**1. Tổng quan về Processor**

* **Vai trò**: processor.py chịu trách nhiệm:
  1. Lấy dữ liệu thô từ Redis (queue beacon\_data).
  2. Xử lý dữ liệu để xác định Gateway gần nhất với mỗi Beacon dựa trên RSSI (Received Signal Strength Indicator) và tần suất (frequency).
  3. Cập nhật trạng thái Beacon vào Redis (hash beacon\_state và queue aws\_queue).
* **Input**: Dữ liệu thô từ beacon\_data (được listener.py đẩy vào).
* **Output**: Trạng thái Beacon (Gateway gần nhất, detected/not detected) lưu vào Redis.

**2. Cách Processor lấy dữ liệu**

**Dữ liệu đầu vào**

* **Nguồn**: Redis queue beacon\_data.
* **Định dạng**: Mỗi phần tử trong queue là một chuỗi JSON:

json

CollapseWrapCopy

{

"beacon\_id": "Beacon1",

"gateway\_id": "GW1",

"rssi": -70,

"timestamp": "2025-03-11T10:00:00"

}

* **Cách lấy**: Processor dùng lpop để lấy từng phần tử từ đầu queue beacon\_data, xử lý tối đa 1000 phần tử mỗi lần để tránh quá tải.

**Quy trình lấy dữ liệu**

1. **Kiểm tra queue**:
   * Dùng redis\_client.llen("beacon\_data") để lấy số phần tử trong queue.
   * Lấy tối đa min(redis\_client.llen("beacon\_data"), 1000) để giới hạn số lượng xử lý mỗi vòng lặp.
2. **Lấy dữ liệu thô**:
   * Dùng vòng lặp for với redis\_client.lpop("beacon\_data") để lấy từng phần tử.
   * Mỗi phần tử được giải mã từ chuỗi JSON (json.loads(data.decode())).
3. **Lưu tạm vào bộ nhớ**:
   * Processor dùng cấu trúc defaultdict để lưu dữ liệu theo beacon\_id và gateway\_id:

python

CollapseWrapCopy

beacon\_data = defaultdict(lambda: defaultdict(lambda: deque(maxlen=MAX\_BUFFER\_PER\_BEACON)))

* + Mỗi Beacon (beacon\_id) có một danh sách dữ liệu RSSI và timestamp cho từng Gateway (gateway\_id), lưu trong deque với kích thước tối đa MAX\_BUFFER\_PER\_BEACON (mặc định 100).

1. **Loại bỏ dữ liệu cũ**:
   * Trong mỗi vòng lặp, Processor kiểm tra timestamp và loại bỏ dữ liệu ngoài WINDOW\_SIZE (mặc định 10 giây):

python

CollapseWrapCopy

current\_time = time.time()

beacon\_data[beacon\_id][gw] = deque(

[(r, t) for r, t in beacon\_data[beacon\_id][gw]

if current\_time - time.strptime(t, "%Y-%m-%dT%H:%M:%S").timestamp() <= WINDOW\_SIZE],

maxlen=MAX\_BUFFER\_PER\_BEACON

)

**3. Cách Processor xử lý dữ liệu**

**Thuật toán xử lý**

Processor xác định Gateway gần nhất cho mỗi Beacon dựa trên **RSSI trung bình** và **tần suất phát hiện** trong khoảng thời gian WINDOW\_SIZE. Các bước xử lý:

1. **Tính toán cho từng Beacon**:
   * Với mỗi beacon\_id trong beacon\_data, Processor duyệt qua tất cả Gateway liên quan.
2. **Kiểm tra ngưỡng tần suất**:
   * Số lượng mẫu dữ liệu (len(rssi\_list)) phải >= FREQ\_THRESHOLD (mặc định 3):

python

CollapseWrapCopy

if len(rssi\_list) >= FREQ\_THRESHOLD:

1. **Tính RSSI trung bình**:
   * Tính trung bình RSSI từ danh sách mẫu:

python

CollapseWrapCopy

rssi\_avg = sum(rssi\_list) / len(rssi\_list)

* + Nếu rssi\_avg > RSSI\_THRESHOLD (mặc định -90), Gateway này được xem xét.

1. **Tính điểm số (Score)**:
   * Dùng hàm calculate\_score để kết hợp RSSI và tần suất:

python

CollapseWrapCopy

def calculate\_score(rssi\_avg, freq):

rssi\_normalized = 100 - abs(rssi\_avg) *# Chuyển RSSI thành giá trị dương (càng gần 0 càng mạnh)*

freq\_normalized = min(freq, MAX\_FREQ) / MAX\_FREQ *# Giới hạn tần suất tối đa (mặc định 10)*

return (W1 \* rssi\_normalized) + (W2 \* freq\_normalized \* 100)

* + W1 (mặc định 0.5): Trọng số RSSI.
  + W2 (mặc định 0.5): Trọng số tần suất.

1. **Chọn Gateway gần nhất**:
   * So sánh điểm số giữa các Gateway, chọn Gateway có điểm cao nhất:

python

CollapseWrapCopy

nearest\_gw = max(scores, key=scores.get)

1. **Xử lý song song**:
   * Dùng ThreadPoolExecutor với MAX\_WORKERS (mặc định 10) để xử lý nhiều Beacon cùng lúc:

python

CollapseWrapCopy

executor.map(process\_beacon, temp\_data)

**4. Cập nhật trạng thái**

**Quy tắc cập nhật**

* **Beacon được phát hiện (detected = 1)**:
  + Nếu có Gateway đạt ngưỡng (score > 0), cập nhật trạng thái mới.
  + Nếu trạng thái thay đổi (Gateway khác hoặc lần đầu phát hiện), đẩy vào aws\_queue.
* **Beacon không còn được phát hiện (detected = 0)**:
  + Nếu không còn dữ liệu hợp lệ trong WINDOW\_SIZE, xóa trạng thái và đẩy thông báo "không phát hiện" vào aws\_queue.

**Chi tiết cập nhật**

1. **Trạng thái mới hoặc thay đổi**:
   * Lưu vào hash beacon\_state:

python

CollapseWrapCopy

beacon\_state[beacon\_id] = {"gateway": nearest\_gw, "start\_time": ts, "last\_time": ts}

redis\_client.hset("beacon\_state", beacon\_id, json.dumps(beacon\_state[beacon\_id]))

* + Đẩy vào aws\_queue:

python

CollapseWrapCopy

redis\_client.rpush("aws\_queue", json.dumps({

"beacon\_id": beacon\_id,

"gateway\_id": nearest\_gw,

"detected": 1,

"timestamp": ts

}))

1. **Cập nhật timestamp nếu không đổi Gateway**:
   * Chỉ cập nhật last\_time trong beacon\_state:

python

CollapseWrapCopy

beacon\_state[beacon\_id]["last\_time"] = ts

redis\_client.hset("beacon\_state", beacon\_id, json.dumps(beacon\_state[beacon\_id]))

1. **Beacon không còn phát hiện**:
   * Nếu không có Gateway nào đạt ngưỡng:

python

CollapseWrapCopy

redis\_client.rpush("aws\_queue", json.dumps({

"beacon\_id": beacon\_id,

"gateway\_id": beacon\_state[beacon\_id]["gateway"],

"detected": 0,

"timestamp": time.strftime("%Y-%m-%dT%H:%M:%S")

}))

del beacon\_state[beacon\_id]

redis\_client.hdel("beacon\_state", beacon\_id)