POLITECNICO
MILANO 1863

Internet Of Things - Wireless Internet

Finding Dory

Andrea Prisciantelli (10618568)
Riccardo Reggiani (10670577)

Computer Science and Engineering
Prof. Matteo Cesana - Prof. Alessandro Enrico Cesare Redondi

AA 2021/22

Introduction

The main purpose of the project is to find the coordinates (X,Y) of the position of Dory by using a **fingerprint dataset** composed of Received Signal Strength Indicator (RSSI) values from six **sniffer-anchors** and the RSSIs emitted by Dory's smartphone in its corresponding coordinates.

The fingerprint dataset has been divided in small fragments of data hidden inside **MQTT publishes/subscriptions** and **CoAP request/response packets** and it was required to obtain all these fragments, clean them and remove the outliers.

After reconstructing the entire fingerprint dataset, it was necessary to find a **model** that was able to determine the position of the device with the most similar fingerprint, comparing the RSSI measurements from the Dory's device with the database's entries.

Fingerprint Dataset

Since the fragments of dataset were hidden inside packets of different protocols, it was necessary to implement different strategies in order to retrieve them in the easiest way.

Constrained Application Protocol (CoAP)

Some of the fragments that were required to obtain are hidden within **CoAP GET/POST/PUT/DELETE** resource responses and **CoAP resource observe messages**.

In order to discover the list of existing resource paths, send requests and view responses in a more efficient way, the client **Copper**, available as a Firefox plugin, has been used. The fragments have been retrieved as responses related to different kinds of **resource requests** (with a short description of its typical usage):

- `GET`: read the value of a resource
- `POST/PUT`: change the value of a resource or create a new one
- `DELETE`: delete a resource
- `OBSERVE`: obtain periodically all the new values of a resource

Most of the fragments were obtained by sending requests related to well known **resource paths**. However, for some positions, it was necessary to send manual requests to **hidden resource paths** that were suggested by looking at MQTT publish message, as described later.
CoAP has been used also in order to retrieve the **RSSI vector of Dory's hacked smartphone**.

The **list of positions** retrieved sniffing the content of CoAP packets is the following:

Position (X,Y)	Request type	Resource path	Additional Info
(2.0,4.0)	PUT	/root/putThis	
(6.0,0.0)	Observe	/root/ObserveThisResource	
(2.0,6.0)	Observe	/root/HardFindingDoryHere	
(10.0,6.0)	Observe	/root/CanYouSeeMe	
(2.0,10.0)	GET	/root/CanYouSpotDory	
(4.0,8.0)	DELETE	/root/DeleteMe7	
(8.0,0.0)	POST	/root/DontDeletePlease	
(4.0,4.0)	GET	/root/GetMe10	
(4.0,10.0)	POST	/root/PSherman42WallabyWaySidney	
(0.0,6.0)	DELETE	/root/PlasticBag	
(0.0,4.0)	PUT	/root/SomethingToDoHereMaybe	
(8.0,8.0)	POST	/root/SomethingUseful	
(6.0,8.0)	DELETE	/root/ThisIsTrash	
(4.0,0.0)	GET	/root/ThisIsUseful	
(6.0,2.0)	PUT	/root/WhereIsDory	
(6.0,10.0)	POST	/root/BarrierReef/Anemone?owner=Marlin	
(0.0,2.0)	POST	/root/BarrierReef/Doctor?problem=memory	
(10.0,0.0)	GET	/root/questions/WasDoryHere?answer=yes	
(2.0,2.0)	POST	/root/PostMe6?search=entry	
(8.0,2.0)	GET	/root/BarrierReef/Apps?fingerprint=True&gps=False	
(10.0,10.0)	GET	/root/BarrierReef/FishLocator?user=Dory	
(8.0,10.0)	GET	/root/BarrierReef/HiddenTreasure	hidden
Dory's RSSIs vector	Observe	/root/BarrierReef/Dory	hidden

Message Queuing Telemetry Transport (MQTT)

Some fragments are hidden inside **MQTT publish/subscribe messages** instead. In order to discover all of them, which is only possible by subscribing to all the topics and listen to their publish messages, the client **Mosquitto** has been used and the following command was typed in the terminal:

```
mosquitto_sub -t '#' -h 131.175.120.117 -p 1883
```

where:

- `-t`: the topic we subscribe to, in this case the multi-level wildcard `'#'` means we are subscribing to all the possible topics in the specified server,
- `-h`: identifies the host we are connecting to (131.175.120.117 is only reachable with PoliMi VPN)
- `-p`: the port to which we are listening for incoming publish messages

MQTT messages have been used also as **hints** for some particular CoAP requests.

Reconstructing the dataset

In order to re-construct the entire fingerprint dataset, it was necessary to manually **clean the messages** and discard the incorrect ones. After this step, a `txt` file containing all the correct fragments has been produced and given as input to a Python script.
This simple Python script (attached to the project `.zip` folder) is responsible for reading this file, parsing the rows and creating a matrix with all the entries of the fingerprint dataset.

Since the obtained fragments are only associated to **even positions**, before retrieving the position of Dory it was necessary to find a way to reconstruct the remaining entries associated to odd positions.

Assuming that no particular obstacles are present at any of the odd positions, it was relatively easy to create a model/algorithm which calculates the RSSIs from the six sniffer-anchors.

The main idea for this algorithm was to compute the RSSIs for each sniffer-anchor as an **average** between the closest known positions. In particular, we distinguish between three different types of positions, based on their `(x,y)` indexes:

1. **Both `x` and `y` are odd**: the cell corresponding to the `(x,y)` position is surrounded by **four even cells** (cells with both x and y values even). The RSSIs vectors can be computed as an average between the vectors of these four positions.
2. **Only `x` is odd**: the cell corresponding to the `(x,y)` position is in the middle of **two even cells** located at `(x+1,y)` and `(x-1,y)`. The RSSIs vectors can be computed as an average between the vectors of these two positions.
3. **Only `y` is odd**: the cell corresponding to the `(x,y)` position is in the middle of **two even cells** located at `(x,y+1)` and `(x,y-1)`. The RSSIs vectors can be computed as an average between the vectors of these two positions.

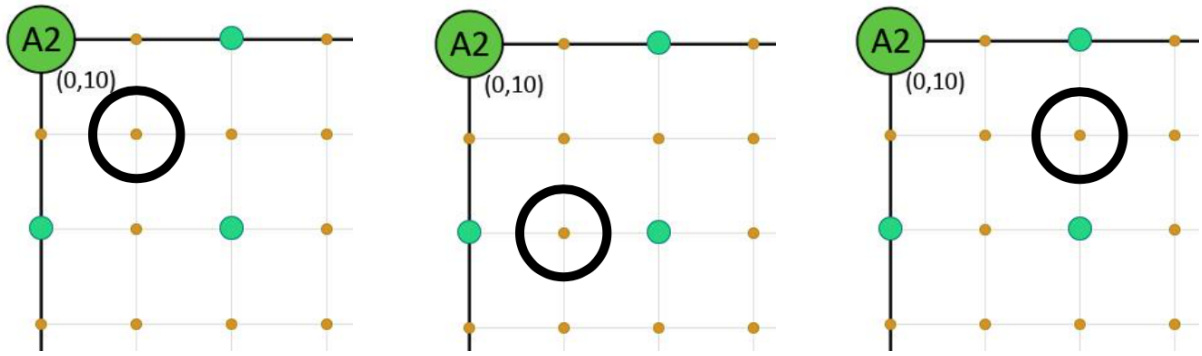


Figure 1: From left to right: case 1, case 2, case 3

After reconstructing the missing fragments associated to odd positions, a `txt` file containing all the entries for each position has been produced and made available within the `.zip` folder.

Final datasets

After cleaning and filtering the fragments, **two files** have been produced:

- `input.txt` (created manually after cleaning fragments), containing all fragments obtained via CoAP and MQTT and given as input to the Python script that computes odd positions and Dory's estimate position
- `output.txt` (obtained with Python), containing the whole dataset (even and odd positions), which is the matrix used by the Python script in order to compute Dory's position

Each row of these files is formatted as follows:

$$x_{pos}, y_{pos} \,||\, anchor_1 \,||\, anchor_2 \,||\, anchor_3 \,||\, anchor_4 \,||\, anchor_5 \,||\, anchor_6$$

where $anchor_i$ is an array containing **5 different RSSI measurements** at the same position for the same anchor i.

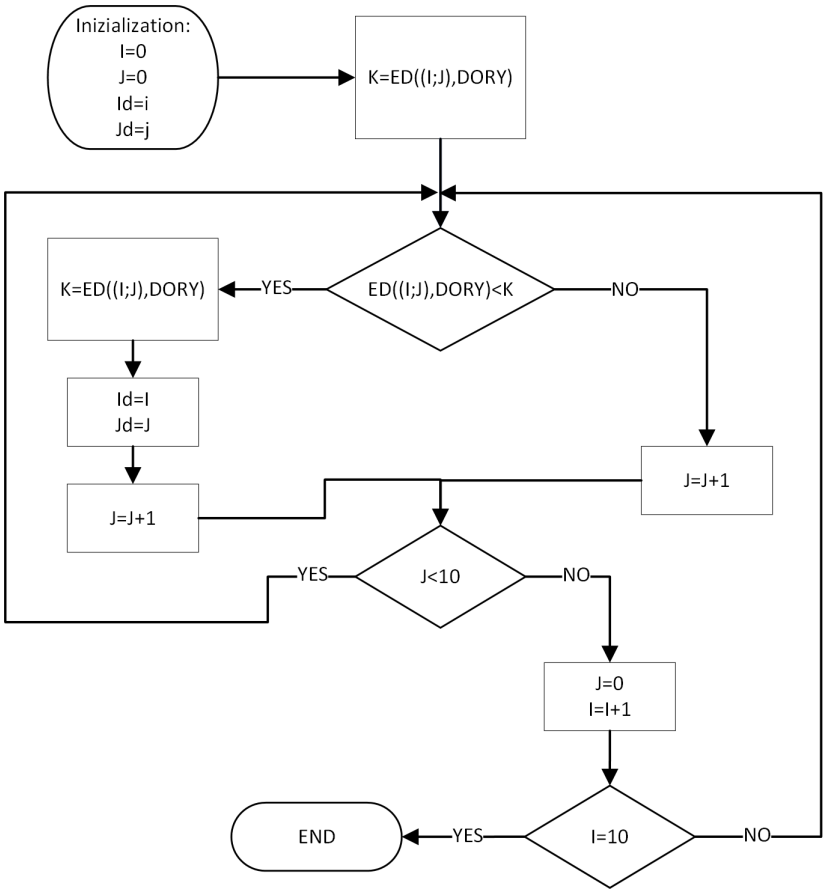
Matching approach

In order to match Dory's RSSI with the found ones and also to localize her we have used the following approach:

1. It was computed the Euclidean distance between Dory's RSSIs vector and the RSSIs vector of the first measurement in position (0,0)
2. The starting position coordinates and the result of the first point were stored respectively in I_d, J_d and K
3. The Euclidean distance between the RSSIs vector at that position and Dory's vector is calculated again for each position . If a k value (Euclidean Distance) for some position (i, j) is found to be lower than the stored one (the lowest K found so far), then the K got updated with the new Euclidean distance and so I_d and J_d with i and j. Otherwise, these values remain unchanged until a better k value is found.
4. This procedure is repeated for every position as described by the **flowchart**.

Matching approach flowchart

I: current row
J: current column
I_d: Dory's row
J_d: Dory's column
K: storage variable
$ED((I;J), DORY)$: Euclidean distance between RSSI vector in position $(I;J)$ and Dory's RSSI



Practical aspects of the matching algorithm

Now that we have given the reader an introduction about the used matching algorithm, we would like to clarify its practical aspects in order to make the reading of the code as much understandable as possible, in particular we will focus on the assumption made on the data we got and on how we dealt with **multiple RSSI measurements** for single cell while Dory only having one.

First of all, we assumed that **all the data we got were correct**, in fact those which looked absurd have been spotted and removed before creating the input file.

Secondly, in order to overcome the fact of having multiple measurements for each cell and only one for Dory we proceeded like this for each comparison:

1. It was computed the Euclidean distance between Dory's RSSIs vector and each one of the different measurements at the same position (i, j)
2. The Euclidean distance $k_{i,j}$ that has to be compared with the stored one (K, the lowest Euclidean distance found so far) is the **lowest** one between the 5 measurements of that position

Dory's location

Given the assumptions and explanations of the paragraphs above, the reader should now be ready to run the code.
As you will be able to see **Dory turned out to be in position (2;3).**