiating user is informed about the outcome of their connection request — whether the communication channel has been successfully established or denied. Both "accept user" and "decline user" include this notification feature, ensuring that users receive consistent feedback on their connection attempts.

### 5.1.4   Process message

When a device receives a packet from another peer, the process begins with the Receive packet use case. This represents the system's initial handling of incoming data. Immediately after receiving, the system executes Decrease packet TTL (Time To Live) and Check packet TTL, which are both included in use cases. The TTL value determines how many hops a packet can make before being discarded, preventing infinite circulation within the mesh. If the TTL reaches zero, the system executes the Drop packet use case, ending the process for that data unit.

If the packet remains valid (TTL > 0), the system proceeds to <Check packet> information. This step involves inspecting the packet's metadata — such as sender ID, destination ID, and sequence number — to determine whether it should be processed or ignored. Several optional or conditional actions extend from this stage. For instance, the <Check block list> ensures that packets from blocked or banned devices are filtered out. Check packet cache prevents duplicate message forwarding by verifying whether the packet has already been processed or forwarded recently. Both checks help maintain security and reduce unnecessary traffic in the mesh network.
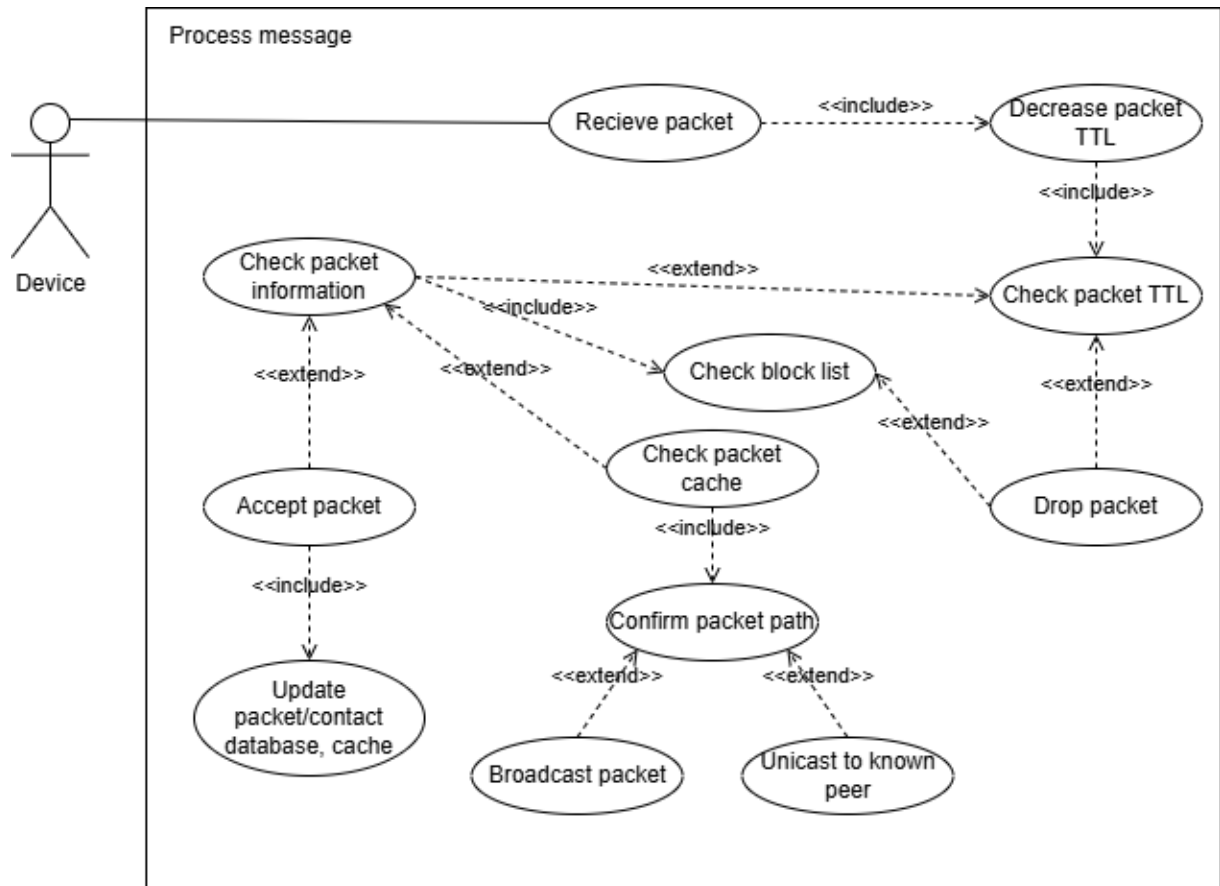
**Figure 8:** *Usecase diagram: Process message*

Once the packet passes these checks, the system moves to Accept packet, which indicates that the packet is legitimate and ready to be processed or forwarded. Accepting the packet includes <Update packet/contact database>, cache, allowing the system to store new peer information and maintain an updated record of recently seen packets and nodes. This helps optimize future routing decisions.

After the packet is accepted, the device must determine how to deliver or relay it. This is handled through the Confirm packet path use case, which validates the intended destination. Depending on the routing outcome, the system may <Broadcast packet> — forwarding it to multiple peers to reach unknown destinations — or <Unicast to known peer>, sending it directly to a specific device if the path is already known. Both of these cases extend from the <Confirm packet path> since they are conditional forwarding strategies determined by the mesh routing logic.

### 5.1.5 Send message
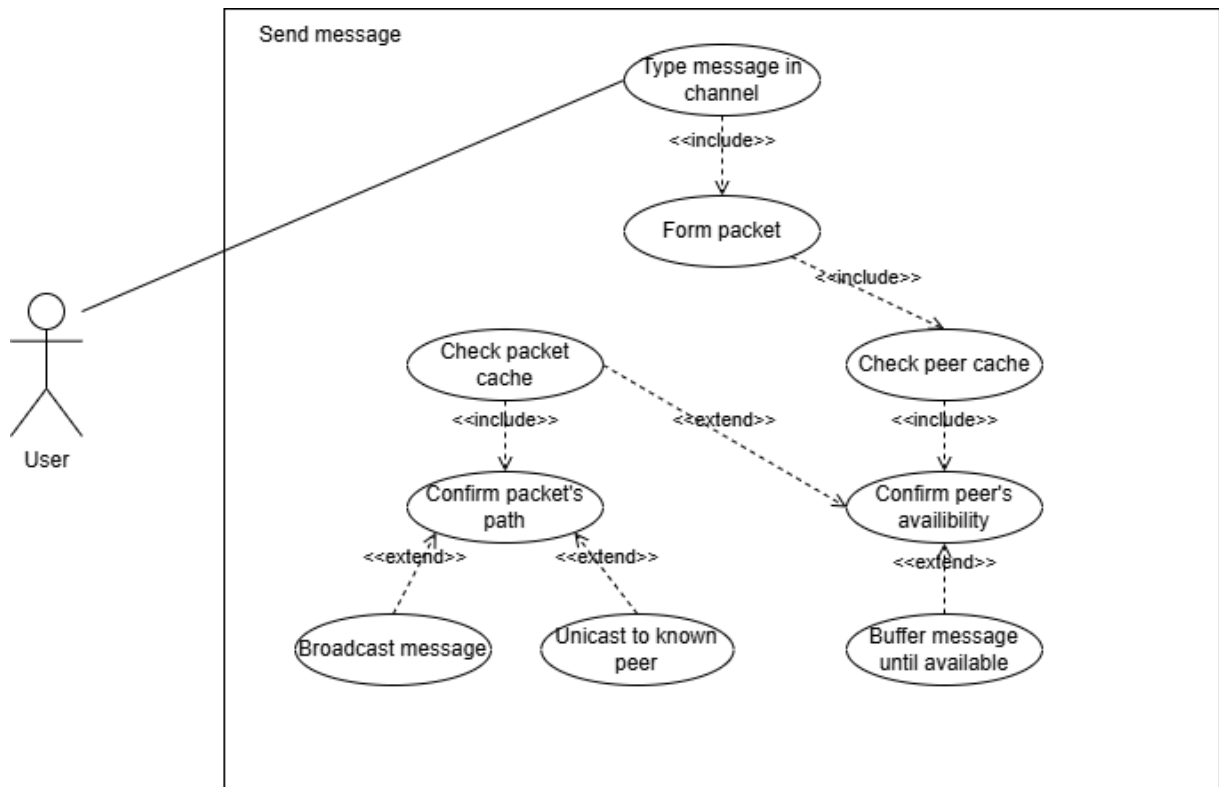


**Figure 9:** *Usecase diagram: Send message*

The process starts when the user initiates the <Type message in the channel> use case. This represents composing a message in the chat interface. Once the user types a message, the system automatically proceeds to <Form packet>, which packages the message content along with necessary metadata such as source ID, destination ID, timestamp, and sequence number. This use case is included because packet formation is always required before any transmission can occur.

After the packet is formed, the system performs two parallel verification checks — Check packet cache and Check peer cache. Checking the packet cache ensures that the message is not a duplicate of a recently sent or processed packet, preventing redundancy and network congestion. At the same time, checking the peer cache verifies that the target peer or device is known and that its address and status are stored locally. Both of these are included in use cases, as they are standard procedures during message preparation.

Next, the system proceeds to confirm the packet's routing and the recipient's readiness. The Confirm packet's path use case determines whether a valid route exists for message delivery. If a known path is available, the message may be <unicasted to known peer>, representing direct one-to-one delivery within the mesh. If no known route exists or if the destination is uncertain, the system may instead broadcast the message, extending its

reach to all nearby nodes, which can forward it until it reaches the intended peer. Both <Broadcast message> and <Unicast to known peer> are extended use cases since they are conditional on the outcome of path confirmation.

In parallel, the <Confirm peer's availability> use case checks whether the target peer is currently reachable in the network. This extends from <Check peer cache> and helps the system avoid transmission failures. If the peer is temporarily unavailable (for example, disconnected or out of range), the system triggers the <Buffer message> until the available use case, which temporarily stores the message and retries sending when the peer reconnects. This ensures reliability and prevents message loss even in a dynamic mesh environment.

### 5.1.6 Fragmentation
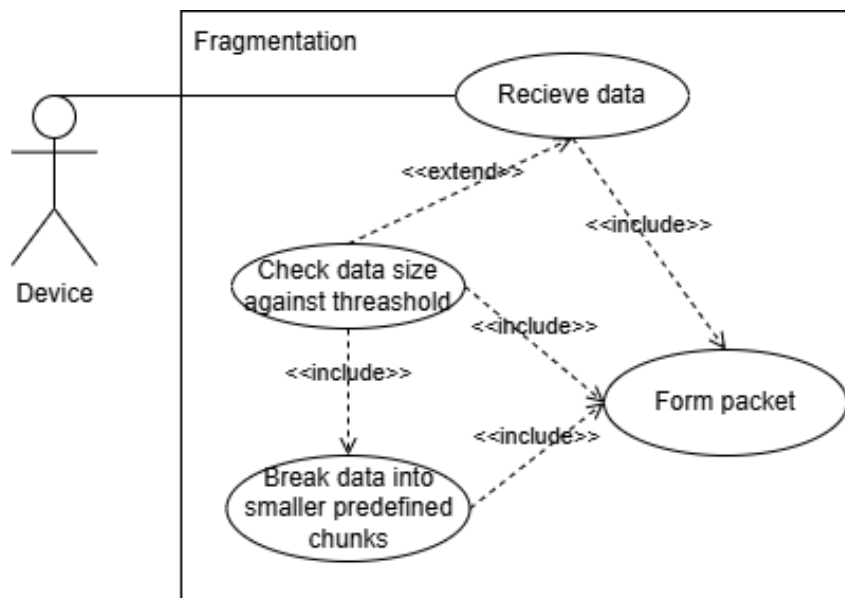


**Figure 10:** *Usecase diagram: Fragmentation*

The main actor here is the device, which could be any node in the mesh network (either a sender or a relay). The process begins when the device <Receives data>, such as a message or file, that needs to be prepared for transmission. This is the primary use case of the diagram.

Once the data is received, the system automatically triggers the Check data size against threshold use case. This step compares the incoming data's size with the system's predefined maximum transmission unit (MTU). If the data size exceeds this threshold, the system must perform fragmentation to ensure compatibility with the mesh network's transmission constraints. This relationship is modeled as an include, since checking data size is an essential step every time data is received.

---

If the threshold check determines that fragmentation is needed, the Break data into smaller predefined chunks use case is invoked. This represents dividing the original message into smaller pieces that conform to the mesh packet size limit. Each chunk is then assigned an identifier and sequence number to allow proper reassembly at the receiver's end. This step is also shown as an include, as it is a required sub-process once large data is detected.

After fragmentation or if the data already fits within the size limit, the process proceeds to Form packet, which organizes the (possibly fragmented) data into transmission-ready packets that include addressing, integrity checks, and sequence metadata. This ensures that the resulting packets can be efficiently propagated through the mesh. The Form packet use case is central to this process and is therefore included in both <Receive data> and <Break data into chunks>.

### 5.1.7 Form packet
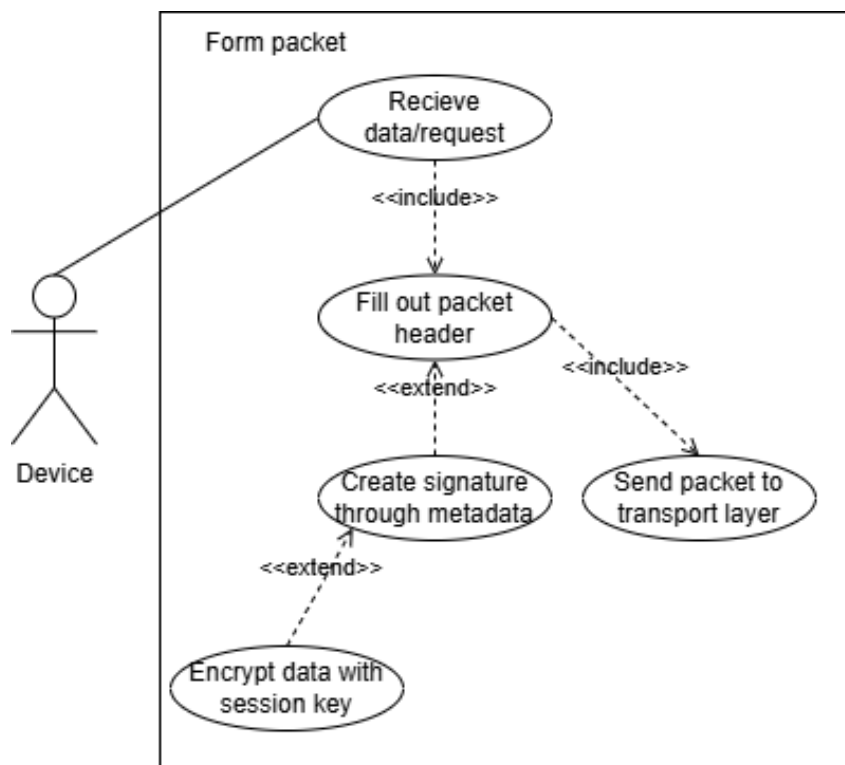


**Figure 11:** *Usecase diagram: Form packet*

The process begins when the device <Receives data/request>, which could originate from higher-level processes such as user input, fragmented data chunks, or control messages. This initial step triggers the packet formation process and is marked as an include relationship since it is always necessary for forming a packet.

After receiving the data, the system proceeds to <Fill out packet header>, where it constructs the essential components of the packet's metadata — such as source and desti-

nation addresses, message type, sequence number, and other routing-related information. This header ensures that each packet can be properly routed through the mesh and identified upon reception.

The <Create signature through metadata> use case is shown as an extension of the header-filling step. This indicates that once the header is completed, the system may generate a digital signature or integrity check based on the metadata. This mechanism ensures authenticity and prevents tampering during transmission across multiple mesh hops. To further enhance security, the <Encrypt data with session key> use case is introduced as another extended relationship. This step ensures that the payload is encrypted before transmission using a shared session key between peers. This encryption maintains confidentiality, ensuring that only authorized nodes can read the message contents even though the mesh operates as a distributed network.

Once the packet has been properly constructed, signed, and encrypted, it is ready to be transmitted. The Send packet to transport layer use case — connected via an include relationship — handles forwarding the completed packet to the underlying transport layer. This layer then manages the packet's transmission across the mesh network, whether via unicast or broadcast, depending on the routing configuration.

## 5.2 Use Case Description

### 5.2.1 Use Case 1: Boostrapping

| Use Case ID | UC-01 |
|---|---|
| Use Case Name | Bootstrapping - Device Network Connection |
| Actors | • Outsider Device (Primary Actor)<br><br>• Devices in the Network (Secondary Actor) |
| Description | The bootstrapping process where an outsider device discovers, authenticates, and joins the Bluetooth mesh network. |
| Preconditions | • Bluetooth is enabled on both devices<br><br>• Network devices are broadcasting<br><br>• Application is installed |

| Postconditions | |
|---|---|
| | • New device is provisioned and can communicate securely<br><br>• Contact database is updated on all devices<br><br>• User appears in network member list |
| **Main Flow** | 1. Outsider device scans for available networks<br><br>2. Network devices broadcast their presence<br><br>3. Outsider device displays list of discovered networks<br><br>4. User selects network and sends join request<br><br>5. Network device validates authentication credentials<br><br>6. Network device accepts request and sends provisioning data<br><br>7. Outsider device receives encryption keys and network configuration<br><br>8. All network devices update their contact database<br><br>9. System displays updated user list on all devices |
| **Alternative Flow** | **3a.** No devices found<br><br>1. System displays "No devices found"<br><br>2. User can retry scanning<br><br>**5a.** Authentication fails<br><br>1. Network device declines the request<br><br>2. System sends rejection notification<br><br>3. Use case ends with failure |

### 5.2.2 Use Case 2: Open single communication channel

| Use Case ID | UC-02 |
|---|---|
| Use Case Name | Open Single Communication Channel |
| Actors | User (Sender), User (Receiver) |
| Description | Establishing a secure one-to-one communication channel between two users in the mesh network. |
| Preconditions | • Both users are provisioned in the network<br><br>• Users are within mesh network range |
| Postconditions | • Secure communication channel established<br><br>• Both users can exchange encrypted messages<br><br>• Connection status displayed to both users |
| Main Flow | 1. Sender opens the application<br><br>2. System displays list of available peers<br><br>3. Sender chooses peer from the list<br><br>4. System sends connection request to receiver<br><br>5. Receiver's device receives connection request<br><br>6. Receiver accepts the connection request<br><br>7. System establishes secure encrypted channel<br><br>8. System notifies both users of successful connection<br><br>9. Communication channel is ready for messaging |

| Alternative Flow | **2a.** No peers available |
|---|---|
| | 1. System displays "No peers found" |
| | 2. System suggests scanning for new devices |
| | 3. Use case ends |
| | **6a.** Receiver declines connection |
| | 1. Receiver selects decline option |
| | 2. System sends decline notification to sender |
| | 3. System displays "Connection declined" message |
| | 4. Use case ends without establishing channel |
| Exception Flow | **4a.** Connection timeout |
| | 1. No response from receiver within 30 seconds |
| | 2. System notifies sender of timeout |
| | 3. Use case ends |
| | **7a.** Channel establishment fails |
| | 1. System retries establishing secure channel |
| | 2. If retry fails, display error message |
| | 3. Use case ends |

### 5.2.3   Use Case 3: Process Message

| Use Case ID | UC-03 |
|---|---|
| Use Case Name | Process Message |
| Actor | Device (System) |
| Description | Receiving, validating, and routing incoming packets through the mesh network. |
| Preconditions | Device is part of mesh network, packet received from another peer. |
| Postconditions | Packet delivered to destination or forwarded to next hop. |

| Main Flow | |
|---|---|
| | 1. Device receives packet from mesh network |
| | 2. System decreases packet TTL (Time-To-Live) |
| | 3. System checks packet TTL validity |
| | 4. System checks packet information (headers, integrity) |
| | 5. System checks packet cache for duplicates |
| | 6. System confirms packet routing path |
| | 7. System accepts packet for processing |
| | 8. System updates packet/contact database cache |
| | 9. System unicasts to known peer or forwards to destination |
| **Alternative Flow** | **4a.** Sender is in block list |
| |    1. System checks block list |
| |    2. System drops packet |
| |    3. Use case ends |
| | **5a.** Packet is duplicate (found in cache) |
| |    1. System drops packet to prevent loop |
| |    2. Use case ends |
| | **7a.** Routing path unknown |
| |    1. System broadcasts packet to mesh |
| |    2. Use case ends |

| | |
|---|---|
| **Exception Flow** | **3a.** TTL expired (reaches 0) |
| | 1. System drops packet |
| | 2. Use case ends |
| | **4b.** Packet corrupted or invalid |
| | 1. System drops packet |
| | 2. Use case ends |

### 5.2.4 Use Case 4: Send Message

| | |
|---|---|
| **Use Case ID** | UC-04 |
| **Use Case Name** | Send Message |
| **Actor** | User |
| **Description** | Sending encrypted message to peer(s) through the mesh network. |
| **Preconditions** | User is connected to mesh network, communication channel established. |
| **Postconditions** | Message delivered to recipient(s), delivery status confirmed. |
| **Main Flow** | |
| | 1. User types message in channel |
| | 2. System forms encrypted packet |
| | 3. System checks packet cache to avoid duplication |
| | 4. System checks peer cache for recipient availability |
| | 5. System confirms peer's availability |
| | 6. System confirms optimal routing path |
| | 7. System unicasts message to known peer |

| Alternative Flow | **5a.** Peer not immediately available |
|---|---|
| | 1. System buffers message until peer available |
| | 2. Message delivered when peer comes online |
| | **6a.** Direct path unavailable |
| | 1. System broadcasts message through mesh |
| | 2. Message routed via intermediate peers |

### 5.2.5 Use Case 5: Fragmentation

| Use Case ID | UC-05 |
|---|---|
| Use Case Name | Fragmentation |
| Actor | Device (System) |
| Description | Breaking large data into smaller chunks to fit Bluetooth mesh packet size limits. |
| Preconditions | Data ready to transmit, packet size threshold defined. |
| Postconditions | Data fragmented into multiple packets ready for transmission. |
| Main Flow | 1. Device receives data to transmit |
| | 2. System checks data size against threshold |
| | 3. System breaks data into smaller predefined chunks |
| | 4. System forms packets from each chunk |
| Alternative Flow | **2a.** Data size within threshold |
| | 1. System forms single packet directly |
| | 2. Skip fragmentation step |
| | 3. Use case ends |

### 5.2.6 Use Case 6: Form Packet

| Use Case ID | UC-06 |
|---|---|

| Use Case Name | Form Packet |
|---|---|
| **Actor** | Device (System) |
| **Description** | Creating network packet with headers, encryption, and metadata for transmission. |
| **Preconditions** | Data/request ready, session keys available. |
| **Postconditions** | Complete encrypted packet ready for transport layer. |
| **Main Flow** | 1. Device receives data or request<br><br>2. System fills out packet header (sender, receiver, timestamp, sequence)<br><br>3. System encrypts data with session key<br><br>4. System creates signature through metadata (integrity check)<br><br>5. System sends packet to transport layer |

## 5.3 Activity Diagrams

The app continuously scans nearby devices over Bluetooth and reads each peer's basic info from advertisements/GATT while also broadcasting its own presence. When the user accepts a peer, the app fetches that peer's network details (e.g., capabilities/keys), then updates a local peer cache with a fresh timestamp/TTL. If the user declines, the temporary record is discarded. The UI then displays the current list of discovered users. This loop repeats so the list stays up-to-date as devices appear or leave.
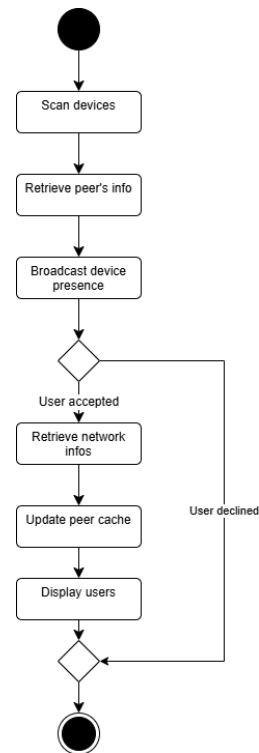


**Figure 12:** *Activity diagram: Bootstrapping*
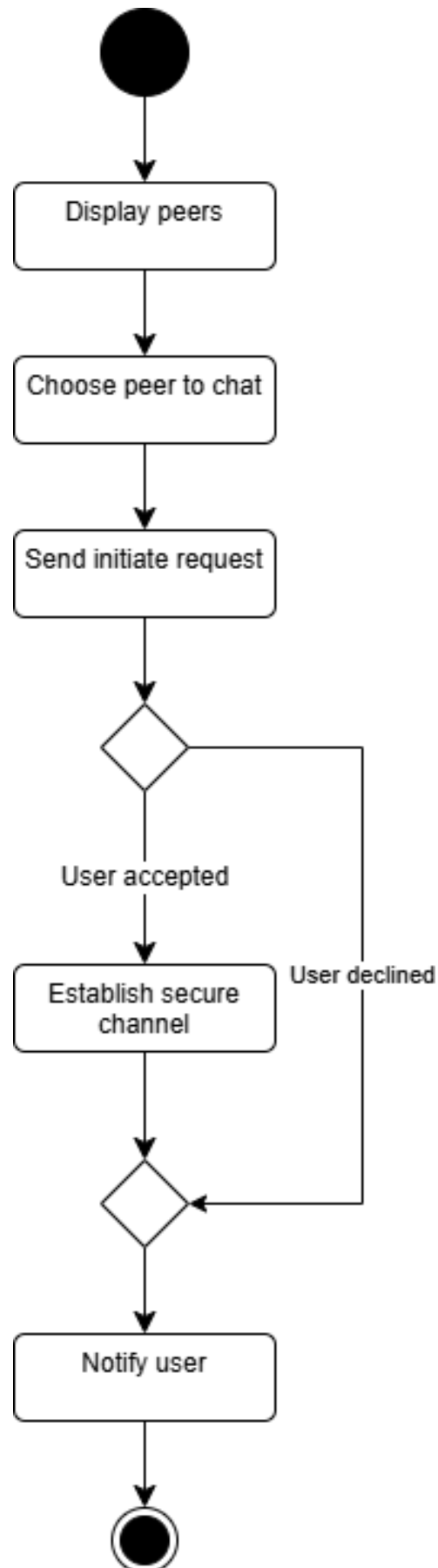
**Figure 13:** *Activity diagram: Open communication channel*
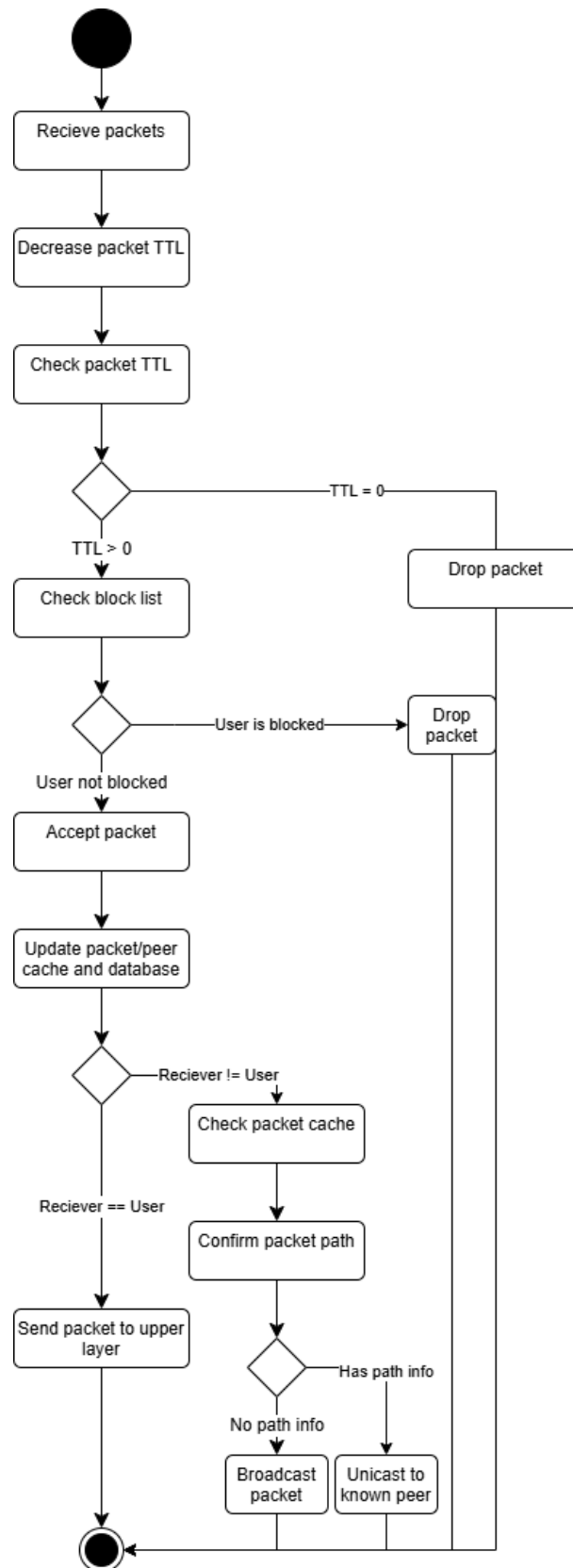
**Figure 14:** *Activity diagram: Process message*
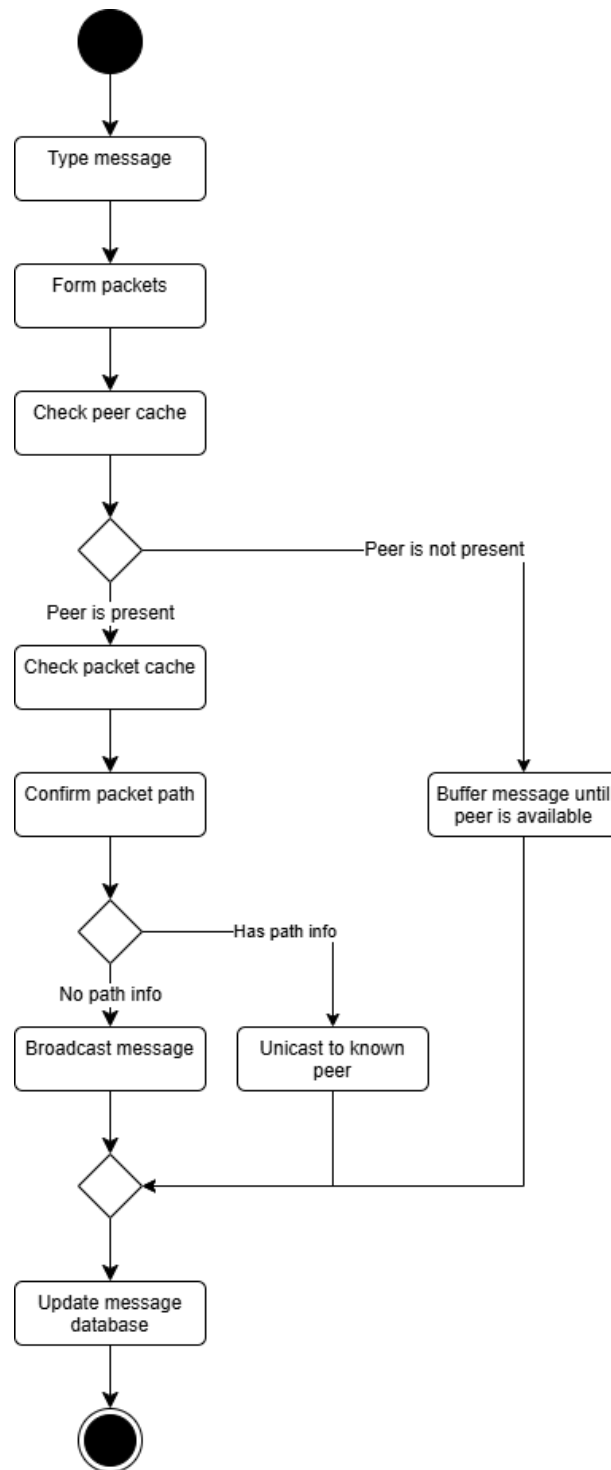
**Figure 15:** *Activity diagram: Send message*
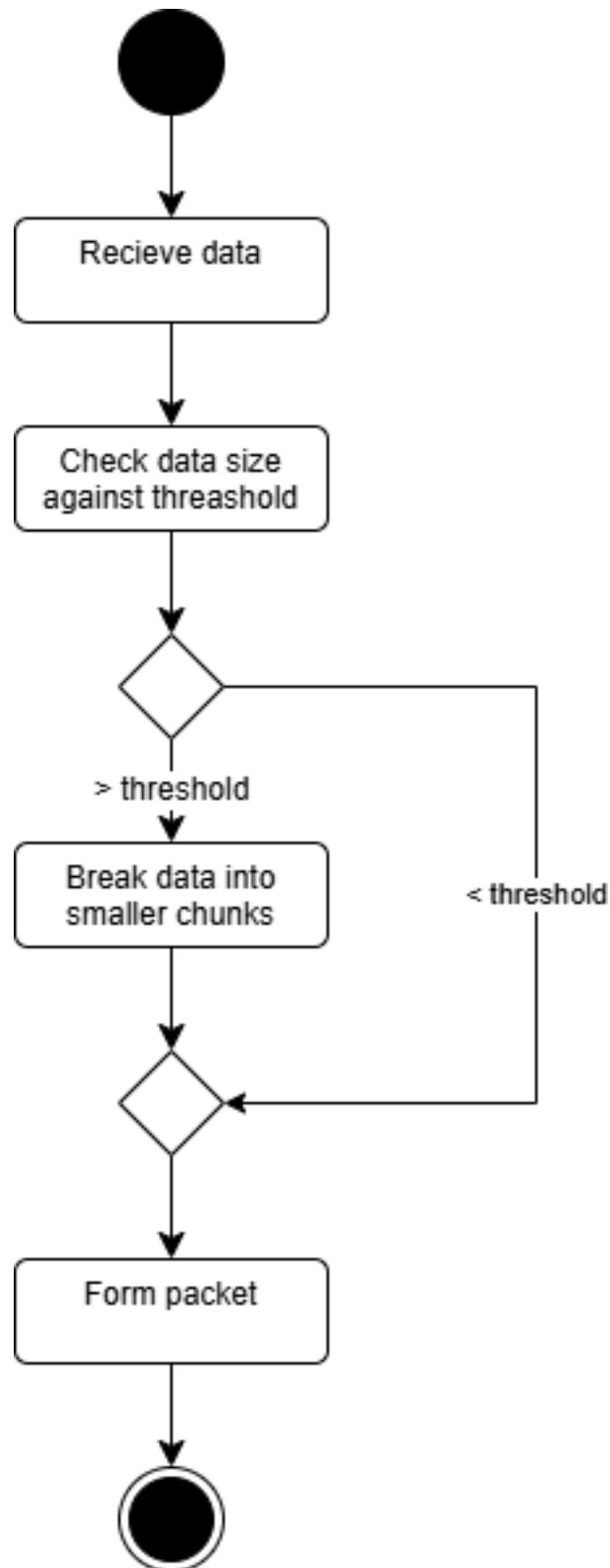
**Figure 16:** *Activity diagram: Fragmentation*
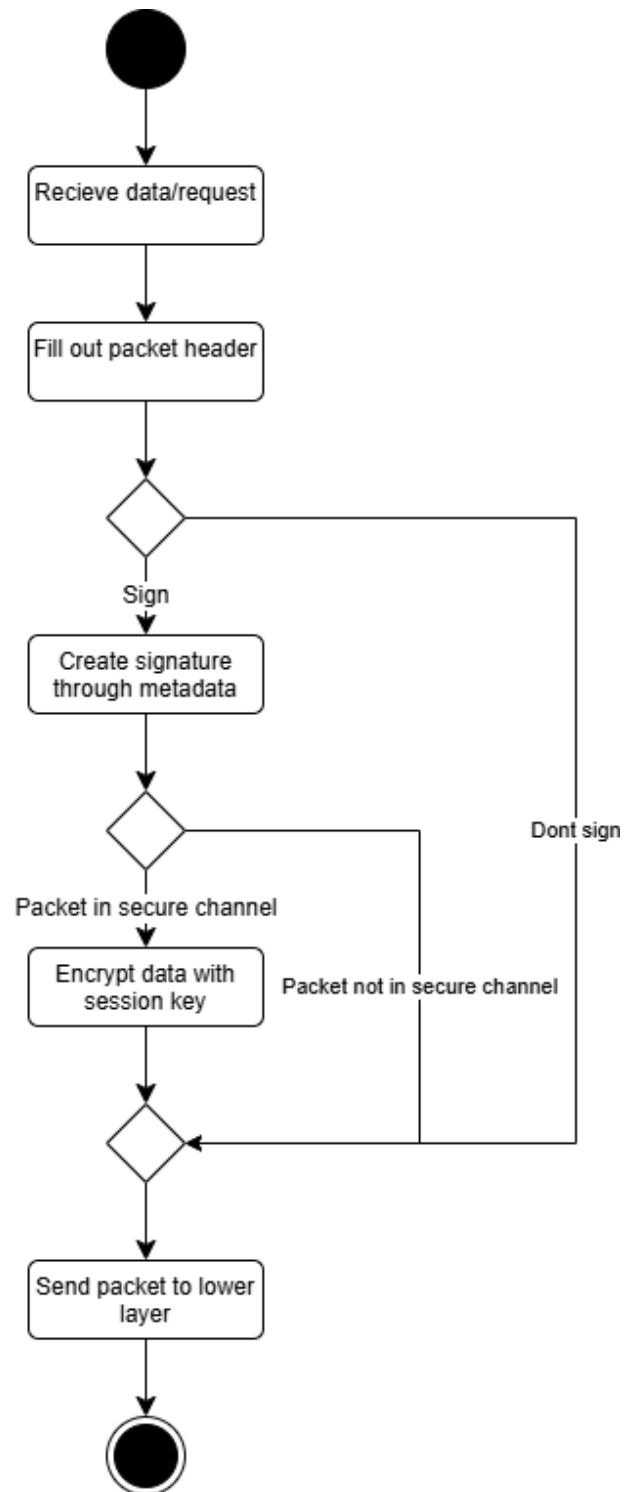
**Figure 17:** *Activity diagram: Form packet*

## 5.4 Sequence Diagram



Bootstrapping

Open single communication channel

**Process Message**

**Send Message**



**Fragmentation**

**Form Packet**



Device

PacketHandler    MetadataManager    CryptoManager    TransportLayer

**Receive request**

sendDataRequest(data/request)

Receive data/request

**Fill out packet header**

Fill out packet header

**Decide signing**

Check if signing is required

alt    [Sign]

Create signature through metadata

Return signature

[Do not sign]

Skip signing process

**Secure channel check**

Check if packet is in secure channel

alt    [Packet in secure channel]

Encrypt data with session key

Return encrypted data

[Packet not in secure channel]

Skip encryption

**Send packet**

Send packet to transport layer

Acknowledge packet sent

Device

PacketHandler    MetadataManager    CryptoManager    TransportLayer

## 5.5 Deployment Diagram



This diagram shows the execution
architecture of the hardware system and the assignment of software artifacts to the hardware. Each device (smartphone) runs the app, which includes several components: Interface, Mesh manager, Encryption engine, Data cache (message queue, encryption keys, etc). Devices communicate directly using BLE in a mesh topology. The Mesh manager handles discovery, relaying, and routing of the payload while the encryption engine keeps things private.

## 5.6 Class Diagram

The Class Diagram illustrated depicts the static structure of the system, defining the attributes, operations, and relationships between the core components.

This diagram combines the entities defined in the sequence diagrams into a unified architectural model. Key components include:

- **UI Layer:** Represented by the `User` and `ChatBox` classes, handling user interactions.

- **Mesh Logic:** The `MeshManager` acts as the central controller, coordinating with `BootstrappingController` for node discovery.

- **Data Handling:** The `MessageProcessor` delegates specialized tasks to the `SecurityHandler` (encryption) and `FragmentationHandler` (packet sizing).

- **Storage:** `ChatStorage` and `Cache` provide local persistence for messages and routing tables, essential for the offline capability.
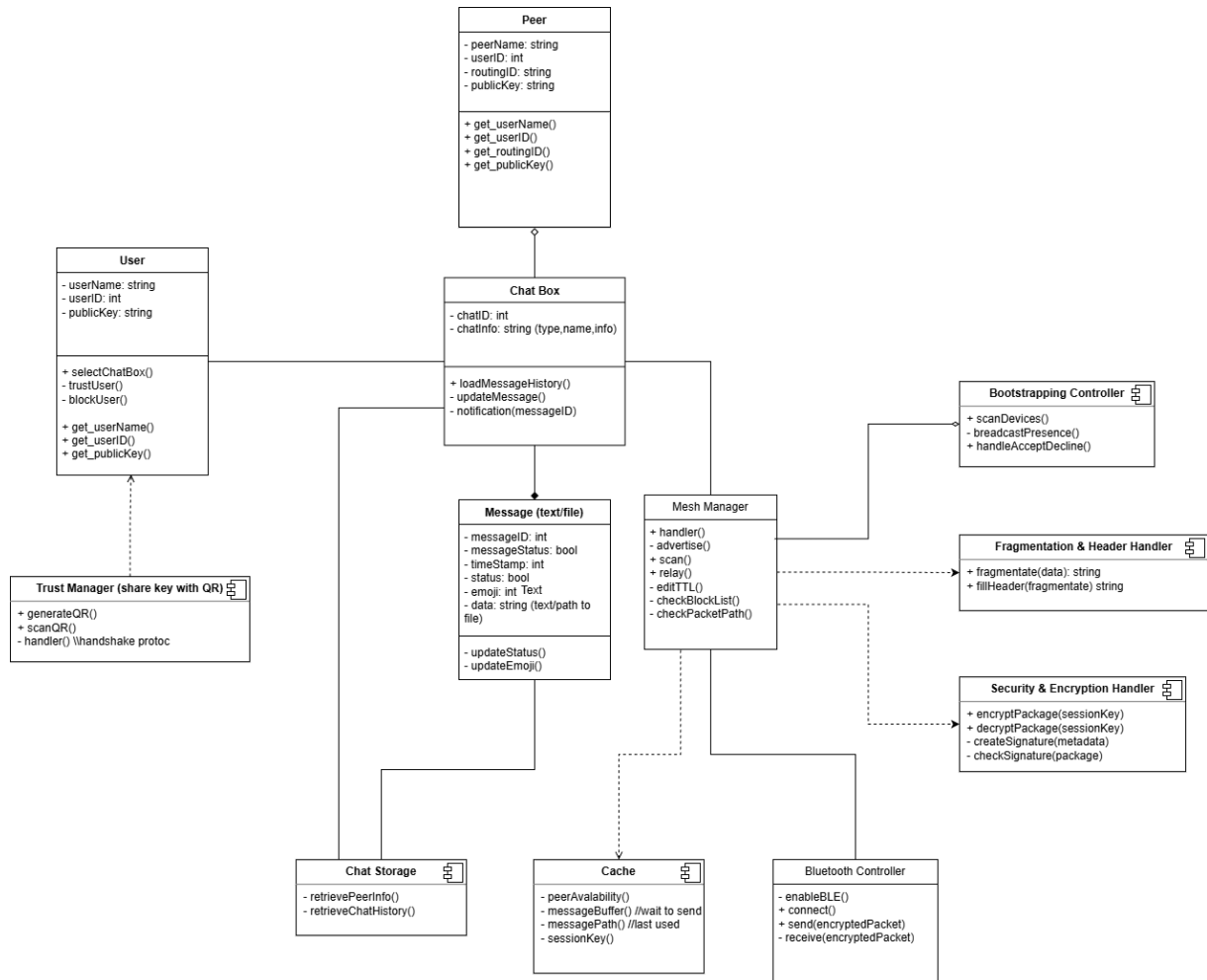
**Figure 18:** *System Class Diagram showing the relationships between the Mesh Manager, Controllers, and Data Entities.*

# 6 System Design

## 6.1 System Architecture

### 6.1.1 Comparison of Architecture Styles and Architecture Decision

Since this project focuses on developing a mobile messaging application over a Bluetooth mesh network, only monolithic architectural styles are considered. This choice is motivated by the constraints of mobile environments, including limited resources, offline operation, and the need for tight integration between networking, security, and application logic.

Table 7 presents a comparison of three commonly used monolithic architecture styles and evaluates their suitability for the Meshenger system.

Based on the comparison, the layered architecture emerges as the most suitable choice for this project. In the context of Meshenger, each layer is designed as a collection of loosely

**Table 7:** *Comparison of Monolithic Architecture Styles*

| Pattern | Pros | Cons | Meshenger Suitability |
|---|---|---|---|
| Layered | Clear separation of concerns (UI, business logic, data). Easy collaboration for teams with different specializations. Widely adopted and well understood. Supports modularity and maintainability. | Rigid dependencies between layers. Performance overhead if layering is overly strict. Changes may ripple across multiple layers. | **Highly suitable (Recommended).** Matches the team's diverse skills. Core services such as Networking, Security, Backend, and Frontend can be structured into layers, improving clarity, maintainability, and testability. |
| Pipeline | Effective for data processing and transformations. Independent and reusable stages. Supports parallel execution. | Complex error handling. Management overhead for non-streaming tasks. | **Partially suitable.** Useful for packet transformation pipelines, but less effective for interactive and stateful components such as peer tables and message queues. |
| Microkernel | Lightweight and stable core. Flexible through plug-ins and extensions. Suitable for frequent feature updates. | High complexity in plug-in contracts. Performance overhead due to indirection. Difficult to apply to tightly coupled services. | **Moderately suitable.** Appropriate for extensible modules, but introduces unnecessary complexity for core services where simplicity and stability are prioritized. |

coupled yet highly cohesive modules, ensuring a clear separation of responsibilities. This approach enhances maintainability and scalability while simplifying testing, as individual modules can be validated independently without affecting other layers of the system.

### 6.1.2   Data Flow Diagram

This data flow diagram illustrates how the Meshenger application processes data and requests, as well as how it structures the overall system. First, the diagram presents the complete architecture and the flow between its processes. Then, each process is examined in detail to highlight its features and explain how the processes interact with one another to handle requests. Note that this data flow diagram only illustrates the architecture and data flows of the application, not the whole topology of the network.

**Overall Architecture**
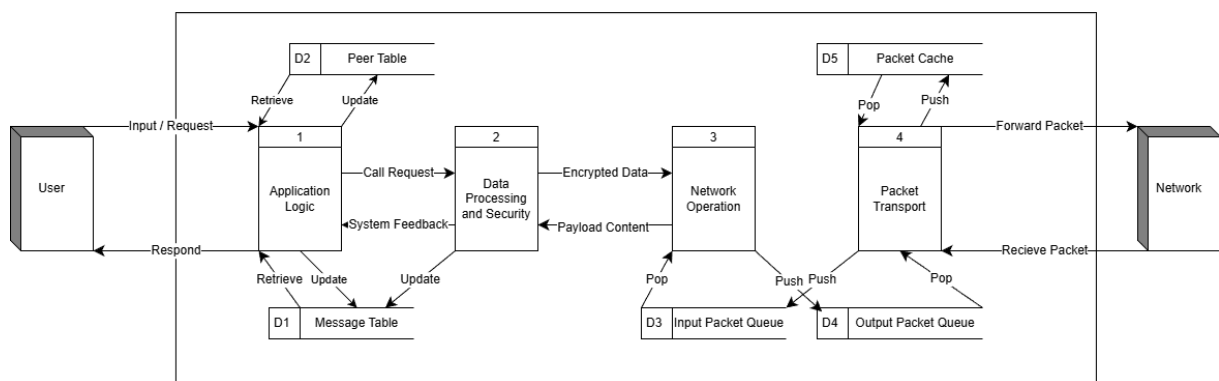


**Figure 19:** *Dataflow diagram 1: Overall Architecture*

The diagram illustrates the overall architecture of the application. The system consists of four main layers:

- **Application Logic:** Responsible for all services and the logic behind the application's features. It receives data or requests from the user and communicates results back through the UI/UX component. This layer also forwards data to lower layers when transmission is required or when configuration changes are needed. Additionally, it can update or retrieve information from the database as part of its operations.

- **Data Processing and Security:** Focused on transforming data into a suitable format for network transfer and parsing payloads received from lower layers. It also manages session establishment between users and ensures secure communication by handling encryption and decryption for end-to-end protection.

- **Network Operation:** Concerned with packet handling. It determines the destination of outgoing packets, constructs them, and parses incoming packets. After routing decisions are made, outgoing packets are placed in a queue for the lower layer to transmit. Similarly, incoming packets can be retrieved and processed when necessary.

- **Packet Transport:** Maintains information about the closest peers that the device can connect to. It also manages packet scheduling, similar to the Network Operation layer, ensuring efficient transmission and reception.

**Application Logic Layer**
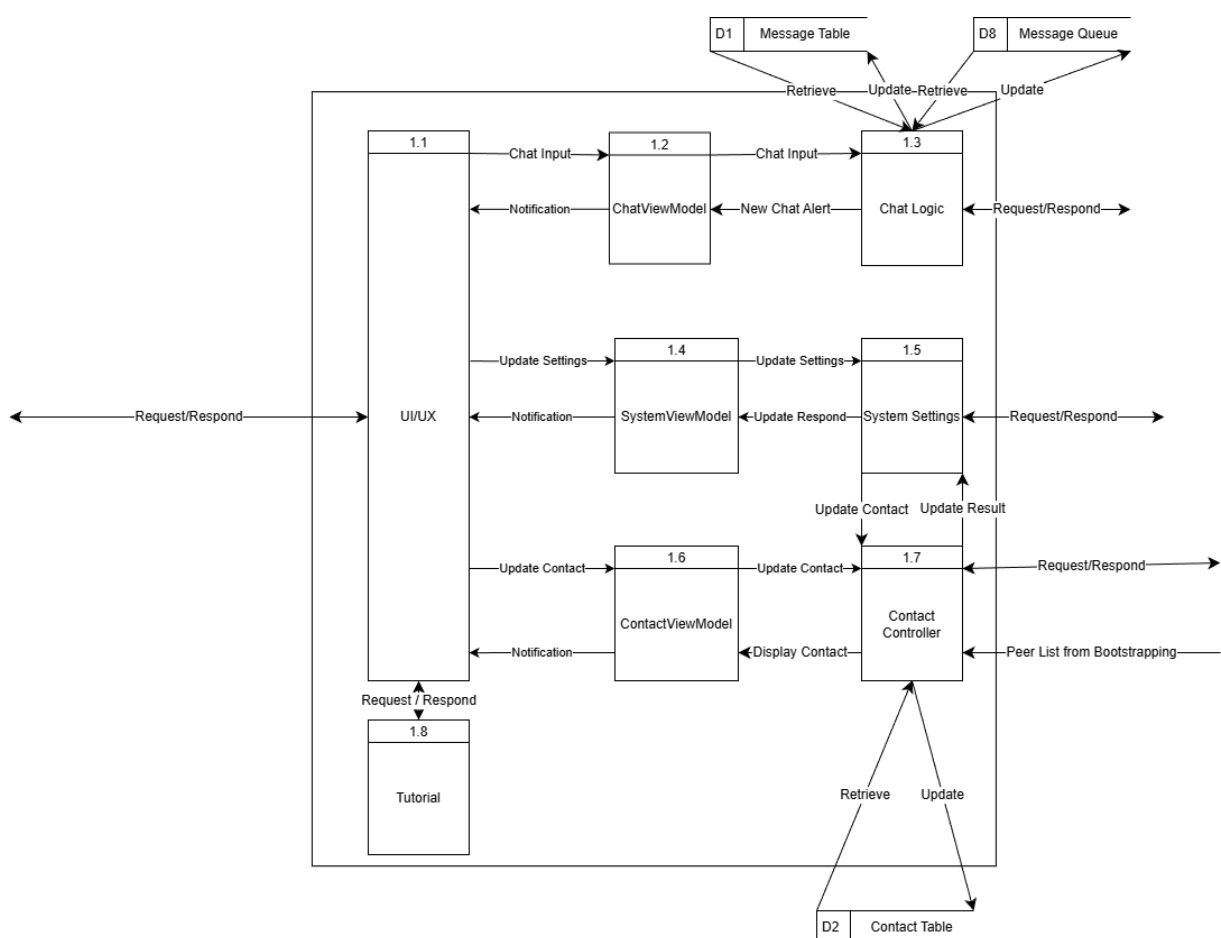


**Figure 20:** *Dataflow diagram 2: Application Logic Layer*

The diagram illustrates the Application Logic Layer and its main processes. The key processes in this layer include:

- **UI/UX:** Provides a simple interface for users to interact with the application. It receives user requests or responses, forwards them to the appropriate process, and displays the results back to the user.

- **ChatViewModel:** Handles requests related to the chat feature and acts as an intermediary between the UI/UX and the Chat Logic. It transfers requests to the appropriate process and updates the UI/UX once a response is received.

- **SystemViewModel:** Manages requests related to system features and serves as a bridge between the UI/UX and the System Settings. It forwards requests to the relevant process and updates the UI/UX when a response is available.

- **ContactViewModel:** Processes requests related to contacts and acts as a mediator between the UI/UX and the Contact Controller. It routes requests to the correct process and updates the UI/UX with the response.

- **Chat Logic:** Serves as the core process of the application. It handles user requests, constructs appropriate payloads, sends them to the lower layer or places them in the message queue, and updates the message table accordingly.

- **System Settings:** Responsible for managing changes in the application according to user preferences. It also propagates necessary configuration changes to the lower layer when required.

- **Contact Controller:** Manages all logic related to the contact list, including displaying potential contacts available in the network. It can retrieve and update the contact table as needed to complete its operations.

**Data Processing and Security Layer**
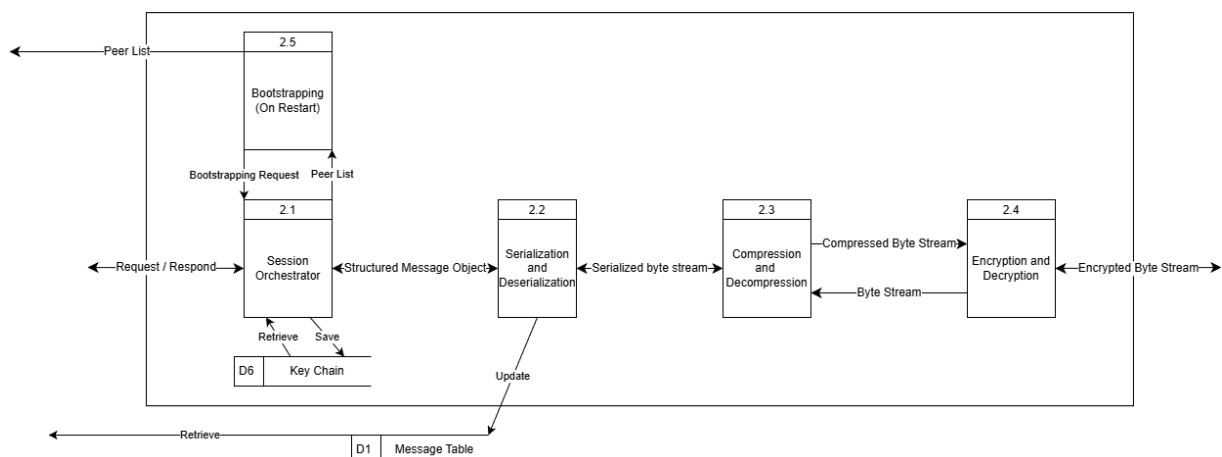


**Figure 21:** *Dataflow diagram 3: Data Processing and Security Layer*

The diagram illustrates the Data Processing and Security Layer, which is responsible for preparing data for transmission and ensuring secure communication. The main processes in this layer include:

- **Bootstrapping:** Constructs a payload containing device information and initiates a request to join the network. This process runs automatically upon restart.

- **Session Orchestrator:** Manages handshakes and key exchanges between communicating parties, returning encryption and decryption objects specific to each channel. This process may be bypassed for broadcast or network-type messages.

- **Serialization and Deserialization:** Converts structured payloads into byte streams for transmission and parses incoming byte streams back into meaningful payloads.

- **Compression and Decompression:** Optimizes network usage by compressing data when necessary and decompressing data upon receipt if it was transmitted in compressed form.

- **Encryption and Decryption:** Secures communication by encrypting and decrypting data based on the type of channel. This process can be skipped for broadcast or network-type messages.

**Network Operation Layer**
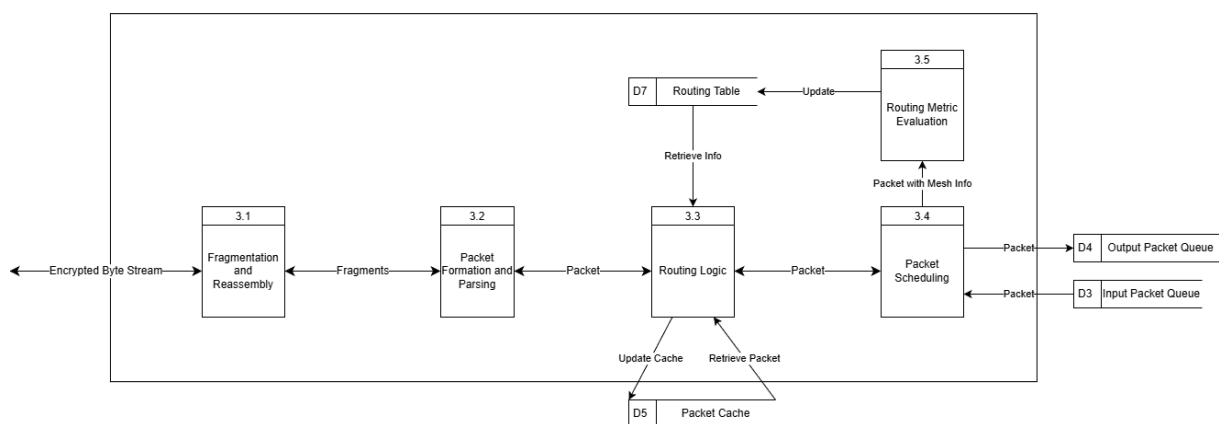


**Figure 22:** *Dataflow diagram 3: Network Operation Layer*

The diagram illustrates the Network Operation Layer, which is responsible for managing packet handling and routing within the system. The core processes of this layer include:

- **Fragmentation and Reassembly:** Provides functionality to fragment payloads that exceed the maximum limit of BLE and reassemble them once received.

- **Packet Formation and Parsing:** Constructs packets using relevant information from higher layers and parses incoming packets into meaningful payloads.

- **Routing Logic:** Serves as the central process of this layer. It retrieves network information from the routing table to determine the next destination for packets. It also updates and consults the packet cache to prevent duplicate packet processing.

- **Packet Scheduling:** Manages when and how packets are taken from the input queue and pushed to the output queue, ensuring efficient transmission.

- **Routing Metric Evaluation:** Analyzes all network-related packets and updates the routing table based on information gathered from the network.

**Packet Transport Layer**
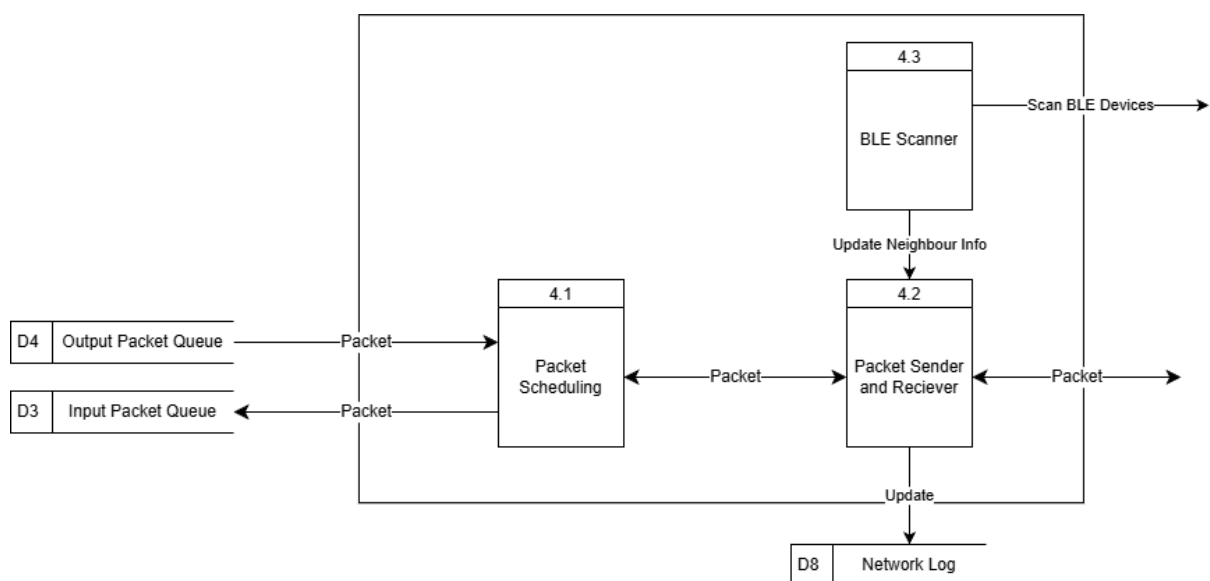


**Figure 23:** *Dataflow diagram 4: Packet Transport Layer*

The diagram illustrates the Packet Transport Layer, which is responsible for managing direct communication with nearby devices and ensuring efficient packet transmission. The core processes in this layer include:

- **Packet Scheduling:** Controls when and how packets are retrieved from the output queue and placed into the input queue, ensuring timely and efficient delivery.

- **BLE Scanner:** Continuously scans for nearby devices running the application, enabling discovery and potential connections.

- **Packet Sender and Receiver:** Maintains active connections with the closest devices, handles the sending and receiving of packets across the network, and records all incoming and outgoing packets in a network log.

### 6.1.3    Mesh Topology in the System

For this project, two types of mesh topology are considered: fully decentralized and hybrid decentralized–centralized. The initial focus is on developing the application to support a fully decentralized topology. In this model, every device in the network functions both as a client and as a router. When a packet is received, the device processes it according to the routing algorithm, deciding whether to accept the packet if it is the intended recipient, drop it if it is invalid or redundant, or relay it to another device if forwarding is required. This approach ensures that all devices contribute equally to routing, resulting in a resilient and infrastructure-independent communication network.

If sufficient time and resources are available, the application will be extended to support a hybrid decentralized–centralized topology. In this model, only devices with higher processing and networking capabilities act as routers, while weaker devices function solely as clients. Requests originating from client devices are routed through their assigned routers, which then forward packets to other routers according to the routing algorithm. This design reduces the computational and networking burden on weaker devices, mitigates bottlenecks, and helps prevent network fragmentation and excessive overhead caused by resource-constrained peers.

To evaluate the effectiveness of these two topologies in a Bluetooth environment, a series of experiments will be conducted to measure key performance metrics, including bandwidth utilization, throughput, round-trip time (RTT), and packet dropping rate. By comparing these results, insights into the trade-offs between resilience and efficiency can be obtained, ultimately guiding the selection of the most practical topology for real-world deployment.

## 6.2 Database Design
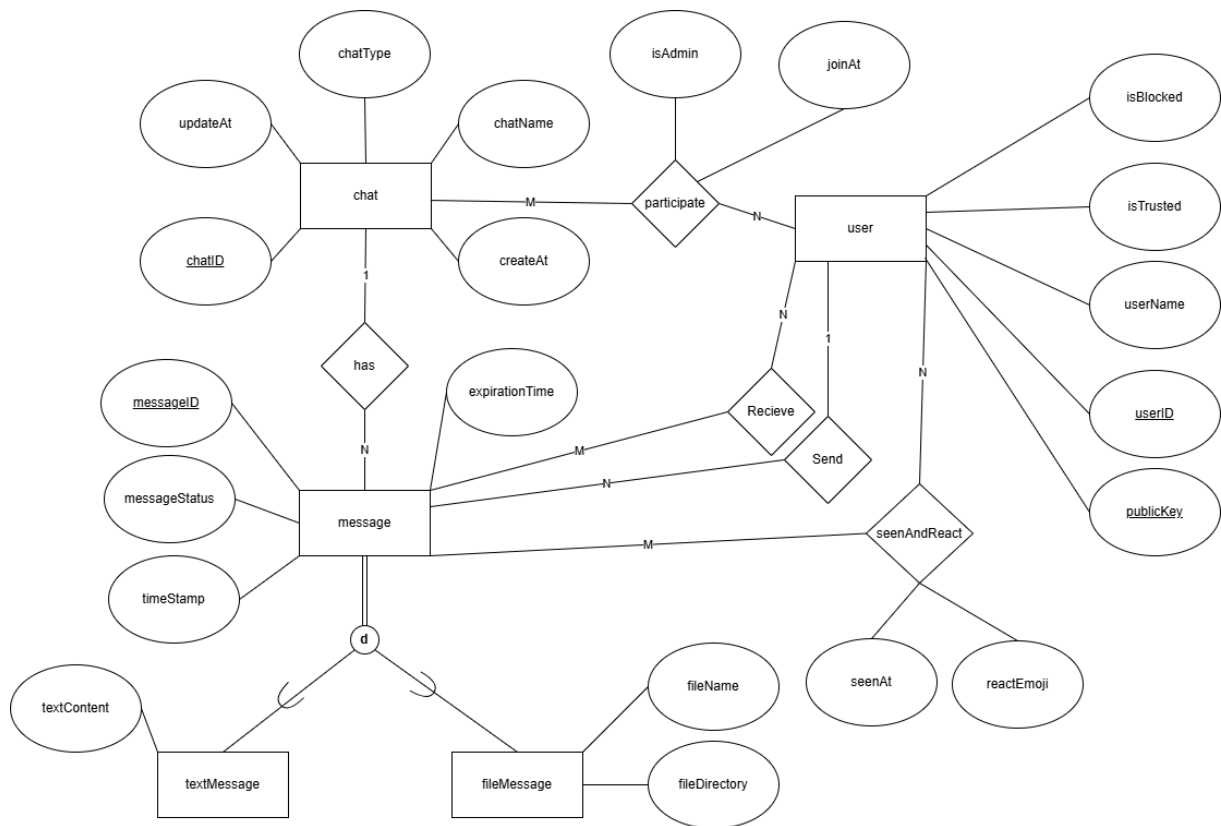


**Figure 24:** *Enhanced Entity-Relationship Diagram*

The messaging application is modeled using an Enhanced Entity-Relationship (EER) diagram that captures the core components and their interactions. The system includes entities such as `chat`, `user`, and `message`, with specialized message types: `textMessage` and `fileMessage`. Users participate in chats via the `participate` relationship, which records roles and timestamps. Messages are linked to chats through the `has` relationship, and users interact with messages through `Send`, `Receive`, and `seenAndReact`, the latter capturing reactions and view times.
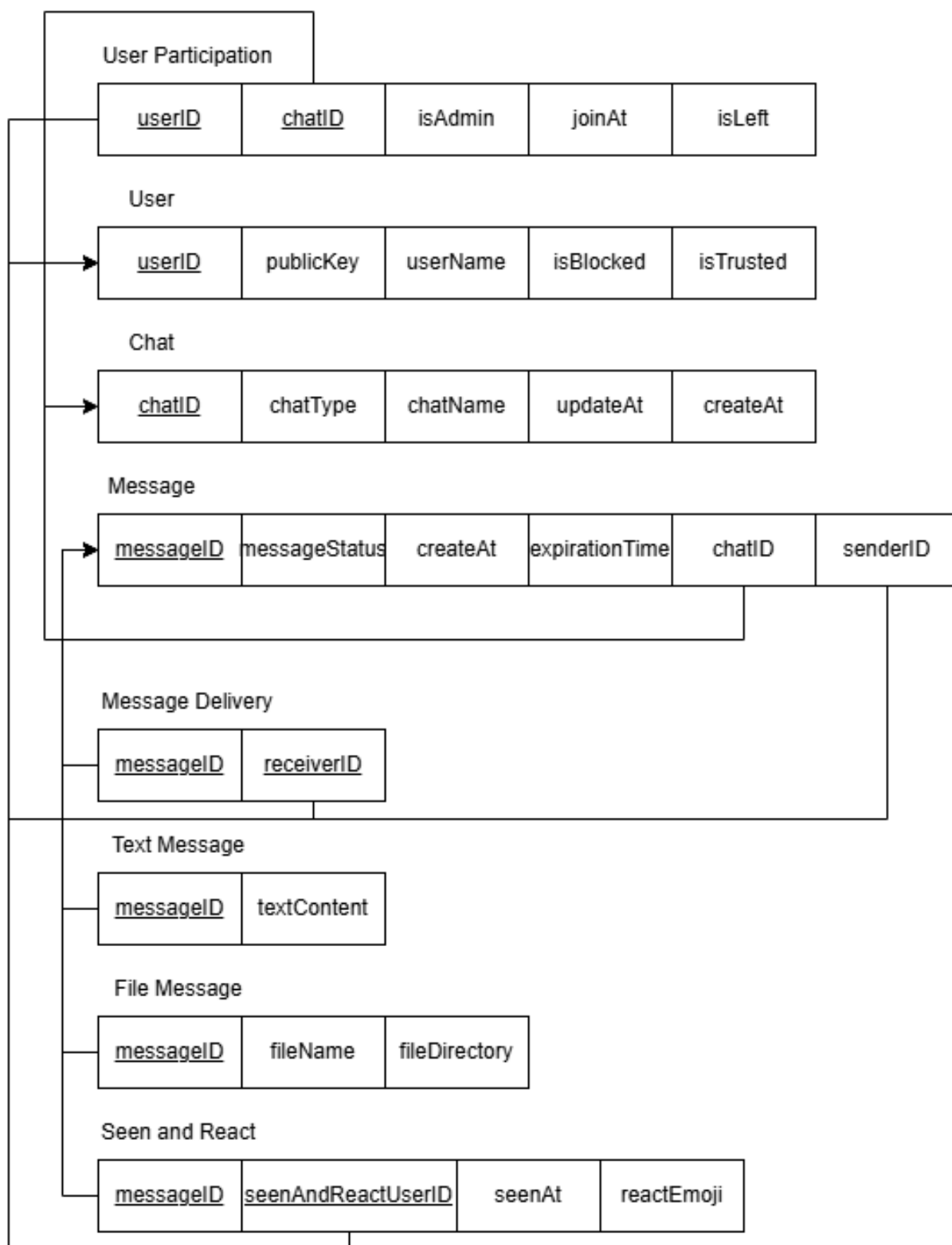
**Figure 25:** *Relational Entity Diagram*

The chat application database is structured around several interconnected entities that support messaging functionality. The `User` entity stores user credentials and trust indica-

tors, while `Chat` defines conversation spaces with metadata such as type and timestamps. User participation is tracked via the `UserParticipation` entity, which records roles, join times, and departure status. Messages are managed through the `Message` entity, linked to both sender and chat, and further extended by `TextMessage` and `FileMessage` for content specialization. Delivery is handled by the `MessageDelivery` entity, mapping messages to recipients. User interactions such as viewing and reacting are captured in the `SeenAndReact` entity, enabling feedback and engagement tracking. This schema ensures modularity, scalability, and clarity in managing peer-to-peer communication.

## 6.3 User Interface Design

### 6.3.1 Onboarding Screens

After the user completes the installation of the Android application, the system displays four onboarding screens only once to introduce the chat application's services and features.
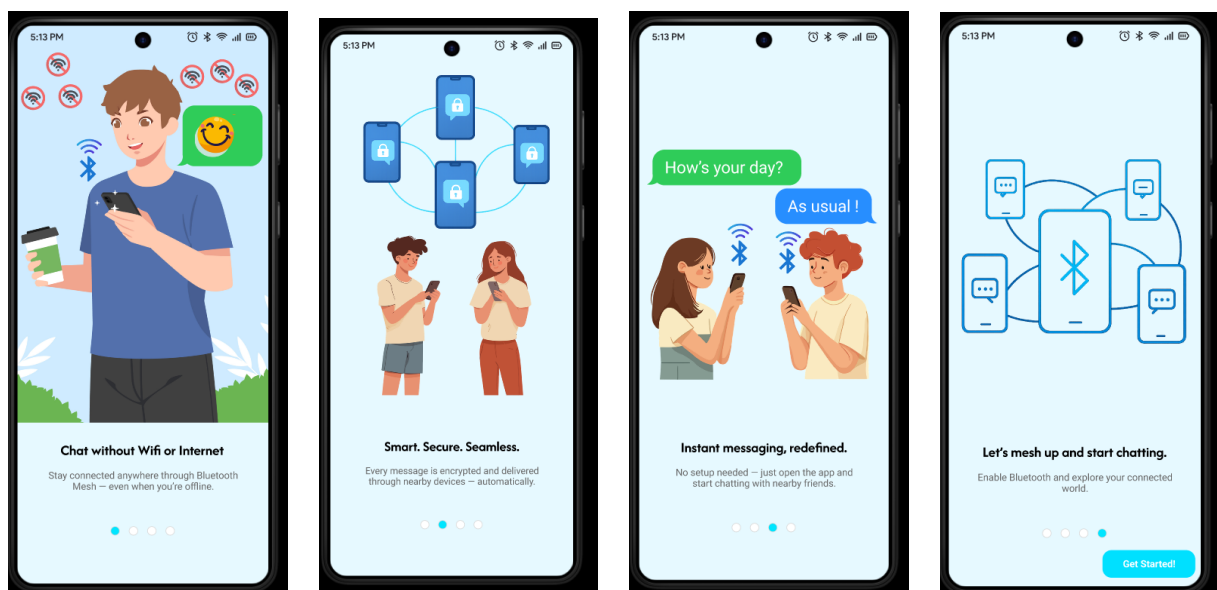


**Figure 26:** *Four onboarding screens of the chat app*

After the fourth onboarding screen, the "Get Started" button appears at the bottom-right corner, and the user is required to click it to navigate to the user profile editing section.

### 6.3.2 User Profile

Before navigating to the user profile section, the system shall check whether Bluetooth is enabled on the mobile device. If Bluetooth is not enabled, the system shall display a

Bluetooth requirement notification prompting the user to enable Bluetooth first.
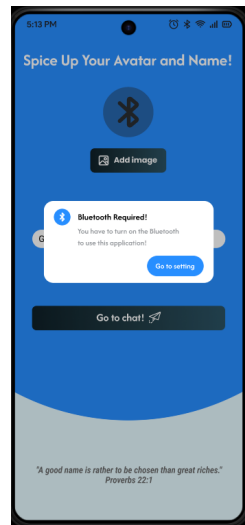


**Figure 27:** *Bluetooth Required Notification*

If Bluetooth is enabled, the system shall render the user profile section, which includes a profile image and a username input field for profile initialization. Profile initialization is optional. If the user leaves the fields unchanged and selects the "Go to Chat" button, the system shall save the default profile image and the device name as the username in the local database. Otherwise, the system shall save the customized profile information.
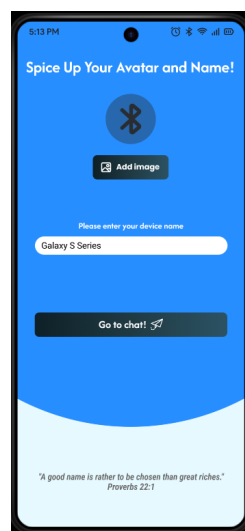


**Figure 28:** *User Profile Section*

### 6.3.3 User Manual

After exiting the user profile section, the chat interface is rendered with the following layout:

- **Header:** contains a *Settings* button, a *Chat* title, and an *Add User* button.

- **Body:** contains a search bar for searching users and messaging accounts. By default, a *Global Channel* is displayed for emergency purposes and cannot be deleted or blocked.

- **Bottom Navigation Bar:** contains an *Import* button to import saved chat history, an *Export* button to export the current chat history to a file (e.g., for device transfer), a *Profile* button to view and edit the user profile, and an *Instruction* button that provides a brief guide for new users.
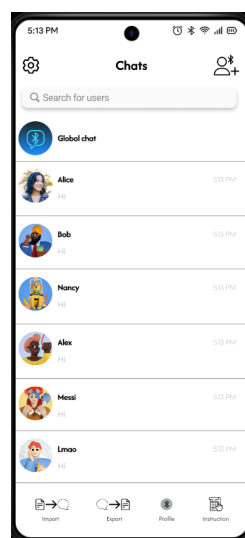


**Figure 29:** *Chat App's Home Interface*

For new users or users who have just downloaded the application, the user manual shall be displayed immediately to guide them on how to use the app. The user manual consists of four guides corresponding to the four main features of the application: adding users, importing messages, exporting messages, and viewing and editing the user profile.
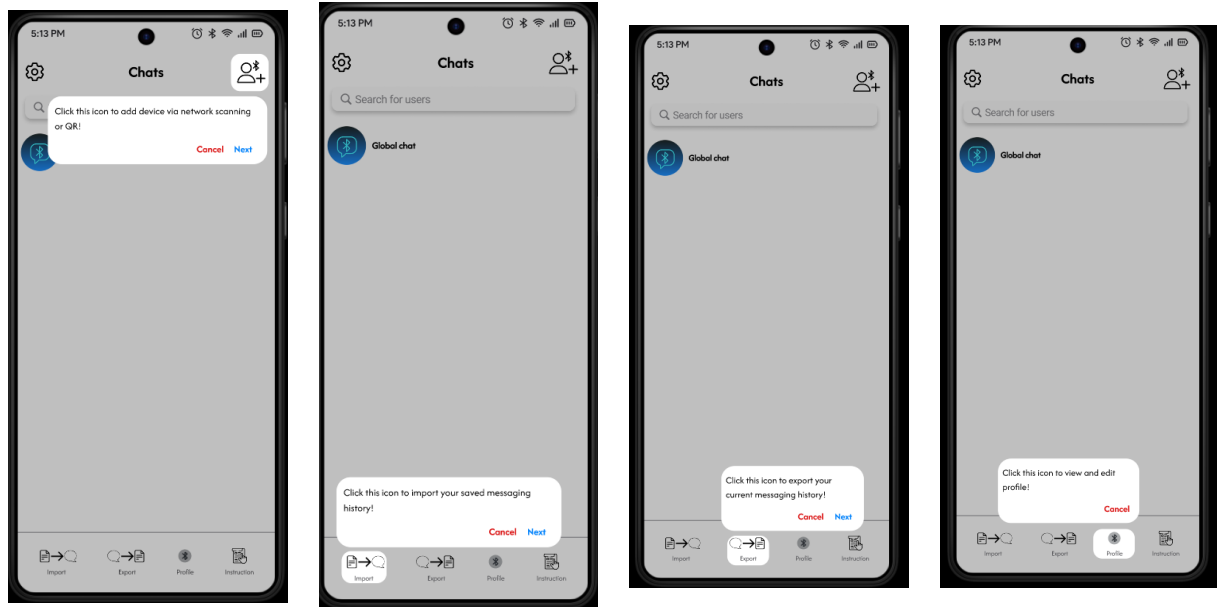
**Figure 30:** *Bluetooth Chat App User Manual*

If the user is unsure how to operate the application, they may select the "Instruction" button to display the user manual again.

### 6.3.4   Add a User

To add a new user to the network, the user shall select the "Add User" button located at the top-right corner of the home screen. The system supports two methods for adding users: network scanning and QR code scanning.
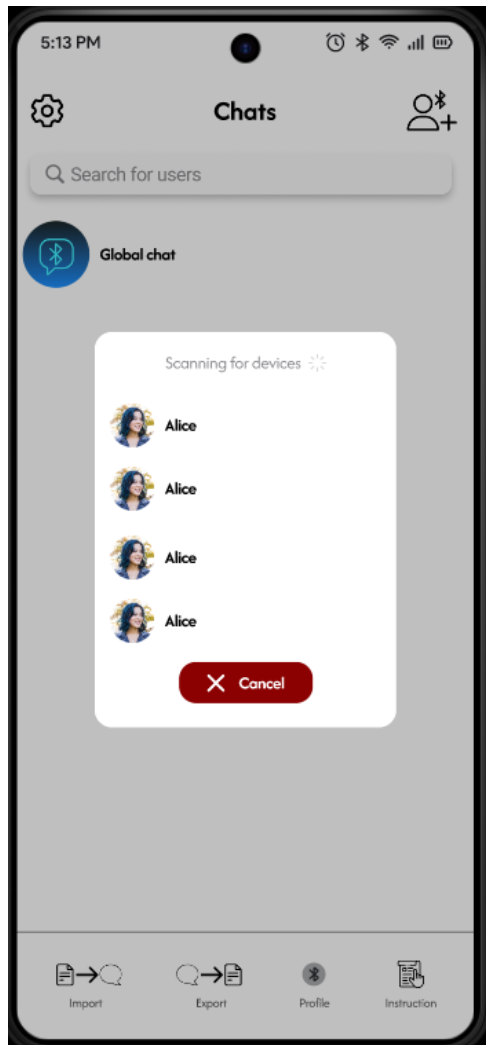
**(a)** *Display nearby users modal*
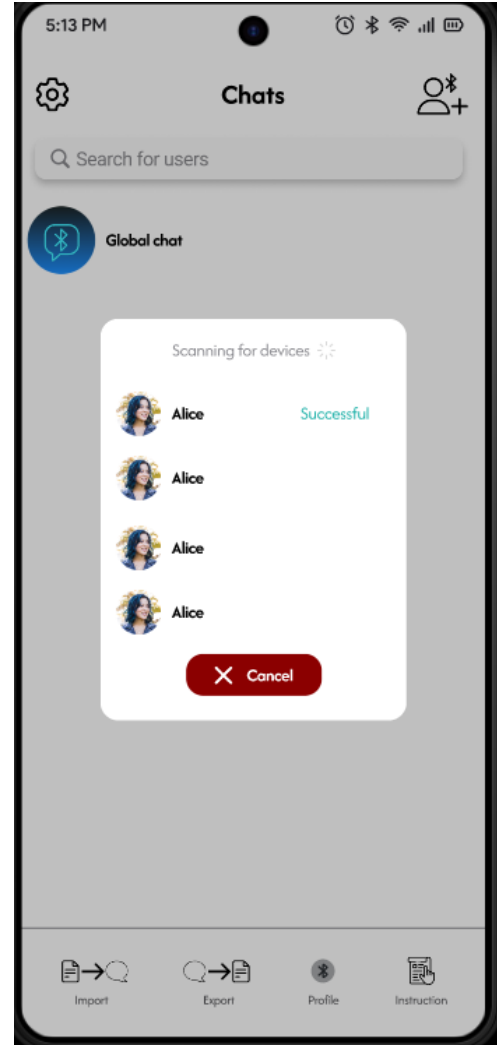
**(b)** *Send add user request successfully*

**Figure 31:** *Add a user by network scanning process*

For the Device Scanning method, a modal dialog shall be displayed, showing a list of nearby Bluetooth users in the network. After the user selects a user from the list, the system shall send a connection request to the selected user, who can accept or decline the request. If the request is sent successfully, the system shall display a success message to notify the user.

If the recipient of the friend request accepts it, the system shall display both users' accounts and usernames on their respective home screens, and establish a communication channel between them.

### 6.3.5 Send Messages

Similar to other messaging applications, users shall be able to communicate by sending messages and receive notifications when they get a reply.
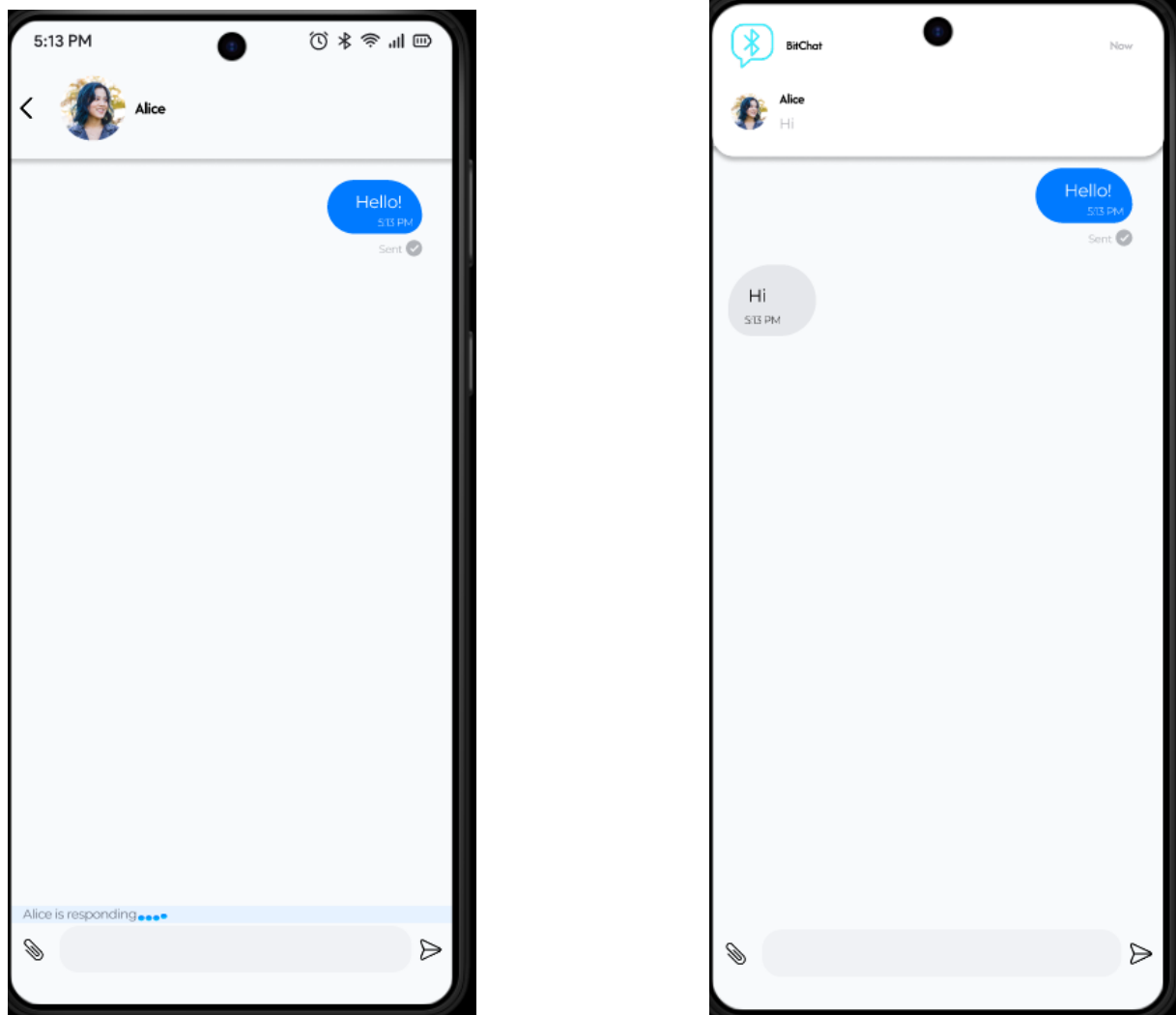
**Figure 32:** *Messaging Interface*

### 6.3.6 Import and Export messages

Since the system operates locally and does not provide online storage, all messaging activities shall be tracked and saved in a local database. Users can then export the messaging history to import it into other devices.

To export messaging history, the user shall select the "Export" icon in the bottom navigation bar. After doing so, the system shall prompt the user to choose a location to save the exported messages.
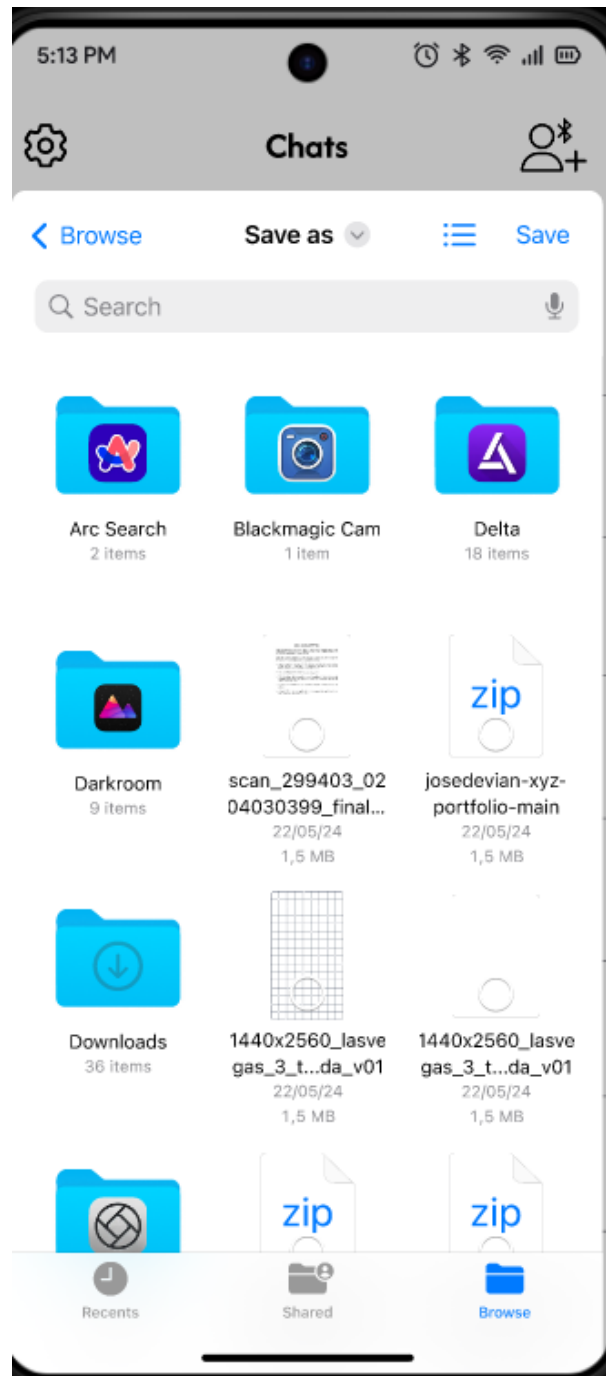
**Figure 33:** *Save location modal (iOS example for Android)*

To import messaging history for device data transfer, the user shall select the "Import" icon in the bottom navigation bar. After doing so, the system shall prompt the user to select the location of the saved data to import the messages to the device.
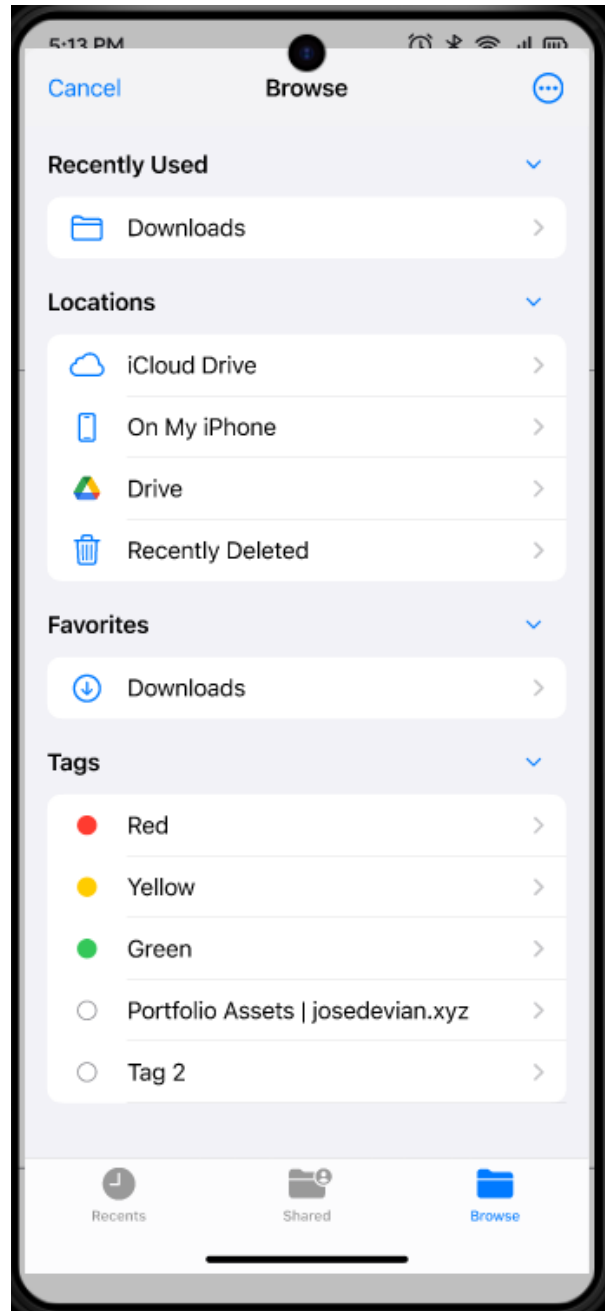
**Figure 34:** *Saved data file location modal for import (iOS example for Android)*

After importing, the system shall read all information from the saved messages, including accounts and messages, and display the account list and messages on the home screen. If there are existing messages on the device, the system shall merge them with the imported data.

## 6.4 Feature Specification

### 6.4.1 Secure Channel Establishment

**Introduction**

The primary objective of the application is to guarantee secure communication between peers. All messages exchanged between users or within channels are protected by end-to-end encryption, ensuring that only the intended participants can access the content. To accommodate different usage scenarios and security requirements, the application supports three distinct types of communication channels with increasing levels of assurance.

**Channel Types and Security Levels**

**Global channel (Low-level security):** The global channel is designed for open communication among all devices connected to the network. Messages in this channel are protected using a basic symmetric encryption scheme managed by the application. To reduce the impact of potential key exposure, the symmetric key is periodically rotated whenever predefined update conditions are met. While this channel ensures confidentiality during transmission, it provides weaker security guarantees and should not be used for sensitive communication.

**Secure channel without out-of-band verification (Medium-level security):** This channel provides stronger protection for one-to-one and group communication. Messages are encrypted end-to-end, and peers are authenticated at the device level within the network. However, the application cannot guarantee that the real-world identity of the user behind a device matches their claimed identity. Although cryptographic authenticity of devices is ensured, social impersonation remains possible. This channel balances usability and strong technical security.

**Secure channel with out-of-band verification (High-level security):** This channel offers the highest level of security. In addition to end-to-end encryption and peer authentication, users verify each other's identities through an external process such as QR code scanning to exchange public keys. By combining cryptographic guarantees with real-world verification, this channel provides strong protection against impersonation and malicious actors.

**Noise Protocol Framework**

The application adopts the Noise Protocol Framework [10] as the foundation for secure channel establishment. Noise is a flexible cryptographic framework based on Diffie–Hellman key agreement, enabling two peers to establish a shared secret over an insecure channel. It defines structured handshake patterns that provide confidentiality, integrity, and authentication. Technical details of the framework are available in the official specification.

A Noise handshake allows peers to agree on shared secrets, which are then expanded into session keys using key derivation functions and cryptographic hashing. These session keys are used to encrypt and authenticate all subsequent communication.

**Handshake Pattern Fundamentals**

A Noise handshake pattern consists of:

- A pre-message pattern for the initiator, describing public keys known in advance by the responder.

- A pre-message pattern for the responder, describing public keys known in advance by the initiator.

- A sequence of message patterns exchanged during the handshake.

In Noise notation, the symbols $e$ and $s$ denote ephemeral and static public keys, respectively. The tokens $ee$, $es$, $se$, and $ss$ represent Diffie–Hellman operations, where the first letter corresponds to the initiator's private key and the second letter to the responder's public key.

**Handshake Patterns Used in the Application**

Based on the application requirements and QR-code-based key sharing, the following handshake patterns are employed:

**XX pattern**

```
-> e
<- e, ee, s, es
-> s, se
```

The XX pattern is the default mechanism for establishing a secure channel between two peers. It ensures end-to-end encryption and mutual authentication of static keys, but does not guarantee the physical authenticity of the device holding the key.

### KK pattern

```
-> s
<- s
...
-> e, es, ss
<- e, ee, se
```

The KK pattern is used to reestablish secure channels between peers that have previously exchanged static keys. Its security depends on whether those keys were verified out-of-band.

### XK pattern

```
<- s
...
-> e, es
<- e, ee
-> s, se
```

The XK pattern is applied during QR-code-based key sharing. If used immediately after key exchange, it establishes a high-security channel. If reused after a timeout, it provides asymmetric trust guarantees while preserving encryption.

### Cryptographic Primitives

The cryptographic primitives selected for secure channel establishment are:

- **Diffie–Hellman:** Curve25519

- **Cipher:** ChaCha20-Poly1305

- **Hash function:** SHA-256

**Rationale for the Design**

The choice to apply three different handshake patterns instead of relying on a single one is driven by the need to support additional features, maintain performance, and ensure fault tolerance. In a mesh network, each device must be uniquely identifiable to its peers. Using a static key is the most effective solution, since cryptographic functions are widely supported across devices. Static keys not only provide a natural means of authentication between peers but also eliminate the need to design a new identifier system, similar to how MAC addresses uniquely identify devices in traditional networks. If only one handshake pattern were chosen, the XX pattern would be the best candidate. It offers end-to-end encryption and mutual authentication, covering nearly all requirements. However, XX requires three message exchanges. As the network grows, this increases the cost of establishing secure channels, since devices must process and relay more packets. For mobile devices where performance is critical, this overhead becomes a problem. To address this, the KK pattern are introduced. It require only two messages, which reduces computational and networking load while still supporting strong security features. To achieve higher security, the application incorporates out-of-band verification via QR code scanning, which leverages known static keys. In this scenario, XX is not ideal, since it does not take advantage of pre-shared keys. The KK pattern is better suited, as it reuses previously exchanged static keys and integrates out-of-band verification to strengthen authentication. Finally, there are cases where only one side successfully retrieves the other's static key, for example, if Alice obtains Bob's key but Bob fails to obtain Alice's. In such situations, the XK pattern ensures fault tolerance by establishing a secure channel that authenticates Bob's device to Alice, even though Bob cannot verify Alice's device.

For selecting the Diffie-Hellman and cipher functions, the rationale is straightforward: since the target platform is Android mobile devices, the cryptographic primitives must be supported by the operating system's libraries and deliver strong performance. Curve25519 for Diffie–Hellman and ChaCha20-Poly1305 for authenticated encryption are chosen because they provide modern, sufficient security while being highly efficient on mobile hardware. Compared to their counterparts, Curve448 and AES-GCM, they are faster, lighter on CPU, and better suited for devices with limited resources. Curve25519 uses smaller key sizes and simpler arithmetic, which reduces computational overhead, while ChaCha20-Poly1305 is designed to perform well even without dedicated hardware acceleration, making it ideal for mobile processors. For hash functions, SHA-256 is selected over SHA-512 primarily for performance reasons. Mobile devices often need to process large volumes of incoming and outgoing packets, and SHA-256's smaller state size and lower computational cost make it more efficient in practice. While BLAKE2s and BLAKE2b

can perform better SHA in certain benchmarks, compatibility across all Android devices is not guaranteed. SHA-256, on the other hand, is universally supported and optimized in Android's cryptographic libraries.

### 6.4.2 Identity Management

The application employs a three-tier identity management framework designed to balance usability, routing functionality, and cryptographic assurance. Each tier provides a distinct level of trust and serves a specific role within the overall security architecture.

**Three-Tier Identity Model**

**Tier 1: Username:** The first tier consists of user-defined usernames. These identifiers are mutable and may be changed over time. As the least trusted tier, usernames are not cryptographically bound to either the device or the individual. Their primary function is usability, enabling users to recognize peers and initiate communication without providing any security guarantees.

**Tier 2: Routing ID:** The second tier is the routing identifier, which functions as the operational address for packet delivery between peers. Routing IDs are dynamically refreshed either periodically or upon application restart. This dynamic behavior enables peer differentiation while concealing the long-term identity of devices. By rotating regularly, routing IDs mitigate tracking and correlation attacks, thereby contributing to privacy and forward secrecy. This tier provides a medium level of trust and is sufficient for routing purposes.

**Tier 3: Static Public Key:** The third and most trusted tier is the static public key. This identifier uniquely binds a device within the network and serves as its definitive cryptographic identity. The static public key is used to derive routing IDs, establish secure communication channels, and anchor authentication mechanisms, providing strong cryptographic guarantees.

**Username**

Usernames are user-defined identifiers displayed in chat interfaces and peer discovery views. To ensure consistency and usability, usernames are restricted to a maximum length of 100 characters. While users are permitted to modify their usernames, a mandatory cooldown period of 48 hours is enforced between changes. This constraint prevents exces-

sive modifications and supports stable peer recognition across the network.

### Routing Identifier

Using a static public key directly as a routing identifier is undesirable, as prolonged observation of network traffic could allow adversaries to correlate packets and track specific devices. To mitigate this risk, the application employs a dynamic routing identifier referred to as the *Mesh Protocol Address (MP Address)*. This approach conceals long-term identities while preserving routing functionality, conceptually similar to the role of IP addresses in traditional networks.

Dynamic routing identifiers provide several important security properties:

- **Replay protection:** Prevents reuse of captured identifiers outside their valid epoch.

- **Proof of possession:** Ensures that only peers holding the corresponding private key can establish secure channels.

- **Forward privacy:** Compromise of a single routing ID does not expose past or future identifiers.

### Mesh Protocol Address Properties

The Mesh Protocol Address is designed to satisfy several fundamental properties. First, it must be cryptographically bound to the device's static public key, ensuring that valid addresses cannot be generated without possession of the corresponding private key. Second, the derivation process must be strictly one-way, preventing recovery of the static key from the address. These properties form the basis for the secure channel establishment mechanisms described in Section 6.4.1.

Additionally, the MP Address must be universally computable by any device running the application, given the required information. To achieve interoperability and decentralization, the address construction incorporates both static and dynamic components.

### Mesh Protocol Address Design and Rationale

The MP Address derivation integrates static elements, including the device's public key, an application-specific salt, protocol metadata, and a static arbitrary key embedded within the application. These elements establish a strong cryptographic binding between

the address and the device identity. Dynamic elements, primarily time-based epochs, introduce regular rotation to enhance privacy and reduce predictability.

### Construction Flow of the Mesh Protocol Address

**Static part.** The static component includes the long-term static public key, protocol metadata, an application-specific salt, and an embedded arbitrary static key.

**Dynamic part.** The dynamic component is defined by a time-based window derived from the current Unix time. Time is divided into fixed two-hour intervals, each corresponding to a unique epoch index. The MP Address is refreshed at each epoch.

**Cryptographic function.** The routing identifier is derived using the HMAC-based Key Derivation Function (HKDF) [11]. HKDF takes as input the hash of the static public key, the salt, protocol metadata, the hash of the static arbitrary key, and the epoch index, producing a one-way derived key securely bound to the original identity.

### Address Beautification

To improve usability, the derived routing ID is transformed into a human-readable format resembling an IP address. Every two bytes of the derived key are grouped into an integer between 0 and 65535, forming an address of the form:

$$X.X.X.X$$

where each segment represents two bytes of the underlying key material.

### Routing Identifier Formula

The MP Address is derived as follows:

$$
\begin{aligned}
\mathrm{MP\_Address} = \mathrm{HKDF}\big( \\
\mathrm{IKM} = \mathrm{SHA256}(\mathrm{PK_{peer}}), \\
\mathrm{salt} = \mathrm{salt\_value}, \\
\mathrm{info} = \mathrm{info\_prefix} \parallel \mathrm{epoch} \parallel \mathrm{SHA256}(\mathrm{PK_{app}}), \\
\mathrm{length} = L\big)
\end{aligned}
\tag{1}
$$

**Static Public Key**

The static public key is generated once during the initial launch of the application using a long-term Diffie–Hellman key pair based on the X25519 elliptic curve, as specified in the Noise Protocol Framework. This key serves as the definitive cryptographic identity of the device. It is used for authentication, secure channel establishment, and routing identifier derivation. For privacy reasons, the static public key is transmitted only in encrypted form during handshake procedures and is never exposed outside secure communication contexts.

### 6.4.3   Routing Algorithm

In a Bluetooth-based P2P mesh system, routing must operate under strict constraints such as low bandwidth, limited Maximum Transmission Unit (MTU), short-range connectivity, and frequent topology changes. Moreover, the system's core use cases—including bootstrapping, peer discovery, message relaying, packet processing, and secure channel establishment—require low-latency forwarding, resilience to intermittent connectivity, minimal control overhead, and support for both unicast and broadcast transmission models.

After evaluating multiple routing approaches, the system adopts *Hybrid Multi-hop Forwarding Over Routing (HMFOR)* as the primary routing algorithm. HMFOR combines selective broadcast and opportunistic unicast forwarding to provide a lightweight and adaptive routing strategy optimized for Bluetooth Low Energy (BLE) environments. Unlike classical Mobile Ad Hoc Network (MANET) protocols, HMFOR does not rely on periodic link-state advertisements or explicit route discovery. Instead, it maintains minimal next-hop knowledge learned passively through normal packet exchanges.

When a valid forwarding path exists, packets are transmitted using unicast to the known next hop. If no route information is available, the system falls back to controlled broadcast forwarding. This hybrid forwarding behavior aligns naturally with the system design and integrates seamlessly with existing mechanisms such as TTL reduction, packet cache filtering, duplicate suppression, and peer availability checks.

**Comparison with Alternative Routing Algorithms**

The following table summarizes the evaluated routing algorithms and highlights the rationale for selecting HMFOR.

**Table 8:** *Comparison of Routing Algorithms for BLE Mesh Networks*

| Algorithm | Pros | Cons | Suitability |
|---|---|---|---|
| Flooding | Very simple and highly resilient to topology changes. | Extremely high transmission overhead and poor scalability. | Limited |
| Gossip | Lower overhead compared to flooding; probabilistic dissemination. | Reduced delivery ratio and unpredictable reliability. | Limited |
| AODV | Efficient unicast routing after route establishment. | Heavy route discovery overhead; unsuitable for BLE constraints. | Not suitable |
| DSR | Simple design using source routing; no periodic control messages. | Route caches become stale quickly in dynamic networks. | Not suitable |
| OLSR | Low-latency routing with proactive maintenance. | Very high periodic control overhead and large routing tables. | Not suitable |
| BATMAN | Fully decentralized and adaptive. | Requires large routing tables and frequent control traffic; not BLE-friendly. | Not suitable |
| **HMFOR (Chosen)** | Hybrid broadcast and unicast forwarding; lightweight and BLE-optimized. | Slightly higher overhead than pure unicast routing. | **Best fit** |

### Core Operating Principles

**Opportunistic Route Learning:** HMFOR does not initiate explicit route discovery. Instead, routing knowledge is learned implicitly from packet forwarding. Each packet carries metadata including the source identifier, destination identifier, previous hop identifier, and a sequence number. By observing the previous hop, a receiving node learns potential next-hop information without generating additional control traffic. This information is stored in a lightweight next-hop table.

**Hybrid Forwarding Strategy:** Forwarding decisions follow a two-tier approach. If a next-hop entry exists for the destination, the packet is forwarded via unicast, minimizing latency and energy consumption. If no route is known, the packet is forwarded using controlled broadcast, excluding the immediate sender to avoid trivial loops. Failed unicast attempts trigger route invalidation and immediate fallback to broadcast forwarding, enabling rapid self-healing.

**Loop Prevention and Packet Management:** To prevent forwarding loops and excessive congestion, HMFOR employs Time-To-Live (TTL) control and duplicate suppression. Each packet carries a TTL value that is decremented at every hop and discarded upon reaching zero. Additionally, nodes maintain a cache of recently seen packet identifiers derived from the source identifier and sequence number. Duplicate packets are dropped immediately.

**Neighbor Discovery and Maintenance:** Neighbor awareness is maintained through periodic lightweight bootstrap or presence packets with a TTL of one hop. Nodes track the last reception time of packets from each neighbor. If a neighbor becomes inactive beyond a predefined timeout, it is removed from the neighbor table, and dependent routes are invalidated automatically. This soft-state mechanism allows the network to adapt naturally to mobility.

**Adaptive Route Refinement:** Routing paths are continuously refined through normal data traffic. Successful forwarding reinforces existing routes, leading to increased use of unicast forwarding over time. Conversely, topology changes naturally degrade stale routes. Failed transmissions result in route invalidation and rediscovery through broadcast, ensuring robustness without centralized coordination.

### Algorithm Behavior Example

Consider a BLE mesh network consisting of nodes A, B, C, D, and E. Initially, node A has no routing information and broadcasts a packet with a bounded TTL. Intermediate nodes

forward the packet while learning reverse paths. When the destination is reached, response packets follow the learned unicast path, reinforcing efficient routes. If an intermediate node becomes unreachable, the sender detects the failure, removes the stale route entry, and falls back to broadcast forwarding, allowing the network to rediscover an alternative path within a single transmission cycle.

### 6.4.4   Payload Structure and Protocol

This section provides a detailed description of the payload structure and the protocol design. The discussion begins with the payload structure, where each field is introduced, described, and assigned a specific purpose within the communication framework. This ensures that the role of every element in the payload is clearly defined and systematically documented.

Following the payload description, the protocol distinguishes between two primary packet types: system packets and user packets. System packets are automatically generated and transmitted by the application to support core operations such as synchronization, routing, and security enforcement. In contrast, user packets are created exclusively in response to user input, serving as the medium for direct communication between peers. This separation of packet types ensures both operational reliability and user-driven interaction within the network.

**Payload Structure**

**Version.** Size: 2 bytes. Purpose: Indicates the protocol version used for compatibility and parsing logic. The current version is `v1`.

**Flag.** Size: 2 bytes. Purpose: Identifies whether special handling is required for the packet. Valid values include `0x00` (no special treatment) and `0x01` (acknowledgment required).

**Type.** Size: 4 bytes. Purpose: Specifies the packet type (e.g., user message or system control) to guide processing behavior.

**TTL (Time To Live).** Size: 2 bytes. Purpose: Limits the lifespan of the packet across the network to prevent indefinite propagation.

**Payload Length.** Size: 2 bytes. Purpose: Defines the size of the payload in bytes for buffer allocation and parsing.

**Total Fragments.** Size: 2 bytes. Purpose: Indicates the total number of fragments

belonging to the complete message, enabling segmentation and reassembly.

**Fragment ID.** Size: 2 bytes. Purpose: Identifies the index of the current fragment for correct message reassembly.

**Timestamp.** Size: 8 bytes. Purpose: Records the Unix time at which the packet was created, supporting ordering, expiration, and replay protection.

**Receiver ID.** Size: 8 bytes. Purpose: Contains the Mesh Protocol (MP) Address of the intended recipient. For broadcast packets, the value is set to `65535.65535.65535.65535`.

**Sender ID.** Size: 8 bytes. Purpose: Contains the MP Address of the sender and is used for reply routing and authentication.

**Signature.** Size: 8 bytes. Purpose: Stores a cryptographic signature used to verify the integrity and authenticity of the packet. For system packets, the signature is computed as a hash over the Sender ID, Receiver ID, Timestamp, Payload Hash, Fragment ID, and Packet Type. Upon reception, the hash is recomputed and compared; mismatched packets are discarded. For user packets, the signature is generated by signing the Sender ID, Receiver ID, Timestamp, Message ID, Packet Type, and Payload. At the receiver, the same fields are verified using the sender's public key.

**Payload.** Size: 0–396 bytes. Purpose: Carries application-level data, which may include encrypted content or structured messages.
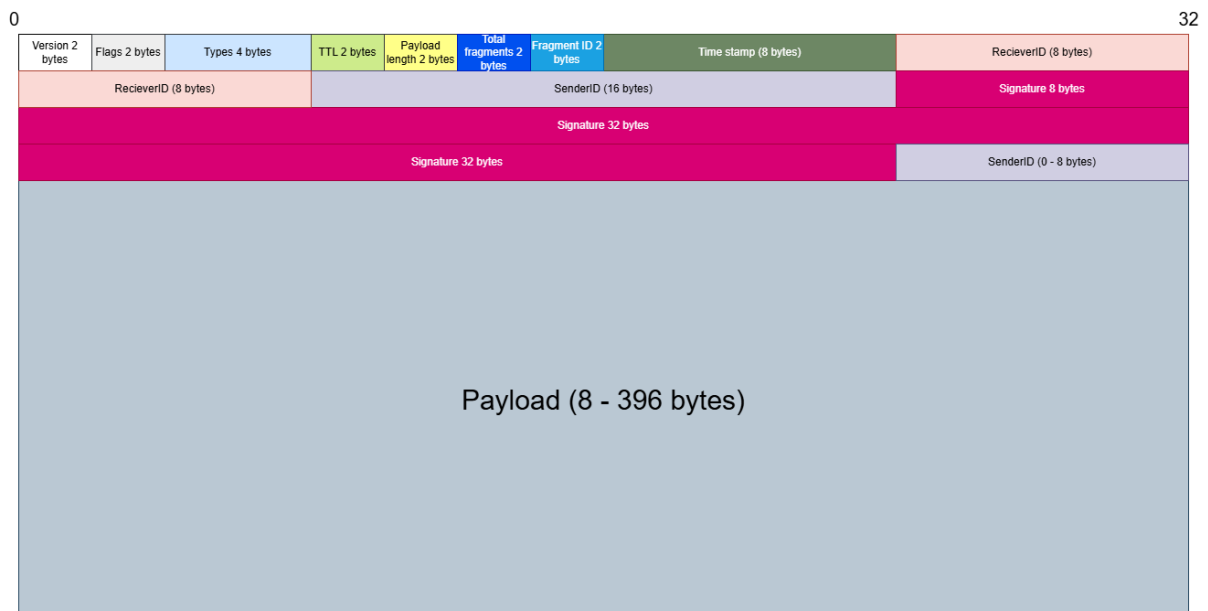
### Protocol

The protocol defines two primary categories of packets: *system packets* and *user packets*. System packets are generated automatically by the application to support network formation, routing, and security operations. User packets, in contrast, are created in response to explicit user actions and carry application-level communication data.

**System Packets** **Bootstrap Packet.** Type: `0x00`. Purpose: Used to initiate network discovery and request to join the mesh network. This packet is broadcast in nature. Description: Upon application startup, the device scans for nearby peers running the same system. If detected, the application broadcasts bootstrap packets containing essential information such as the username, Mesh Protocol (MP) address, and routing-related attributes. These packets are transmitted periodically until an acknowledgment (ACK) is received or a predefined timeout is reached. Upon successful acknowledgment, the discovered peer information is displayed to the user. If the process fails, the user may manually re-initiate the bootstrap procedure.

**Acknowledgment Packet (ACK).** Type: `0x01`. Purpose: Confirms the successful reception of packets that require acknowledgment. Description: This packet is sent in response to any packet flagged as requiring acknowledgment. The payload contains only the signature field and is delivered directly to the original sender. When acknowledging user packets transmitted over secure channels, the ACK may be encrypted.

**Session Packet.** Type: `0x02`. Purpose: Supports secure channel establishment. Description: Session packets carry the necessary handshake data required to establish a secure channel between peers. Depending on the handshake stage, these packets may be encrypted, except for the initial message in certain handshake patterns (e.g., XX).

**Control Packet.** Type: `0x03`. Purpose: Exchanges network and routing-related information among peers. Description: Control packets disseminate knowledge about network topology and peer state, supporting routing decisions and protocol coordination.

**User Packets** **User Message Packet.** Type: `0x10`. Purpose: Transmits user-generated messages. Description: The payload carries the user message to be delivered. When sent over a secure channel, the packet requires encryption and acknowledgment. A retransmission timer governs repeated delivery attempts until an ACK is received or a timeout occurs. Undelivered messages are placed in a queue and resent when the recipient is detected to be online, or may be manually retried or discarded by the user.

**Seen or Reaction Packet.** Type: `0x11`. Purpose: Indicates message read status or user reactions. Description: When a message is marked as seen or receives a reaction, a corresponding packet is generated and transmitted to the original sender. These packets require encryption and acknowledgment and follow the same reliability mechanisms as user message packets.

**File Transfer Packet.** Type: `0x12`. Purpose: Transfers file content between peers. Description: The payload contains raw file bytes. When transmitted over a secure channel, the packet requires encryption and acknowledgment and follows the same retransmission and reliability mechanisms as user message packets.

**Reply or Quote Packet.** Type: `0x13`. Purpose: Sends contextual replies referencing earlier messages. Description: The payload includes a reference to a prior message identifier along with new content. Delivery semantics are identical to those of user message packets.

### Priority Order

This subsection defines the priority assigned to each packet type and provides the rationale behind the prioritization strategy.

### 6.4.5 Peer Online Detection

The peer online detection mechanism is implemented in a lightweight and decentralized manner. The system maintains a hash table of known peers, indexed by their Message Passing (MP) addresses and associated with their static public keys. These public keys may be exchanged through QR codes or established during previous communication sessions.

To track peer activity, the system caches MP addresses observed in packets relayed through the network. Whenever a packet is received, the MP address of the sender or relay node is recorded, allowing the system to identify peers that have been recently active.

By comparing the arrival time of the most recent packet from a peer with the current system time, the application computes a *last seen* value, defined as:

$$\Delta t = t_{\text{now}} - t_{\text{arrive}} \tag{2}$$

This time difference provides a simple yet effective approximation of peer presence.

**Table 9:** *Packet Priority Classification*

| Packet Type | Code | Priority | Rationale |
|---|---|---|---|
| Bootstrap Packet | `0x00` | Medium | Required for initial network discovery and formation. |
| Acknowledgment Packet | `0x01` | High | Ensures reliability and prevents unnecessary retransmissions. |
| Session Packet | `0x02` | Medium | Essential for timely establishment of secure channels. |
| Control Packet | `0x03` | Low | Supports routing and network awareness but is less time-sensitive. |
| User Message Packet | `0x10` | High | Core communication functionality that must be delivered promptly. |
| Seen / Reaction Packet | `0x11` | Low | User experience enhancement that can tolerate minor delays. |
| File Transfer Packet | `0x12` | High | Involves large data transfer requiring prioritized and reliable delivery. |
| Reply / Quote Packet | `0x13` | High | Provides contextual continuity, though slightly less urgent than direct messages. |

Although it does not guarantee continuous online availability, it enables the application to present meaningful activity indicators in a fully decentralized environment.

# 7 System Implementation

## 7.1 Technology Stack

This section outlines the key technologies, programming languages, frameworks, and tools utilized in the development of the P2P chat application, each component is selected to meet the project's requirements.

### 7.1.1 Programming Language: Kotlin

The application is developed entirely in Kotlin, the modern, statically typed programming language recommended by many for Android development.

- **Reason for selection:** Kotlin was chosen for its interoperability with Java and its modern syntax. Crucially for this project, Kotlin provides robust APIs for accessing BLE features and managing background services, which are critical for maintaining the mesh network while preserving battery life.

- **Role:** Used for all backend logic (Mesh Manager, Fragmentation, Security) and frontend UI implementation.

### 7.1.2 IDE: Android Studio

**Android Studio** serves as the primary development platform, providing a unified environment for coding, debugging, and building the application.

- **Build System:** Gradle (used for dependency management and build automation).

- **Role:** Facilitates the integration of Android SDK tools and manages the complex lifecycle of Android components (Activities, Services) required for the mesh architecture.

### 7.1.3 UI/UX Design: Figma and Kotlin

The user interface design followed a two-step process to ensure a user-friendly experience for non-technical users in offline scenarios.

- **Prototyping (Figma):** Figma was used to design high-fidelity wireframes and user flows before implementation.

- **Implementation (Kotlin):** The Figma designs were translated into code using Kotlin (leveraging Android's UI toolkit). This ensures the interface is responsive and follows requirement guidelines for accessibility and dark mode support.

### 7.1.4   Testing and Simulation: Android Studio Emulator

Testing a P2P mesh network requires validating connectivity between multiple nodes.

- **Android Virtual Device:** The Android Studio Simulator is used to emulate different device configurations and screen sizes.

- **Mesh Simulation:** While hardware testing is required for actual Bluetooth RF signals, the logic for packet routing, message fragmentation, and database storage is validated using the emulator's debugging tools to simulate application states and data flows.

### 7.1.5   Data Persistence: SQLite

- **Technology:** SQLite (via the Room Persistence¨library).

- **Role:** As indicated in the system design, the app requires local storage for the Peer Database (caching known routes and peer identities) and Chat Storage (saving message history offline). Room provides an abstraction layer over SQLite to allow robust database access while using Kotlin Coroutines for asynchronous queries.

### 7.1.6   Connectivity: Android BLE

- **Technology:** BLE stack.

- **Role:** The core transport layer relies on the native Android Bluetooth adapter to handle Advertising (broadcasting presence) and Scanning (discovering peers), enabling the infrastructure-independent communication required by the project scope.

## 7.2   Plan for Next Phase

The development plan shown in table 10 and the Gantt Chart of figure 35, 36 and 37 is organized into eight sprints spanning from January to May 2026. Each sprint focuses on a specific phase of building a decentralized Bluetooth-based messaging app. The early sprints (1-4) concentrate on design finalization and foundational features such as account

management, connectivity, and basic messaging. Mid-phase sprints (5-6) introduce security mechanisms, advanced routing, and complete group chat functionality. The final sprints (7-8) are dedicated to testing, optimization, feature refinement, and preparing the app for release and presentation.

**Table 10:** *Project Development Plan by Sprint*

| Sprint | Task | Start Date | End Date |
|---|---|---|---|
| Sprint 1 | Finalize design, class diagram, system architecture, UI/UX | 05/01/26 | 22/01/26 |
| Sprint 2 | Implement account features, data/packet processing, BLE connectivity | 23/01/26 | 11/02/26 |
| Sprint 3 | Routing, peer identity, messaging, continuous testing | 12/02/26 | 03/03/26 |
| Sprint 4 | Chat history, customization, peer management, group chat prototype | 04/03/26 | 23/03/26 |
| Sprint 5 | Packet scheduling, advanced routing, secure channels | 24/03/26 | 10/04/26 |
| Sprint 6 | Message queuing, TCP-like messaging, full group chat | 11/04/26 | 29/04/26 |
| Sprint 7 | Regression testing, beta testing | 30/04/26 | 08/05/26 |
| Sprint 8 | UI polishing, new features, release prep, deployment, report | 09/05/26 | 27/05/26 |
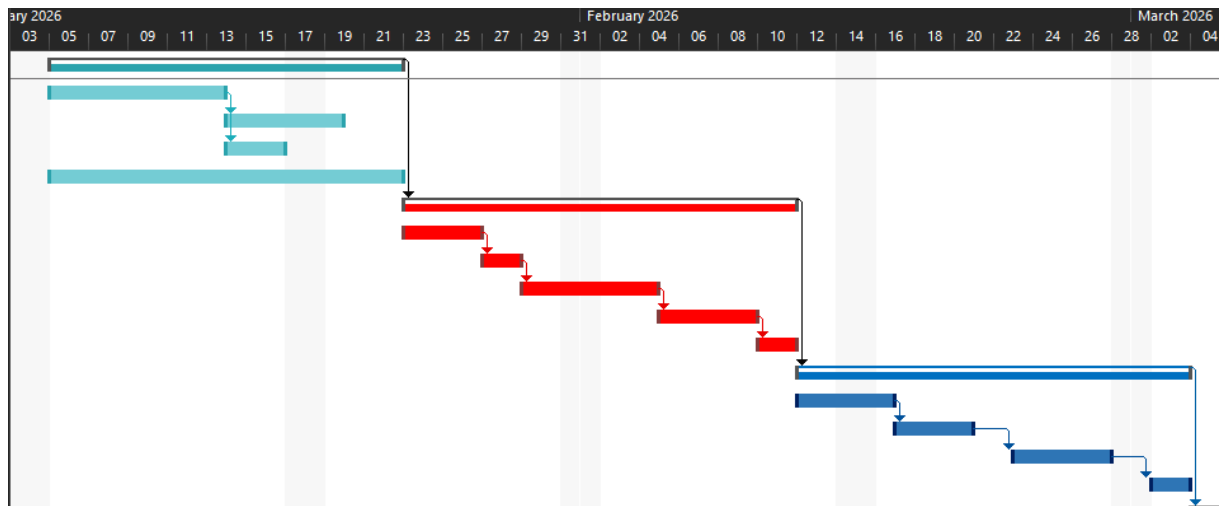


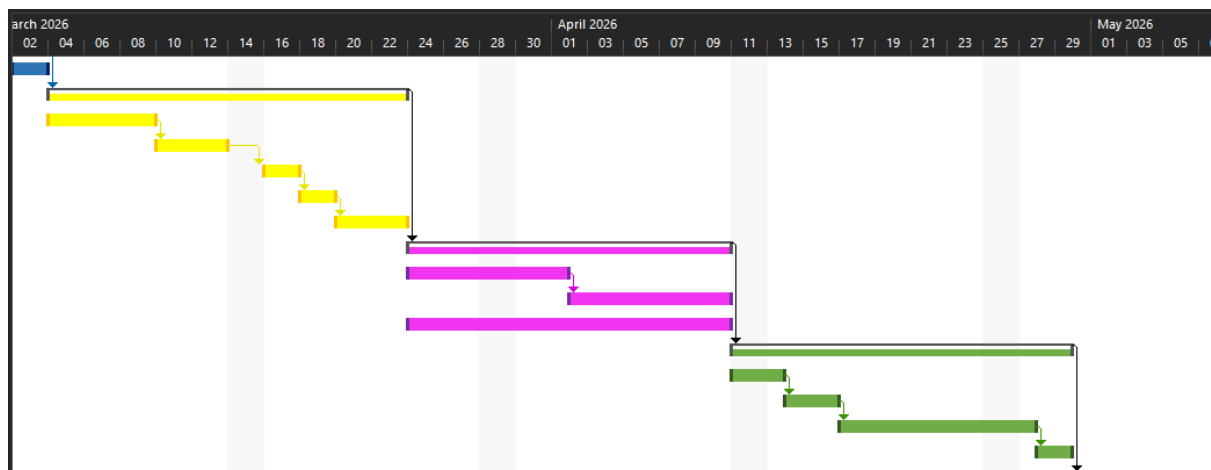**Figure 35:** *Gantt Chart of phase 1, 2 and 3*

**Figure 36:** *Gantt Chart of phase 4, 5 and 6*
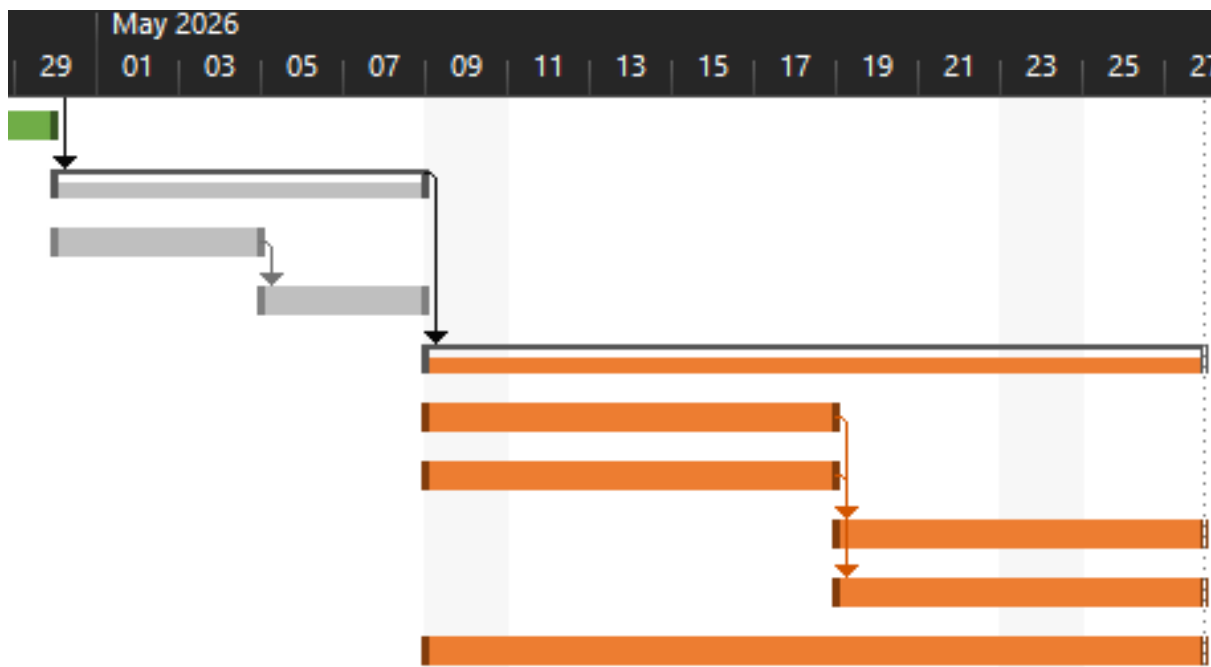


**Figure 37:** *Gantt Chart of phase 7 and 8*

# 8    Conclusion

This project presents the design and development of a decentralized peer-to-peer messaging application operating over Bluetooth mesh networks. By leveraging Bluetooth Low Energy technology and implementing end-to-end encryption, the system addresses a critical gap in communication infrastructure by enabling secure and reliable messaging in environments where traditional internet or cellular networks are unavailable or unreliable. The application demonstrates practical solutions for scenarios such as natural disasters, remote areas, large public gatherings, and situations in which communication infrastructure is deliberately restricted.

The system architecture adopts a layered monolithic design that balances modularity, maintainability, and performance constraints inherent to mobile devices. Through careful selection of cryptographic primitives, including Curve25519 for key exchange, ChaCha20-Poly1305 for authenticated encryption, and the Noise Protocol Framework for secure channel establishment, the application supports three distinct security levels to accommodate different usage scenarios and trust requirements. The implementation of the Hybrid Multi-hop Forwarding Over Routing (HMFOR) algorithm enables efficient message propagation across the mesh network while minimizing control overhead and dynamically adapting to topology changes.

The three-tier identity management system consists of usernames, dynamic routing identifiers, and static public keys. This approach successfully reconciles the competing demands of usability, privacy, and cryptographic assurance. By deriving Mesh Protocol Addresses dynamically from static keys using HKDF with time-based epochs, the system achieves forward privacy and mitigates device tracking while preserving routing functionality.

The protocol design includes system packets for network maintenance and user packets for application-level communication, providing a comprehensive framework for reliable and secure data exchange in a fully decentralized environment. Implementation using Kotlin and Android Studio, together with SQLite for local data persistence and native BLE APIs for connectivity, demonstrates the feasibility of deploying sophisticated mesh networking capabilities on standard consumer devices without requiring specialized hardware.

The user interface design prioritizes simplicity and accessibility, allowing non-technical users to communicate effectively in challenging environments without complex configuration procedures. However, several challenges and limitations remain. Bluetooth's inherent range constraints, limited bandwidth, and susceptibility to interference impose fundamental restrictions on network scale and message throughput. The current imple-

mentation does not fully address scenarios involving very large networks beyond 100 to 500 nodes, prolonged device mobility, or advanced adversarial threats such as Sybil attacks or targeted jamming. In addition, features such as voice communication, large file transfers, and cross-platform support remain areas for future development.

Looking ahead, the next development phase will focus on completing the implementation roadmap outlined in the project plan. This includes comprehensive security testing, performance optimization under real-world conditions, and evaluation of a hybrid centralized-decentralized topology variant. Further research is required to assess system behavior across diverse deployment scenarios and to refine routing strategies based on empirical performance data. The integration of additional out-of-band verification methods and the exploration of energy-efficient duty cycling mechanisms are expected to enhance both security guarantees and battery longevity.

In conclusion, this project demonstrates that infrastructure-independent and secure peer-to-peer communication over Bluetooth mesh networks is both technically feasible and practically valuable. By combining modern cryptographic techniques with adaptive networking protocols, the system provides a resilient communication platform that enables individuals to maintain connectivity when conventional infrastructure fails. The insights gained from this work contribute to a broader understanding of decentralized communication systems and provide a foundation for future innovations in off-grid and privacy-preserving messaging technologies.

# References

[1] Computerworld. (2024, July 18). *What's a peer-to-peer (p2p) network?* https://www.computerworld.com/article/1355655/whats-a-peer-to-peer-p2p-network.html

[2] Bluetooth SIG Regulatory Expert Group. (2023, March). *Bluetooth low energy – regulatory aspects (rad)* (Informational Publication) (Prepared by the Regulatory Expert Group). Bluetooth SIG. Retrieved December 21, 2025, from https://www.bluetooth.com/

[3] Android Developers. (2025). *Bluetooth low energy overview* [Official Android Developer Documentation]. Retrieved December 21, 2025, from https://developer.android.com/develop/connectivity/bluetooth/ble/ble-overview

[4] Ding, C. H., Nutanong, S., & Buyya, R. (2011, May 24). *Peer-to-peer networks for content sharing.* https://doi.org/10.4018/978-1-59140-429-3.ch002

[5] , I. (2025, November 17). *End-to-end encryption.* https://www.ibm.com/think/topics/end-to-end-encryption

[6] Dorsey, J. (2025). *Bitchat.* Retrieved December 21, 2025, from https://github.com/permissionlesstech/bitchat

[7] Briar Project. (2025). *Briar project.* Retrieved December 21, 2025, from https://briarproject.org/

[8] Gnutella Developer Forum. (2003). *The annotated gnutella protocol specification v0.4* [Document Revision 1.6, Clip2]. Retrieved December 21, 2025, from https://rfc-gnutella.sourceforge.net/developer/stable/

[9] Gnutella Developer Forum. (2002). *Gnutella protocol specification v0.6 (draft)* [Draft specification hosted on SourceForge]. Retrieved December 21, 2025, from https://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html

[10] Perrin, T. (2018). *The noise protocol framework* (tech. rep. No. Revision 34) (Official/unstable specification). Noise Project. Retrieved December 21, 2025, from https://noiseprotocol.org/noise.html

[11] Krawczyk, H., & Eronen, P. (2010, May). *Hmac-based extract-and-expand key derivation function (hkdf)* (RFC No. 5869) (Category: Informational). Internet Engineering Task Force (IETF). Retrieved December 21, 2025, from https://datatracker.ietf.org/doc/html/rfc5869