Monitoring Building Occupancy

By Alan, Xiang, Ada, Leo

Background:

The COVID-19 pandemic highlighted the critical importance of building occupancy monitoring, with social distancing measures requiring strict control over enclosed spaces. However, accurate occupancy monitoring extends beyond pandemic scenarios, benefiting public health, emergency response, energy efficiency, and space utilization.

Traditional monitoring methods (manual counting, cameras) have limitations in privacy, accuracy, and scalability. This project proposes an innovative solution using Bluetooth signal detection and advanced algorithms for non-intrusive, scalable, and accurate occupancy monitoring.

Significance:

Our project aims to provide a cost-effective occupancy monitoring solution using widely available hardware: 4 Raspberry Pi Zero W devices for BLE scanning, connected via WiFi hotspot hosted by a Raspberry Pi 4. This system offers:

1. Cost-effectiveness through use of affordable, readily available components
2. Non-invasive monitoring, preserving privacy and requiring minimal space
3. Scalability, with potential for easy expansion in larger spaces
4. Versatility for various applications, from pandemic safety to daily space management

By combining BLE and WiFi technologies, we've created a fast, effective system with room for future enhancements.

Conclusion of Project Scope:

Our occupancy monitoring system utilises distributed BLE scanners (Raspberry Pi Zeros) to detect devices, a preprocessing device (Raspberry Pi 4) for data aggregation, and server-side storage and analysis (laptop). Key technical approaches include frequent BLE scanning, data preprocessing, triangulation for device localization, and DBSCAN clustering to estimate occupancy.

While the system shows promise, we acknowledge current limitations such as MAC address randomization and signal reflection issues in indoor environments. Future work could focus on refining algorithms to address these challenges, potentially incorporating machine learning for improved accuracy. Furthermore, due to hardware complication we were set back 1 week which would have impacted the quality of the results as the testing period was reduced *.

**Raspberry Pi 4 sd card reader broke and 1 Raspberry Pi Zero*

Hardware Setup and Data Flow:

Each Raspberry Pi Zero W functions as a Bluetooth Low Energy (BLE) scanner, conducting scans at 20-second intervals to detect nearby BLE devices. There are 4 in total to cover a certain coordinate in the room. This frequency was chosen through lots of testing to be efficient and also fast since we want regular updates.

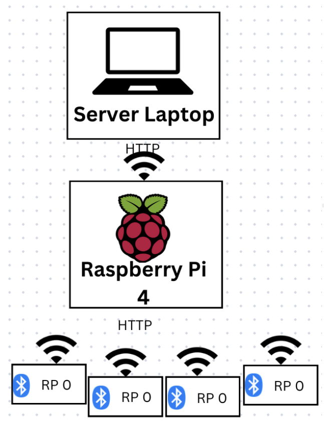


Figure 1: Architectural design

```
while True:
    devices = scanner.scan(20.0)
```

```
return f"ID:{self.device_id},MAC:{dev.addr},RSSI:{dev.rssi},Timestamp:{elapsed_ms}"
```

Network Architecture:

The Raspberry Pi 4 serves as the central hub, hosting a Wi-Fi hotspot with a custom SSID and password. All Raspberry Pi Zeros connect to this network, creating a private, localized data collection system. While this approach simplifies setup, it is not the most secure method since there is no encryption.

The collected data is transmitted from the Raspberry Pi Zeros to the Raspberry Pi 4 using HTTP POST requests. We chose HTTP for its simplicity and wide support, though more robust protocols like MQTT could be considered for larger-scale deployments.

Raspberry Pi 4:

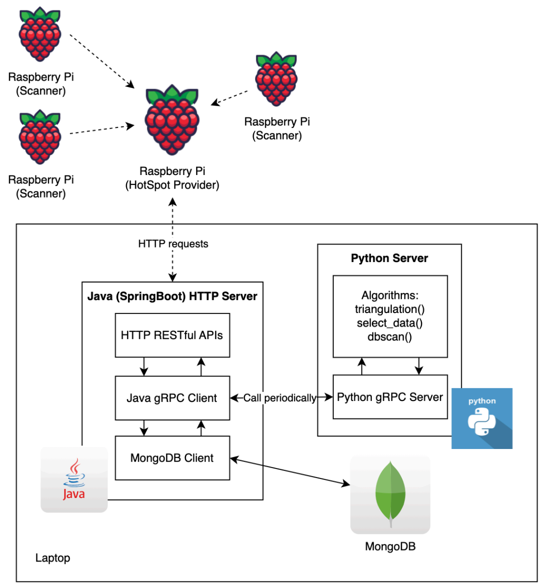
The Raspberry Pi 4 runs a Flask server, chosen for its lightweight nature and ease of use. It performs several key functions:

1. Data Aggregation: Receives and consolidates data from all scanners.
2. Preprocessing: Prepares data for further analysis (explained in architectural design)
3. Health Checking: Implements a `check_laptop_connection` function to ensure a connection to the laptop server:

```
def check_laptop_connection():
    url = f"http://{LAPTOP_IP}:{LAPTOP_PORT}/health"
    try:
        response = requests.get(url, timeout=5)
        return response.status_code == 200
    except requests.exceptions.RequestException:
        return False
```

System Architectural Design

Overview



The architectural design of our project, as shown in the diagram above, is a data-driven distributed system involving multiple Raspberry Pi devices and a laptop provides three services: a Java SpringBoot HTTP server to collect and manage data, a Python server to run algorithms, and a MongoDB database for data storage.

Components and Interactions

- **Raspberry Pi devices**:
    - **Role**: Mentioned earlier to be a BLE scanner as well as a hotspot provider to send data through .
- **Java SpringBoot HTTP server**:
    - **Role**: The Java SpringBoot server serves as the central component responsible for handling HTTP RESTful API requests from Raspberry Pi devices and scheduling gRPC calls to the Python algorithm server to calculate the result. It also integrates with MongoDB for data storage and data management.
    - **Components**:
        - **HTTP RESTful APIs**: Provide HTTP RESTful APIs for the Raspberry Pi devices to send collected data.
        - **gRPC Client**: Maintain high-performance communication with the Python server, calling specific functions periodically to process data.
        - **MongoDB Client**: Manages interactions with the MongoDB database to store and manage data.
- **MongoDB database**:
    - **Role**: MongoDB is the system's database, storing raw and processed data collected from the Raspberry Pi devices.
- **Python algorithm server**:
    - **Role**: The Python algorithm server handles advanced data processing using specific algorithms and provides a gRPC server interface for communication.
    - **Components:**
        - **gRPC Server**: Exposes algorithms below as gRPC services that can be called by the Java Spring Boot HTTP server.
        - **Triangulation algorithm**: Using average RSSI from four different scanners to calculate the position of a certain device.
        - **Data selection algorithm**: Select one record for a certain device to proceed with DBSCAN.
        - **DBSCAN algorithm**: Performs clustering using the DBSCAN algorithm to identify the number of people in the room.

Data Flow

1. **Data Collection**: Raspberry Pi Zero scanners scan for Bluetooth signals and send HTTP requests to the Raspberry Pi 4 collector. Collector integrates data from scanners and sends it to the Java server.
2. **Raw Data Storage**: The Java server runs on the laptop, receives the data through its RESTful APIs, and saves it in MongoDB collection via an integrated MongoDB client.
3. **Data Processing**: The Java gRPC Client periodically calls the Python gRPC server to process the collected data using the algorithms mentioned above (i.e., triangulation(), select_data(), and dbscan()).
4. **Processed Data Storage**: After getting processed data from the Python algorithm server, the Java server saves it in the MongoDB database.
5. **Periodic Updates**: The system ensures continuous data collection, processing, and storage by periodically updating and calling different services across the components.

Data Structure

The raw data sent to the Java server includes the following properties. Each record represents a scan result for a certain device scanned by a Raspberry Pi Zero scanner:

- **Timestamp**: Batch identifier for server processing
- **Device ID**: Scanner identifier
- **MAC Address**: Unique hardware identifier (with privacy considerations)
- **Average RSSI**: Mitigates skewed results from movement
- **Count**: Frequency of MAC address detection
- **First Seen**: Initial detection timestamp
- **Last Seen**: Most recent detection timestamp
- **Duration**: Presence duration (used to filter transient detections)

The processed data in the database will be added with the following properties:

- **X, Y, and Z**: Position of this device
- **In Cube**: If this device is inside the room or not

The DBSCAN result saved in the database includes:

- **MAC Address**: The MAC address of a certain device
- **Cluster ID**: The cluster this device belongs to

Scalability and Extensibility

The system architecture is designed to be scalable and extensible, ensuring it's adaptability in large-scale scenarios:

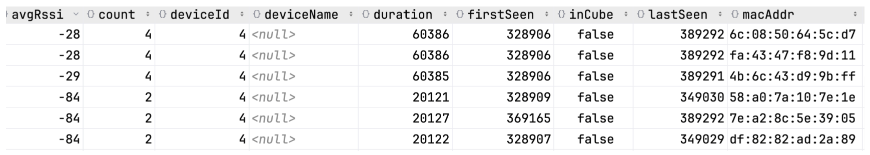
- **Scalability**:
    - Additional Raspberry Pi scanners can be easily integrated to expand data collection endpoints.
    - Multiple collectors can be deployed in different rooms, and the server can process data from different rooms by connecting to the hotspot provided by different Raspberry Pi collectors.
    - The Java and Python servers can be scaled horizontally to handle increased loads.
    - The Java and Python servers can be deployed on different machines to efficiently handle large-scale data.
- **Extensibility**:
    - New algorithms or data processing procedures can be easily added to the Python server without changing the gRPC API.
    - The algorithms on the Python server can be easily replaced with advanced data processing techniques if required.
    - New RESTful APIs and services can be added to the Java Spring Boot HTTP server without breaking the other parts of the system.

Performance Analysis:

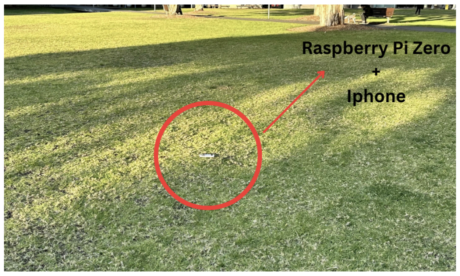
Testing shows an almost 100% data transmission rate done with all 4 Raspberry Pi Zeros sending data at the same time with no little to no latency. This can up to around 7-10 Raspberry Pi Zeros before any delay or loss of connection occurs.

Challenges:

One of the key challenges faced with BLE scanning was the fact that most modern devices randomise MAC addresses as shown below



avgRssi	count	deviceId	deviceName	duration	firstSeen	inCube	lastSeen	macAddr
-28	4	4	<null>	60386	328906	false	389292	6c:08:50:64:5c:d7
-28	4	4	<null>	60386	328906	false	389292	fa:43:47:f8:9d:11
-29	4	4	<null>	60385	328906	false	389291	4b:6c:43:d9:9b:ff
-84	2	4	<null>	20121	328909	false	349030	58:a0:7a:10:7e:1e
-84	2	4	<null>	20127	369165	false	389292	7e:a2:8c:5e:39:05
-84	2	4	<null>	20122	328907	false	349029	df:82:82:ad:2a:89



Where the first three rows all have similar avg rssi values and the same number of count but different MAC addresses. This was tested in an open field with only 1 scanner and 1 iphone next to the scanner as shown in the images. To solve this our algorithms will count the multiple MAC addresses as just one cluster.

Another problem I faced is the power situation of how these devices will be powered. We decided to use power banks to power the Raspberry Pi Zeros as they use barely any power around 100 mah per hour or 1%.

Future Improvements:

To improve the design and setup, having batteries directly connected to the Raspberry Pi Zero would be much more efficient and productive as this would remove dangling cables. Also having better encryption as well using MQTT for improved scalability and reliability.

Methodology and Algorithm

Triangulation:

From RSSI to distance

By using the Free Space Path Loss Model, we can get an equation which could transfer the rssi value to physical distance.

$$RSSI = RSSI1M - 10n\log_{10}d$$

$$\log_{10}d = \frac{RSSI1M - RSSI}{10n}$$

$$d = 10^{\frac{RSSI1M - RSSI}{10n}}$$

RSSI1M is the signal strength measured at 1 meter (usually a known reference value).
RSSI is the received signal strength.
n is the path loss exponent (depending on the environment, generally between 2 and 4).

d is the distance.

The model is based on the following assumptions:
1. The signal propagates in free space, i.e., without the influence of obstacles, reflections, etc.
2. The signal strength decays with increasing distance.

After multiple tests at multiple locations in the school, we came up with values for RSSI1m and n that are suitable for most locations in the school.
(When actually applied to different scenarios, it may be necessary to measure in advance to determine the values that best suit the scenario)

```
RSSI_1m = -66  # RSSI value at 1 meter (dBm)
n = 4  # Environmental factor

# RSSI to distance conversion function
def rssi_to_distance(rssi, RSSI_1m, n):
    return 10 ** ((RSSI_1m - rssi) / (10 * n))
```

Code implementation

From distance to coordinate

When designing this project, we thought of establishing a coordinate system to determine whether the target is in the monitored space. As we all know, two points determine a straight line, three points determine a plane, and four points can determine a space, so using four devices to build a coordinate system became the final solution.
During the test, we assumed that the monitoring space was an ideal cube, which is more conducive to our deployment of Bluetooth model receiving devices.We use the four vertices of the cube as the relative positions for triangulation, which the coordinate are O(0,0,0), A(L,0,0), B(0,L,0), C(0,0,L). L is the length of the cube.

```
# Objective function for least squares
def equations(vars):
    x, y, z = vars
    eq1 = (x - O[0])**2 + (y - O[1])**2 + (z - O[2])**2 - d1**2
    eq2 = (x - A[0])**2 + (y - A[1])**2 + (z - A[2])**2 - d2**2
    eq3 = (x - B[0])**2 + (y - B[1])**2 + (z - B[2])**2 - d3**2
    eq4 = (x - C[0])**2 + (y - C[1])**2 + (z - C[2])**2 - d4**2
    return [eq1, eq2, eq3, eq4]

# Initial guess
initial_guess = [1, 1, 1]

# Solve the system of equations using least squares method
result = least_squares(equations, initial_guess)
x, y, z = result.x

# Print the position of the Bluetooth device (for debugging purposes)
print(f"The position of the Bluetooth device is: x={x:.2f}, y={y:.2f}, z={z:.2f}")
```

Given the estimated distances di from the RSSI values, we formulate the error equations to quantify the difference between the squared estimated distances and the squared Euclidean distances. For each reference point, the error equation is:

$$(x - xi)^2 + (y - yi)^2 + (z - zi)^2 - di^2 = 0$$

These error equations represent the squared difference between the actual Euclidean distance and the estimated distance derived from the RSSI value.

The goal is to find the position (x, y, z) that minimizes the sum of the squares of these error equations. This is achieved using the least squares method, which iteratively adjusts ((x, y, z)) to reduce the total error. Finally, the target point is determined to be inside the cube by comparing the size of L and x y z.

DBSCAN Clustering

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm that can discover clusters of arbitrary shapes and identify noise points. We determine the number of people in a room by calculating the number of clusters produced by DBSCAN. Moreover, feature selection and parameter adjustment are crucial steps that will have a significant impact on the outcome of our project. To address this issue, we process feature engineering first and adjust the parameters of DBSCAN dynamically and automatically using silhouette score when moving to a new environment. The following is an explanation of our parameter selection and adjustment, as well as feature selection.

DBSCAN Principle

1. **Core Point**:
    - A point is a core point if there are at least `min_samples` points (including itself) within a given radius (ε).
2. **Border Point**:
    - A point is a border point if it is not a core point but lies within the ε neighborhood of a core point.
3. **Noise Point**:
    - A point that is neither a core point nor a border point is considered a noise point.

Parameter Selection

1. **eps (ε):**
    - Represents the radius of the neighborhood. Determines how many points must be within a point's ε-neighborhood for it to be considered a core point.
    - We use grid search (`np.arange(0.1, 1.5, 0.1)`) to select the optimal `eps` value, trying values from 0.1 to 1.5 with a step size of 0.1.
2. **min_samples:**
    - Determines the minimum number of points required within a point's ε-neighborhood for it to be considered a core point.
    - We use grid search (`np.arange(2, 10)`) to select the optimal `min_samples` value, trying values from 2 to 10.
    -
3. **Selecting Optimal Parameters:**
    - The silhouette score is used to evaluate the effectiveness of each parameter combination. The silhouette score ranges from [-1, 1], with higher values indicating better clustering results.

      s(i)= b(i)−a(i) /max(a(i),b(i))

Where:

- a(i) is the mean distance between the sample i and all other points in the same cluster.
- b(i) is the mean distance between the sample i and all points in the nearest cluster that the sample i is not a part of.

- The `eps` and `min_samples` values that yield the highest silhouette score are selected as the optimal parameters.

Feature Selection

In this project, we selected the following six features for DBSCAN clustering analysis: `count`, `avg_rssi`, `duration`, `x`, `y`, `z`. Each feature has its specific reasons and significance. Below is a detailed discussion of these features.

1. **Count**
   - Description: Number of times the device signal is detected.
   - Significance: Reflects device activity level. Frequent appearances indicate active movement in the area.
   - Purpose: Distinguish between stationary and frequently moving devices.
2. **Avg_RSSI**
   - Description: Average RSSI value of the device signal.
   - Significance: Indicates received signal strength and device proximity to the receiver.
   - Purpose: Estimate distance between the device and receiver, aiding in spatial location.
3. **Duration**
   - Description: Duration of the device signal.
   - Significance: Reflects the time the device stays within the monitoring area.
   - Purpose: Identify long-staying and transient devices, reducing noise points.
4. **x, y, z**
   - Description: Three-dimensional coordinates of the device.
   - Significance: Provides device position in space, crucial for clustering analysis.
   - Purpose: Aid in identifying spatially clustered devices.

Data Preprocessing

- **Application of StandardScaler**: We use `StandardScaler` from scikit-learn to standardize the features.
- **Scaling Process**: This scaler transforms the data such that each feature has a mean of 0 and a standard deviation of 1.
- **Purpose of Standardization**: This step is crucial to ensure that all features contribute equally to the clustering process. Without standardization, features with larger scales could dominate the distance calculations in the DBSCAN algorithm, leading to biased results. Standardizing the features helps in achieving a balanced contribution from each feature, improving the accuracy and reliability of the clustering results.

Experimental Setup

The following equipment and tools were used in our experiments:

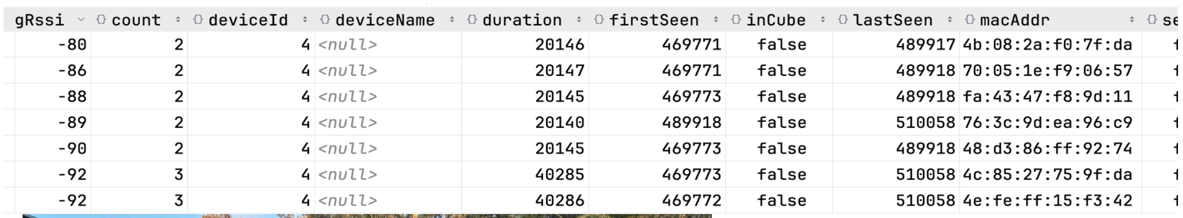
1. Bluetooth Devices: Four Bluetooth receiving devices placed at fixed locations.
    - Device O: Located at the origin (0,0,0).
    - Device A: Located at (L, 0, 0).
    - Device B: Located at (0, L, 0).
    - Device C: Located at (0, 0, L).

    Where L is the distance between two devices.

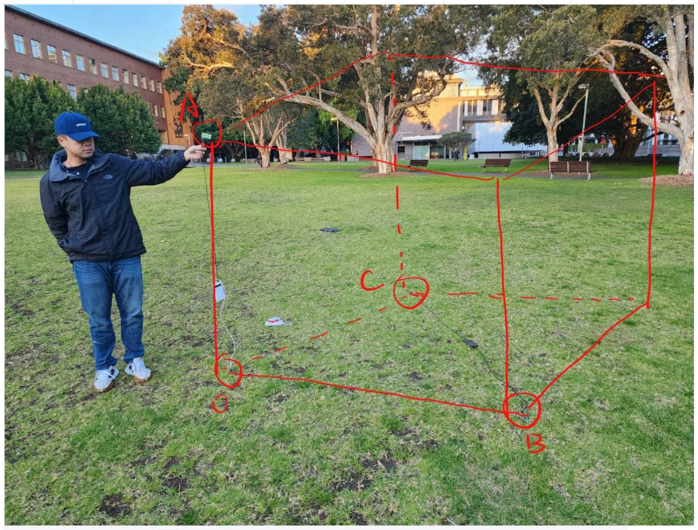
2. Computer: Used for data collection, processing, and running the triangulation method and the DBSCAN algorithm.

Outdoor Environment: The outdoor environment provides a more open and less controlled space, designed to evaluate the algorithm's performance under different conditions. The key characteristics of the outdoor environment are:

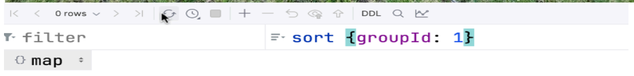
- **Size and Dimensions**: The outdoor space is a large open area with a side length of L meters.
- **Open Space**: The outdoor environment has fewer obstacles, reducing signal reflection and multi-path effects. This allows for a clearer signal propagation path and tests the algorithm's performance in ideal conditions.



gRssi	count	deviceId	deviceName	duration	firstSeen	inCube	lastSeen	macAddr	se
-80	2	4	<null>	20146	469771	false	489917	4b:08:2a:f0:7f:da	{
-86	2	4	<null>	20147	469771	false	489918	70:05:1e:f9:06:57	{
-88	2	4	<null>	20145	469773	false	489918	fa:43:47:f8:9d:11	{
-89	2	4	<null>	20140	489918	false	510058	76:3c:9d:ea:96:c9	{
-90	2	4	<null>	20145	469773	false	489918	48:d3:86:ff:92:74	{
-92	3	4	<null>	40285	469773	false	510058	4c:85:27:75:9f:da	{
-92	3	4	<null>	40286	469772	false	510058	4e:fe:ff:15:f3:42	{

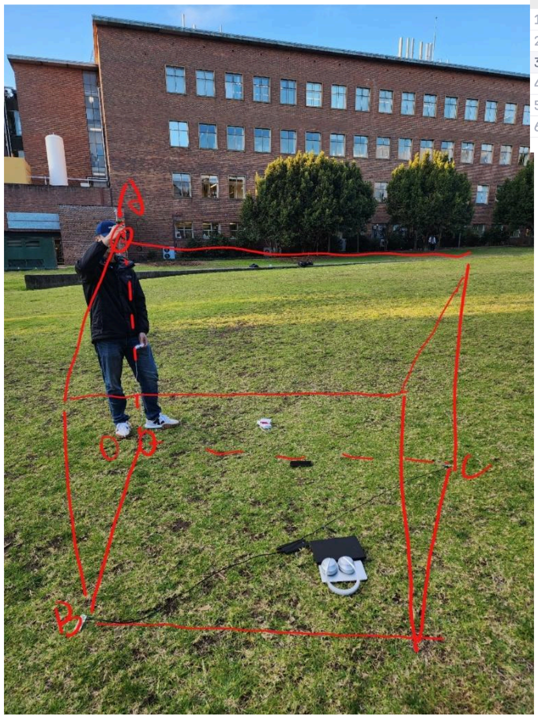


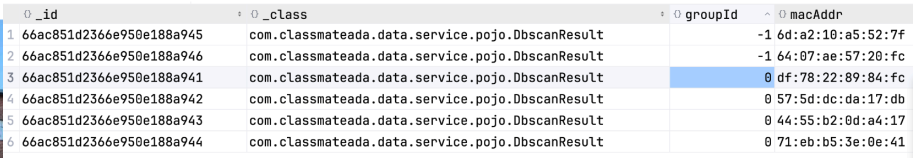
In case one, we used four Raspberry Pi Zero W devices to form a cube with L = 2 which means the distance between 2 Raspberry Pi Zero is 2 meters, with no other devices inside the cube. Data from our database shows that all inCube values are false, indicating that while nearby devices were detected, none of them were inside the cube. Moreover, no data in the database underwent DBSCAN clustering, as only devices inside the cube would be scanned and clustered. This aligns with our expected results, as we did not place any devices inside the cube.



```
0 rows
filter                sort {groupId: 1}
map
```

Result from algorithm scanning with no devices





_id	_class	groupId	macAddr
66ac851d2366e950e188a945	com.classmateada.data.service.pojo.DbscanResult	-1	6d:a2:10:a5:52:7f
66ac851d2366e950e188a946	com.classmateada.data.service.pojo.DbscanResult	-1	64:07:ae:57:20:fc
66ac851d2366e950e188a941	com.classmateada.data.service.pojo.DbscanResult	0	df:78:22:89:84:fc
66ac851d2366e950e188a942	com.classmateada.data.service.pojo.DbscanResult	0	57:5d:dc:da:17:db
66ac851d2366e950e188a943	com.classmateada.data.service.pojo.DbscanResult	0	44:55:b2:0d:a4:17
66ac851d2366e950e188a944	com.classmateada.data.service.pojo.DbscanResult	0	71:eb:b5:3e:0e:41

Result from algorithm scanning with devices

In this case, we placed four devices inside the cube: a mobile phone, a laptop, a headphone, and a tablet. After running the algorithm, we can see from the database results that there are four devices inside the cube, all belonging to cluster 0. This means that these four devices either belong to the same person or have very similar characteristics and distances because we placed them very close together. Also, we can see the presence of cluster -1, indicating that these devices are noise points. They do not have sufficient features to be considered as core points, which means they cannot represent devices belonging to the same person. We talked about this before,environmental factors, such as signal interference or the dynamic nature of devices (e.g., MAC address changes), can also cause this issue.

Limitations

The triangulation method requires the parameters RSSI1M and the environmental factor n to be re-measured each time the device is deployed. Determining n involves multiple measurements and comparisons, which is inconvenient for real-world applications. Additionally, n can vary due to factors like signal reflection and device density, even within the same environment. We use a standard cube for simplicity, but the coordinate system can be adapted to spaces of various shapes if accurately measured and adjusted for the algorithm's incube conditions.

Signal interference, reflection, and changes in device density can cause DBSCAN results to be unstable. For example, variations in RSSI and n within the same environment affect clustering accuracy. Devices that frequently change their MAC addresses create multiple noise points, impacting results even when classified as noise. Moreover, calculating the optimal `eps` and `min_samples` values for each new data set enhances adaptability but increases computational costs.

Conclusion

Project Achievements

Cost-Effectiveness: Utilizing affordable hardware, including Raspberry Pi Zero W devices for BLE scanning and a Raspberry Pi 4 for WiFi, reduces costs while maintaining effectiveness. **Non-Invasive**

Monitoring: BLE signals preserve privacy and require minimal space, providing a discreet occupancy monitoring solution. **Scalability**: The system design supports easy expansion, suitable for larger spaces and diverse environments, with additional Raspberry Pi scanners seamlessly integrated. **Versatility**: Applicable in various scenarios, from pandemic safety to daily space management and energy efficiency improvements.

Technical Insights

Data Preprocessing: Standardizing features with StandardScaler ensures equal feature contribution, enhancing clustering accuracy and reliability. **DBSCAN Clustering**: DBSCAN identifies clusters of arbitrary shapes and detects noise points, crucial for determining room occupancy. **Triangulation and Localization**: Using RSSI values to estimate distances and establish coordinates allows accurate device localization within the monitored space.

Challenges and Limitations

Environmental Factors: Signal interference, reflection, and changes in device density can affect DBSCAN results. Variations in RSSI and environmental factor n impact clustering accuracy. **MAC Address Randomization**: Frequent MAC address changes create multiple noise points, impacting results despite being classified as noise. **Computational Cost**: Automatically calculating optimal `eps` and `min_samples` for each data set increases adaptability but also computational costs. **Power Supply Issues**: Ensuring a stable power supply for Raspberry Pi devices was challenging. Power banks provided a temporary solution, but a permanent setup is needed for long-term deployment.

Future Work

To enhance the system, future improvements could focus on:

- **Robustness Against Noise**: Developing methods to handle MAC address randomization and other noise sources for reliable occupancy detection.
- **Improved Security**: Enhancing data transmission security, possibly using protocols like MQTT.
- **Power Optimization**: Exploring efficient power solutions for continuous Raspberry Pi operation without external power banks.

Final Thoughts

Our project demonstrated a scalable, non-invasive, cost-effective solution for building occupancy monitoring using BLE technology. Despite challenges, the system's adaptability and the insights gained pave the way for future advancements in smart, efficient, and safe space management solutions. By refining and expanding this technology, we can significantly improve real-time occupancy monitoring and its applications across various domains.

GITHUB LINKS - NEEDS PERMISSION FROM ADA (since its private):

https://github.com/classmateada/6733-algorithm-scaffold

https://github.com/classmateada/6733-server