Using Flutter + Dart
Even if you don't need rescue, hold out the phone so SOS messages can pass through!

For Natural Disasters Like ⬇️
Earthquakes, Tsunamis, Volcanic Eruptions, Floods, Hurricanes, Tornadoes, Wildfires

Three Level
Red: Critical
Yellow: Injured
Green: None

The user cannot edit the messages to be sent (can only edit the options)

iOS Section

iOS supports iPhone 6 and above

iOS 17+ up to 527 bytes (524 bytes) (13, 14, 15, 16, 17)
iOS 15- 16 up to 247 bytes (244 bytes) (8, x, 11, 12)
iOS 10 - 14 up to 185 bytes (182 bytes) (6s, SE, 7)

iOS Low Power mode cannot be forced; developers cannot programmatically override an iOS device's system-level power saving modes. The app should detect if the iPhone is in Low Power Mode to notify the user that they must turn it off manually for the mesh functionality to work reliably. isLowPowerModeEnabled / NSProcessInfoPowerStateDidChange 🪫

Epidemic Routing (Store-and-Forward)

1. User A (Trapped) posts an SOS.
2. Use B (Walking by) comes within range. Their phone silently downloads User A's message.
3. User B keeps walking. They walk until they get cell service or meet a rescuer.
4. User B's phone automatically uploads the message to the Cloud/Rescue.r

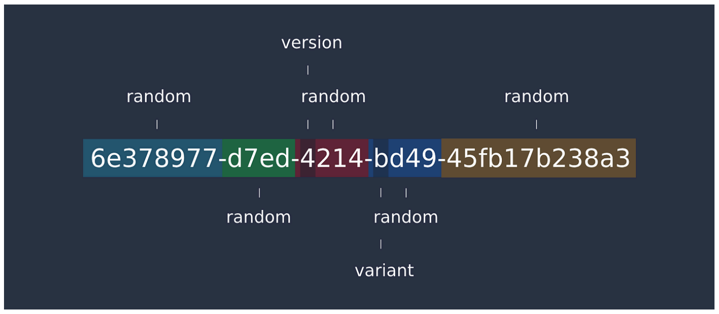
The application automatically activates SOS for mesh networking mode when it detects that the internet connection to Google has failed. 🛜 🆘

Bluetooth Low Energy (BLE) (10 - 30 or 30 - 100 ft (30.48 m) indoor) (100 - 400 meters + or 330 - 1300 ft (396.24 m) open air)
Maximum Transmission Unit (MTU) - The Largest packet size that can be sent between two devices
Android devices or iOS devices can only maintain 3 to 7 connections at the same time

Header (3 bytes)

Payload (actual data)
Android (Manual)
iOS (Automatic)

UUID (Universally Unique Identifier) is a 128-bit number used to uniquely identify information or objects in computer systems uniquely



- BLE (Open) iOS scan everything nearby (serviceUUIDs: nil) (saves scanned devices' UUIDs)
- BLE (Background) Request UUID to scan a specific device (Uses saved scanned UUIDs)

Detect if the app is in the background; if true, scan saved UUIDs. If the user opens the app again, stop the background scan, restart the scan, add new UUIDs found to your list, and remove the old ones that are no longer needed.

Integer (10^7) Formula
- Latitude x 10^7 = 40.6424741
- Longitude x 10^7 = -73.8941060

244 / 8 = 30.5; exactly 30 coordinates could be stored.

Limit BLE MTU to 244 bytes
- Android Max MTU 514 bytes
  - Android control Programmatic (requestMTU)
- iOS max MTU 524 bytes for newer models
  - iOS Automatic System-managed

An iOS device can scan for other beacons while it is broadcasting.

Enable the Bluetooth peripheral and Bluetooth central background modes in your app's Info.plist to allow continuous operation.

State Preservation and Restoration to ensure the system relaunches your app if it is terminated due to memory pressure.

iOS Always On Display: UIApplication.shared.isIdleTimerDisabled = true. This prevents the screen from dimming or locking while your app is open.

Smart Switch for iOS
Foreground (Screen Always On)
Set UIApplication.shared.isIdleTimerDisabled = true
While the app is visible, it can broadcast a standard iBeacon signal that any device (Android or iOS) can easily find

Background (Screen Off / Phone Locked)
iOS automatically sets isIdleTimerDisabled = false when the user leaves the app, Bluetooth Peripheral background mode is enabled, and the beacon continues to work.

To locate you in the background, iOS devices can only scan for you if they are also iOS devices; other iOS devices can scan for you even in the background, as they are explicitly scanning for your UUID.

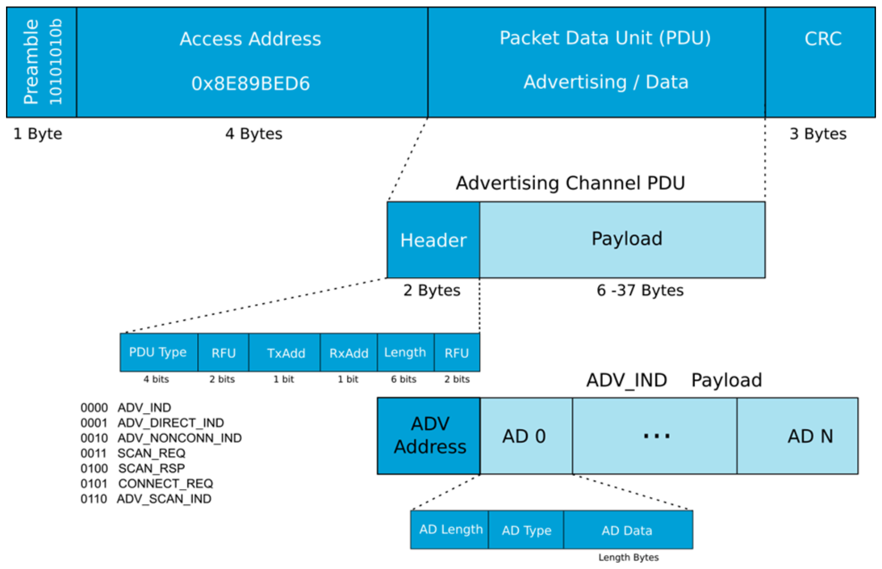
You cannot send large broadcast messages in the background. Instead, another phone must connect to your device briefly to read the messages stored in GATT characteristics.

The UUID remains constant; however, the Bluetooth MAC address rotates every 15 minutes. (Looking for a fixed Service, UUID will still recognize you)

When one phone is in the background, it stops sending messages.

Connection Features
BLE Advertising (30-100m)
BLE GATT Connection (30-100m)
Ultrasonic (1-5m)
Flashlight Strobe (Line of Sight)

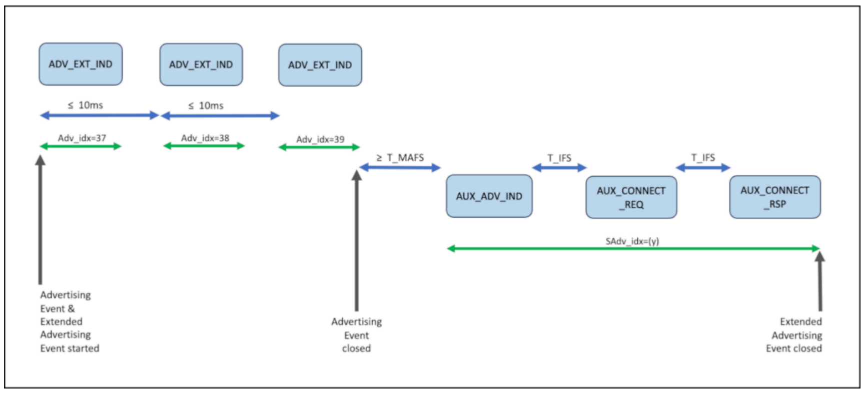
BLE



BLE (Legacy) Advertising (Bluetooth 4.x)

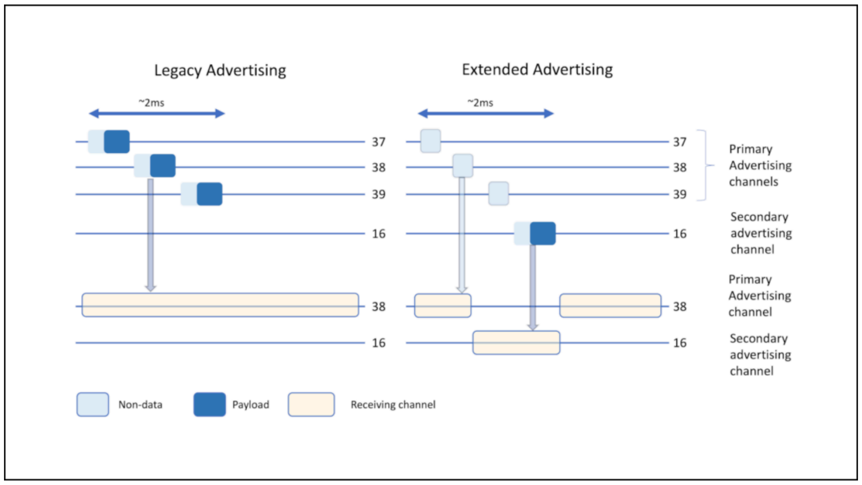
Include: A 2-byte header (Length & AD Type) plus the actual data.
Channels: Only three primary channels (37,38,39)
Payload Size: Limited to about 31 bytes
Compatibility: Supported by nearly all BLE devices.
Use Case: Simple, basic broadcasting

Extended Advertising (Bluetooth 5.0+)



Channels: Use primary channels for small header, then offloads larger data to secondary channels (37 data channels)
Payload Size: Up to 255 bytes in a single packet, or over 1600 bytes with packet chaining
Compatibility: Requires Bluetooth 5.0 or later on the scanner side.
Use Case: Richer sensor data, asset tracking, firmware updates, and bright lighting.

Packet Chaining: Breaking large amounts of advertising data into smaller packets and linking them together.


Legacy Advertising
Extended Advertising
~2ms
~2ms
37
38
39
16
38
16
Primary Advertising channels
Secondary advertising channel
Primary Advertising channel
Secondary advertising channel
Non-data
Payload
Receiving channel

Android Sections

Android nearby connection (100 meters)

WifiManager.isWifiEnabled() for Android to check the status of the device's Wi-Fi. If enabled, and Google's ping isn't responding three times in a row, active SOS mode is triggered. 🛜 🆘

iOS doesn't allow this ^

The app requests that the user exit Android battery optimization settings, because it restricts background activity, which interferes with the app's ability to maintain a mesh connection or listen for SOS messages reliably. 🪫

Google Nearby Connections: Android to Android

Undecided Features

Handle Spam vs Real
^
Cannot block Spam completely without risking blocking a real person
^
An SOS signal is marked Unverified by default. If three other phones in the nearby mesh detect the same SOS signal, they can confirm it. Rescues prioritize confirmed over single isolated signals (Risky)

Everyone is a Carrier: Phone A meets Phone B, they swap all encrypted SOS packets they have. Now both phones carry both packets.

When a phone detects an internet connection, it uploads all packets. Once uploaded, the rescue server sends back a tiny Acknowledgement. When phone B receives this Acknowledgement, it deletes the packet from its storage to prevent it from bouncing around forever.

Foreground / Background
Foreground: The Device acts as a beacon that other phones (iOS or Android) can find and read.

Background: When the app is closed or the screen is locked, standard iBeacon broadcasting stops. Custom BLE continues, but has been relocated to the overflow area. Only other iOS devices specifically scanning for your app's service UUID can find it; most Android devices will not see it

To check internet connectivity in apps, Android uses the ConnectivityManager and NetworkCallback classes for real-time status updates. In contrast, iOS uses NWPathMonitor (modern) or SCNetworkReachability for robust checks; however, the most reliable method across both is to ping a server, such as Google.

Phone 2 Rescue
1. Pressed the button: SOS Active
2. First Hop: Signal Picked Up! Carried by 1 Nearby Node
3. Acknowledgment: Confirmed. Rescue Team has your location.

Phone 2 Phone
1. Phone A sends an SOS packet
2. Phone B receives it and checks the file to ensure the data isn't corrupted.
3. If the data is good, Phone B sends a tiny acknowledgment message back to Phone A
4. Phone A receives that acknowledgment (sendPayload) (onPayloadTransferUpdate)
5. Every time the phone successfully handshakes with a new person, the counter goes up.
   ^
   5 mins: Connected to iPhone 15. Handshake Complete. (2 copies)
6. If it sends the packet but doesn't get the Ack back, in 10 seconds, it assumes the transfer failed

Determining Proximity ("Hot or Cold") ♨️ 🧊

RSSI (Received Signal Strength Indication)
- RSSI jumps around
- Option 1: Kalman Filter
- Option 2: Simple Moving Average (SMA) or Exponential Moving Average (EMA)
  - Convert RSSI (dBm) to Meters (d)
    - Distance = 10((MeasuredPower - RSSI)/(10*N))

        - RSSI value at exactly 1 meter usually –69 is a good default
        - N (Environmental Factor)
            - 2 = Open Space (Field)
            - 4 = Rubble/ Buildings
            - N = 3 as a Disaster Average

Determining direction 🧭

- ~~Angle of Arrival (AoA) and Trilateration~~
    - ~~It requires Bluetooth 5.1+ hardware arrays and driver support~~
    - ~~It requires three static beacons with known GPS locations to pinpoint a 4th target~~
- Solution: LocBLE relies on humans to be the scanner (Sensor Fusion) (Compass/Gyroscope)
    - By using flutter_compass to get the phone's heading (0 to 360)
    - The user holds the phone and spins in a circle
    - At specific degrees, RSSI is -90 (Weak)
    - At specific degrees, RSSI is -60 (Strong)

Heatmaps 🗺️

- (City View) One giant heatmap with the number inside
- (Street View) A giant circle splits into different smaller circles with the number inside
- (Close Up) Individual dots of specific people
- Able to manually filter it

Deduplication

- Duplicate, save the first UUI, D remove duplicate UUIDs

Chatting with a nearby device??

Manage Flooding

1. Keep a simple list in memory
2. Check the ID in the list when the message arrives
3. If seen before, ignore it
4. If new, then process it and rebroadcast

How to prevent messages from being sent and traveling infinitely around the world?

If 100 people hear SOS at the same time and they all re-shout it instantly, the radio waves jam. Beforere-shouting, wait a random time.
^
Await Future,delayed(Duration(milliseconds:Random().nextInt(500)));

How the Data Moves

It is called Manufacturer Data. When your phone shouts it sends a tiny packet of public data that anyone can read without asking permission

- SOS Packet might look like this:
    - UUID
    - Payload: HELP|LAT|LON|ID

Phone A: Broadcasts this packet 3 times a second
Phone B: Scanning / Listening
Phone B: Sees the packet floating in the air reads the text and saves it to the database
Phone B Starts broadcasting it

If 100 people are near

- You can't handle 100 people at once
- So you put them in a line (Queue) Start sending to the one with the strongest of signal
- Process 1,2,3,
- Then process 4,5,6

Loop Alogirthm

- Your phone listens to air waves for few seconds
- It hears 50 shouts from nearby phones
- Look at the list of 50 messages, ignore the ones you have seen before
- Saves those messages to local database (SQLite)
- Now you become the shouter
- Take messages from database and broadcast them

Public SOS: unlocks and shows the info
Private DM or someone specifically: It stays locked. App saves it and rebroadcasts it (user never sees what it says)

Sender Code

- Broadcast Manufacturer Data
- Content: [MyID, MyGPS, Status]
- Change content every minutes

Receiver Code

- Scan with Services: [ResQ_UUID]
- Hear packet
- Is it new? Save to DB
- Is it old? Ignore
- Add it to the Shouted queue

Automatic Triggers (Smart Switch)

- Mesh wouldn't run 24/7
- API Alert: background watcher sees Magnitude 6.0 Earthquake = Auto Start Mesh
- ~~Internet Loss: If the phone tires to ping Google and fails for 1 hours = Auo Start Mesh~~
- Manual Trigger: user can press activate rescue mode

Power Saving Mode Scan!!!

Andorid Solution

  Ignore Battery Optimzations

    Allow App to stay connected in the background

  android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS

When user clicks allow, the app gets permission to keep scanning

iOS Solution

  Warm inside the app: Warning: Power Saving detected. Please disable it for the Mesh to work reliably

Automatic Turn on Before outage

App runs tiny background tasks every 15 minutes using Work Manager
Check USGS Earthquake API or Weather API for warning issued in users' zip code location
Action: The app wakes up fully, sends notification: Storm detected. Mesh Mode Activated

Check connection: Ping Google (Only when app is opened) Ping fails 3 times in a row
Action: App assumes internet died. Auto activates Mesh Mode

Smart Detection (Only work for Android)

Check WifiManager.isWifiEnabled() returns False

  Do nothing

Check WifiManager.isWifiEnabled() returns True Ping (Google) Fails

  Activate Rescue Mode

Auto Start

Android Boot Receiver

- When phone restarts, Android sends signal BOOT_COMPLETED
- Catches this signal and silently starts a background Listener service. User never sees the app open, but it is running (Privacy Issue, and Battery)

iOS Silent Push

- USGS detects an Earthquake or Hurricane Warning from NOAA
- Server sends a special APNs Notification with content-available: 1
- Wakes up every iPhone with your app installed in the background for 30 seconds (without turning on the screen)
- App have 30 seconds to quickly turn on Bluetooth Scanner. Check for SOS signals and Post local Notification

Most rescue apps are One-Way. This will be A Two way

- Upstream (SOS): You → Nearby → Rescue Team
- Downstream (Command): Rescue Team → Neighbor → You

Target Message

- Rescue Team select your specific dot on their map
- They broadcast a packet specifically labeled for you
- The Mesh relies on everyone
- Stranger pass to another stranger until it reach you

Supply Drop / Rescue Zone (Global Broadcast) (Safety Issue) (Hold)

- Rescue Team Broadcast to All
- Packet
    - Target Everyone
    - Content: Supply GPS: 40.7, -74.0

iOS / Android check User is connected to the internet

- Check if internet actually reach the outside world
- Package: connectivity_plus
- Package: internet_connection_checker_plus

Android: Allows you to listen to stream even when the app is in the background using Work Manager task

iOS: Prescription about background checks
Strategy: Cannot run this loop forever in the background on iOS, run this check immediately when the app launches or when silent push wakes it up.

When detected Internet
Store and Forward Strategy

- When the apps scan BLE SOS, do not upload it immediately.
- Save it instantly to the local database
- Mark record as synced: false

Upload and Delete

- App asks the local database to give all records where synced: false
- Sends a POST request to your server one by one
- Waits server to reply with 200 OK
- Delete/Update after getting 200 OK, it goes back to the local database and marks those records as synced: true and deletes them (When upload fails halfway, you do not delete. You keep them to try again later)

Handling Background and Foreground

- App is open: Logic above runs instantly. The moment internet returns, the map updates.
- App is Closed (Background)
- Android: Use (Work Manager) wakes up every 15 minutes to run Ping Check. If it finds internet, it runs

- iOS: to Use background Fetch, iPhone OS wakes up the app briefly when it thinks the user is likely to use it, or when the system has resources allowing it to run

Battery Saving Mode? (Able to turn on)

Eco Mode
1. Scan Window: 5 Seconds (Listen for help)
2. Sleep Window: 55 seconds (Turn off the radio, save battery)
3. Repeat

Active Time: 8% of the hour. Battery Drain: ~1-2% per hour. Total Life: 30+ Hours

Data Corruption Protection
Phone A shouts #1; Phone B hears it; Corrupted
Phone B drops it (Does not save)
Phone A shouts #1 Again; Phone B hears is; Clean
Phone B processes it

Legacy / Extended Hybrid Plan
Default: Legacy (31 Bytes) Guaranteed to work on every phone.
Optional: Extended (255 Bytes)
The Warning: Enabling Extended Mode means old phones CANNOT hear you. Use only if you have detailed info to send.

Moving Target Fix
Check both ID + Counter
Every time user's location changes >10 meters, increment a tiny number inside the packet
Packet A (10:00 AM) ID:987654321 | 1 | Loc: A
Packet B (10:05 AM) ID:987654321 | 2 | Loc: B

Hears both 987654321 | Check if counter > than previous counter
Yes: Re broadcast it
No: Ignore it
If new message 1 < 2 old messages ignore

Spreading Stop
You are rescued, or you don't need rescue: Press green I AM SAFE button
Packet ID: 987654321 | 3 | SAFE (0×00)
Neighbors hear Alice | Count: 3 > 2 | Sees Status: SAFE
Action: Stop shouting Help, update map to show green dot
Pass SAFE message too nearby to clean network

Automatic Kill Switches
1. As soon as app detects the internet: Upload data to the server and turn SOS off
2. Since Wi-Fi are back or Cellar, then they can just call 911

3. If packet is older than 24 hours, drop it

Automatic Assign Unique ID
Generation: On first app launch, generate a random 4 byte INT
Storage: Save to Shared Preferences (Android) / UserDefaults (iOS)
Re-install: Generates New ID

4 Bytes Timestamp
Unix Time (Seconds since 1970) fits in 4 Bytes (Issue: January 19, 2038, the value will overflow)

When app receives a packet
1. Read Packet. Timestamp
2. Get Phone. Current Time
3. Age = Current Time - Packet Time
4. If Age > 86400 (Seconds in 24h) Delete and Ignore

Status Codes Examples:
- 0x00: SAFE
- 0x01: SOS
- 0x02: Medical Emergency
- 0x03: Trapped
- 0x04: Need Water / Food

Broadcast Logic
Carry multiple neighbors without jamming the packet (Allowed to change the random ID)
- [Me, A, B, C]
- Time 0ms Advertise Me (0xA1…)
- Time 300ms: Advertise A (0xB2)
- Time 600ms: Advertise B(0xC3)
- Time 900ms: Advertise C (0xD4)
- Time 1200ms: Loop back to me

Note: If you receive Status: Safe packet for A, remove A from this queue

Store and Forward (Internet Sync)
Goal: Upload data when the internet returns
1. Listener: Watch for Wi-Fi / Cellular connection
2. Trigger: If connected, Ping Google
3. Action (if Ping Success): Upload Post all records in SQLite to rescue server
4. Automatically Create a new packet for Me: ID: Me | Status 0×00 (SAFE) Counter: Current Counter + 1

Echo Feedback (Know nearby people got your message)

We listen for your own voice coming from someone else

1. Shout: ID: 0xA1 | Counter: 1
2. Pone B: hears it, saves it and re broadcasts it
3. You (while scanning) hear a packet with ID 0xA1
4. Your ID and it's coming from outside signal strength (RSSI -80) That means someone else is repeating

Parsing (Binary Reading): Bluetooth chip gives a Byte Array (List of numbers)

Blueprint

Byte Index	Field Name	Size	What it holds
0-1	App Header	2 B	0xFFFF Secret handshake
2-5	User ID	4 B	Random ID 0xA1B2C3D4
6-7	Sequence	2 B	Counter: 1, 2, 3
8-11	Latitude	4 B	(Int) 34052200
12-15	Longitude	4 B	(Int) -1182437
16	Status	1 B	Code 0x01 (Hurt) 0x00 (Safe)
17-20	Timestamp	4 B	Unix Seconds (17361500)
Empty	Empty	Empty	Empty

Lag: No, Why? Bluetooth runs on a separate physical chip. It does not sue main CPU, scanning for beacons is extremely lightweight. It will not make lag.

Battery Strategy for Android

Mode	Scan (ON)	Sleep (OFF)	Battery Life	Reaction Time
SOS Active	Always On	Always On	6 - 8 hrs	Max
Bridge Mode (default)	30 seconds	30 seconds	12 + hrs	Fast updates
Background / Batter Saver	1 Minutes	1 Minutes	24+ Hours	Efficiency
Customization Mode	Custom	Custom	Depends	Customization

Mode	iOS Technical Implementation
SOS Active	Keep App in Foregorund (Disable Screen Lock) scan allowDuplicates: true

Bridge Mode	Standard Foreground Scan (Disable Screen Lock)
Battery Saver	Standard Foreground Scan (Enable Screen Lock) 30 seconds On and Off
Background	Call scan(serviceUUIDs: [YourID])

For iOS when app is moved to the background, check if user are still broadcasting to specific device, if so use UUID and background to send it.

Phone wakes up
Scan for 2 seconds, Any SOS signals
No: Goes back to sleep for 30 seconds
Yest: Switches to High Alert mode instantly

Smart Trigger
WorkManager (Android) / Background Fetch (iOS)
- Schedule a task to run every 30 minutes
- API Call: Fetch Earthquake data
- Check Is there Mag > 6.0 event in your zip code
- No: Go back to sleep

DEFCON System (Defense Readiness Condition)

Level 5: Peace
- Every 30 minutes, tiny background job fetches data from USGS and NWS APIs based on Zip Code
- No Pinging. No Bluetooth Scanning

Level 3: Warning Issued (Two time verification)
- API says “Earthquake Warning in your Zip Code”
- Verification Mode
- Wake up: launches foreground service
- Ping Google one time / Min for 10 minutes.’
- If all failed: app SOS unlock

To pass first verification one: Earthquake Magnitude 6.0+ or Hurricane Category 3+ or Tornado EF2+ or Tsunamis or Volcanic Eruptions or Flood or Wildfires

Level 1: Disaster Confirmed

99.9% of the time
- Bluetooth: OFF
- GPS: OFF

- Scanning OFF
- Only Activity: API check every 30 minutes

0.1% of the time
- Bluetooth: ON

Pocket Problem
- On Android
    - Use Foreground Service. Permanent notification in status bar (App Active)
    - When Disaster Confirmed Android Keeps the app alive in background (5 seconds every minute), pick up SOS, and vibrate your phone to tell you someone needs help.
- On iOS
    - Use Background Service UUID Scanning
    - App tells iOS: Wake up if you see service ID 0xFFFF (ID is same for everyone using the app)
    - Even if the app is closed, the Bluetooth chip stays awake. When it hears neighbor shouting on channel 0xFFFF, it wakes up for 10 seconds to handle the messages and alert user.

iOS UUID
Device UUID (identifierForVendor) unique to every phone.
Service UUID (The Channel) Not unique to the phone, unique to the APP

How it works
- Hardcode a specific ID in code, let’s say 0xFFFF. Every copy of your app, on every iOS in the world, knows this numbers
- When neighbor’s iPhone shouts SOS, it attaches serviceUUID
- iPhone told iOS to wake up if you see 0xFFFF
- Bluetooth chip sees flag. Wakes up the app
- App reads the rest of the packet and notifies the user

Both Android and iOS are listening to the ServiceUUID (Same UUID for both platforms)

Victim will have the app OPEN (Foreground) to shouts SOS
But:
Mesh chain of 3 people: Victim A → Neighbor B → Rescuer C
A opens the app and send SOS
- State: Foreground (App Open)
- Behavior: Her Phone shouts the UUID

B (The Relay): B is safe. He is just walking around. His phone is in his pocket (Locked)
- State: Background
- We need B’s phone to shout (re broadcast) even when it is locked in his pocket

C (Rescuer): Looking at the map

- State: Foreground

iOS background shout

Foreground Shout

- iPhone shouts: I am service 0xFFFF

Background Shout (App Closed / Locked)

- iPhone hides 0xFFFF UUID. puts it in a special overflow area (hidden pocket in the packet)
- It shouts: I am an Apple Device but whispers 0xFFFF secretly
- Risk of getting ignored
- Fix: Android scanner configured with Scan Filter to specifically look for hidden whisper

iOS can scan and shout at the same time in the background

- Scan: Listens for the 0xFFFF UUID (Wake up if someone sends SOS)
- Shout: Broadcast the 0xFFFF UUID (Relay A's message to C)

A: The UUID is like a Club Card. ID will be in the settings of both the Android and iOS apps

1. Open the app's settings file Info.plist and check Uses Bluetooth LE Accessories (Central): Listen in the background
2. Acts as Bluetooth LE Accessory (Peripheral): Lets your app shout in the background

B: When Shout Must attach the Service UUID to the broadcast packet

C: Scanning: Specifically, tell manner only to wake up for Service UUID 0xFFFF

For Android

A: Add permissions in the AndroidManifest.xml file

1. To use Bluetooth
2. To keep app running in the background

B: In advertising setting, add the Service UUID to the data packet
C: Scan Filter: Tell Android to Look inside the hidden data (overflow) for 0xFFFF

Locked iPhone vs. Locked Android

1. iPhone (Shouting)
    - Sends out a whisper signal
    - Signal says: I am an Apple device
2. Android (Listening)
    - Running a background Service
    - Has filter on. Catches the whisper
    - Grabs the packet and processes the SOS

Android Shouting vs. Locked iPhone

1. Android (Shouting)
    - It sends a loud signal: 0xFFFF
2. iPhone (Listening)
    - Bluetooth hardware sees the 0xFFFF
    - Wake up the app
    - The iOS app wakes up for ~10 seconds to handle the SOS

iOS Hidden Hashed UUID

Background

iOS forces the packet to change. To save space, it moves your UUID into a special section called the "Overflow Area."

It looks like this: [Flags] [Apple Manufacturer Data] [... Hashed Bitmap ...]

0xFFFF is turned into a mathematical "Hash" (a specific bit set to 1) inside that bitmap

Waking Up Loop Solution

allowDuplicates: false is FORCED to false in the background.

1. A shout 0xFFFF
2. Phone Wakes Up
3. New device (save to database)
4. A shouts 0xFFFF again
5. Ignores it

You can manually turn off the bridge

Manually disable it

The Connectivity Manager detects that Wi-Fi/Cellular is back.
Auto-Submits the SOS data to the Cloud
Automatically switches the local Bluetooth signal to Status: SAFE