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. CompSience Council
**SUPERVISOR:** Assoc. Prof. Trương Tuấn Anh, PhD
**—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

**Authors**

*Trần Trung Vĩnh*

*Nguyễn Trần Huy Việt*

*Nguyễn Anh Quân*

*Lê Phan Nhật Minh*

# ACKNOWLEDGEMENTS

**Authors**

*Trần Trung Vĩnh*

*Nguyễn Trần Huy Việt*

*Nguyễn Anh Quân*

*Lê Phan Nhật Minh*

# 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

In recent years, the increasing reliance on centralized communication infrastructures has exposed significant vulnerabilities during emergencies and connectivity outages. Natural disasters, political unrest, and network restrictions often render traditional internet-based communication useless, leaving people isolated and unable to coordinate or request help. Furthermore, growing privacy concerns highlight the risks of centralized servers that can collect, monitor, or expose user data.

This project is motivated by the need for a secure, privacy-preserving, and infrastructure-independent communication system. By leveraging Bluetooth Mesh technology and end-to-end encryption, the proposed system enables direct, resilient, and private communication between users, even in the absence of mobile networks or Wi-Fi.

## 1.2 Objectives

The primary objectives of this project are as follows:

- **To design and develop** a decentralized peer-to-peer 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 peer-to-peer 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

### 1.4.2 Significance from the Scientific Perspectives

## 1.5 Report structure

# 2 Background Knowledge

# 3 Technology Stack

# 4 Related Works

# 5 Requirements Elicitation

## 5.1 System users

## 5.2 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).

## 5.3 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.

## 5.4 Data requirements

# 6 System Analysis

## 6.1 Use Case Diagrams

### 6.1.1 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.

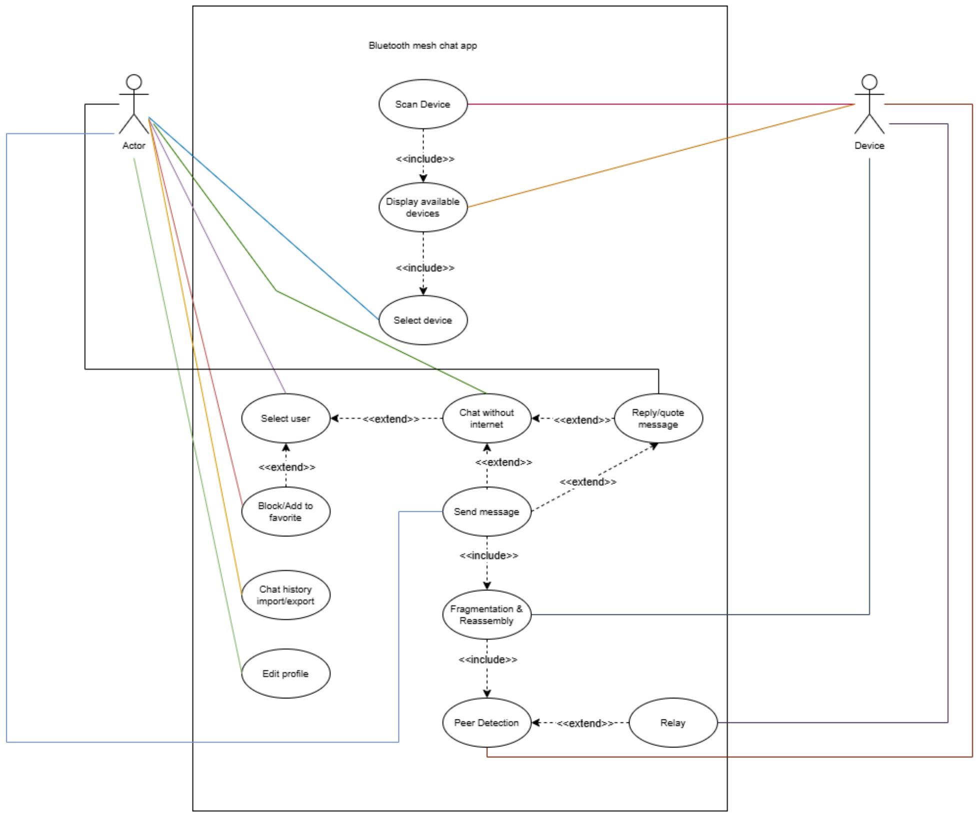


**Figure 1:** *Usecase diagram: Whole system*

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.

### 6.1.2 Bootstrapping

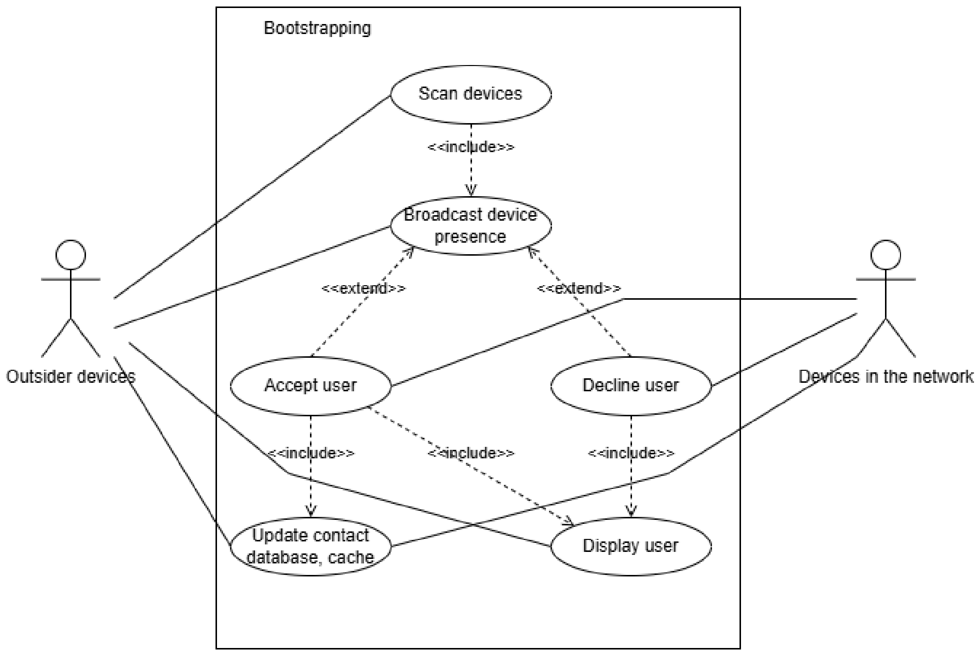


**Figure 2:** *Usecase diagram: Bootstrapping*

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.

### 6.1.3 Open single communication channel

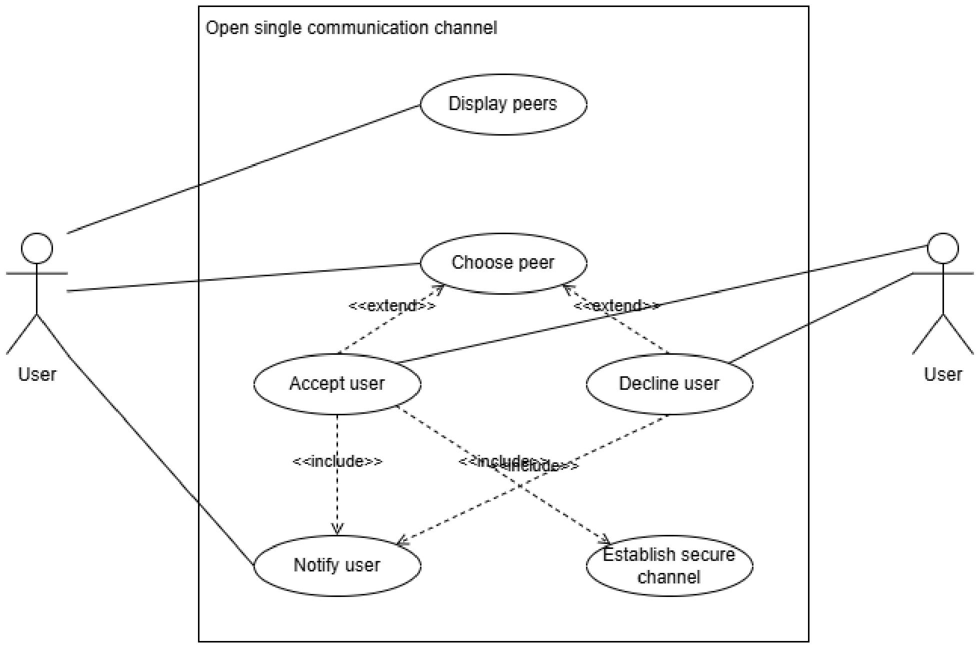


**Figure 3:** *Usecase diagram: Open single communication channel*

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.

### 6.1.4 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.

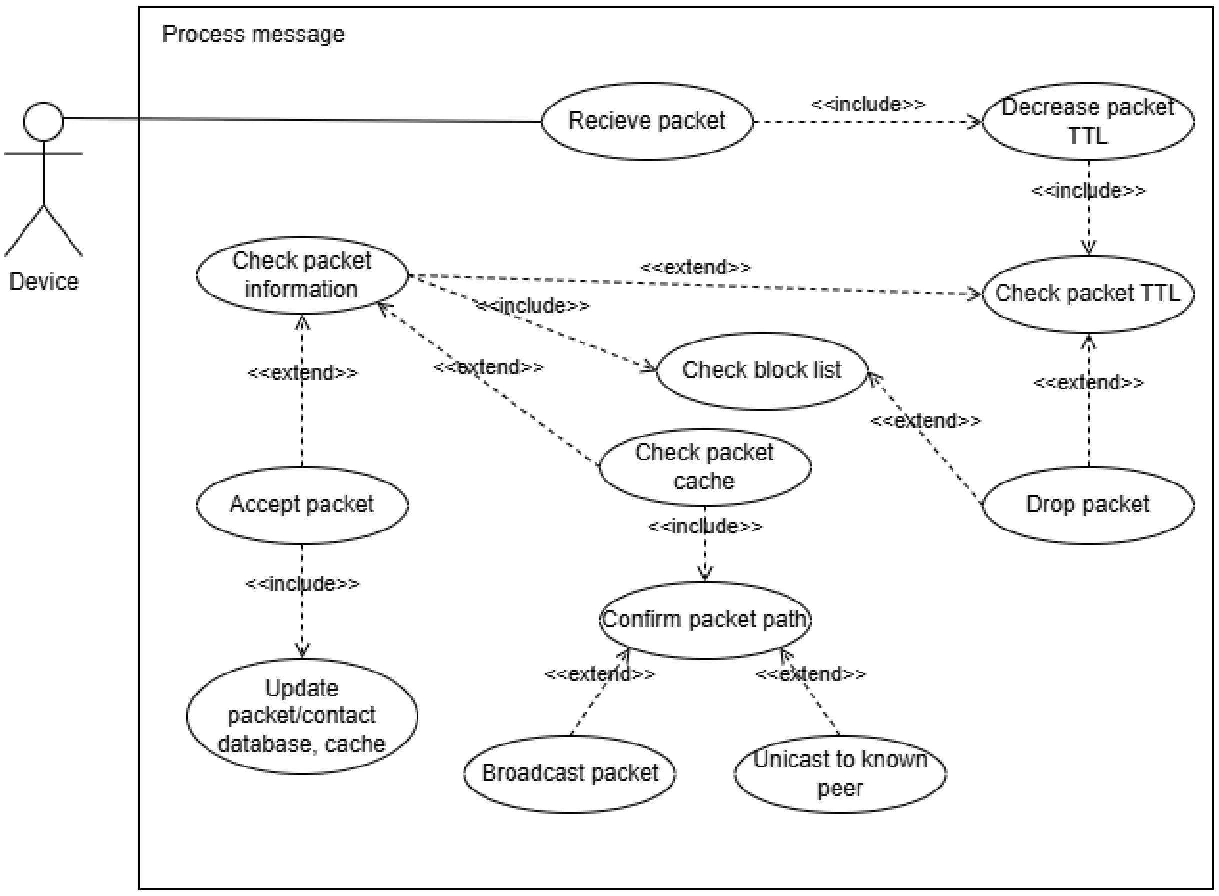


**Figure 4:** *Usecase diagram: Process message*

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.

### 6.1.5 Send message

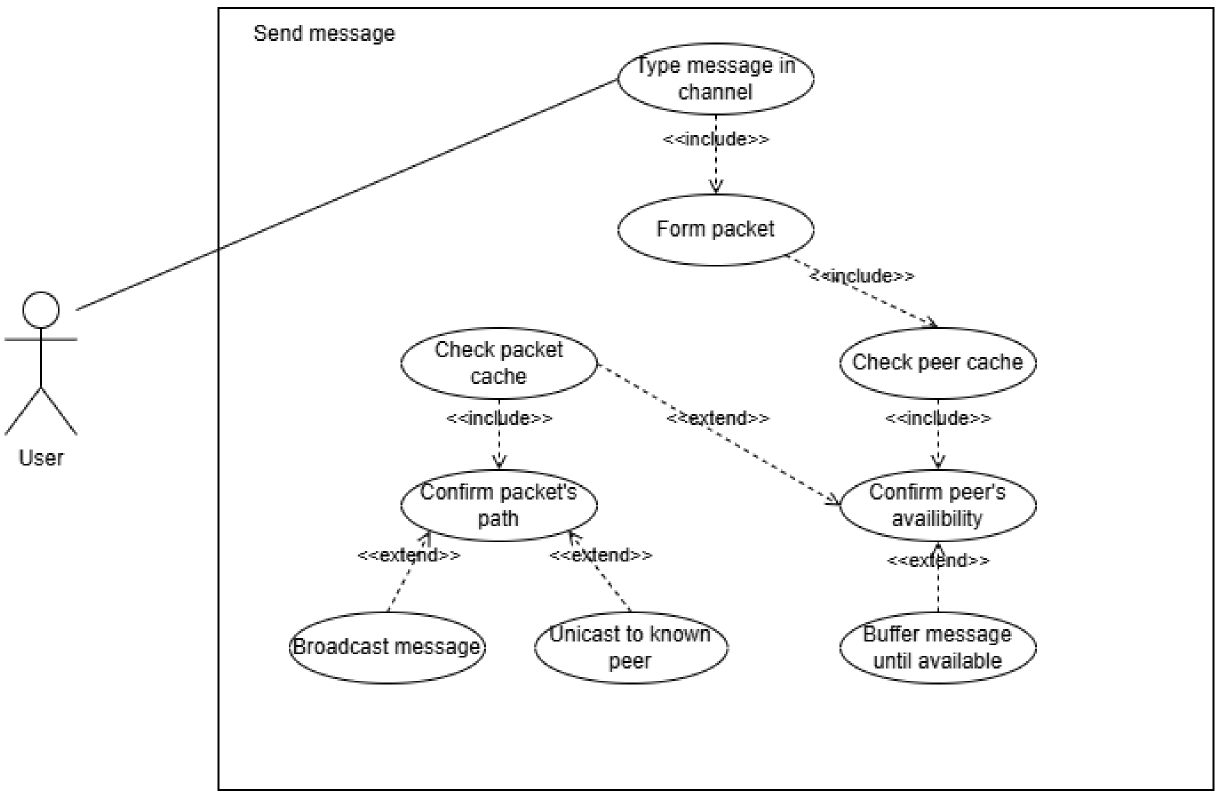


**Figure 5:** *Usecase diagram: Send message*

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.

### 6.1.6 Fragmentation

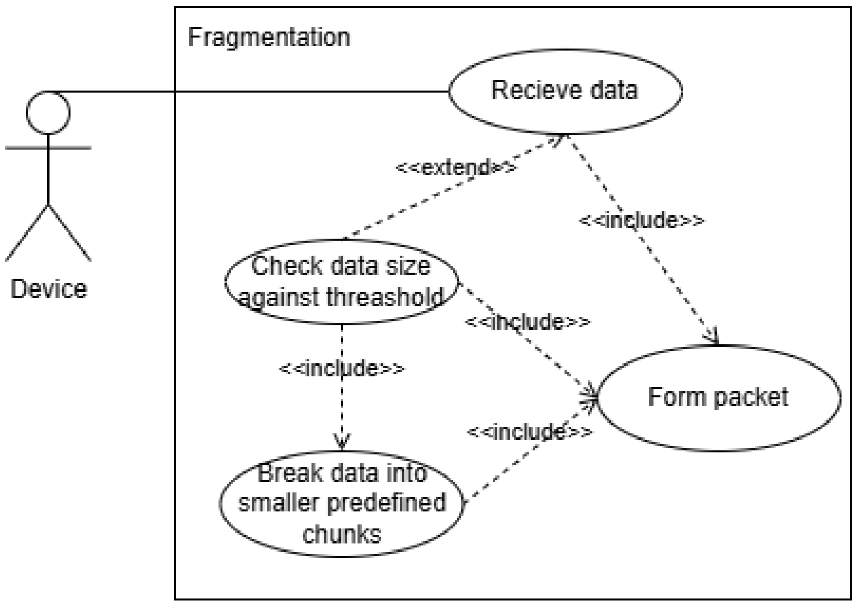


**Figure 6:** *Usecase diagram: Fragmentation*

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>.

### 6.1.7 Form packet

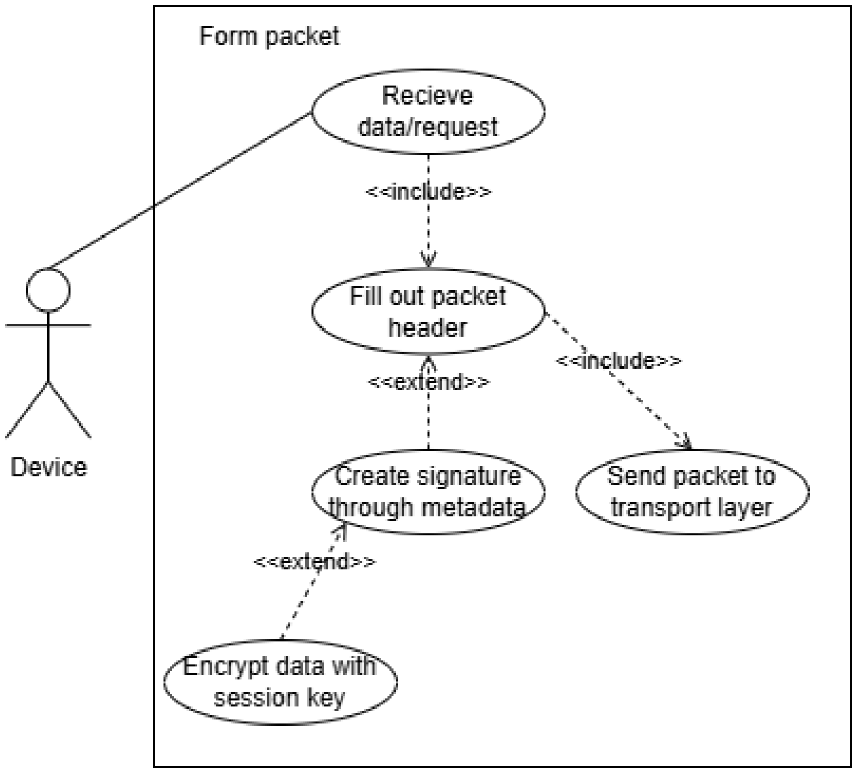


**Figure 7:** *Usecase diagram: Form packet*

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 destination 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.

## 6.2 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.

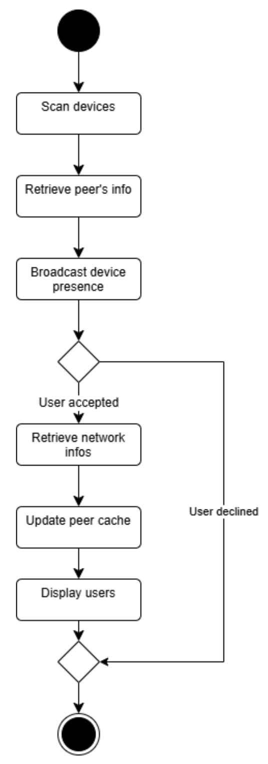


**Figure 8:** *Activity diagram: Bootstrapping*

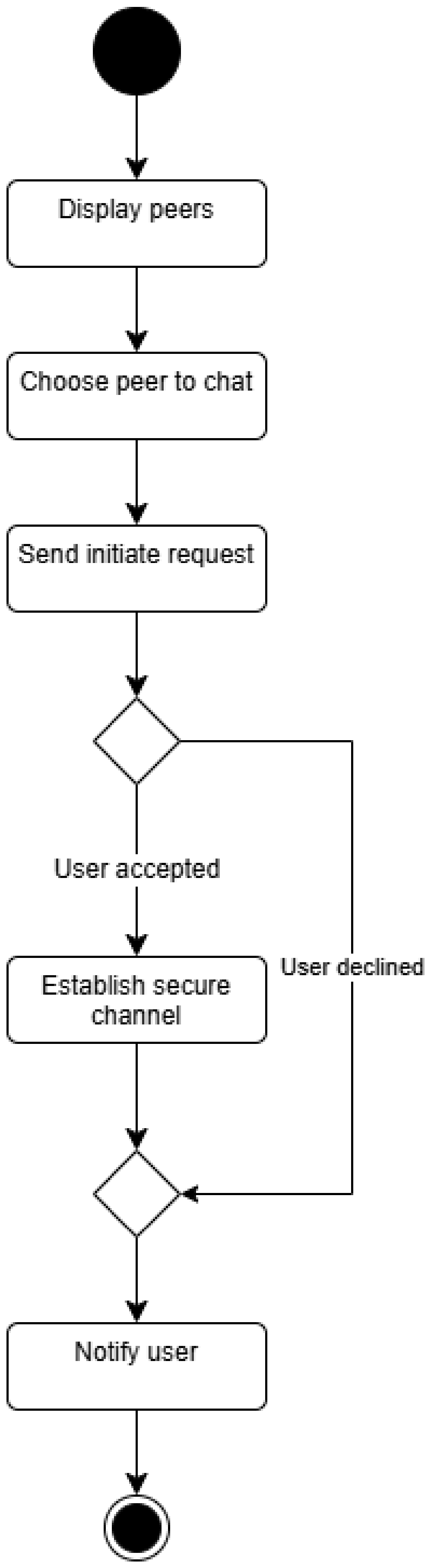


**Figure 9:** *Activity diagram: Open communication channel*

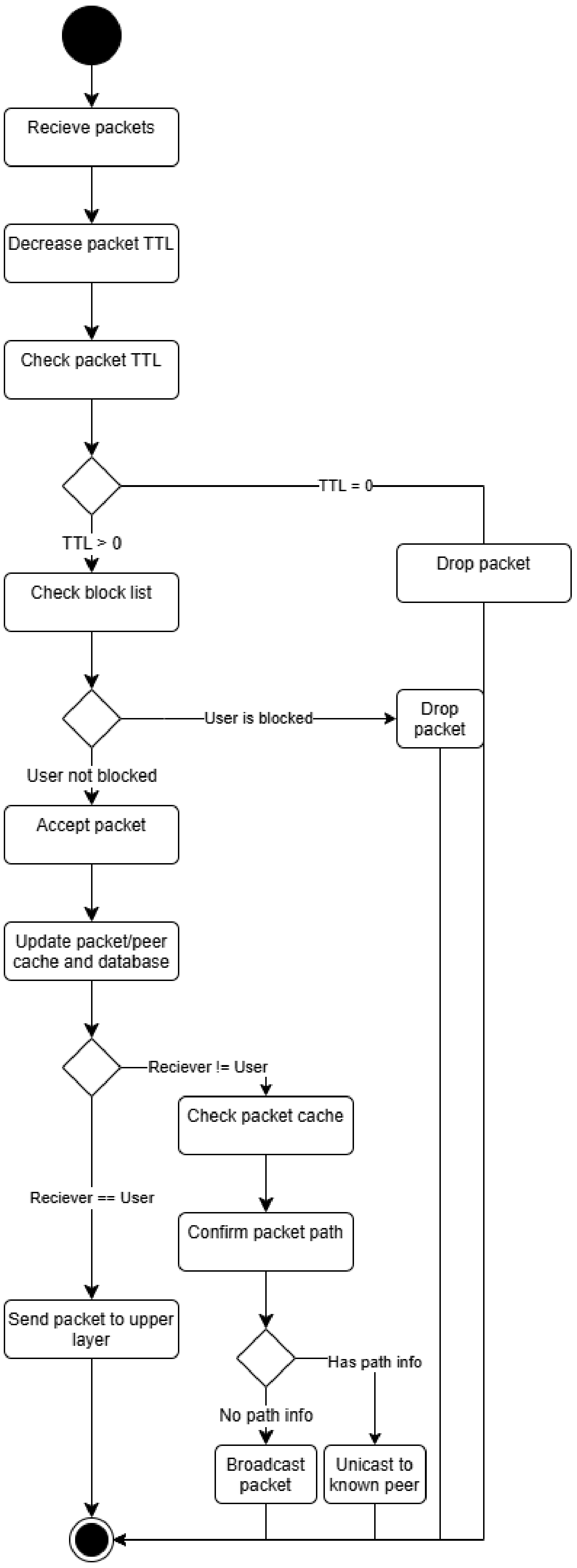


**Figure 10:** *Activity diagram: Process message*

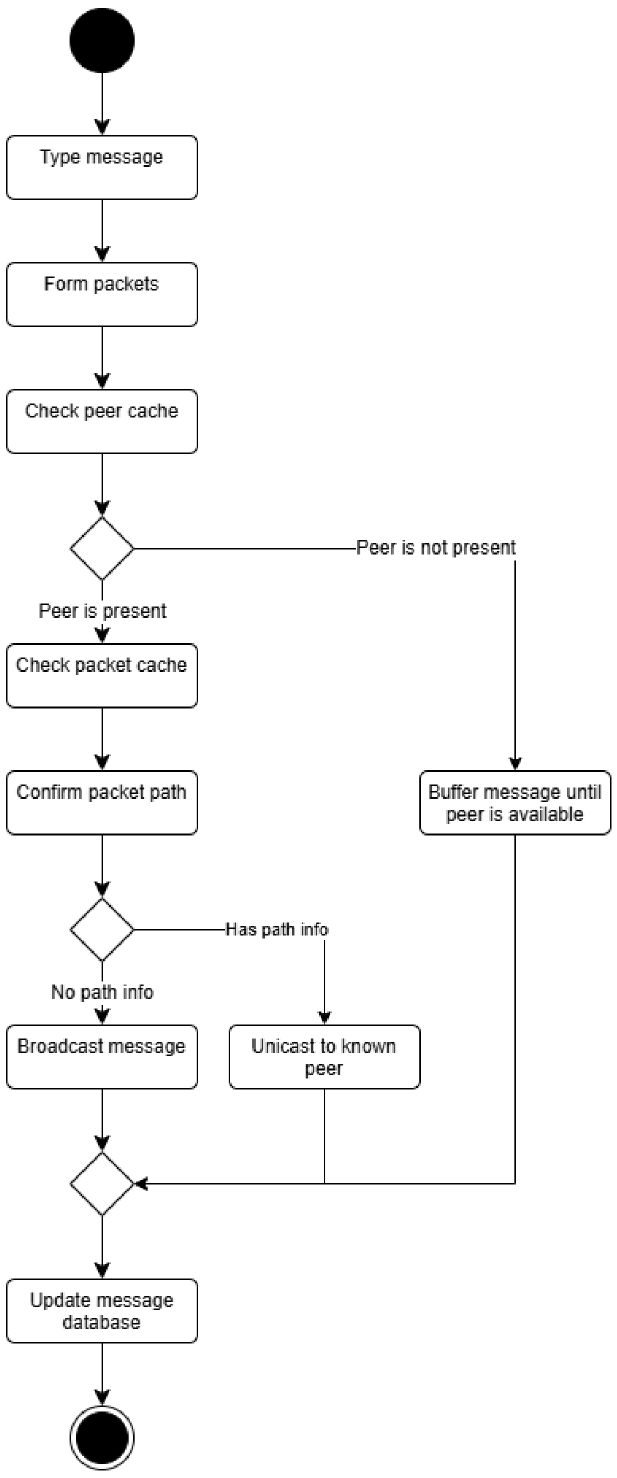


**Figure 11:** *Activity diagram: Send message*

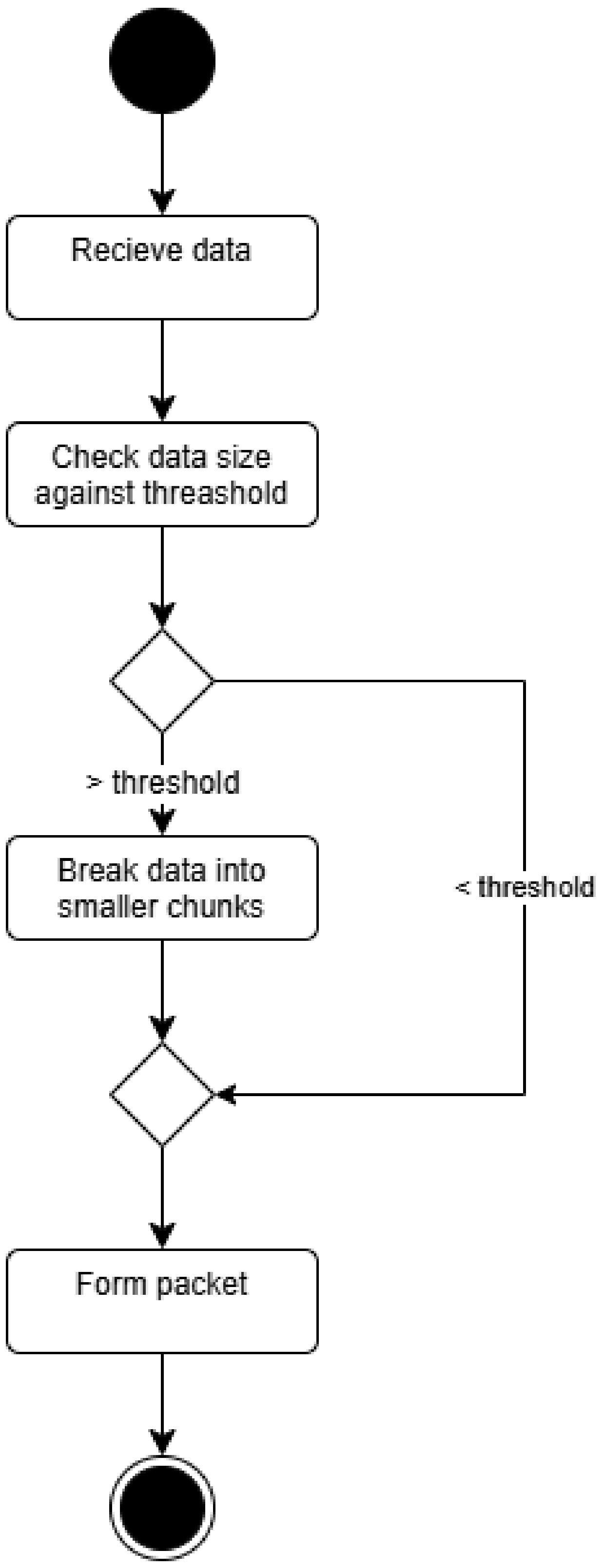

Recieve data
Check data size against threashold
> threshold
< threshold
Break data into smaller chunks
Form packet

**Figure 12:** *Activity diagram: Fragmentation*

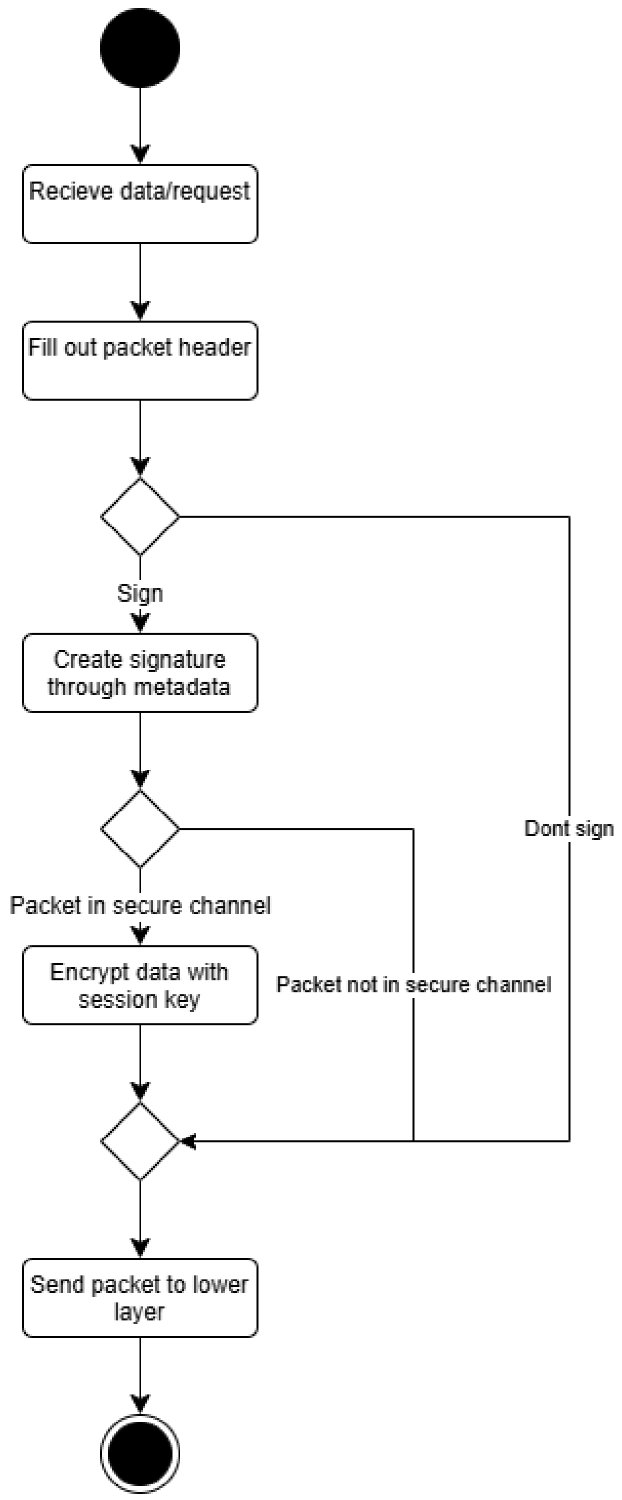
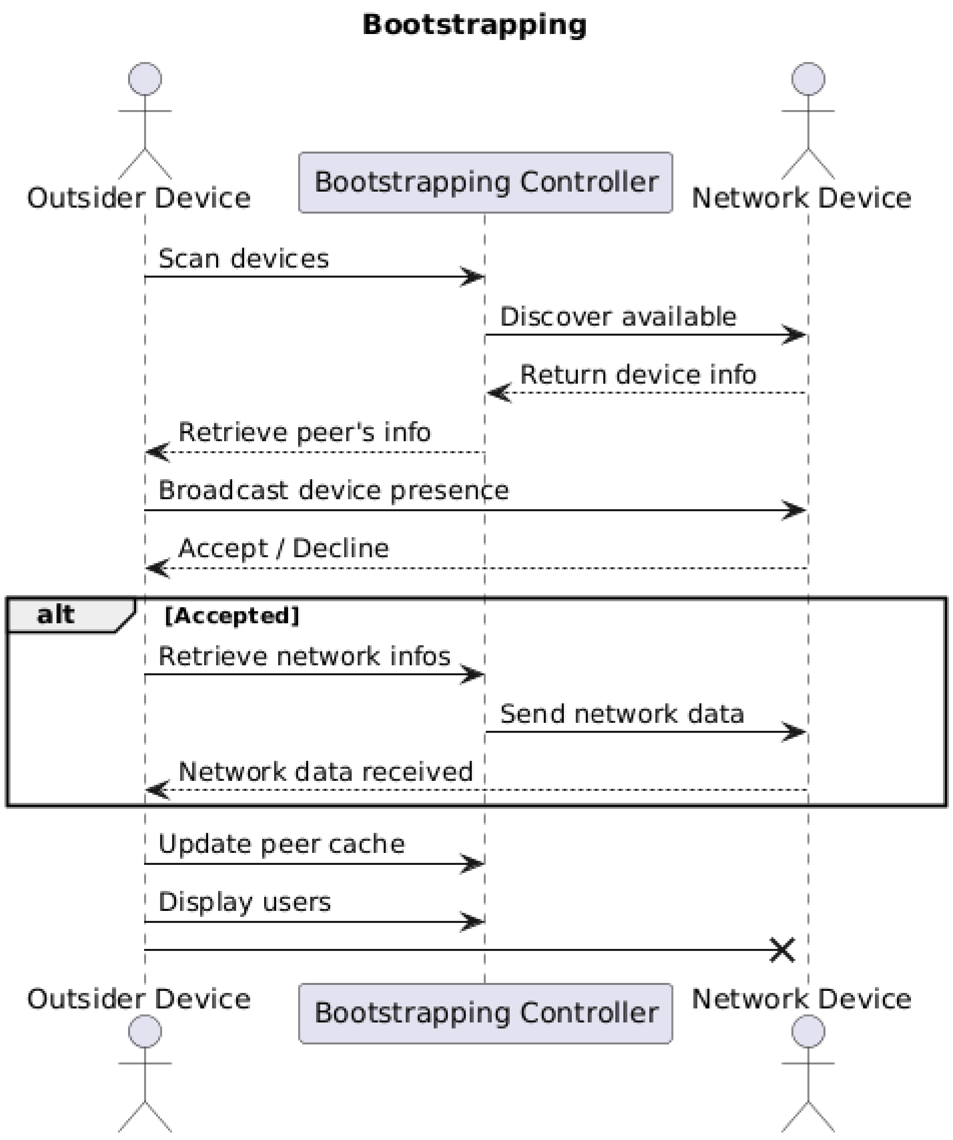
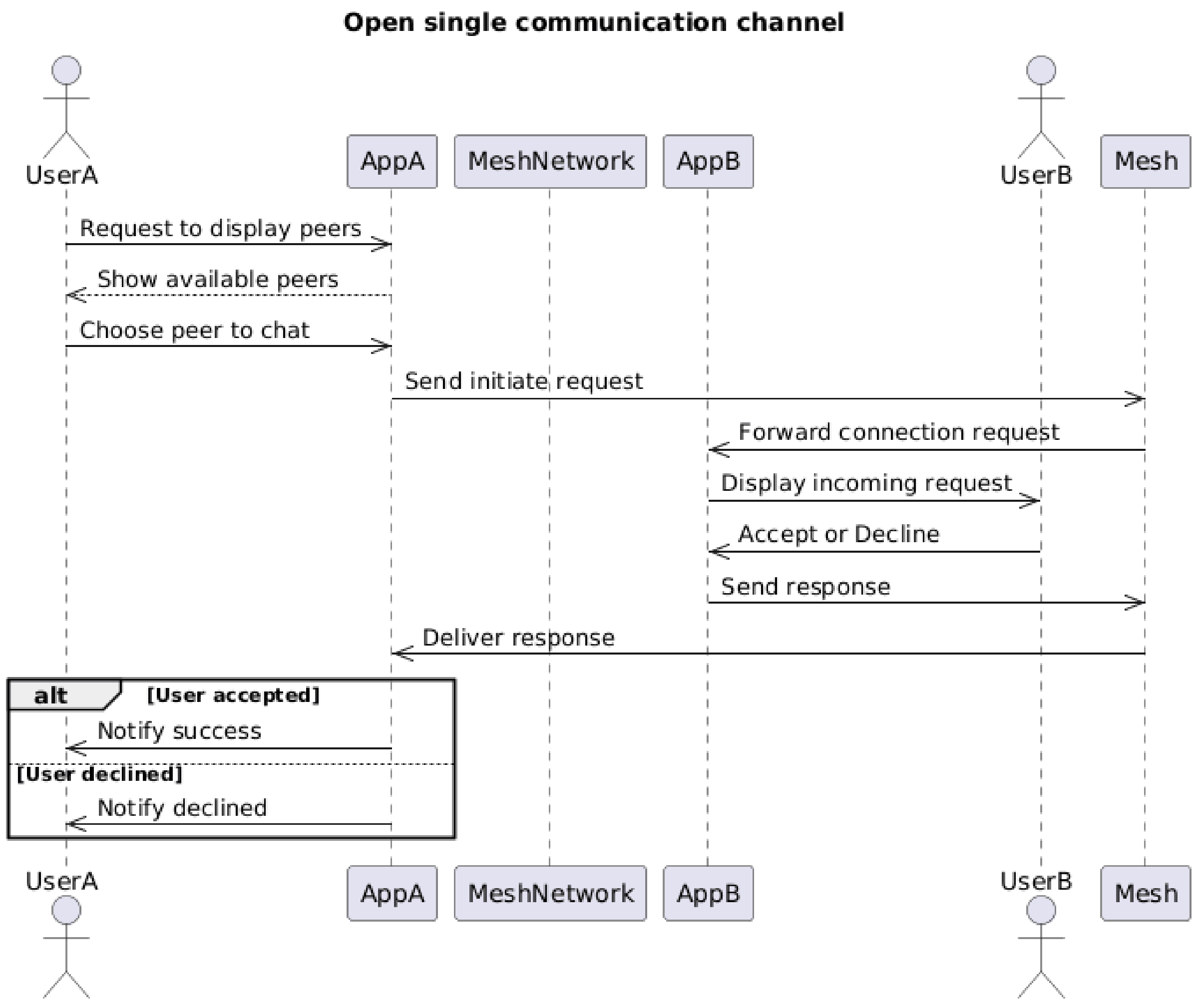
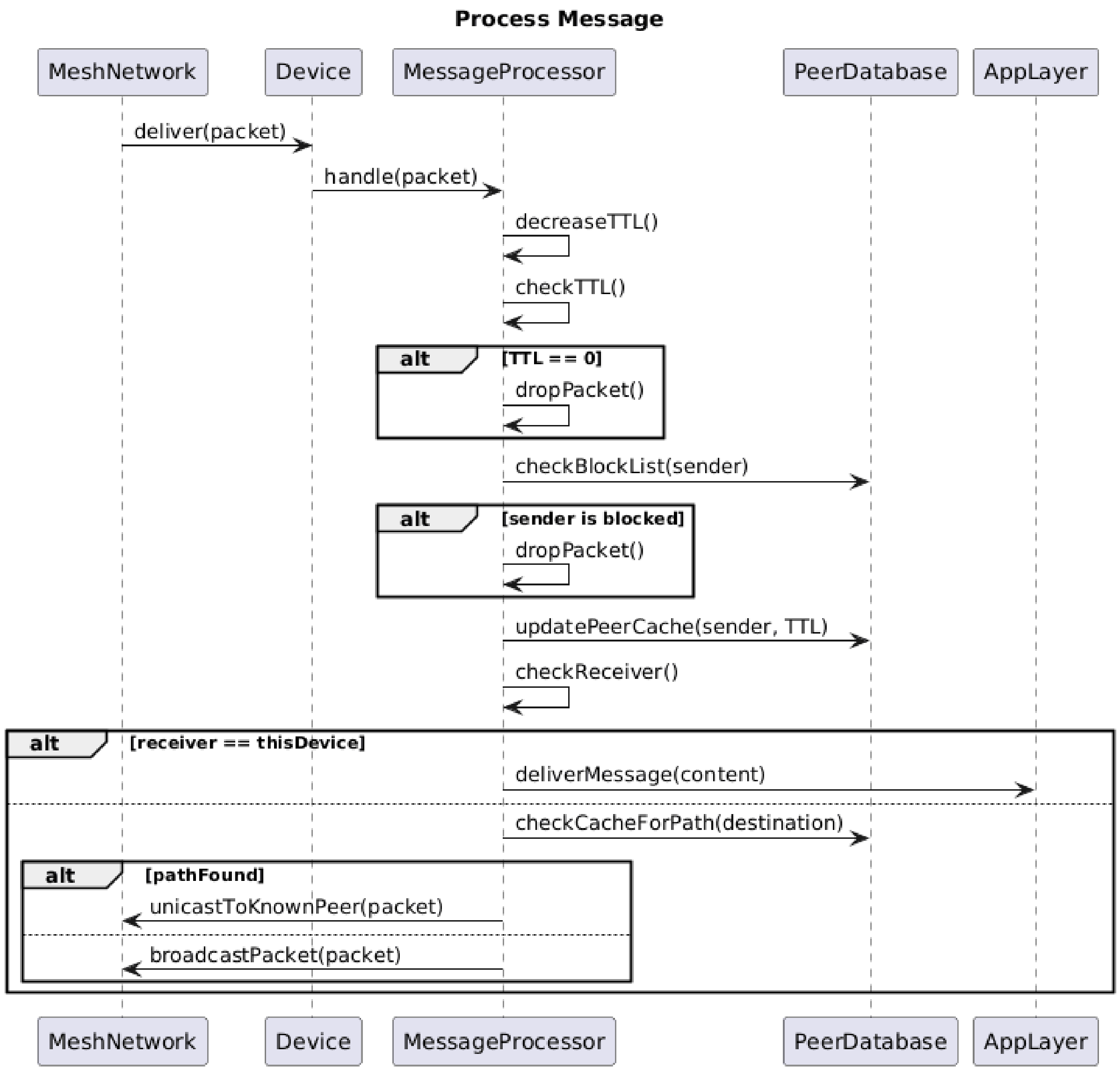

Recieve data/request
Fill out packet header
Sign
Dont sign
Create signature through metadata
Packet in secure channel
Packet not in secure channel
Encrypt data with session key
Send packet to lower layer

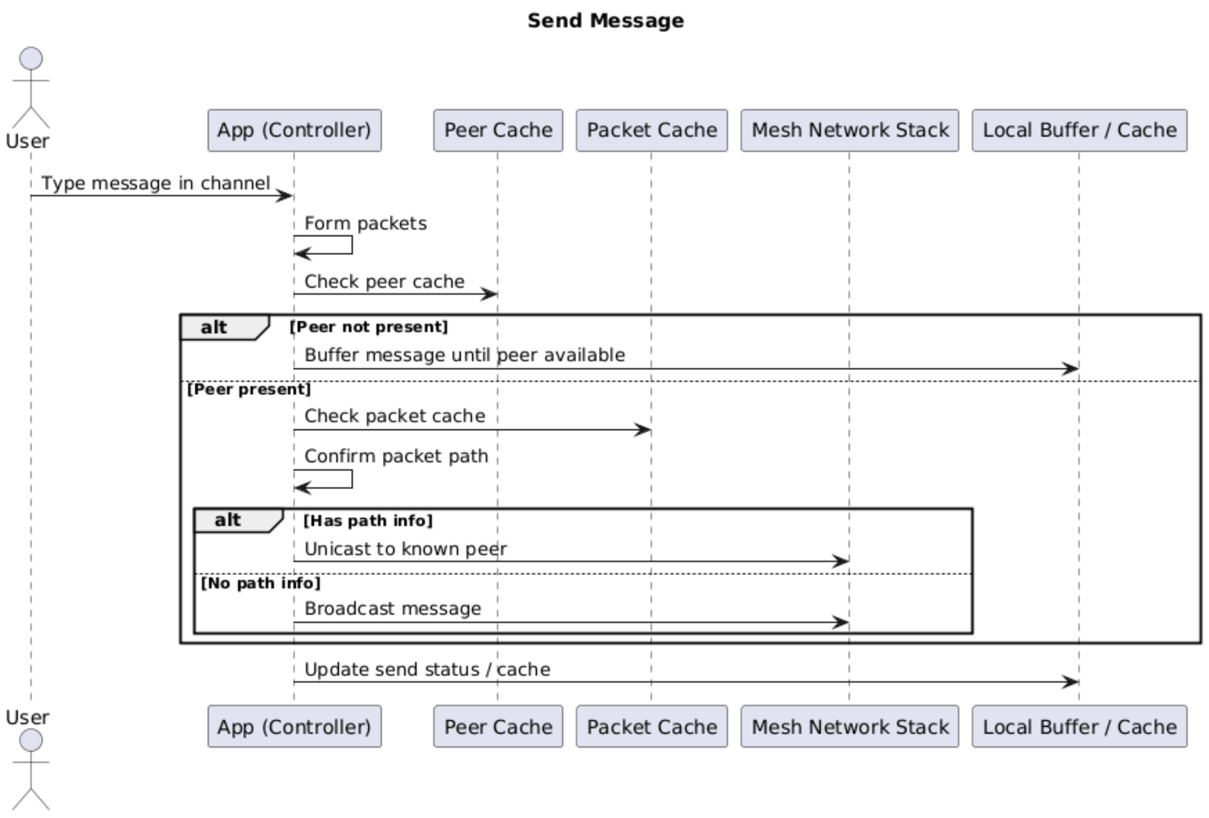
**Figure 13:** *Activity diagram: Form package*

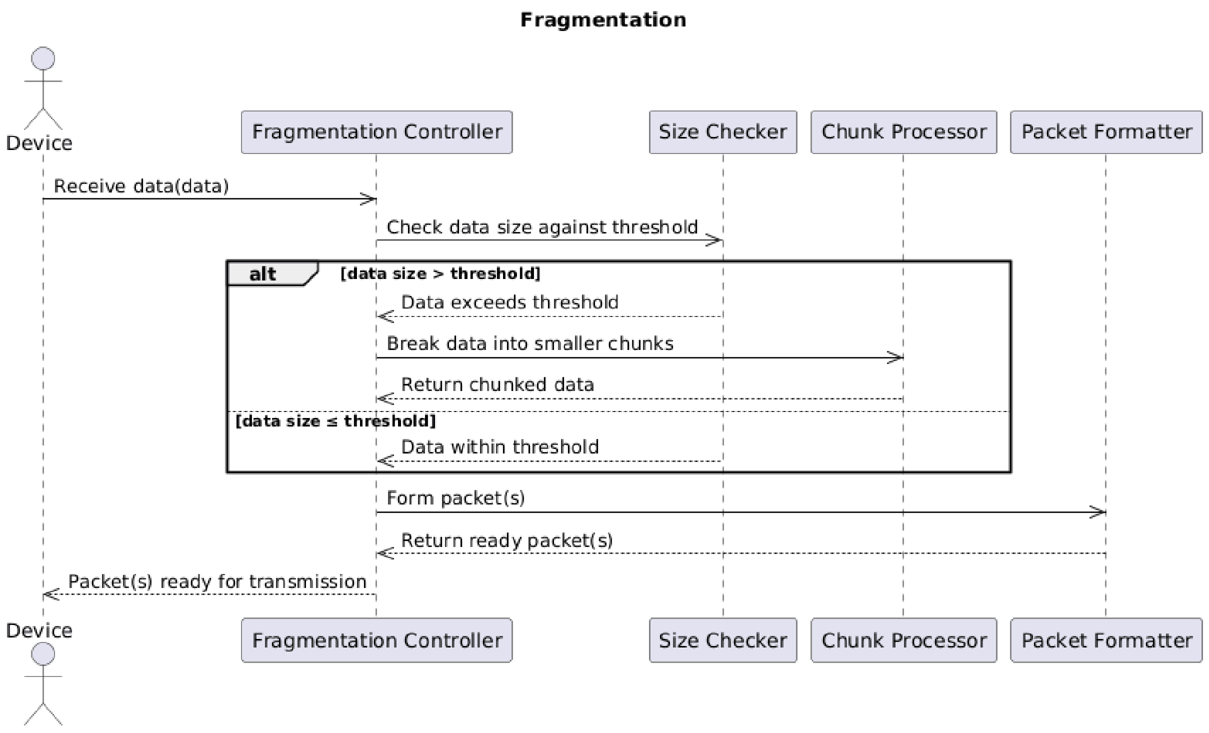
## 6.3 Sequence Diagrams

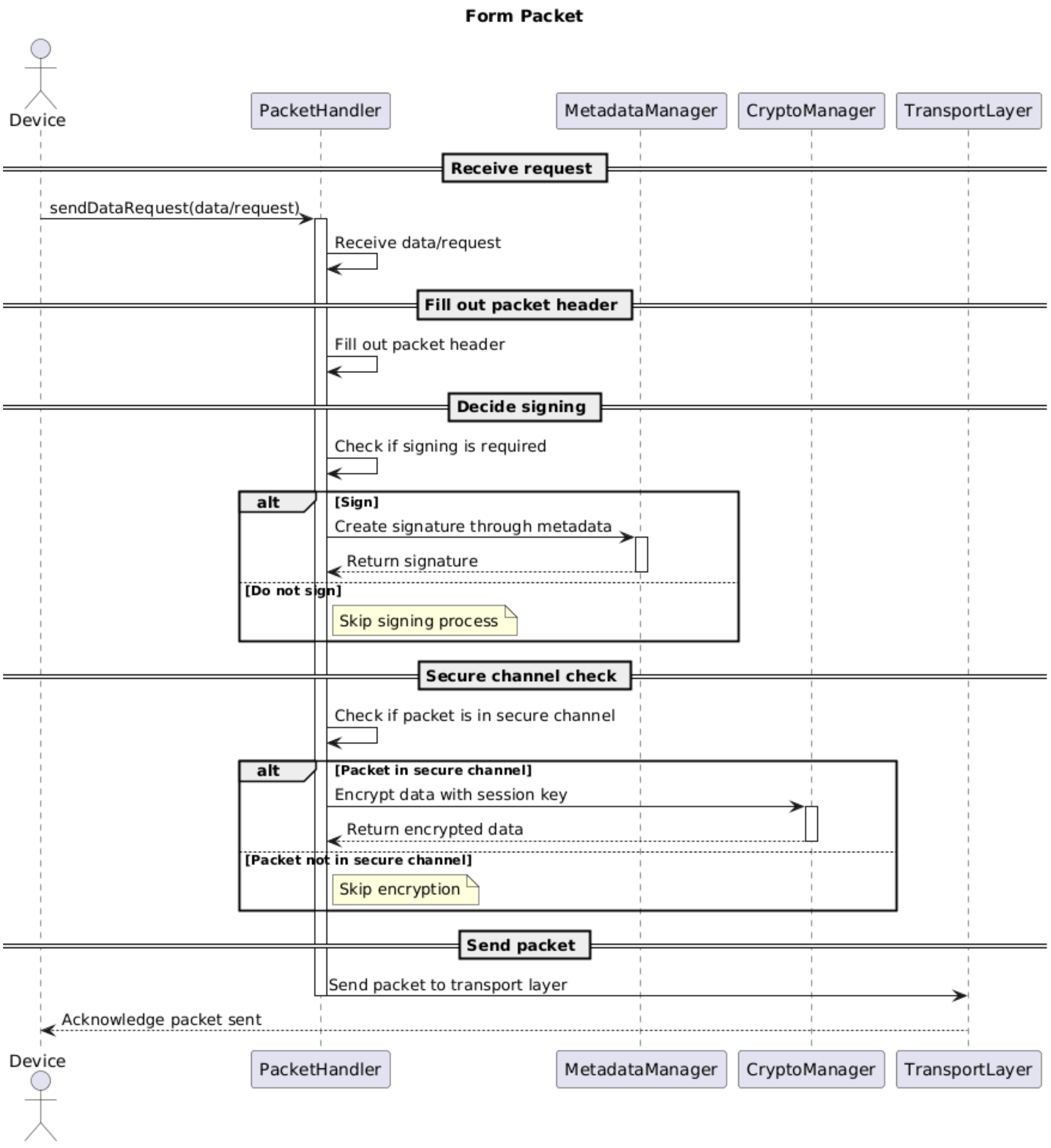



Open single communication channel
UserA
AppA
MeshNetwork
AppB
UserB
Mesh
Request to display peers
Show available peers
Choose peer to chat
Send initiate request
Forward connection request
Display incoming request
Accept or Decline
Send response
Deliver response
alt
[User accepted]
Notify success
[User declined]
Notify declined

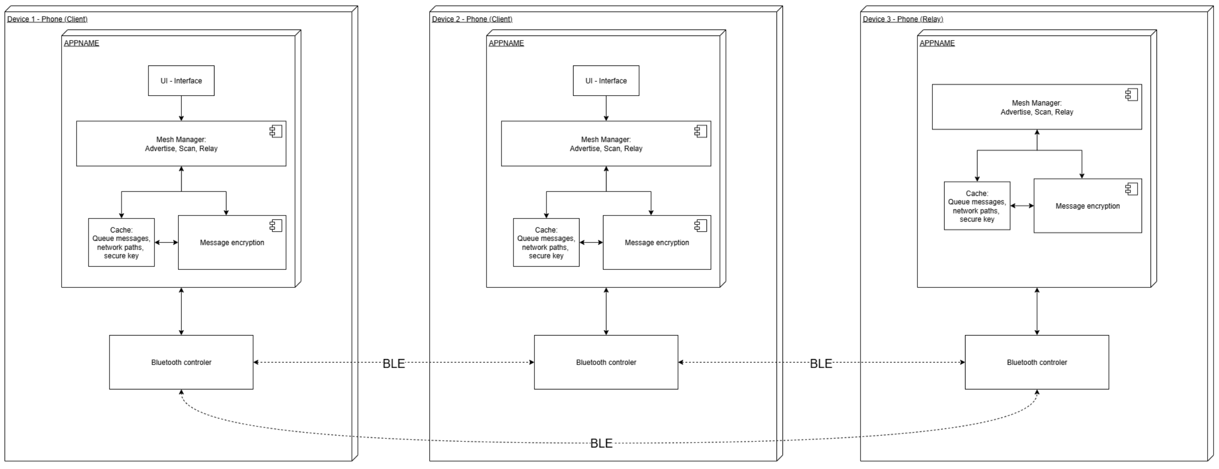

Process Message
MeshNetwork
Device
MessageProcessor
PeerDatabase
AppLayer
deliver(packet)
handle(packet)
decreaseTTL()
checkTTL()
alt
[TTL == 0]
dropPacket()
checkBlockList(sender)
alt
[sender is blocked]
dropPacket()
updatePeerCache(sender, TTL)
checkReceiver()
alt
[receiver == thisDevice]
deliverMessage(content)
checkCacheForPath(destination)
alt
[pathFound]
unicastToKnownPeer(packet)
broadcastPacket(packet)


Send Message
User
App (Controller)
Peer Cache
Packet Cache
Mesh Network Stack
Local Buffer / Cache
Type message in channel
Form packets
Check peer cache
alt
[Peer not present]
Buffer message until peer available
[Peer present]
Check packet cache
Confirm packet path
alt
[Has path info]
Unicast to known peer
[No path info]
Broadcast message
Update send status / cache


Fragmentation
Device
Fragmentation Controller
Size Checker
Chunk Processor
Packet Formatter
Receive data(data)
Check data size against threshold
alt
[data size > threshold]
Data exceeds threshold
Break data into smaller chunks
Return chunked data
[data size ≤ threshold]
Data within threshold
Form packet(s)
Return ready packet(s)
Packet(s) ready for transmission


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

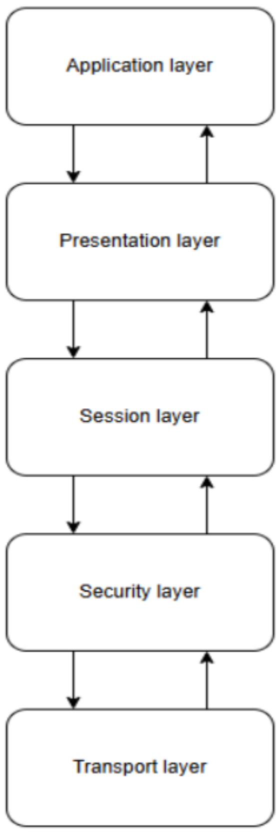
## 6.4 Deployment diagram



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 package while the encryption engine keeps things private.

# 7 System Design

## 7.1 System Architecture



The system is structured into five interacting layers, each responsible for a set of tasks:

- **Application layer**
  - Provide user interface
  - Manage notifications
- **Presentation layer**
  - Handle formatting
- **Session layer**
  - Maintain active peer connection across the mesh

  - Control session establishment, routing, and sequencing between nodes
  - Manage acknowledgements, retransmission, and relay within the mesh

- **Security layer**
  - Implement end-to-end (E2E) security using cryptographic protocols
  - Manage keys, authentications, and data integrity

- **Transport layer**
  - Provide the BLE transport function
  - Handle advertising, scanning, connection establishment, and packet delivery

This architecture ensures that each function of the app is independent and replaceable, enhancing scalability and maintainability.

## 7.2 Technology Selection

### Why Kotlin?

On the Android platform, Kotlin provides the most control over BLE and background executions, which is critical for reliability and battery life. Kotlin is also a modern and productive platform with a wide range of APIs and available documentation. Scaling path is also a big reason why we chose the platform; the app can be upgraded or moved to iOS with minimal work.

#### Other Solutions

Some other solutions that can provide similar services have many disadvantages that our system can overcome.

- **Conventional Internet-based messaging apps:**
  - Require internet or mobile data.
  - Expose user metadata and routing paths to service providers.
  - Stop functioning in offline or low-connectivity environments (disasters, protests, rural areas).

- **Local Wi-Fi or Hotspot:**
  - Only provide short-range offline messaging.
  - High power consumption.

  - Often requires manual connection.

- **SMS or Radio communication:**
  - Depend on pre-existing infrastructure.
  - Not end-to-end encrypted, easily monitored or intercepted.
  - Limited in data capacity and flexibility for file or image sharing.

#### Why Bluetooth Mesh is Superior

- **Fully Decentralized:** No servers, SIM cards, or internet needed. Every device becomes a node and relay, ensuring communication continues even when infrastructure fails.
- **End-to-End Encryption:** Messages remain private and secure, even when relayed through other users' phones.
- **Energy-Efficient and Automatic:** Uses Bluetooth Low Energy (BLE) for background operation with minimal battery drain.
- **Resilient and Scalable Mesh:** Messages hop through nearby peers; the more peers, the more reliable and powerful the system becomes.
- **Privacy by Design:** No centralized storage or tracking; data stays local on the device.
- **Perfect for Offline Scenarios:** Ideal for natural disasters, remote areas, festivals, protests, or emergency conditions.

## 8 System Implementation

# 9 Conclusion

## 9.1 Achievements

## 9.2 Limitations

## 9.3 Further improvements