Comunicação em rede AD-Hoc para auxílio em situação de emergência

Redes e Sistemas Autónomos
Universidade de Aveiro
DETI

Mauro Filho, (103411)
Adalberto Júnior, (105589)

Conteúdo

1 Objectives of the work

The proposed project aims to develop a robust solution for communication in emergency scenarios, where traditional network infrastructure may be compromised. To achieve this, the proposal is to implement an ad hoc network that, with the help of an application, allows the exchange of messages between users, even in the absence of cellular coverage. In addition to exchanging messages, the system will maintain a distributed cache of user information. This means that even if a user loses connection, their information will still be available to other users. This functionality can be crucial in rescue and identification operations, providing valuable information about individuals in emergencies. Sending frequent messages between users (nodes) allows network members to keep their information current. Nodes analyze the security level of the area around you with the help of sensors and communicate this information to others so that everyone can map the safe places and move there.

2 Architecture

Our architecture consists of 5 OBUs representing people or mobile objects in an emergency situation, which we almost always call nodes. We use NAP-Vanetza to exchange CAM messages and MQTT to exchange customized messages, as Vanetza accepts a specific type of message and discards those that do not follow the standards. The entire simulation takes place on the host PC. We simulate GPS, to obtain the location of the nodes as well as the mapping of the space (map) of the simulation, and sensors to obtain environmental data, we do not specify the sensors to be used, but rather the smoke sensors, Temperature sensor and camera are examples of useful sensors for a more realistic project implementation.

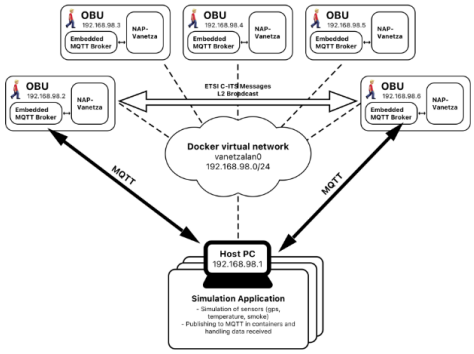


Figura 1: Software Architecture

3 Implementation

3.1 Data Generation for simulation

Since we are not working with real sensors that are reading values from the real world, the need for a simulated world arose and our solution for that was to create a script that would generate sensors readings for each coordinate of the simulated world, this way the OBUs would be capable of knowing the sensor value according to their position in the map by reading the file generated by the script.

The script "GenerateSensors.py"is designed to generate simulated heat map data by creating sensor readings based on proximity to randomly placed heat points within a grid. The process begins by accepting user input for the grid size, a base value, a variation range, and the number of heat points to be used in the simulation.

Once these parameters are set, the script randomly places the specified number of heat points within the grid. These heat points serve as the focal areas from which the sensor readings are derived. For each coordinate on the grid, the script calculates the distance to the nearest heat point. This distance is used to determine a sensor reading value for that coordinate. The value is calculated by subtracting the product of the minimum distance and the variation range from the base value, ensuring that the result does not fall below zero. This simulates how heat intensity might decrease with distance from the source.

Before saving the data, the script checks if the output file already exists and deletes it if necessary to ensure that the data stored is fresh and not appended to any old data. The sensor readings, along with their corresponding coordinates, are then written to a text file in a structured format.

3.2 Simulation Visualization

The script "generateHeatMap.py"is responsible for the visualization of the sensor data as a heat map and animate the movement of points (OBUs) based on real-time location updates received via MQTT. It combines data processing, plotting, and live updates to create the visualization.

Initially, the script takes user input for a sensor data file and an IP address, which are used to determine the source of the sensor data and the target for receiving location updates, respectively. In other words, the user must choose an OBU to "observe"through the visualization app and this will be the one sending the needed updates to it.

The script begins by reading sensor data from the sensor file mentioned in the previous section. The data representing the sensor readings in each GPS location is loaded into a matrix. This data is then used to plot a heat map, with a background map image to provide context.

Once the heat map is plotted, the script sets up an MQTT client to listen for location updates. It connects to the MQTT broker at the OBU that it is observing and subscribes to the topic "simulation/locationUpdate"for receiving

these updates. When a message is received, the script updates the coordinates of the specified object on the animated map.

To provide a dynamic visualization, the script uses Matplotlib's animation capabilities. It defines an animation function that updates the position of points on the plot based on the received location data. The self (observing OBU) coordinates and other object (remaining OBUs) coordinates are displayed. In addition to that, two circles are also drawn, the outer one representing the communication range of the nodes, and the inner one representing the sensor range.

The animation runs in real-time, continuously updating the plot with new data. The MQTT client operates in a separate thread to ensure that location updates are received and processed without interrupting the main plotting process.

3.3 OBU behaviour

Now lets focus on the script "obu.py", which controls the behaviour of the OBUs.

3.3.1 Message Exchanging

As mentioned before, the project mainly uses MQTT for sharing messages between nodes, and therefore the nodes must be the ones responsible for sending and receiving/ processing most of them. The messages exchanged are:

- vanetza/out/cam: CAM message used for sharing GPS location of nodes
- sensors/evaluation: message for sharing the safety evaluation of the area in which the node finds itself at a given moment. This safety evaluation is done by reading all the sensor values withing the node's sensor radius and calculating an average. These evaluations are stored in a dictionary in each of the nodes and are then used to decide safe locations where they should move to, or not.
- report/action: message for sharing the current state of nodes. There are 4 possible states a node could be in which affects their behaviour during the simulation. Those will be explained later in this report.

All messages exchanged are periodic, CAMs and sensor evaluations are sent every second and state updates are sent every 2 seconds.

Apart from the messages mentioned in this section, there is also the simulation location update message, which is only consumed by the simulation app as previously mentioned. This one is sent whenever a new location is known to the OBU, be it its own location or of another node.

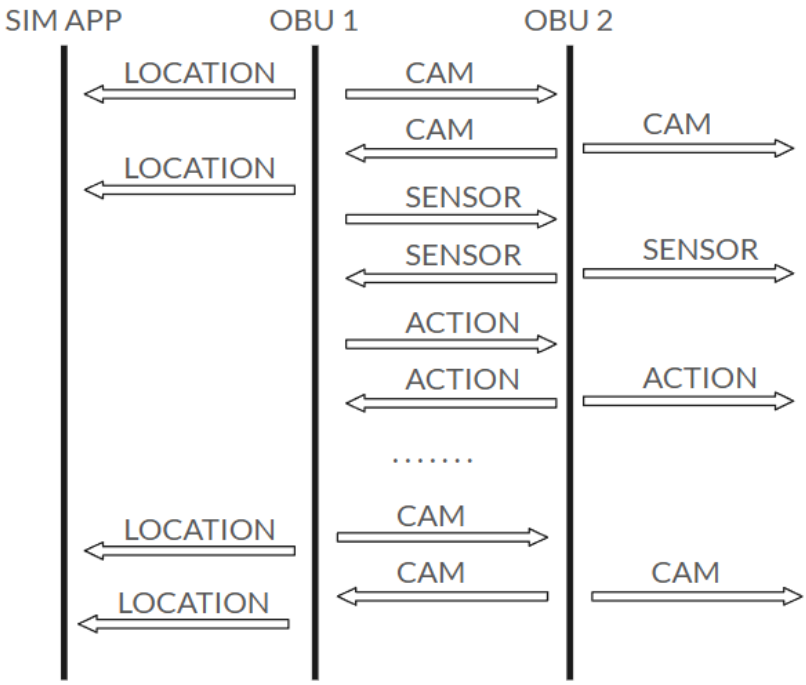


Figura 2: Message sequence diagram

Since we are running all the OBU docker containers on the same machine, all messages forwarded by Vanetza will always reach all nodes, therefore the need arose for us to implement a way to simulate "out-of-range"situations in our project. To do that, we created a function that analyzes recursively at each node, what other nodes are reachable. We do that by calculating the distance between nodes and checking which ones are within range, this way we can know what nodes are directly (or indirectly) connected and create a list of nodes from which the current node will accept messages from. If a message arrives from a node outside of this list, it will be skipped and will not be processed.

Still on the topic of simulating communication range, we also track the last update time for each node, this way its possible for us to know when a node that was previously connected goes missing. Nodes are considered to be missing after 10 seconds of not sending any updates.

3.3.2 States of a Node

In order to facilitate the understanding of the logic, we will split the explanation by the possible states of the OBU:

- STATIONED: This is the state in which the nodes transition to when they have found a safe spot and have reached it too. When in this state, the node will still send the usual messages, but now it won't move anymore, therefore its updates will be always the same. The only way a node that is stationed will move again is if a new safer spot is found, in which case it will try to reach that point instead.

- MOVING TOWARDS SAFETY: When in this state, the node has found a safe spot but it is not yet there, it is moving towards the location. In order to reach the desired location it uses a search algorithm (A*) to define the best path using the known locations of the map while using the immediate sensor readings to refine the movement when passing through areas that are unknown. By moving like this we can ensure that the node is moving towards the correct direction while also avoiding dangerous points by using its sensor. The movement algorithms shall be detailed later in this report.

- SEARCHING FOR SAFETY: When in this state the node has not yet found a safe spot and is doing a solo exploration of the map to try and find it. Usually when all nodes are in dangerous zones, none of them will immediately know of a safe spot, and can take a while until one of the nodes in the network finds one and advertises it on the network, thats how you may find yourself with all 5 nodes in this state. Another possibility is if you have a node that is not in the range of communication with the remaining ones, since it will only know of its own visited locations and will not receive better options through the network, the only option is then to explore the map alone.

- MOVING OUT OF DANGER: So far we have been talking about following "safe"spots found collaboratively by the nodes, and if no safe spot is found the nodes will keep exploring the map in order to find one. However, for nodes that are in urgent need of rescue, even if no "safe"spot is found, they can still follow a target that is in a less dangerous situation for an immediate relief in its safety. What this means in practical terms is that nodes that are currently in dangerous or very dangerous areas will take known positions and follow the safer one as if it was safe, even though it may not be. Once the node is out of the immediate danger, it may restart its solo exploration for a better spot.

3.4 Movement Algorithms

3.4.1 When following an objective

To reach the desired objective, we use a search algorithm (A*) to define the best path using known map locations and immediate sensor readings to refine movement when passing through unknown areas. To do this, we send as an argument the current position, objective, and the map, where the map is composed of all coordinates and the sensor reading for each coordinate, but only

the readings in the node's sensor range and in other coordinates in Since the node is unaware of the sensor values, a very high and invalid danger value has been added. To find the best path, the real cost function (g(n)) was combined with a heuristic (h(n)) to guide the search. In which the heuristic is given by the distance between the coordinate in question and the objective, and the real cost is given by the values (state of the environment) read in the sensor range. Remember that in coordinates outside the sensor range, a very high danger value (404) is given. In the end, if there is a path to reach the objective, the next coordinate that the node should move to (within the limit of the map and the sensor range) is returned and if there is not, or if it is already at the objective, it is returned to current coordinate. And the node changes state to stationed state.

3.4.2 When exploring the map alone

To explore the map alone, nodes without a concrete objective look for the next jump in which the values obtained by reading their sensors are less dangerous, they always go from coordinate to coordinate until they find an area considered safe, and when they find it, they end the exploration. To prevent the node from getting stuck between two coordinates, we create a memory of the coordinates with the corresponding readings from sensors already visited, which is also used to allow the node to return to the least dangerous path already visited, if it reaches a place of extreme danger. In this situation, A* is used to find the best (safest) known path to the objective (the least dangerous place among those that have already been visited), after reaching the objective the map is explored in another direction. Remembering that the situation on the map may vary over time, the node upon returning may find a safe situation or receive information from other nodes about safer locations.

4 Flow of the demo

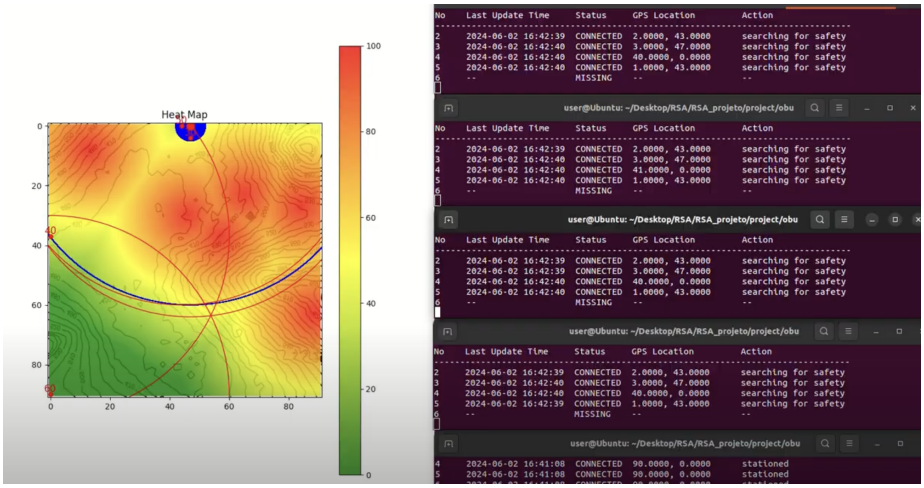


Figura 3: Demo preview

In the "full demo"video made available in the links attached to this report, we can observe:

1. Nodes 20, 30, and 50 go online. Node 30 is in a medium safety spot, so its doing a solo exploration of the area to try and find safer spots. Nodes 20 and 50 are in a high danger area and therefore will try to reach node 30 even though it is not in a safe area.

2. Node 40 goes online. Node 40 does a solo exploration to find a safe spot. In this case, it will not try to reach node 30, nor will all the other nodes try to reach node 40 because their safety situation is similar and not worth the risk of moving away from their current situation.

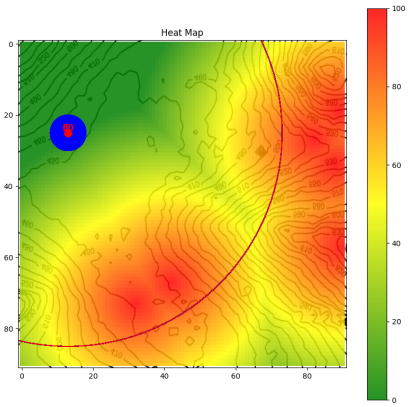
3. Node 40 finds a safe spot. Nodes 20, 30 and 50 will now follow this objective and move towards that direction.

4. Node 60 goes online. Node 60 is in a safe spot and advertises it to all other nodes. Since this position is safer than all the previously known ones, all nodes will move towards it.

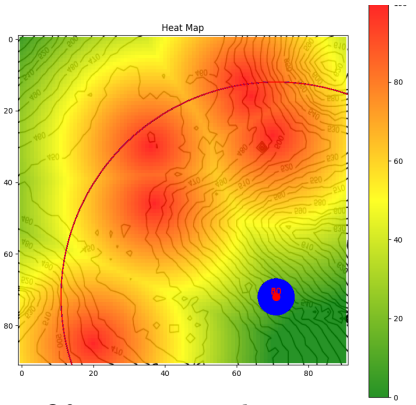
5 Results

In order to test the whole system we generated two maps using our generation tool and ran the simulation 5 times for each map, with nodes randomly positioned. The maps generated were 90x90 in size, had 5 randomly placed heat points and had a sensor variation range of 2 points. Moreover, nodes have a

communication radius of 60 points and a sensor radius of 5 points. These are the results observed:



Attempt	Time taken to reach safety stability
1	1:25.33 min
2	0:55.97 min
3	0:58.26 min
4	1:24.11 min
5	1:24.11 min
Average	1:13.96 min



Attempt	Time taken to reach safety stability
1	1:51.60 min
2	1:41.64 min
3	1:52.23 min
4	1:23.43 min
5	1:40.87 min
Average	1:41.55 min

Of course many things can impact the results obtained from the simulation, most of them in the environment setup, like number of heat points, size of the map and the sensor variation range. However, considering the maps generated for the tests, one of the most important aspects for the simulation, from what we could observe, is the placement of nodes. If all nodes are in dangerous positions, it can take significantly longer for them to find a safe spot and start moving there, which will delay reaching stability between all nodes. Moreover, sometimes a node can be stuck in between dangerous zones and may not be able to leave, which then turns the objective of getting all nodes out "alive"impossible.

6 Links

Link to code repository: `https://github.com/mauromarques/RSA_projeto`
Link to demo videos: `https://shorturl.at/MU82s`