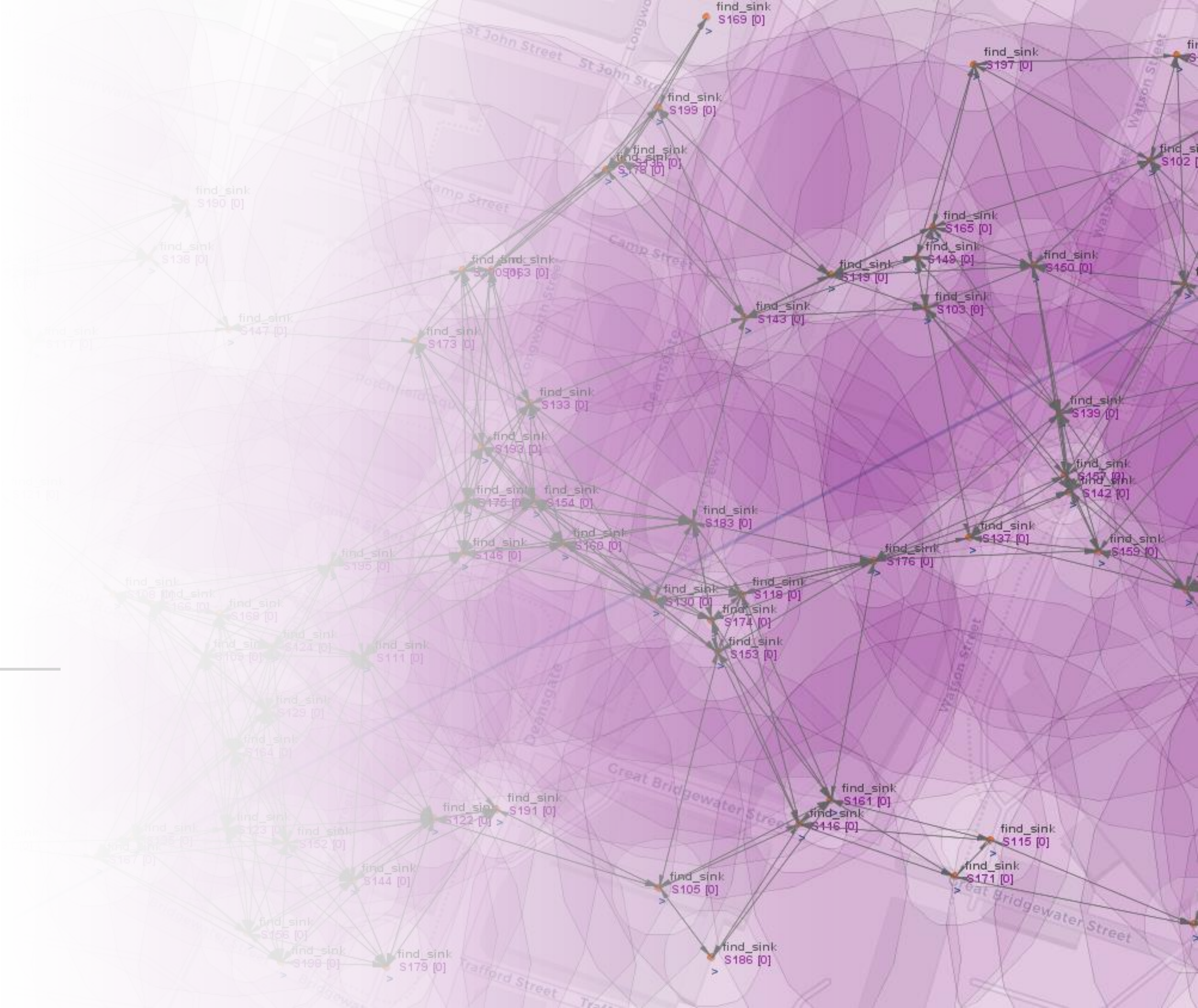
# Multiple Sink Discovery

IoT Part 1 Final Project

Made by Ciro Ogliastro



## The problem to solve

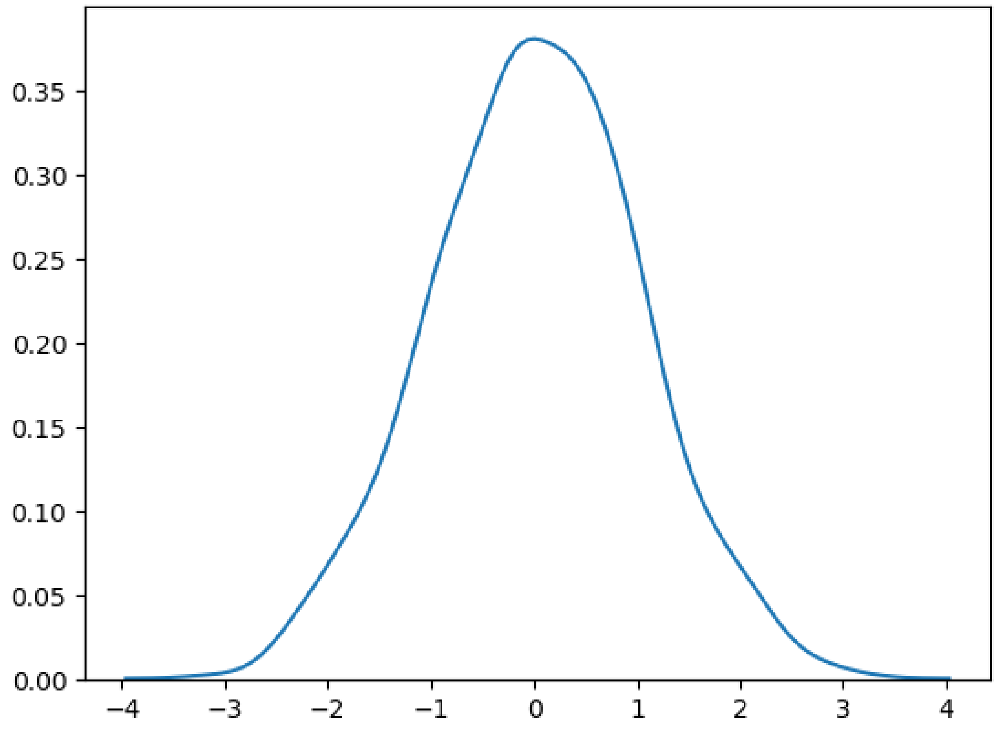
- Given a dense WSN with at least 50 nodes, identify approximately 10 candidate nodes for sink roles.
- Sink nodes must be located on the network's external perimeter and be evenly distributed geographically (more or less).

## Approach to the Solution

- How can we distinguish external perimeter nodes from internal ones?
- My proposed solution is based on this observation: **on average, perimeter nodes have less neighbors than internal nodes**
- Therefore, the problem becomes to **identifying nodes with fewer neighbors than the average.**
- These nodes are **likely** to be on the network's external perimeter.

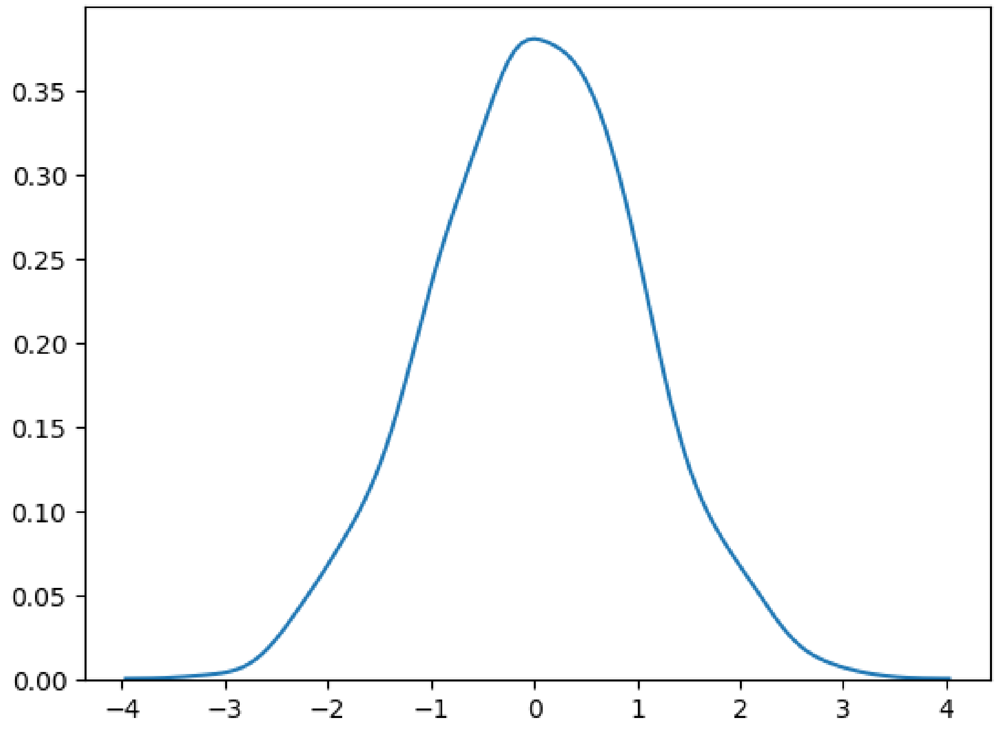
# Reasoning on the solution

- Nodes are randomly generated following a normal distribution, meaning their number of neighbors also follows a normal distribution.
- I'm interested in nodes with neighbor counts falling on the **left side** of the distribution.
- Therefore I considered only a **portion of the average**
- All the nodes that have a number of neighbors less than this threshold are likely external perimeter nodes.
- Computing the **local average** it's a good enough approximation



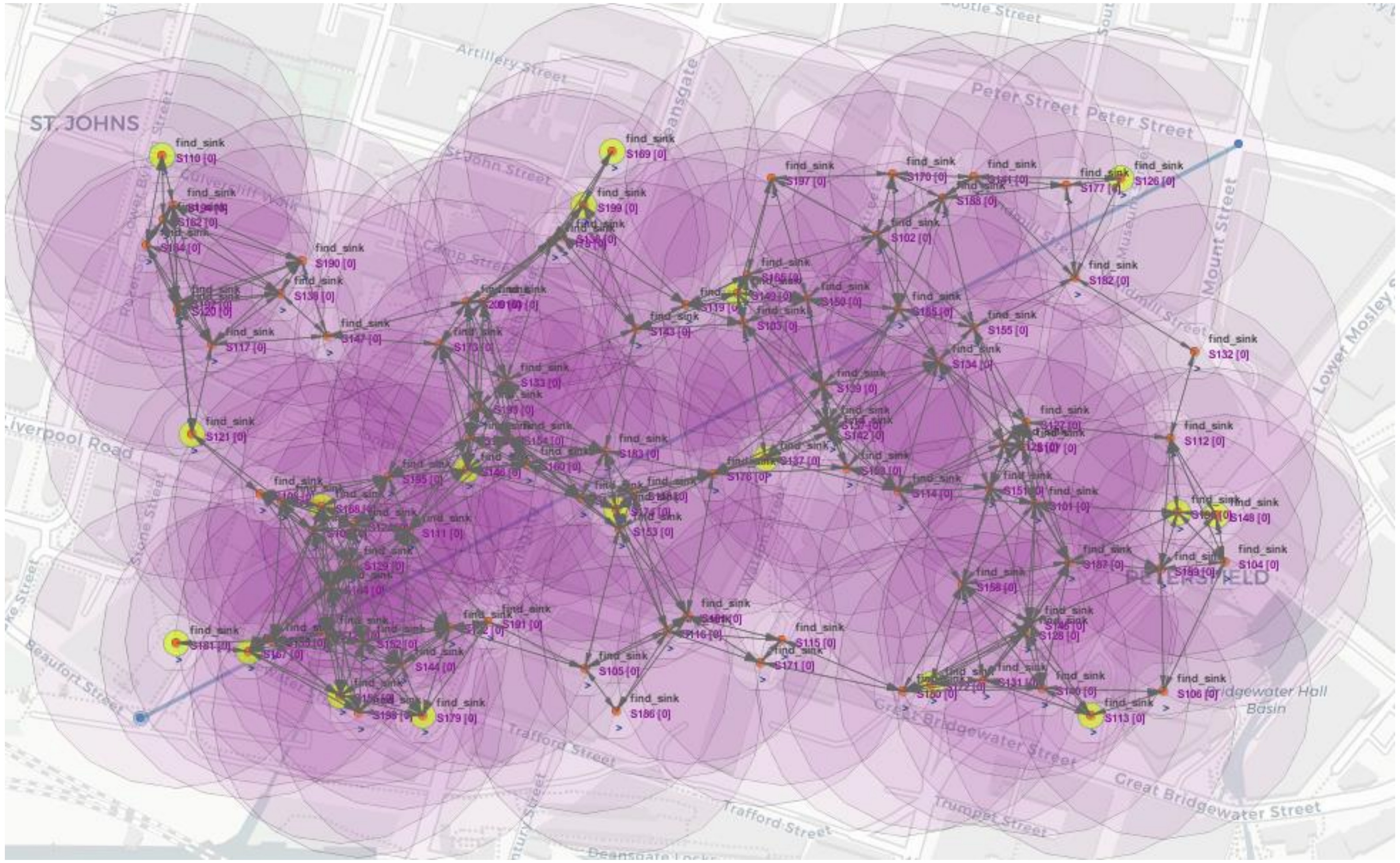
# Parameters of the Algorithm: Sensitivity

- The first parameter is called "**sensitivity**"
- It's a tunable parameter between **0 and 1** that is multiplied by the local average of each node
- **Higher sensitivity** (closer to 1) selects nodes nearer to the average (center of the Gaussian).
- **Lower sensitivity** (closer to 0) selects nodes nearer to the lower tail of the distribution.



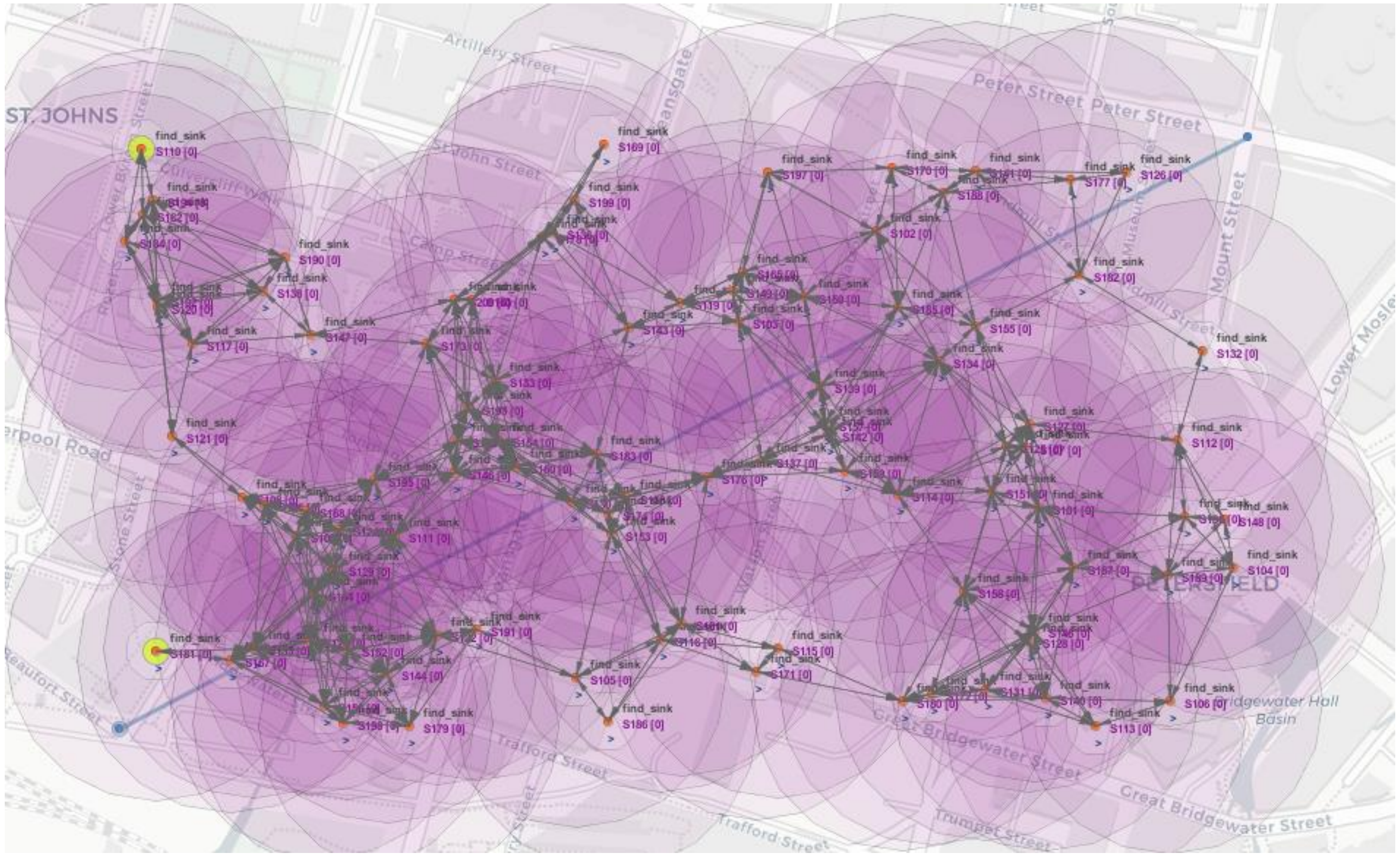
## Simulation with Sensitivity=0.90 (Too High)

- Too many nodes marked.
- Internal nodes are marked



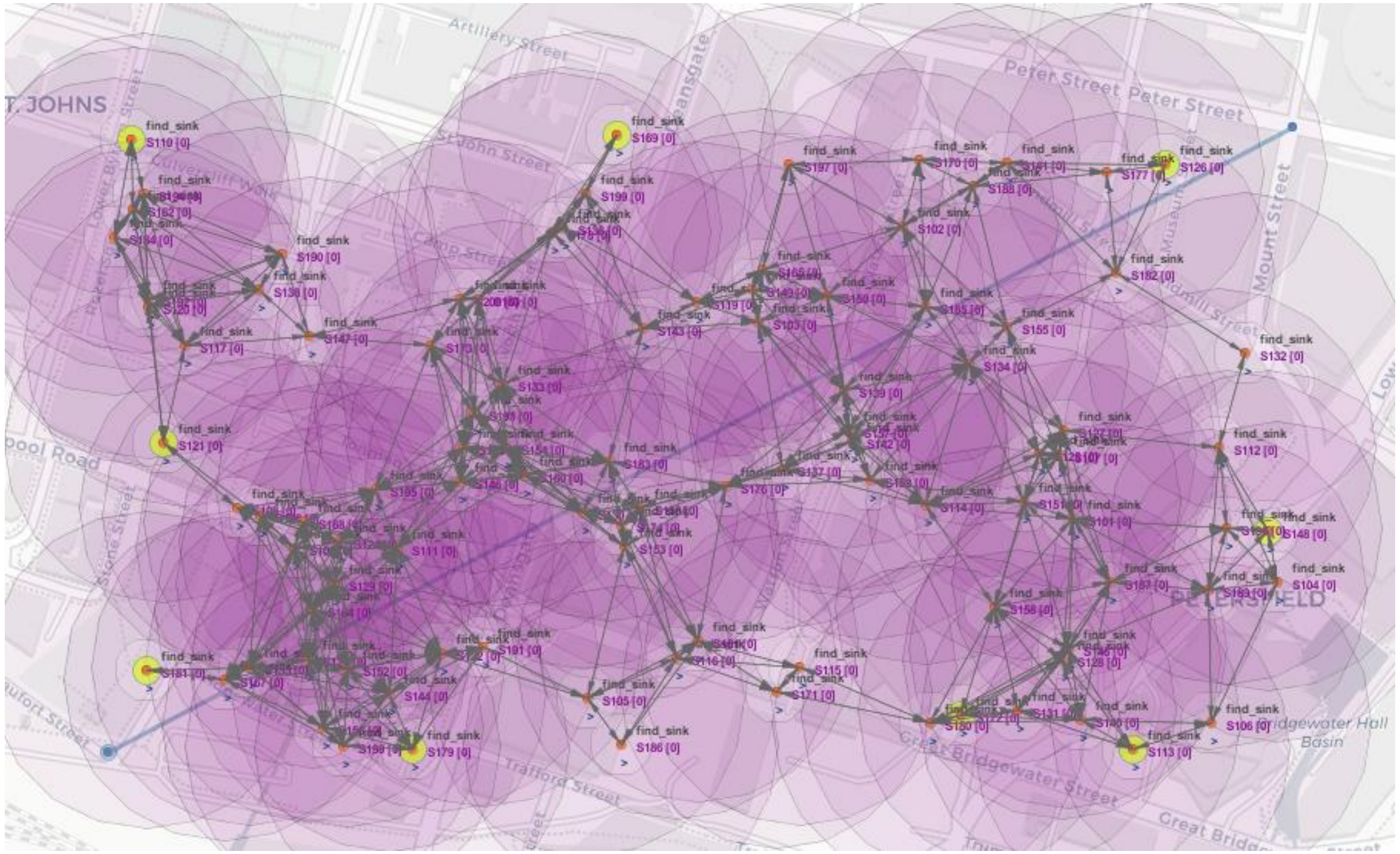
## Simulation with Sensitivity=0.50 (Too Low)

- Too few nodes marked (the goal is about 10 nodes)
- Only external perimeter nodes are marked



## Simulation with Sensitivity=0.74 (Optimal)

- After a few trial and error
- 9 marked nodes,
- All marked nodes on the external perimeter



## Parameters of the algorithm: RSSI

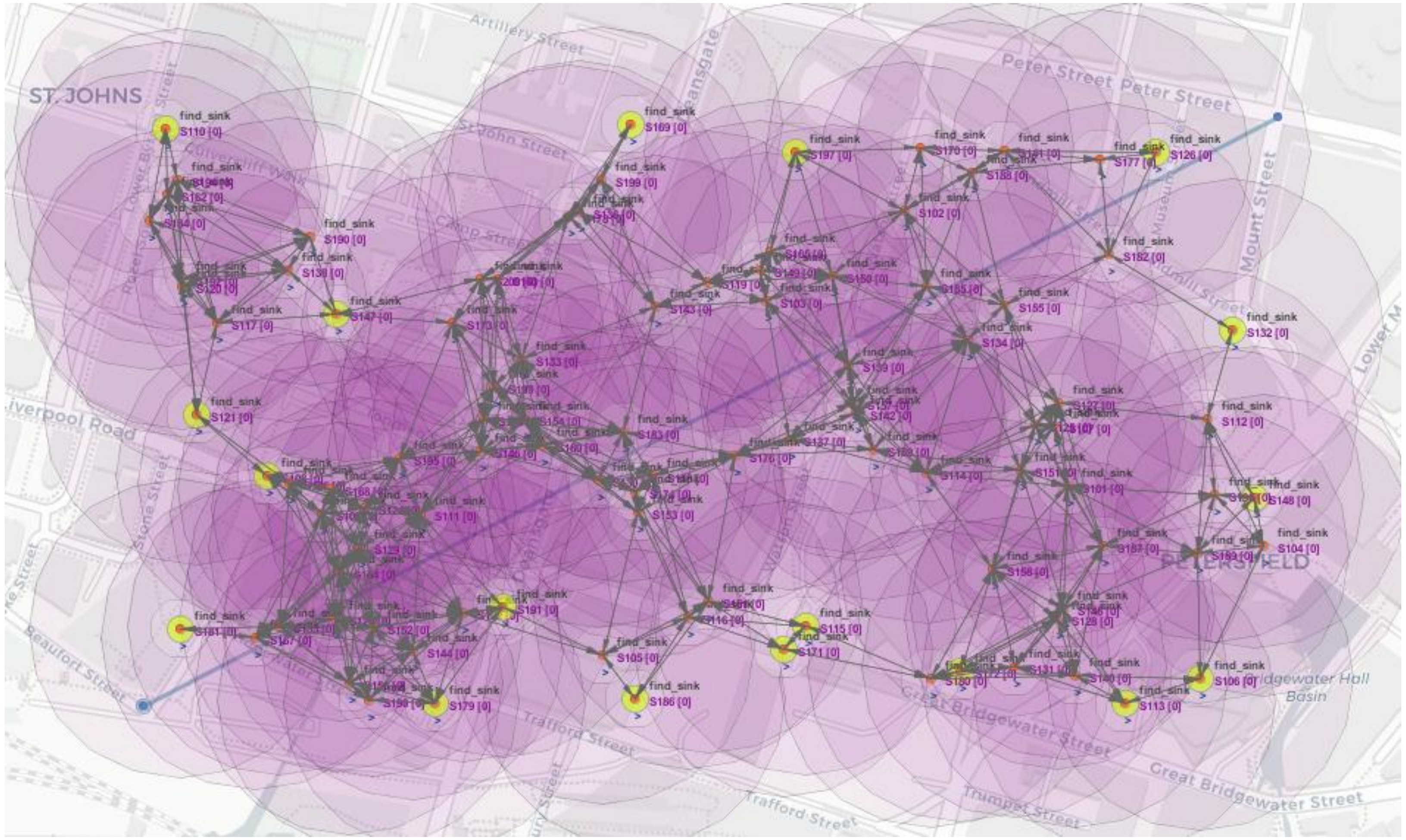
- Sometimes, internal nodes may have fewer neighbors than average, causing the algorithm to mark them incorrectly.
- To improve accuracy, the **local average RSSI** it is considered
- An improvement is to consider the **local average RSSI**
- The RSSI (Received Signal Strenght Indicator) measures the power level of a received wireless signal
- It can help differentiate perimeter nodes from internal ones

## Parameters of the algorithm: RSSI

- Nodes on the perimeter in general have a similar **local average RSSI**
- For more accuracy, I considered two **thresholds**: a minimum RSSI and maximum RSSI
- A node is marked as a sink if (both are true):
    1. It has **fewer neighbors than the local average**.
    2. Its **local average RSSI falls between the two thresholds**.

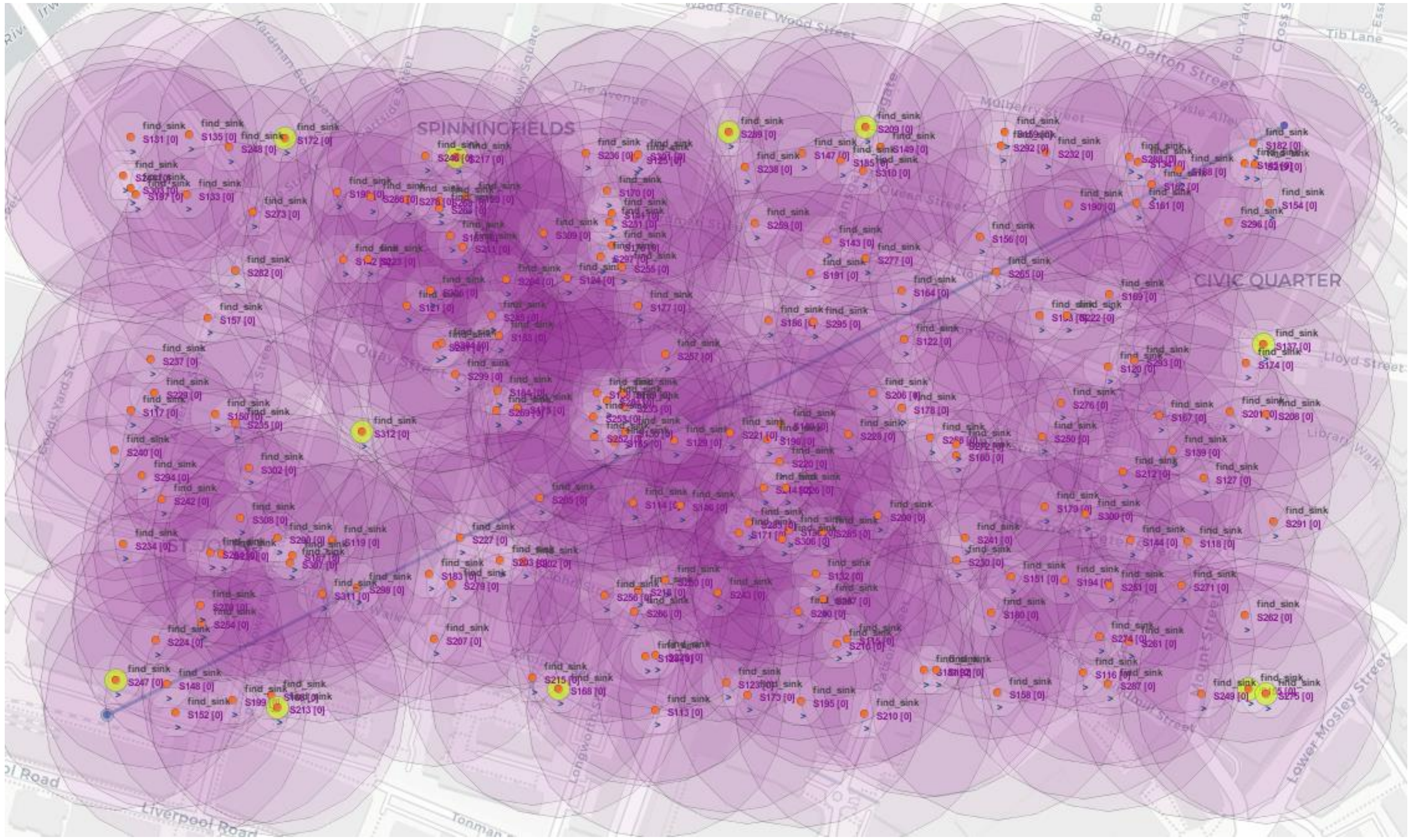
## Simulation - no RSSI threshold

- RSSI min=0
- RSSI max =100
- Sensitivity=0.74
- 18 marked nodes



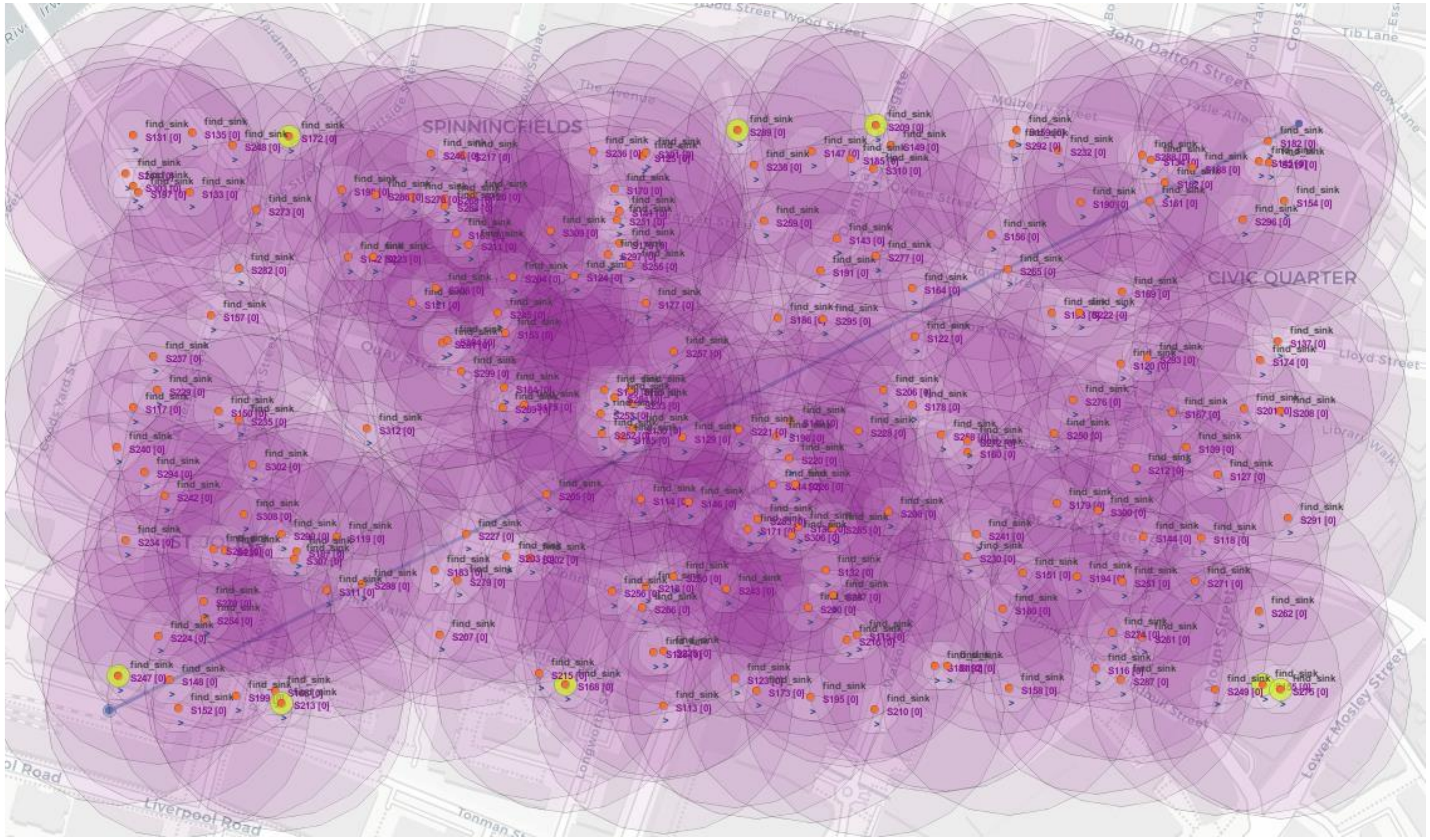
## Simulation - Different Network with 200 Nodes

- RSSI min = 0
- RSSI max = 100
- Sensitivity = 0.67
- Notice that 1 internal node is **incorrectly marked** as sink



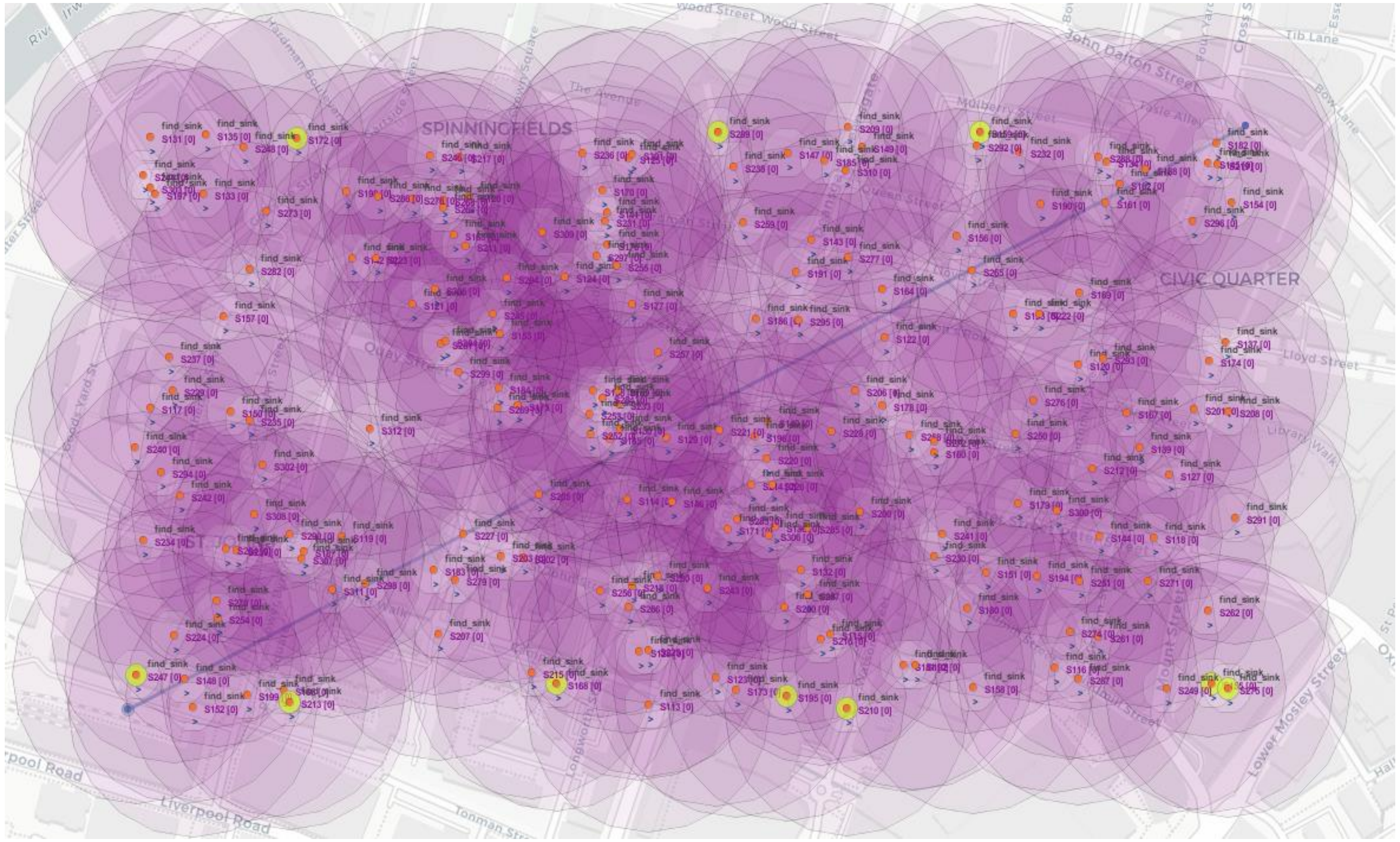
# Simulation – Lowering maximum RSSI

- RSSI min = 0
- **RSSI max = 65**
- Sensitivity=0.67
- **Internal node is no longer marked**
- But we lost some good perimeter nodes.



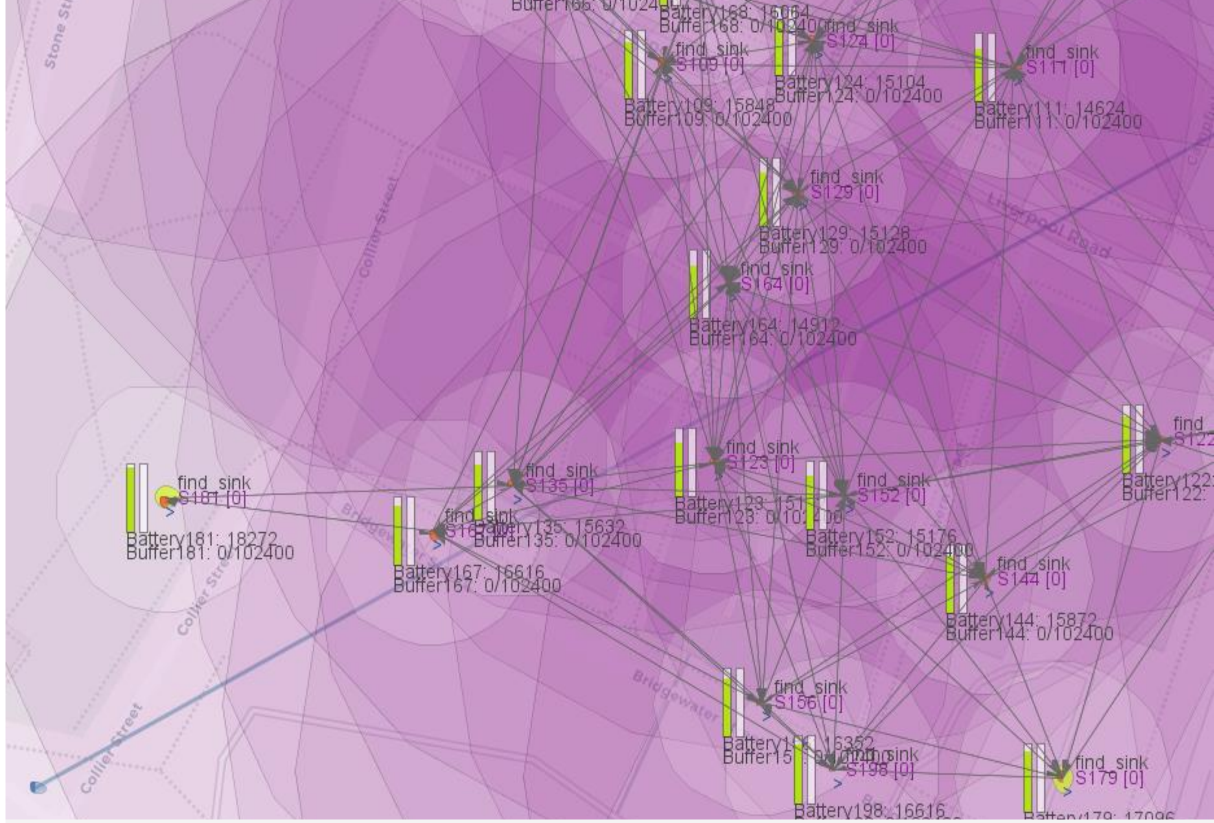
# Simulation – Optimal Parameters

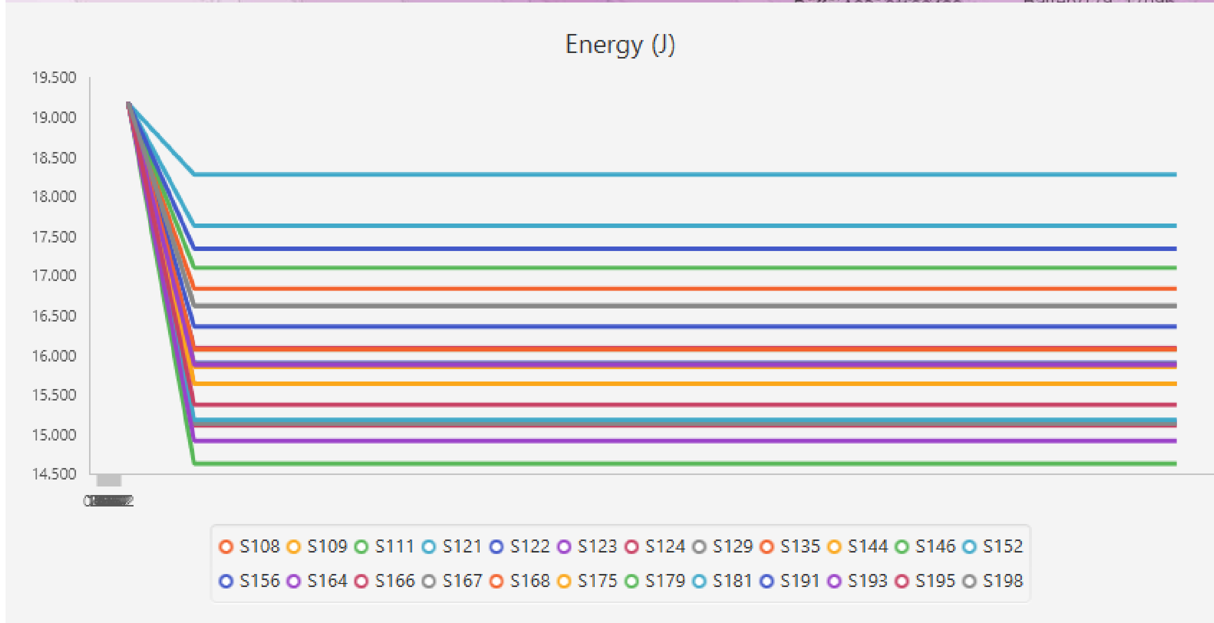
- RSSI min = **45**
- RSSI max = **65**
- Sensitivity = **0.70**
- After tweaking the parameters, **exactly 10 nodes were correctly marked**.



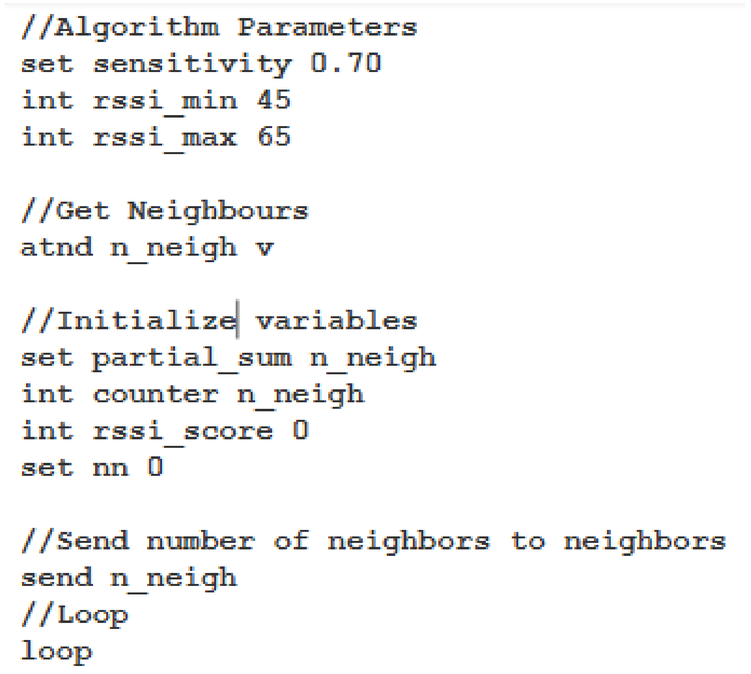
# Energy Consumption & Network Lifetime

- For **energy consumption,** I considered the following values **E_TX=6, E_RX=3**
- The algorithm doesn't require much energy, nodes makes simple arithmetic operations, transmit one time and receive multiple times.
- Internal nodes tend to consume more energy because they receive more packets.
- Energy consumption grows as the number of nodes and number of connections grows.





# Comments to the SenScript Code

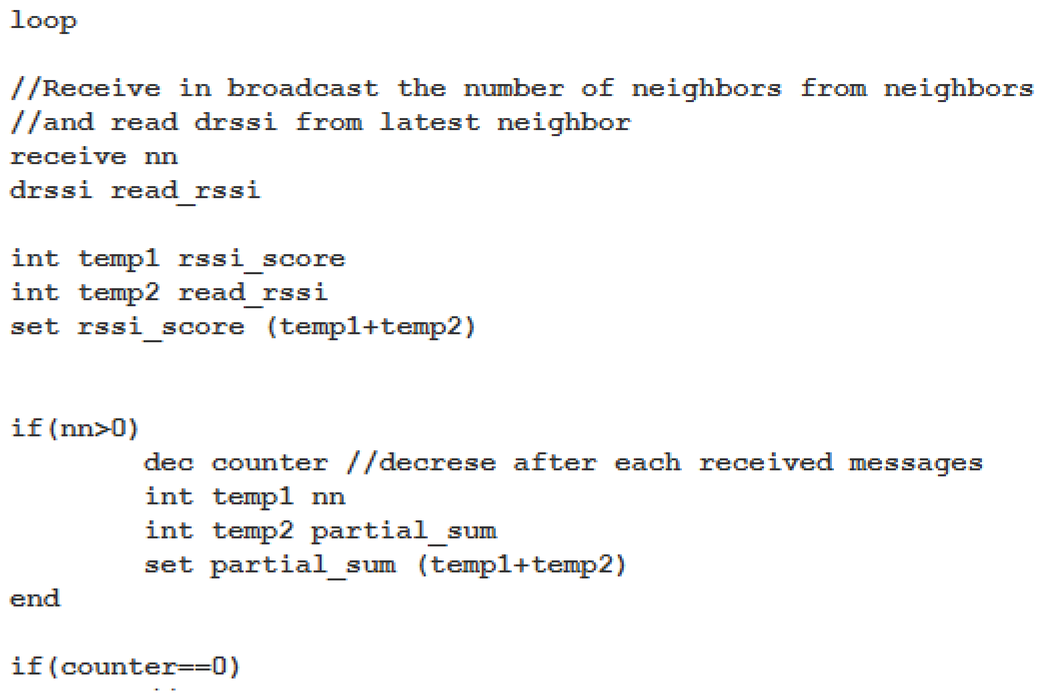


```
//Algorithm Parameters
set sensitivity 0.70
int rssi_min 45
int rssi_max 65

//Get Neighbours
atnd n_neigh v

//Initialize variables
set partial_sum n_neigh
int counter n_neigh
int rssi_score 0
set nn 0

//Send number of neighbors to neighbors
send n_neigh
//Loop
loop
```



```
loop

//Receive in broadcast the number of neighbors from neighbors
//and read drssi from latest neighbor
receive nn
drssi read_rssi

int temp1 rssi_score
int temp2 read_rssi
set rssi_score (temp1+temp2)


if(nn>0)
        dec counter //decrese after each received messages
        int temp1 nn
        int temp2 partial_sum
        set partial_sum (temp1+temp2)
end

if(counter==0)
```

# Comments to the SenScript Code

```
if(counter==0)
        //Compute average number of neighbors
        int temp1 partial_sum
        int temp2 n_neigh
        set average temp1/temp2


        //Compute a portion of the average
        set temp1 average*sensitivity
        int paverage temp1


        //Compute the local average DRSSI
        int temp1 rssi_score
        int temp2 n_neigh
        set avg_rssi_score temp1/temp2

        //Debug
        //print "rssi_score=" avg_rssi_score " n_neigh=" n_neigh " pdaverage=" paverage

        //Check if the number of neighbors is less than "paverage"
        //and if it's between the two drssi thresholds
        if((n_neigh<paverage)&&(avg_rssi_score>rssi_min)&&(avg_rssi_score<rssi_max))
                mark 1
        end
        stop
end

delay 1000
```

## Pros and Cons of the solution

### Pro

- ✓Energy efficient
- ✓Few data transmission
- ✓Very Fast
- ✓Works on every dense network

### Cons

- Cannot mark an exact number of nodes, depends on network topology.
- Accuracy varies with network structure and parameter selection (Sensitivity & RSSI).
- Isn't accurate on networks with fewer nodes

## Possible Improvements

- **Considering battery levels** as an additional factor
- Internal nodes have more neighbors, receive more packets, and **consume more energy**.
- The problem with this approach is that it would require **more thresholds**, making the algorithm **more complex and** requires **more parameter tuning**

Thank you for listening