IINERNET OF THINGS: final project proposal

Andrea Guzzon

1 Main idea

The main idea of the project is to develop a lattice of sensors distributed around a city in order to help people at risk (e.g. elderly ones) to avoid potential security threats.
The city should be imagined like a matrix $N \times M$ where each cell has a set of sensors (both environmental and for the stores) that communicate with the sentral server

2 Entities:

To reach the aim these are the entities involved:

2.1 Air Sensors

These are $S_i i \in \{1, ..., N_{sens}\}$ sensors responsible of measuring the air quality in the city. *Each sensor is a node* and each node has four kind of sensors: **temperature**, **humidity**, **PM**$_{2.5}$ and **PM**$_{10}$. For the **PM**$_{2.5}$ ones a threshold of 12 is fixed for a warning notification and 35 for a danger notification. For the **PM**$_1 0$ 40 is the threshold for warning and 80 the one for danger. For the **temperature** ones a threshold of 30 degree is fixed it the **humidity** is too high
Whenever is detected a value above one of the threshold the sensor sends a CoAP message to the server, advising it about the trigger.
Sensors can also be queried by the server: a scenario include also the possibility for the application to ask the current status of the air.

2.2 Stores sensors

These are distributed at the entrances of the above mentioned stores. They keep track of how many people there are in the store (i.e. *every time someone enter* $count + 1$ *and every time someone exits* $count - 1$)
They also have the information about how big a store is and what kind of store they are located at, so that they can calculate *how a store is crowded proportionally* (i.e. $\frac{people}{m^2}$) and **data can be classified by type of store**
When requested by the server they send the registered value and the coordinates in the matrix
The stores are categorized by type (*pharmacy, grocery store...*) so that when users need to go to a specific kind of store the server provides a list of all the available ones (i.e. not too crowded) for that type, classifying them depending on **distance, how crowded they are, and traffic**

2.3 Traffic sensors

Imagining the city like a matrix as represented below, where each cell has its traffic sensor, a score from 0 to 10 depending on data is assigned to each cell (*the more that zone is trafficked, the highest is the score*)
Doing so it is possible, when a path to a store is needed, to take into account the best route also considering traffic.
As an example, imagine a user that needs to go from A to one of the B-store

A	9	9	9	7	8
1	7	8	7	6	4
1	8	7	B_1	4	2
1	6	5	4	5	4
1	7	6	7	B_2	8
2	B_3	5	6	5	3

So considering not going in diagonal we can see that the less trafficked path is to B_3 (6), while B_1(17), and B_2(24) are more trafficked. This information is used by the server to give back the list of the stores.

2.4 Central server

It is responsible of the handling of users requests and notification.
It has *a database* where pseudo-anonymous IDs are stored: these are the IDs of people that are using the application and are used whenever it is necessary to notify them about possible dangers. In an hypothetical scenario some sensors measure a risky level of PM, sends values and alert to the server which `notify_all` the users about it

It also stores information about where the stores are located and what type they are; this can helps people when they need to avoid crowded markets:

1. the users through the application tell the server in which kind of store they need to go (for the example, `type`)
2. The server asks to the closest `type` store sensor about how crowded it is. If it is below a risk factor R_f the server sends an `ok` to the application
3. Otherwise, if the store is too crowded the server sends a request to all the `type` stores within a fixed radius in KM.
4. It assigns to each store a score, based on the previous mentioned R_f weighted with the **distance** D_i of the store and the **traffic**
5. It then sends back to the application the list of all the near `type` stores, ordered from the best

2.5 Application

It is the way people have to interact with the sensors; it can be used both to receive notifications about security alerts or *to send a request to the air sensors and know actual air quality*
It also can be used to avoid crowded stores, requesting to the server the best solution based on store type, distance and traffic
The application saves **the location** of the users, in order to calculate the best path to the store based on distance and traffic.
Users could also use the application to **book a place** in a selected store, so that the application could notify the store and it takes into account that.
If users need to go to different kind of stores they could provide a shop list to the application, which will calculate **the best path to all the stores**, taking into account crowd and traffic situation.