29/04/2024

Braceloc

Technical Document

Florian Krasulja,
Grace Ndoko-ounounou,
Nicolas Meyer,

Engineer student, GPSE

Project tutor:
Rodolphe Weber
To:
Jury's director

Summary

Table of illustrations

1. Introduction

This project embodies a fusion of technology and practicality to ensure safety and connectivity for individuals in various outdoor scenarios, such as hiking in remote areas, school outings, or even leisure activities like mushroom foraging in dense forests.

These bracelets, when paired, allow users within a group to share and track each other's locations in real-time, even in the most challenging environments where traditional network signals fail.

Braceloc leverages state-of-the-art GPS technology embedded in wearable bracelets. Our vision is to provide peace of mind and enhance safety for our users, enabling them to locate their friends, family, or any group member instantly over considerable distances.

This document aims to articulate the intricate details of Braceloc's design, functionality, and operational mechanics. Through a clear and comprehensive presentation of technical specifications, we intend to provide a thorough understanding of the project's architecture, component integration, and user interface.

2. Project Analysis

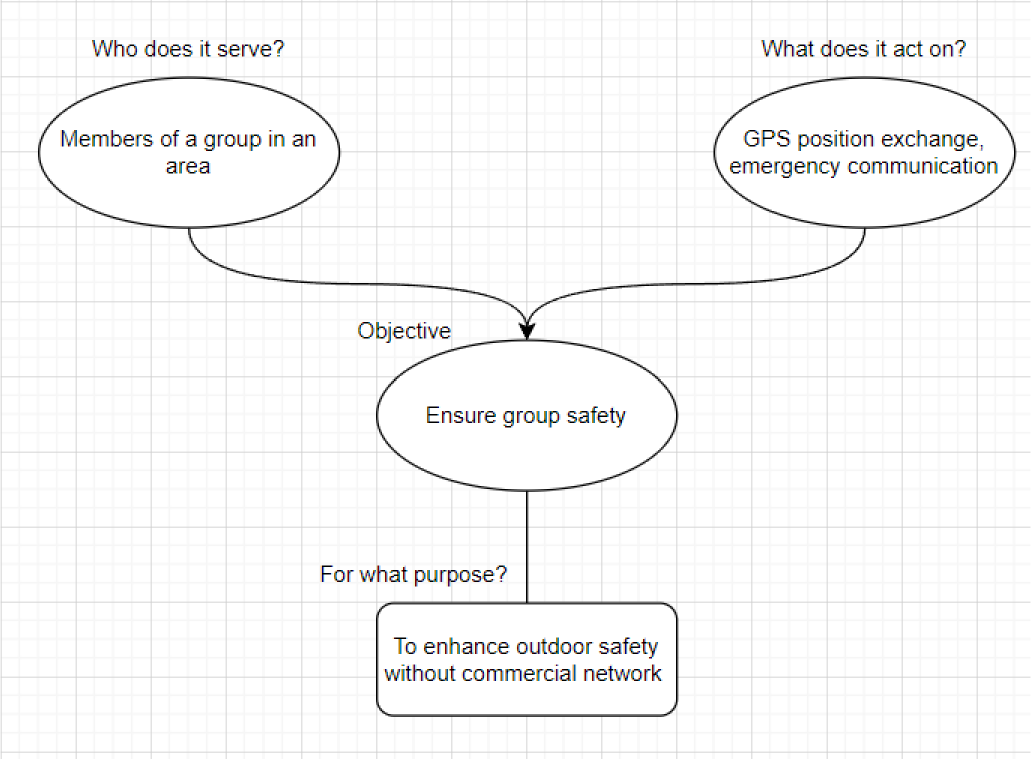

Who does it serve?
Members of a group in an area
What does it act on?
GPS position exchange, emergency communication
Objective
Ensure group safety
For what purpose?
To enhance outdoor safety without commercial network

Figure 1 : Need analysis

This innovative solution is designed to enhance safety and connectivity for groups in remote areas.

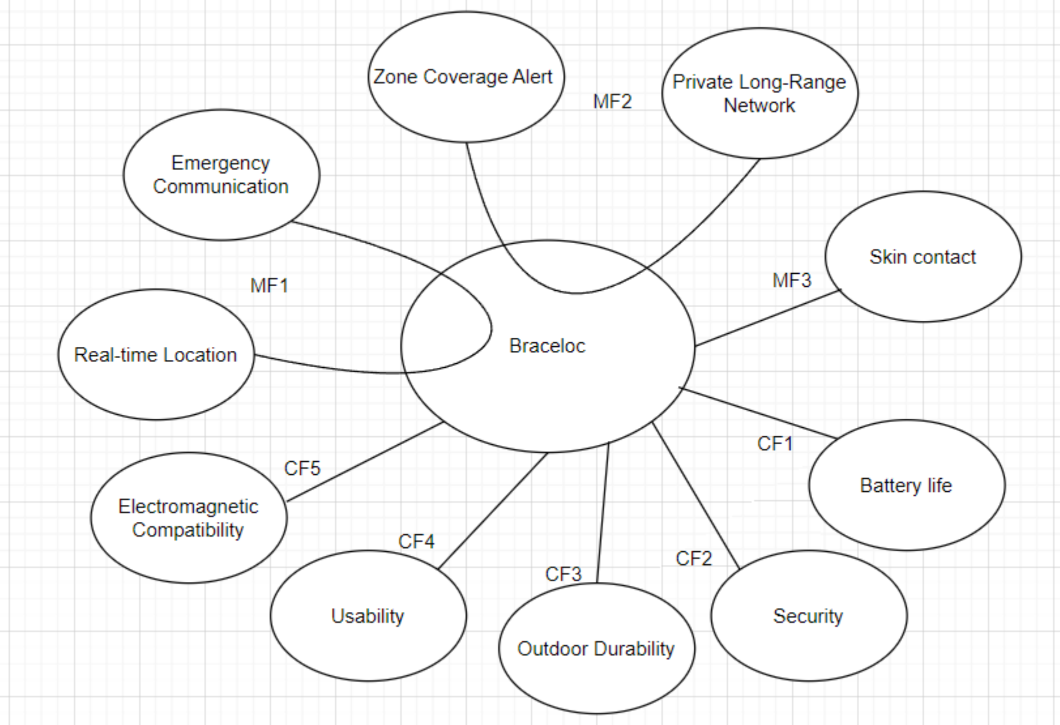


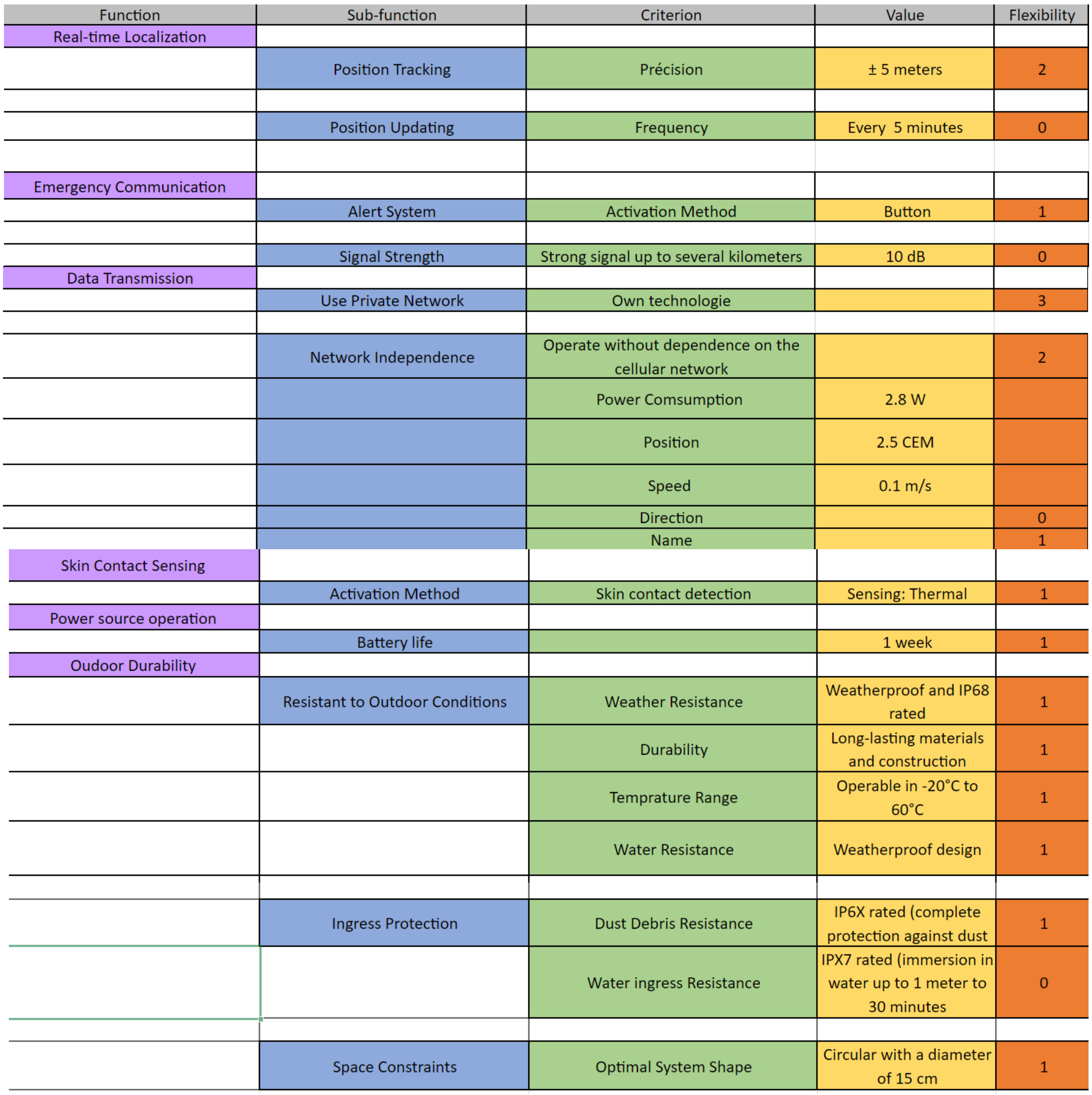
Figure 2 : Interaction graph

The provided table translates the conceptual graph into a structured format, aligning each main function (MF) and constraint function (CF) with specific criteria and priorities that define Braceloc's capabilities. The graph's interconnectivity is mirrored in the table by directly associating each function with its operational importance, from 'High' to 'Low' priority, thus establishing a direct correspondence between the visual map and the tabulated specifications. This tabular representation allows for a quick reference and understanding of how each feature contributes to the overall functionality of Braceloc, as depicted in the graph.

Category	Name	Criterion Details	Priority
MF1	Emergency Communication	Capability for distress signals, independent of cellular service	High
MF2	Private Long-Range Network	Proprietary network for long-distance communication	High
MF3	Real-time Location	Ability to provide or track the precise location in real-time.	High
MF4	Skin contact	Ensure the bracelet functions only when in contact with the skin	High
CF1	Battery Life	Long-lasting battery support	Medium
CF2	Security	Ensures communication and data security	High
CF3	Outdoor Durability	Resistant to outdoor conditions	Medium
CF4	Usability	Easy to use in various conditions	Medium
CF5	EMC (Electromagnetic Compatibility)	Compliance with EMC standards	Low
CF6	Skin contact	Temperature differential for skin contact detection	High

Following the table, we can understand that the diagram and the table collectively present a multi-faceted view of Braceloc's design. The diagram shows the relational aspect of Braceloc's features, indicating how each function and factor is a part of an integrated system. In contrast, the table provides a detailed hierarchy of these features, outlining their criteria and relative priorities.

These functions allow us to build the following specifications:



Function	Sub-function	Criterion	Value	Flexibility
Real-time Localization				
	Position Tracking	Précision	± 5 meters	2
	Position Updating	Frequency	Every 5 minutes	0
Emergency Communication				
	Alert System	Activation Method	Button	1
	Signal Strength	Strong signal up to several kilometers	10 dB	0
Data Transmission				
	Use Private Network	Own technologie		3
	Network Independence	Operate without dependence on the cellular network		2
		Power Comsumption	2.8 W	
		Position	2.5 CEM	
		Speed	0.1 m/s	
		Direction		0
		Name		1
Skin Contact Sensing				
	Activation Method	Skin contact detection	Sensing: Thermal	1
Power source operation				
	Battery life		1 week	1
Oudoor Durability				
	Resistant to Outdoor Conditions	Weather Resistance	Weatherproof and IP68 rated	1
		Durability	Long-lasting materials and construction	1
		Temprature Range	Operable in -20°C to 60°C	1
		Water Resistance	Weatherproof design	1
	Ingress Protection	Dust Debris Resistance	IP6X rated (complete protection against dust	1
		Water ingress Resistance	IPX7 rated (immersion in water up to 1 meter to 30 minutes	0
	Space Constraints	Optimal System Shape	Circular with a diameter of 15 cm	1

Figure 3 : Specifications

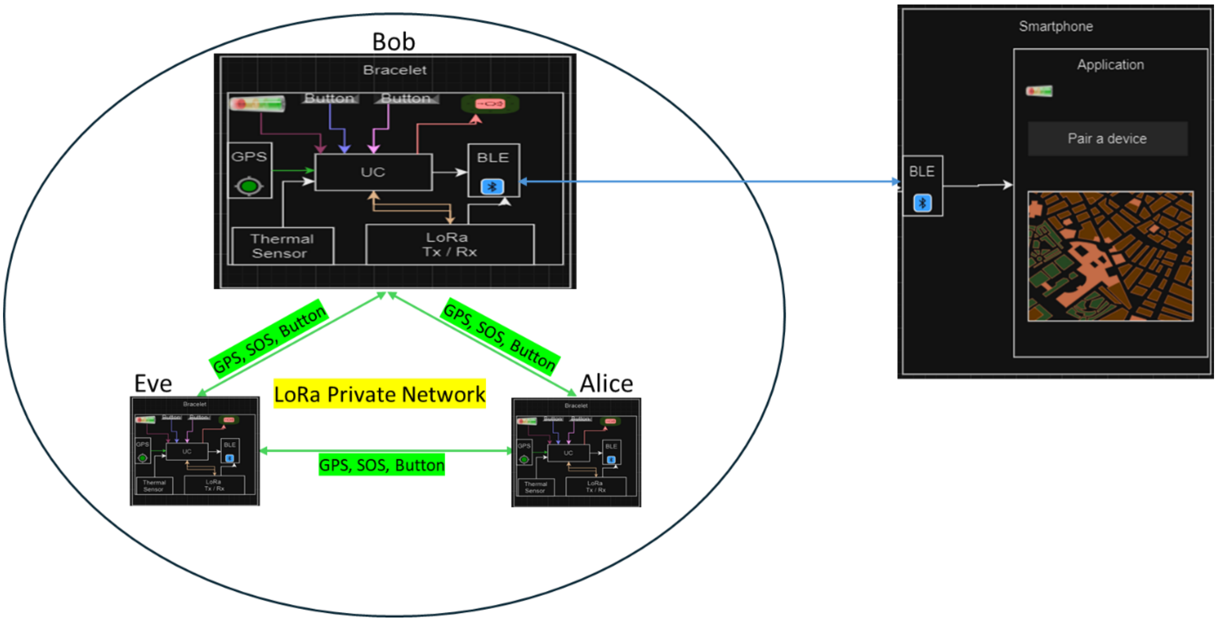


Figure 4 : System architecture

Data Transmission:

- **Thermal sensor to Microcontroller (I2C protocol):** temperature/pressure/humidity
- **GPS to Microcontroller (UART protocol):** latitude/longitude
- **Microcontroller to Application (BLE):** username/UserId/longitude/latitude/CodeParty(Host)
- **Application to Microcontroller (BLE):** username/CodeParty(member)
- **Bob and Alice (LoRa):** username/CodeParty/UserId/longitude/latitude
- **Alice and Eve (LoRa):** username/CodeParty/UserId/longitude/latitude
- **Eve and Bob (LoRa):** username/CodeParty/UserId/longitude/latitude

3. Mesh with meshtastic project

3.1 Data transmission

This technical report introduces the architectural nuances and operational dynamics of Braceloc's LoRa mesh network, engineered to foster long-range communication in a self-sustaining and autonomous manner.

The cornerstone of Braceloc's network architecture is the mesh topology, characterized by its decentralized nature and peer-to-peer communication model. Unlike traditional star-topology networks, where each node connects to a central hub, Braceloc's mesh network ensures that each wearable device can act as an independent node capable of both sending and receiving data. This intrinsic flexibility endows the network with self-healing properties; it can reconfigure itself in real-time as nodes become unavailable or new nodes join, maintaining uninterrupted communication even in dynamically changing environments.

Furthermore, Braceloc's use of LoRa technology is instrumental in surmounting the conventional barriers of distance and power efficiency. LoRa's low-power, wide-area network (LPWAN) capabilities allow Braceloc devices to communicate over extended distances, far surpassing the limitations of traditional wireless technologies, while operating on minimal power consumption. This is paramount for wearables that are expected to perform consistently and reliably without the need for frequent recharging

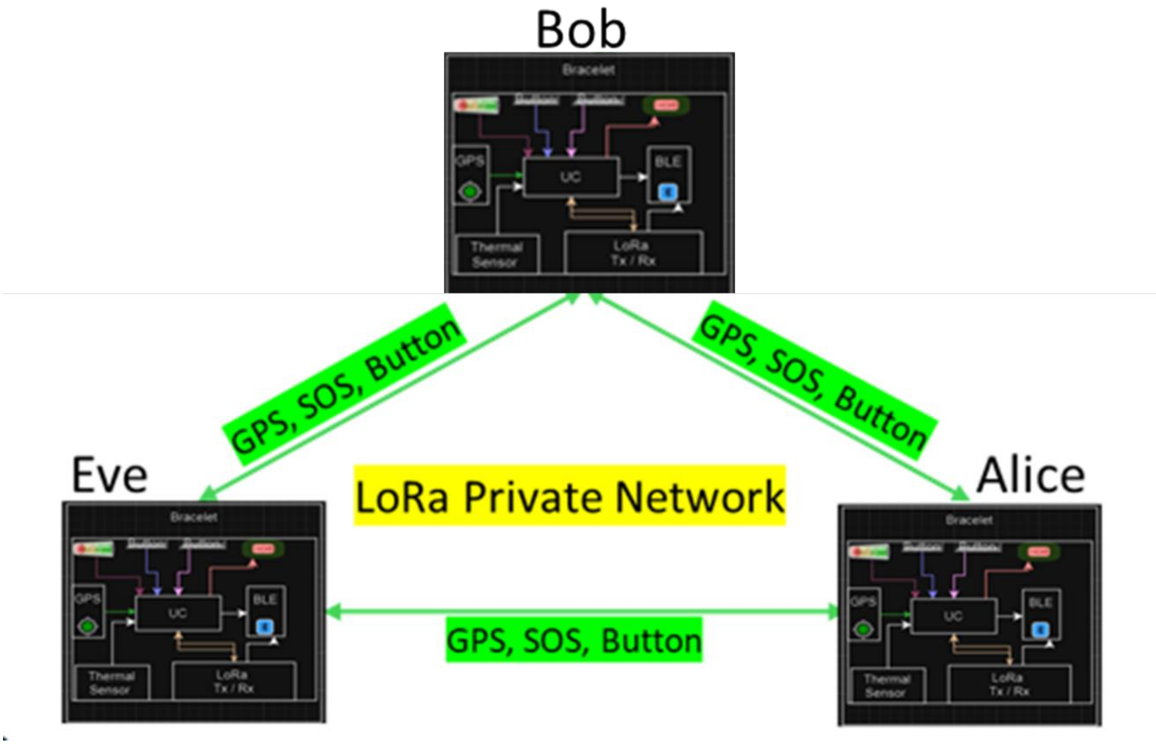


Figure 5 : Communication Lora mesh

LoRa transmission technology is designed for long-range communication with a trade-off on bandwidth. It employs a chirp spread spectrum technology which has several advantages, particularly in terms of robustness to interference and signal degradation over distance. In a mesh network, each node, such as the bracelets in your scenario labeled 'Eve' and 'Alice', acts as a transmitter (Tx) and receiver (Rx), capable of sending and receiving messages.

The LoRa modulation allows for a range that can exceed 10 km in rural areas with a clear line of sight, but in urban or dense environments, the range is typically around 2-5 km. Despite the potential reduction in range due to physical obstructions, mesh topology can overcome these limitations as data can hop from one node to another to reach its destination.

The transmission via LoRa is also characterized by its low power consumption. This is critical for wearable devices that need to operate for extended periods on limited battery power. LoRa's duty cycle—the ratio of the transmitters "on" time to the total time—is managed to comply with regional regulations, such as the European ETSI regulations, ensuring that the bracelets will not cause undue interference and will maintain a reliable communication channel.

LoRa's spread spectrum approach also provides inherent security benefits. The chirp signal is less prone to interception and jamming compared to other modulation techniques. Additionally, LoRaWAN, the network protocol designed for LoRa, includes end-to-end encryption ensuring that data transmitted across the network, such as GPS locations and SOS signals, are secure.

For the technical implementation, each bracelet's microcontroller would manage the timing of transmissions to optimize network efficiency and avoid collisions—a critical consideration in mesh networks. This is achieved through adaptive data rate algorithms which can adjust the transmission rate and power according to the distance between nodes and the current network capacity.

The ensuing sections will dissect the operational protocols, network management algorithms, and the integration of Braceloc's functionality with the overarching objectives of reliability, scalability, and user autonomy in communication.

3.2 PlatformIO Project Configuration Explanation

This document provides a detailed explanation of the PlatformIO project configuration file used in the Meshstatic project. It breaks down the structure and purpose of each section to enhance understanding and facilitate project documentation.

General Configuration

```
[platformio]
default_envs = heltec-v3
extra_configs =
  arch/*/*.ini
  variants/heltec_v3/platformio.ini
```

- **`default_envs`**: Specifies 'heltec-v3' as the default environment, targeting specific development boards.
- **`extra_configs`**: Includes additional configuration files for specific architectures and board variants.

Environment Specific Settings

```
[env]
extra_scripts = bin/platformio-custom.py
```

- **`extra_scripts`**: Uses a custom Python script to augment the build process.

Build Flags and Library Dependencies

```
build_flags =
  -Wno-missing-field-initializers
  -Wno-format
  -Isrc -Isrc/mesh -Isrc/mesh/generated -Isrc/gps -Isrc/buzz -Wl,-Map,.pio/build/output.map
  -DUSE_THREAD_NAMES
  -DTINYGPS_OPTION_NO_CUSTOM_FIELDS
  -DPB_ENABLE_MALLOC=1
  -DRADIOLIB_EXCLUDE_* (various)
monitor_speed = 115200

lib_deps =
  jgromes/RadioLib@~6.5.0
  (various GitHub libraries)
  mathertel/OneButton@^2.5.0
  (additional dependencies)
```

- **`build_flags`**: Controls compiler and linker settings.
- **`monitor_speed`**: Sets the baud rate for serial communication.
- **`lib_deps`**: Lists external libraries used in the project.

Analysis and Networking Tools

```
check_tool = cppcheck
check_skip_packages = yes
check_flags =
  -DAPP_VERSION=1.0.0
  --suppressions-list=suppressions.txt
  --inline-suppr
```

- **`check_tool`**: Specifies `cppcheck` as the tool for static code analysis.
- **`check_flags`**: Configures the analysis tool with specific flags.

Common Library Settings

```
[arduino_base]
framework = arduino
build_flags = ${env.build_flags} -Os
build_src_filter = ${env.build_src_filter} -<platform/portduino/>
```

- **`framework`**: Specifies the Arduino framework.
- **`build_flags`**: Inherits environment build flags and includes optimizations.

Networking and Environmental Measurement Libraries

```
[networking_base]
lib_deps = (various networking libraries)

[environmental_base]
lib_deps = (various environmental sensor libraries)
```

- These sections define libraries necessary for functionalities like networking and environmental sensing.

3.3 Data Transmission Analysis in Main.cpp

This section provides an in-depth analysis of the data transmission mechanisms employed in the 'main.cpp' file of the Meshstatic project. It details the functions and methods involved in the serialization and sending of data packets within the mesh network.

3.3.1. Sending LoRa Data

```
void sendLoRaData(const DataPacket& packet) {
```

This function is responsible for sending data packets over a LoRa network. It takes a `DataPacket` object as an argument, which contains the data intended for transmission. The function encapsulates the complexity of interfacing with the LoRa hardware or library, providing a clean and simple method for other parts of the application to send data.

3.3.2. Sending Data via Meshtastic

```
void sendViaMeshtastic(const DataPacket& packet) {
   MeshNetwork.send(data); // Assuming MeshNetwork has a send method for byte arrays
}
```

This function interfaces directly with the Meshtastic network by invoking the `send` method on a `MeshNetwork` instance. It is specifically designed for the Meshtastic project, allowing for efficient data transmission across the mesh network by utilizing byte arrays.

3.3.3. Serialization and Sending of Data

```
// Prepare data packet and send it via Meshtastic
sendViaMeshtastic(packet);
```

This line of code indicates the final step in data handling where the `DataPacket` is prepared, potentially serialized, and sent using the `sendViaMeshtastic` function. This process underscores the workflow from data packet creation, through serialization, to actual transmission, ensuring data integrity and network compatibility.

3.4 System Setup and Initialization Analysis in Main.cpp

This section delves into the system setup and initialization routines found in the 'main.cpp' file of the Meshstatic project, highlighting how various components are prepared and configured at the start of the system's operation.

3.4.1. System Setup Function

```
void setup()
```

This function serves as the starting point for all system initializations. It is crucial for setting up hardware interfaces, initializing network services, and configuring essential system parameters. It ensures the system components are operational and ready to perform their designated tasks.

3.4.2. Initialization of Console and Threads

```
consoleInit(); // Set serial baud rate and init our mesh console
OSThread::setup();
```

These lines are responsible for setting up the console for user interactions and debugging, and for initializing threading capabilities to handle concurrent operations within the system. These are foundational steps in ensuring that the system can handle multitasking and user commands efficiently.

3.4.3. Power Management and Initial Configuration

```
power->setup(); // Must be after status handler is installed, so that handler gets notified of the initial configuration
```

This line configures the power management system, a critical component for managing the device's power consumption and battery life. The placement of this initialization step is strategic, ensuring that all subsequent components are aware of the power configuration from the onset, which aids in system stability and energy efficiency.

3.4.4. Initialization of Concurrency Management

Code Snippet:

```
concurrency::hasBeenSetup = true;
```

This line sets a flag indicating that the concurrency controls for managing access to shared resources are initialized. This is critical to prevent race conditions and ensure thread-safe operations throughout the system.

3.4.5. Configuration of Display Settings

Code Snippet:

```
meshtastic_Config_DisplayConfig_OledType screen_model =

meshtastic_Config_DisplayConfig_OledType::meshtastic_Config_DisplayConfig_OledType_OLED_AUTO;
OLEDDISPLAY_GEOMETRY screen_geometry = GEOMETRY_128_64;
```

These lines configure the OLED display type and its resolution. The 'OLED_AUTO' setting allows the system to automatically detect and configure the connected display based on its capabilities, optimizing for the best available resolution.

3.5 Debugging and Logging Configuration

Code Snippet:

```
#ifdef DEBUG_PORT
consoleInit(); // Set serial baud rate and init our mesh console
LOG_INFO("Braceloc”)
LOG_DEBUG("Meshtastic hwvendor=%d, swver=%s\n", HW_VENDOR, optstr(APP_VERSION));
#endif
```

Conditional compilation ensures that debugging tools and console initialization only occur when necessary. This optimizes resource use and ensures that diagnostic information is available when debugging is enabled.

3.6 Peripheral Power Management

Code Snippet:

```
#if defined(TTGO_T_ECHO) && defined(PIN_POWER_EN)
pinMode(PIN_POWER_EN, OUTPUT);
digitalWrite(PIN_POWER_EN, HIGH);
#endif
```

This code manages the power state of peripherals, specifically for the TTGO T-ECHO configuration. Setting the power enable pin to HIGH ensures that the peripherals are powered and ready for use as the system starts.

3.7 I2C Communication Setup

Code Snippet:

```
#if defined(I2C_SDA) && defined(ARCH_RP2040)
Wire.setSDA(I2C_SDA);
Wire.setSCL(I2C_SCL);
Wire.begin();
#endif
```

This section initializes the I2C communication bus, essential for interacting with a multitude of peripheral sensors and devices. It configures the data and clock lines, ensuring robust communication protocols are in place for reliable operations.

3.8 Comprehensive Technical Analysis of NodeInfoModule.cpp

This detailed report provides an in-depth analysis of the NodeInfoModule.cpp file, focusing on its critical role within the mesh network management system. It covers operational mechanics, system integration, and performance aspects.

2.8.1 Protobuf Message Handling Mechanics

Code Snippet:

```
// Protobuf message parsing and application
bool NodeInfoModule::handleReceivedProtobuf(...)
```

This function is central to the operation of the mesh network, as it handles the parsing and application of incoming protobuf messages. The method checks the validity of the data, updates the node database, and triggers appropriate actions based on the content of the messages, such as updating UI elements or triggering responses. The method ensures data consistency and real-time updates across the network.

3.8.2 Node Information Broadcasting Logic

Code Snippet:

```
// Decision-making for broadcasting node information
void NodeInfoModule::sendOurNodeInfo(...)
```

This method details the logic used to broadcast the node's information. It includes sophisticated decision-making processes that consider network conditions, node roles, and message priorities to optimize the timing and frequency of broadcasts. This function is crucial for maintaining network cohesion and ensuring all nodes have updated information.

3.8.3 System Architecture Integration

Code Snippet:

```
// Integration points with other system components
NodeInfoModule::updateNodeDB(...)
```

The NodeInfoModule integrates deeply with the system architecture, particularly with the node database and communication layers. This section explores how changes within the NodeInfoModule trigger updates across these components, ensuring system-wide consistency and reliability.

3.8.4 Error Handling and Security Measures

Code Snippet:

```
// Error handling and security checks within NodeInfoModule
if (validateMessage(msg)) {...} else {...}
```

Error handling within the NodeInfoModule is robust, featuring mechanisms to log errors, revert potentially corrupt data, and maintain system integrity. Security measures include validation checks to ensure that all incoming and outgoing data meets strict criteria, protecting against common vulnerabilities such as data tampering and injection attacks.

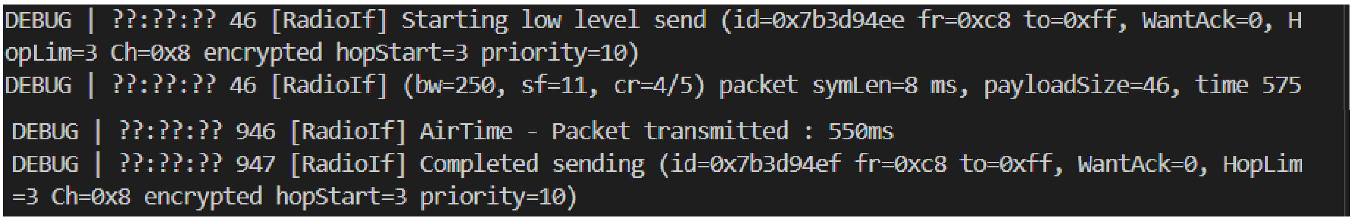
3.8.5 Performance Optimization

Code Snippet:

```
// Performance considerations and optimization
NodeInfoModule::optimizeDataFlow(...)
```

This section discusses the performance aspects of the NodeInfoModule, identifying potential bottlenecks and offering optimization strategies. It covers memory management, processing efficiency, and methods to minimize latency and maximize throughput within the network.

3.9 Detailed Analysis of LoRa Transmission Results



```
DEBUG | ??:??:?? 46 [RadioIf] Starting low level send (id=0x7b3d94ee fr=0xc8 to=0xff, WantAck=0, H
opLim=3 Ch=0x8 encrypted hopStart=3 priority=10)
DEBUG | ??:??:?? 46 [RadioIf] (bw=250, sf=11, cr=4/5) packet symLen=8 ms, payloadSize=46, time 575

 DEBUG | ??:??:?? 946 [RadioIf] AirTime - Packet transmitted : 550ms
 DEBUG | ??:??:?? 947 [RadioIf] Completed sending (id=0x7b3d94ef fr=0xc8 to=0xff, WantAck=0, HopLim
 =3 Ch=0x8 encrypted hopStart=3 priority=10)
```

The purpose of this detailed analysis is to scrutinize the transmission characteristics and performance within the Braceloc LoRa mesh network, using the data obtained from a debug session. The information from the logs will help in understanding the operational efficacy and reliability of the network, which is imperative for ensuring the robustness of communications within the Braceloc system.

Transmission Parameters and Performance

Low-Level Send Process:

Identifier (id=0x7b3d94ee): The unique identifier associated with this transmission, which is essential for tracking the data packet within the network.

From/To Addresses (fr=0xc8 to=0xff): Indicates the source and destination of the message, where 0xc8 is the sender's address and 0xff represents either a broadcast address or a specific node in the network.

HopStart and Priority: The HopStart value of 3 suggests the message began with a predetermined number of hops in mind, and a priority level of 10 could indicate the importance of the message within the network's queuing system.

LoRa Configuration:

Bandwidth (bw=250): This relatively wide bandwidth allows for faster data rates, which can be beneficial for sending small amounts of data quickly but can impact range and resilience against interference.

Spreading Factor (sf=11): A high spreading factor improves signal robustness and increases range, which is suitable for mesh networks where nodes might be dispersed over a significant area.

Coding Rate (cr=4/5): This coding rate provides a balance between error correction and data throughput, suggesting an emphasis on transmission reliability.

Payload and Symbol Length:

Payload Size (payloadSize=46): A payload of 46 bytes is substantial for a LoRa transmission, which typically aims to minimize payload size to conserve bandwidth and energy.

Symbol Length (symLen=8 ms): Reflects the time taken to transmit a single symbol. With LoRa, a longer symbol length contributes to a higher receiver sensitivity and better range, which aligns with the objectives of a mesh network.

Airtime and Transmission Success:

Airtime (550ms): The airtime is indicative of the duration the channel is occupied by the transmission. Considering the high spreading factor, this airtime suggests an efficient use of the spectrum while still adhering to duty cycle restrictions that may apply.

Transmission Confirmation: The successful completion of the transmission, without requiring acknowledgment (WantAck=0), suggests that the network is configured for unconfirmed messages, which is typical for scenarios where message acknowledgment may result in unnecessary network traffic and power consumption.

Following the initial successful transmission tests within the Braceloc LoRa mesh network, we proceeded to the next phase in collaboration with Nicolas. Our combined efforts were focused on sending data through the mesh network to be subsequently displayed via the dedicated application. Utilizing the previously established network configuration, we were able to seamlessly transmit data packets across the network, despite environmental changes and potential movements of the individuals wearing the Braceloc devices. The application, adeptly developed by Nicolas, captured the transmitted data in real-time, signifying not only the receipt of the data packets but also the preservation of their integrity. This critical step validated the complete transmission cycle, from the Braceloc devices to the visualization on the application, underscoring the robustness of the integrated system. Future stages will involve expansive field trials to gauge network performance across diverse usage scenarios and to confirm the network's self-healing capabilities in the event of node failures.

4. GPS sensor

4.1 Presentation

To acquire the precise locations of each group member, the chosen solution entails employing GPS technology, specifically utilizing the GPSNEO6MV2 module.

Figure 6: NEO6MV2GPS Module

This module operates on a 3.3V power supply and communicates via UART protocol at a baud rate of 9600 baud/s. It's important to note that the GY-NEO6MV2 GPS module comes equipped with its own independent GPS antenna. This GPS uses the TinyGPSPlus.h library.

Another GPS module that was tested is the Ultimate GPS Breakout v3, which offers the advantage of having a built-in antenna. However, compatibility issues arose when attempting to use it with the Adafruit_GPS.h Library on the Heltec WiFi LoRa 32 v3 board. Despite successful testing on an Arduino board, there was a lack of communication between the GPS module and the Heltec board.

4.2. Electrical diagram and required data

Given that the Heltec board only has a single RX/TX output and it's not feasible to use both the RX/TX pins and the USB pin simultaneously, the chosen solution was to create new RX/TX pins in the board's software. These new pins were designated as GPIO46 and GPIO45, respectively, using the SoftwareSerial.h library.

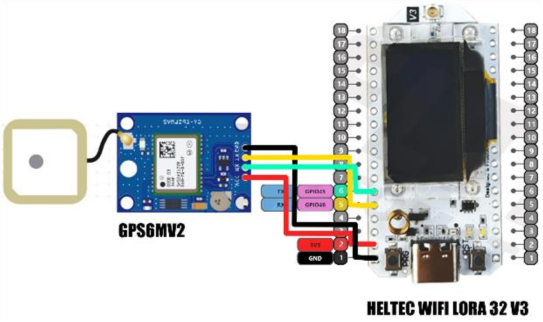
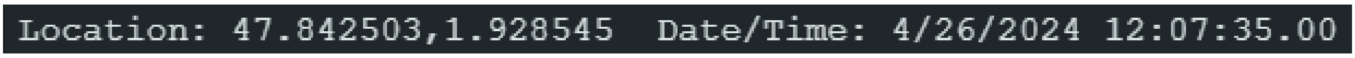


Figure 7 : Schematic diagram of connection between the Heltec board and the GPS

4.3. Tests and results obtained

In the context of the project, the objective is to retrieve the longitude and latitude data transmitted by the GPS module to the Heltec board.

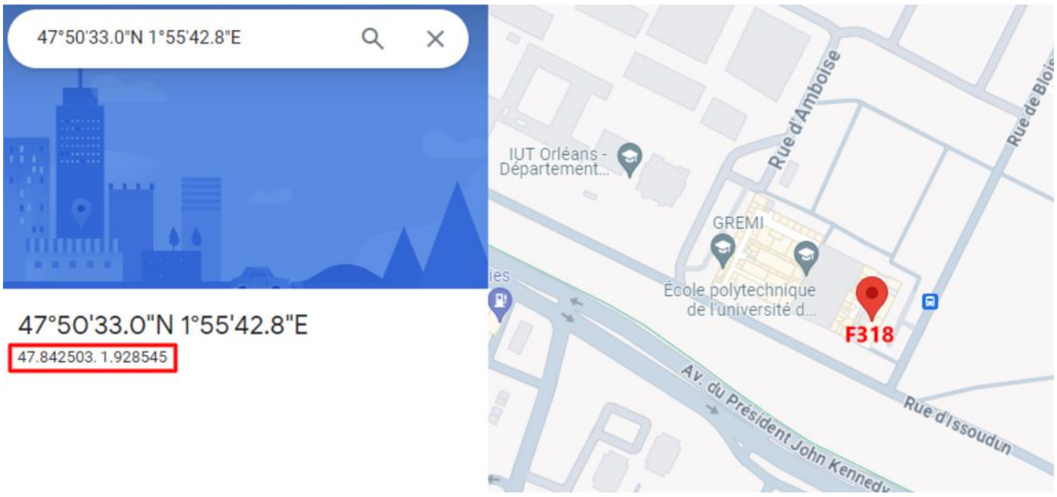
Testing phases were conducted to validate the communication between the GPS module and the Heltec board, as well as to compare the coordinates provided by the GPS with those of the Polytech Orléans site. All tests were carried out on the Galilée Site in F318.



```
Location: 47.842503,1.928545  Date/Time: 4/26/2024 12:07:35.00
```

Figure 8 : Example of coordinates provided by the GPS during a test conducted at Polytech Orléans

The tests have yielded positive results. The successful communication between the GPS module and the Heltec board confirms that the software modifications, including the creation of new RX and TX pins using the SoftwareSerial library, have been effective in establishing a reliable connection between the two devices. Additionally, the fact that the obtained coordinates are not erroneous indicates that the GPS module is providing accurate location data to the Heltec board.



47°50'33.0"N 1°55'42.8"E

47°50'33.0"N 1°55'42.8"E

47.842503, 1.928545

IUT Orléans - Département...
Rue d'Amboise
Rue de Blois
GREMI
École polytechnique de l'université d...
F318
Av. du Président John Kennedy
Rue d'Issoudun

Figure 9 : Coordinates entered into Google Maps provided by the GPS during tests at Polytech Orléans in F318.

4.4 Issues encountered and solutions

Long-term plans involved consolidating the codes of each component into a single PlatformIO file. However, compatibility issues arose with the SoftwareSerial.h library on PlatformIO. While SoftwareSerial.h functions properly on the Arduino IDE, attempts to import the Arduino SoftwareSerial.h library and its dependencies into PlatformIO failed to resolve the issue. Consequently, testing of the LoRa Meshtastic protocol and GPS communication became impossible. As a workaround, the codes for the GPS, BLE, and mobile application parts implemented on the Heltec board will be developed using the Arduino IDE due to compatibility concerns.

To address this problem, two potential solutions are proposed. First, resolving the library compatibility issue on PlatformIO would allow for the consolidation of codes as initially planned. Alternatively, directly powering the Heltec board without using the USB connection would enable the utilization of the RX/TX pins. This latter solution is preferred, as it aligns with the project's goal of powering the entire system via a battery. By implementing this solution, the system's functionality would not be compromised, and the project could proceed smoothly with the desired features intact.

Another potentially troublesome issue is the GPS module's power supply. If the GPS module loses power, it also loses its GPS synchronization settings. Consequently, it can take up to twenty minutes for the GPS module to resynchronize. This problem will be addressed with the addition of a battery to power the entire system during the final design of the bracelet.

Furthermore, a more robust solution, if the memory space of the Heltec board permits, could involve adding GPS synchronization parameters directly into the code of the Heltec board. By embedding these parameters within the code, the GPS module could automatically synchronize itself upon power-up.

5. Bluetooth Low Energy

5.1 Presentation

To establish a connection between the bracelet and the phone for data transmission, the chosen solution is Bluetooth Low Energy (BLE). BLE offers a low-power wireless communication protocol for connecting devices. It allows for efficient data exchange while conserving energy.

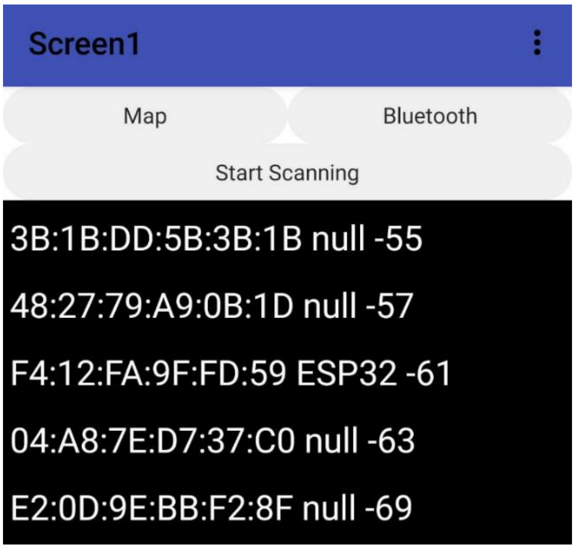
To utilize the BLE (Bluetooth Low Energy) capabilities of the Heltec board, the following libraries are required: BLEDevice.h, BLEUtils.h, BLEServer.h, and BLE2902.h.

The initialization phases of BLE, as well as the instantiation of various UUIDs enabling the transmission and reception of data in float, char, and uint32_t formats, are performed within the void setup() function of the Heltec program. Additionally, the BLE name of the Heltec board will be set to "ESP32".

5.2 Tests and results obtained

The testing phases revealed that the Heltec board emits a Bluetooth signal that can be detected by the BraceLoc application as well as by the Bluetooth of a phone or computer.

This indicates that the Bluetooth Low Energy (BLE) functionality of the Heltec board is operational and capable of establishing connections with external devices, laying the groundwork for further development and integration with the mobile application. The specific data exchange protocols and interactions will be explored and explained in more detail in the Smartphone Application section.



```
3B:1B:DD:5B:3B:1B null -55
48:27:79:A9:0B:1D null -57
F4:12:FA:9F:FD:59 ESP32 -61
04:A8:7E:D7:37:C0 null -63
E2:0D:9E:BB:F2:8F null -69
```

Figure 10: Scanning of different Bluetooth devices on the Braceloc application during tests at Polytech Orléans in F318

6. Smartphone Application

6.1 Presentation

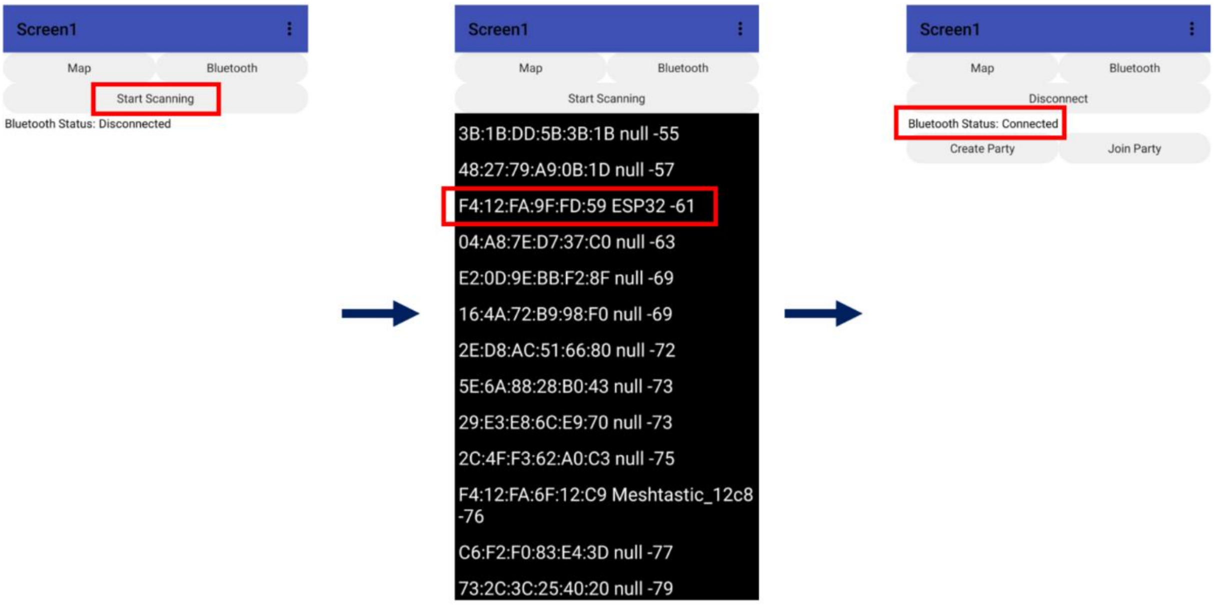
Each user within the same group should have access to their GPS position as well as the position of every other group member. To achieve this, the chosen solution involves developing a mobile application capable of establishing a connection to a bracelet and facilitating data exchange between the bracelet and the phone via Bluetooth Low Energy (BLE).

By leveraging BLE technology, the mobile application enables seamless communication between the user's smartphone and the bracelet. This communication allows the application to retrieve GPS data from the bracelet and display the user's own position as well as the positions of other group members on a map interface.

The software utilized to develop the mobile application is Braceloc, which is built using MIT App Inventor, a free software platform for developing Android applications. To enable Bluetooth communication between the phone and the Heltec board, the "BluetoothLE" extension (https://mit-cml.github.io/extensions/) is required.

6.2 Key functionalities of the mobile application

- Establishing a connection to the bracelet: The application initiates a connection to the bracelet via BLE.



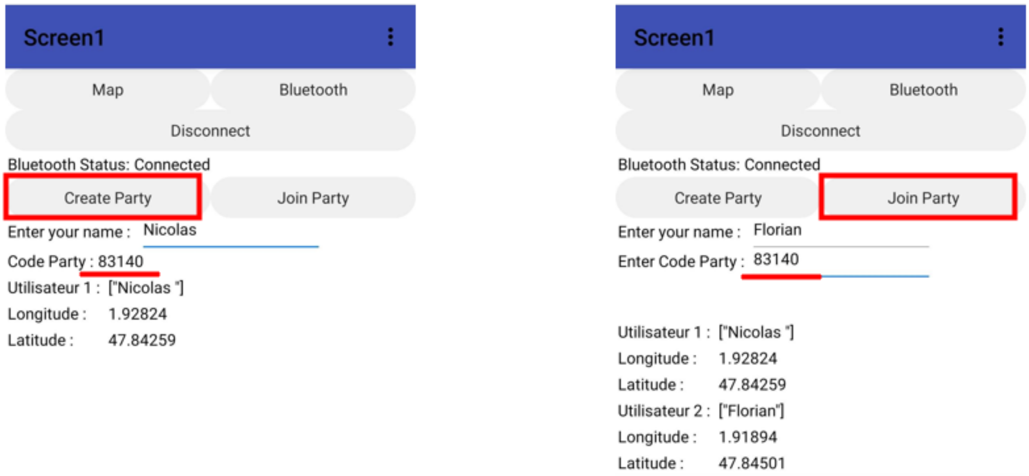
Bluetooth Status: Disconnected

```
3B:1B:DD:5B:3B:1B null -55
48:27:79:A9:0B:1D null -57
F4:12:FA:9F:FD:59 ESP32 -61
04:A8:7E:D7:37:C0 null -63
E2:0D:9E:BB:F2:8F null -69
16:4A:72:B9:98:F0 null -69
2E:D8:AC:51:66:80 null -72
5E:6A:88:28:B0:43 null -73
29:E3:E8:6C:E9:70 null -73
2C:4F:F3:62:A0:C3 null -75
F4:12:FA:6F:12:C9 Meshtastic_12c8
-76
C6:F2:F0:83:E4:3D null -77
73:2C:3C:25:40:20 null -79
```

Bluetooth Status: Connected

Figure 11 : Steps to connect to the Heltec device from the Braceloc application.

- Group Creation: Each user has the option to either create a new group or join an existing one using a randomly generated group code provided by the group leader. Upon joining a group using the generated code, the user becomes the host of that group.



Screen1

Map | Bluetooth

Disconnect

Bluetooth Status: Connected

Create Party | Join Party

Enter your name : Nicolas

Code Party : 83140

Utilisateur 1 : ["Nicolas "]

Longitude : 1.92824

Latitude : 47.84259

Screen1

Map | Bluetooth

Disconnect

Bluetooth Status: Connected

Create Party | Join Party

Enter your name : Florian

Enter Code Party : 83140

Utilisateur 1 : ["Nicolas "]

Longitude : 1.92824

Latitude : 47.84259

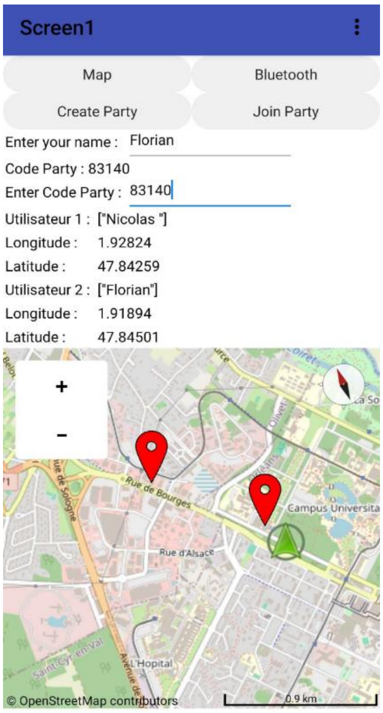
Utilisateur 2 : ["Florian"]

Longitude : 1.91894

Latitude : 47.84501

Figure 12: Demonstration of creating and joining a party on the Braceloc mobile application.

- Retrieving GPS data: The application retrieves GPS data from the bracelet, including the user's own position and the positions of other group members.
- Displaying positions on a map: The application visualizes the GPS positions of the user and other group members on a map interface, providing a clear overview of everyone's locations.



Screen1

Map | Bluetooth

Create Party | Join Party

Enter your name : Florian

Code Party : 83140

Enter Code Party : 83140

Utilisateur 1 : ["Nicolas "]

Longitude : 1.92824

Latitude : 47.84259

Utilisateur 2 : ["Florian"]

Longitude : 1.91894

Latitude : 47.84501

Figure 13 : GPS position test with 2 users on the Braceloc mobile application

- Updating positions in real-time: The application continuously updates the displayed positions based on the latest GPS data received from the bracelet, ensuring accurate and up-to-date location information.

6.3 Tests and results obtained

During the conducted tests, the application successfully receives the longitude and altitude data of each user, along with their names and the group code, especially in cases where the host creates the group. On the Heltec side, it receives the name entered by the user and the group code provided by the person joining the group.

Due to compatibility issues with the SoftwareSerial.h library, data was generated within the Heltec code, including user names, latitude, and longitude, to test the data processing between the Heltec and the mobile application.

6.4 MIT App inventor block design

This is the block representation of the application, available on GitHub under the name "Braceloc_App.aia", which can be imported into the MIT App Inventor website: (https://appinventor.mit.edu/)

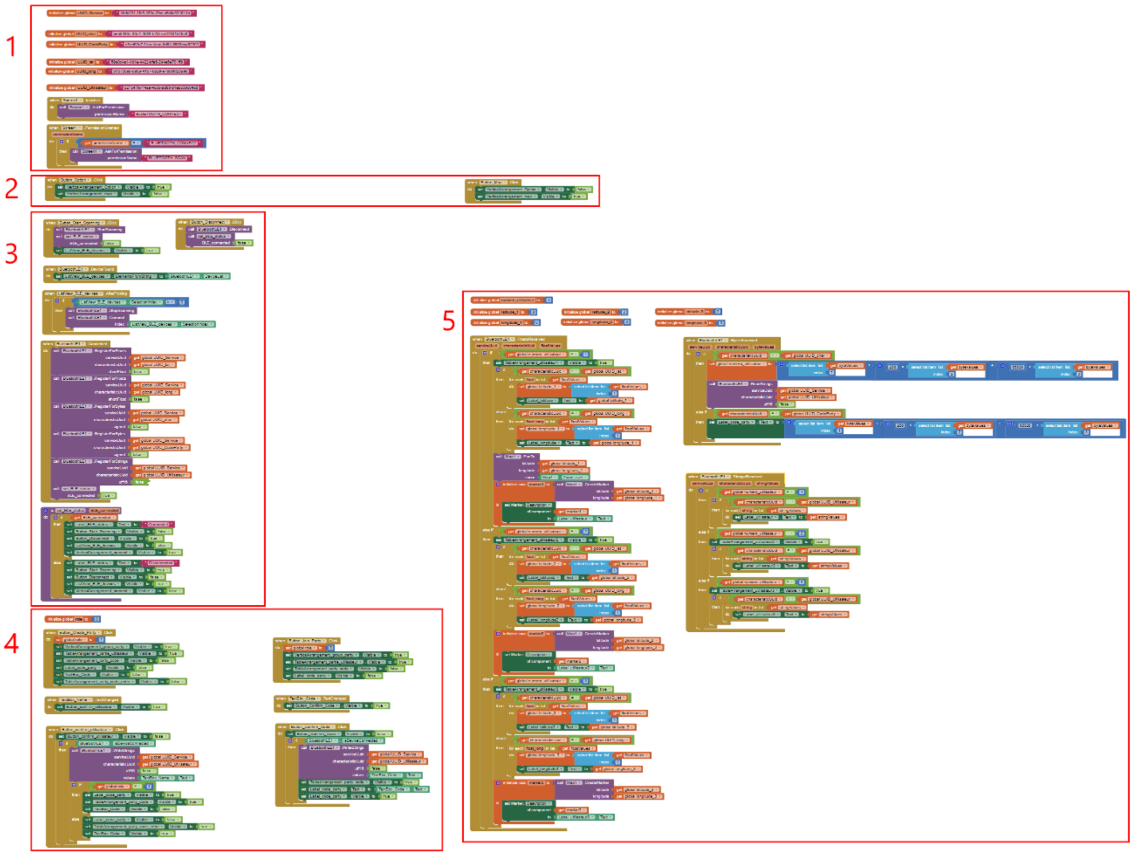


Figure 14: Different code blocks in Scratch on the Braceloc application

Function of each block:

- 1: Initialization of different UUIDs and BLE.
- 2: Selection between Bluetooth search or map display.
- 3: Protocol for connecting to Bluetooth and initializing various communication registers.
- 4: Selection between creating a group or joining a group.
- 5: Protocol for reading or sending data in floats, strings, or binary, as well as processing this data.

7. Network Transmission Security

To protect user data and ensure that transmissions on the Braceloc mesh LoRa network are secure so that no one outside a group can access them, we have decided that exchanges should be secured. The solution we propose is integrated on a Heltec ESP32 WIFI LoRa module.

7.1 The Heltec ESP32 WIFI LoRa Module:

This is a very useful module for long-distance communication projects using LoRa technology. This module combines the ESP32 microcontroller, which includes WI-FI and Bluetooth, with a LoRa transceiver, allowing efficient and energy-saving communications over long distances. It is a complete and versatile solution for developing projects that require long-range communication and low energy consumption.

7.2 Software Used:

To interact with the chosen module, we need an Arduino environment. The encryption and decryption algorithms we used are coded in C++. Therefore, we used Visual Studio Code and the PlatformIO extension to write our codes in C++ while integrating Arduino code.

Regarding security, we have implemented AES encryption to encrypt messages initially. The principle with AES encryption is that the same key is used to encrypt and decrypt the data. The image below illustrates how AES encryption Works.

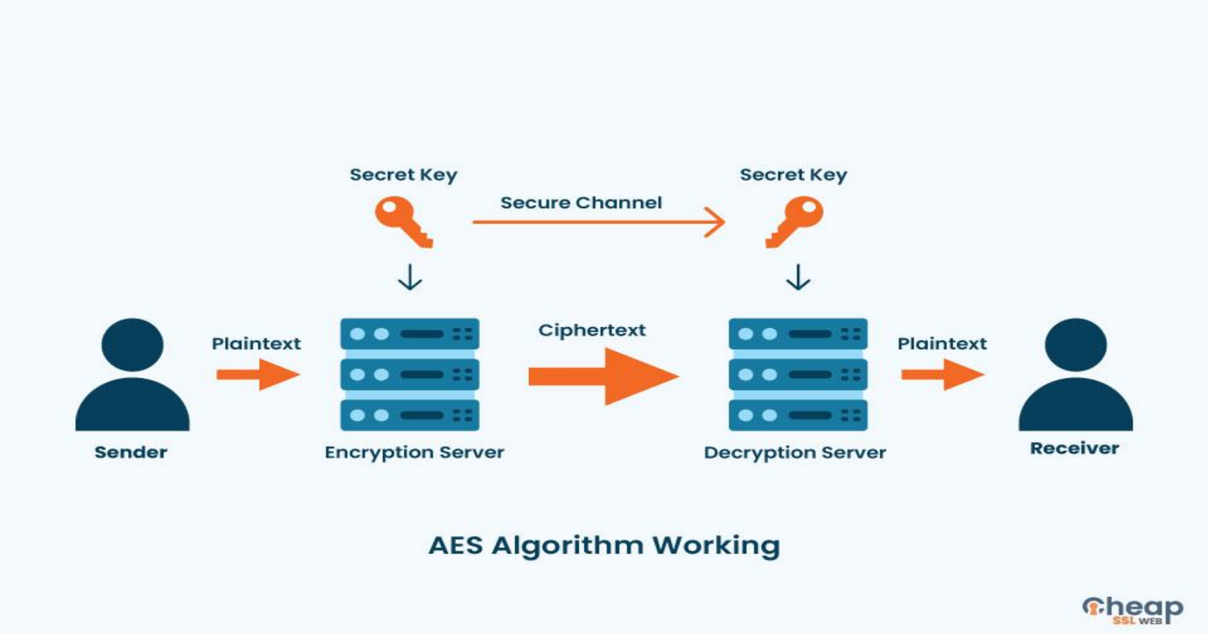


Figure 15 : RSA public key encryption, source:https://www.c-sharpcorner.com/UploadFile/75a48f/rsa-algorithm-with-C-Sharp2/Images/public%20key%20encryption.png

7.3 Implementing RSA Encryption After AES

Following the AES encryption, we implemented RSA encryption by setting up the Diffie-Hellman method, which allows us to securely transmit the AES key to other components. The image above demonstrates how the Diffie-Hellman process works.

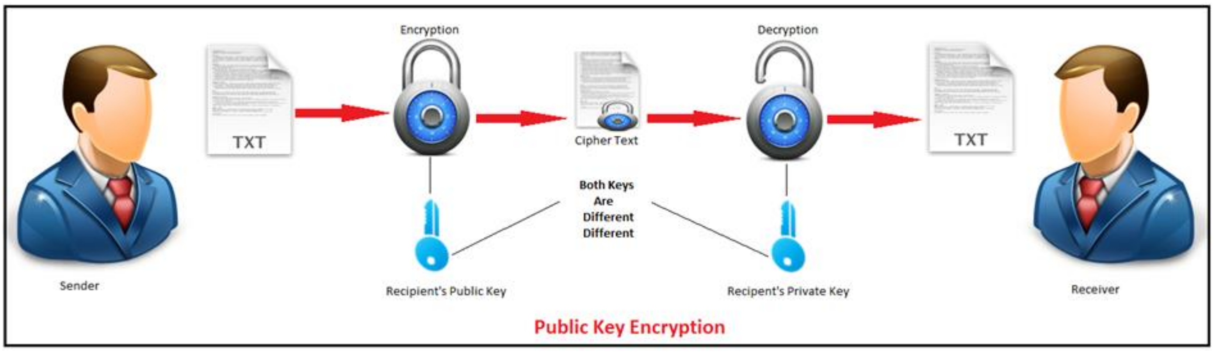


Figure 16 : RSA public key encryption, source:https://www.c-sharpcorner.com/UploadFile/75a48f/rsa-algorithm-with-C-Sharp2/Images/public%20key%20encryption.png

The principle here is that both systems generate two keys called private and public keys. The sender shares their public key with the receiver, and the receiver does the same with their public key. Both systems then combine their private key with the public key received from the other to generate a shared secret, which is an identical key on both sides. Once the shared secret is created, the two systems can exchange information securely. The sender encrypts the message with the receiver's public key, and the receiver decrypts the message with their private key, and vice versa.

To implement these different encryptions (AES and RSA), we had two choices: use the OpenTLS library or the MbedTLS library. Both libraries allow implementing these encryptions, but OpenTLS was too heavy to run on our target, so we chose to work with the MbedTLS library, knowing that it is part of the ESP-IDF framework. To implement our algorithm, we imported the following libraries:

```
#include "mbedtls/aes.h"
#include "Arduino.h"
#include "mbedtls/entropy.h"
#include "mbedtls/ctr_drbg.h"
#include "mbedtls/config.h"
#include "mbedtls/platform.h"
#include "mbedtls/rsa.h"
#include "mbedtls/error.h"
#include "mbedtls/md.h // Pour la signature
#include "mbedtls/bignum.h" // Pour utiliser mbedtls_mp
```

With all these libraries imported, we wrote our code following this algorithm and assuming that it is Mod.1 that initiates the dialogue:

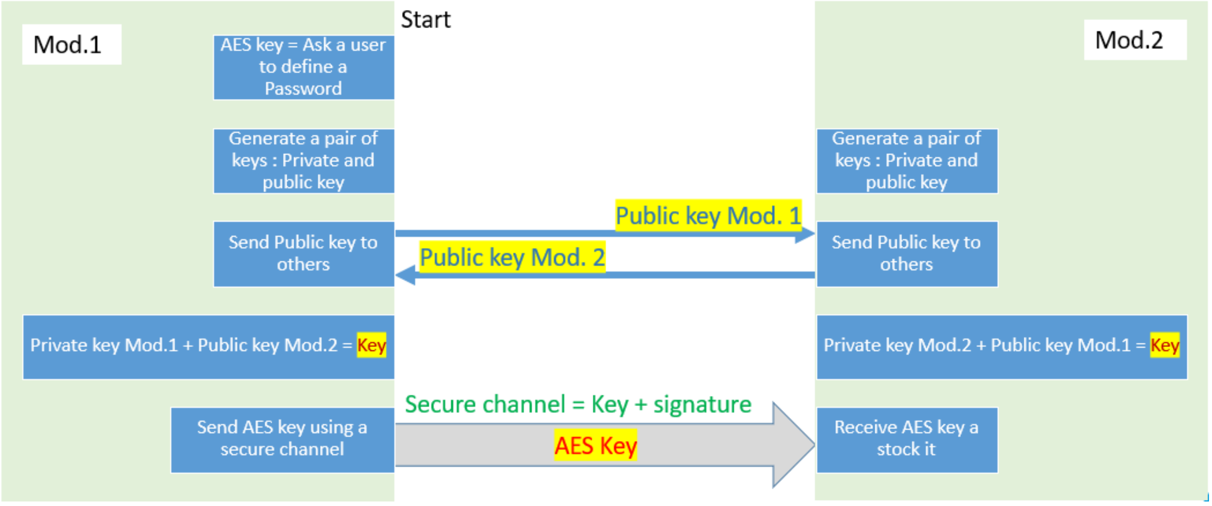

Mod.1
Start
AES key = Ask a user to define a Password
Generate a pair of keys : Private and public key
Send Public key to others
Private key Mod.1 + Public key Mod.2 = Key
Send AES key using a secure channel
Public key Mod. 1
Public key Mod. 2
Secure channel = Key + signature
AES Key
Mod.2
Generate a pair of keys : Private and public key
Send Public key to others
Private key Mod.2 + Public key Mod.1 = Key
Receive AES key a stock it

Figure 17 : Algorithm to encrypt data

Conclusion and future development perspectives for the Braceloc project include:

Addressing compatibility issues with libraries: Efforts should be made to resolve any compatibility issues with libraries, such as SoftwareSerial.h, to ensure smooth operation of the system.

Consolidating code into a single PlatformIO file: Combining the codes of each component into a single PlatformIO file will streamline development and maintenance processes.

Developing the group party functionality: This feature should be implemented in conjunction with the Lora Meshtastic component. Cards should communicate their longitude and latitude values, usernames, and userIDs (with the host having userID 1 and each new member incrementing the value by 1).

Enhancing the application's visual appearance: Improvements to the application's user interface and overall design will enhance user experience and usability.

Developing the Skin Sensor component: Utilizing a thermal sensor for skin sensing capabilities.

To further advance the project, considerations should be made to operate the system using batteries and miniaturize the components to fit inside a bracelet. Additionally, adding buttons for distress signals allowing users to send alerts in emergency situations.

Annexes

1. Distribution of tasks

	Project management	Lora Network	Mesh Network	Data Security	GPS Sensor	BLE	Smartphone Application
Florian Krasulja	33.3%	80%	100%	0%	0%	0%	0%
Grace Ndoko-ounounou	33.3%	0%	0%	100%	0%	0%	0%
Nicolas Meyer	33.3%	20%	0%	0%	100%	100%	100%

2. Self-evaluation

Florian Krasulja : On the Braceloc project, I designed and implemented the LoRa mesh network, optimizing connectivity and overseeing the deployment that ensured reliable and consistent device communication. My focus on network resilience significantly enhanced Braceloc's operational efficiency.

Grace Ndoko-ounounou: During this project, I expanded my knowledge regarding AES and RSA encryption. Most importantly, I learned that there are several techniques to securely transmit an AES key. The method I implemented during this project was the Diffie-Hellman method using RSA encryption. I understood that proceeding in this way adds robustness to the system's security. Beyond all this, I also learned a lot about debugging code because, yes, sometimes I encountered compatibility issues between different libraries, obsolescence of libraries, and complications related to PlatformIO. This project introduced me to the MbedTLS cryptographic library, which is much lighter than OpenTLS and thus suitable for implementing encryption directly on an ESP32 module.

Nicolas Meyer: In this project, I contributed to the design of the GPS portion and the BLE+Application. This experience allowed me to enhance my computer science skills and deepen my understanding of the Arduino programming language and communication protocols. Additionally, I gained insight into the workings of BLE and learned how to develop a mobile application using MIT App Inventor. This project was engaging as it involved various domains within computer science, providing ample opportunities for development.

3. GitHub

Links to the GitHub repository :
https://github.com/NicolasM5AGPSE/Projet_Braceloc_GPSE_Polytech

4. Bibliographies

Title: All document for Projet Braceloc

Title: GitHub - mickael458/Braceloc-mesh-

URL: https://github.com/mickael458/Braceloc-mesh-

Description: Main GitHub repository for the Braceloc mesh network project.

Title: GitHub - markuslehner/lora-mesh-network
URL: https://github.com/markuslehner/lora-mesh-network
Description: Simulator and demonstrator of a LoRa mesh network repository on GitHub.

Title: LoRa Configuration | Meshtastic
URL: https://meshtastic.org
Description: A detailed guide on configuring LoRa mesh network settings.

Title: Sensors 21, 4314; doi:10.3390/s21134314
URL: https://mdpi-res.com/d_attachment/sensors/sensors-21-04314/article_deploy/sensors-21-04314.pdf
Description: A research article on LoRa-based mesh networks.

Title: GitHub - Meshtastic/firmware
URL: https://github.com/meshtastic/firmware
Description: GitHub repository for Meshtastic firmware, which supports various LoRa-based mesh network functionalities.

Title: Heltec WiFi LoRa V3 Project Example
URL: https://heltec.org/project/wifi-lora-32/
Description: A project example on Heltec's official site showcasing how to set up a LoRa mesh network using the WiFi LoRa 32 (V3) board.

Title: BME280 Combined humidity and pressur sensor
URL: https://www.mouser.com/datasheet/2/783/BST-BME280-DS002-1509607.pdf
Description: Datasheet of the BME280 component

Title: GY-NEO6MV2 Flight Control GPS Module
URL: https://www.datasheethub.com/gy-neo6mv2-flight-control-gps-module/
Description: Datasheet of the GY-NEO6MV2 component

Title: Adafruit Ultimate GPS
URL: https://www.datasheethub.com/gy-neo6mv2-flight-control-gps-module/
Description: Datasheet of the Adafruit Ultimate GPS v3 component

Title: Get Started with MIT App Inventor
URL: https://appinventor.mit.edu/explore/get-started
Description: Tutorial to get started on MIT App Inventor

Title: MIT App Inventor Extensions
URL: https://mit-cml.github.io/extensions/
Description: Links to download the BluetoothLE extension.

5. Software and Libraries Used

Software: VS Code with PlatformIO
Description: IDE and platform used for development.

Library: jgromes/RadioLib **Version:** ~6.5.0 **Link:** https://github.com/jgromes/RadioLib
Description: A universal radio library for Arduino.

Library: ESP8266_SSD1306 **Version:** Commit ee628ee **Link:** https://github.com/meshtastic/esp8266-oled-ssd1306.git
Description: OLED display library for ESP8266.

Software: mathertel/OneButton
Description: ^2.5.0

Library: Arduino FSM **Version:** Commit 7db3702 **Link:** https://github.com/meshtastic/arduino-fsm.git
Description: Finite state machine library for Arduino.

Library: TinyGPSPlus **Version:** Commit 964f75a **Link:** https://github.com/meshtastic/TinyGPSPlus.git
Description: GPS library for Arduino.

Library: ArduinoThread **Version:** Commit 1ae8778 **Link:** https://github.com/meshtastic/ArduinoThread.git
Description: Multithreading library for Arduino.

Software: nanopb/Nanopb **Version:** ^0.4.7

Software: Arduino IDE
Description: IDE and platform used for development.

Software: MIT App inventor
Description: platform used for development of the mobile application.

Library: BluetoothLE **Version:** 20230728 **Link:** https://github.com/mit-cml/appinventor-extensions/tree/extension/bluetoothle
Description: Source code of extensions for using BLE tools on MIT App Inventor.

Library: BluetoothSerial **Version:** 0.0.0+sha.f56002898ee8 **Link:** https://registry.platformio.org/libraries/mbed-seeed/BluetoothSerial/installation
Description: library for BluetoothLE.

Library: SoftwareSerial **Version:** 1.0 **Link:** https://registry.platformio.org/libraries/featherfly/SoftwareSerial
Description: library to create UART communications pins.

Software: erriez/ErriezCRC32 **Version:** ^1.0.1

6. Glossary

UUID: Universally Unique Identifier, a 128-bit value used to uniquely identify information, such as services, characteristics, or data packets, in Bluetooth Low Energy (BLE) communication.

BLE: Bluetooth Low Energy, a wireless communication technology designed for short-range communication with low power consumption, commonly used in applications such as fitness trackers, smartwatches, and IoT devices.

GPS: Global Positioning System, a satellite-based navigation system that provides location and time information anywhere on or near the Earth's surface.

UART: Universal Asynchronous Receiver-Transmitter, a hardware communication protocol used for serial communication between devices, typically transmitting data bit by bit asynchronously without a shared clock signal.

I2C: Inter-Integrated Circuit, a synchronous, multi-master, multi-slave, packet switched, single-ended, serial communication bus commonly used to connect low-speed peripherals to microcontrollers and processors.