universidade de aveiro deti departamento de eletrónica, telecomunicações e informática

FireMesh

Vehicular ad-hoc networks in extreme situations for crisis response units' communications

Project developed within the scope of the *Networks and Autonomous Systems* course at the *University of Aveiro*.

Authors

88801	Diogo Vicente Mendes
93444	David José Araújo Ferreira

Orientation

Professor Susana Sargento

Professor Pedro Rito

June 2023

Table of Contents

Motivation

Wildfires pose serious difficulties in terms of coordination and communication. Usually spanning large, remote, geographical areas, involving up to hundreds or even thousands of operatives divided by multiple sectors and performing multitudes of different missions, coordinating operations to attain a successful outcome in the shortest possible time can prove difficult.

These difficulties can present themselves in various manners, being the most obvious the lack of situational awareness, meaning the reduced information relative to the status of the operations and environment in all of the affected areas. Also, the lack of communication infrastructures in many of these remote locations severely limits communication to faulty or outdated instruments like SIRESP and VHF/UHF radios.

With these difficulties in mind, the goal was to design a system that does not rely on existing infrastructure but is self-sufficient and self-organizing. An *ad-hoc* network between the nodes is the ideal solution if firefighting vehicles were to organize themselves at the time of the event to keep, not only communications but to gather and construct communal knowledge, useful to observe the evolution of the fire and the status of each team on the field.

Environment Architecture and Features

Vanetza

VANETZA stands for Vehicular Ad hoc Network for Zero Accidents. It is a software framework designed to enable communication and coordination among vehicles in a vehicular ad hoc network (VANET) environment. In general, VANETs are wireless networks where vehicles act as mobile nodes, allowing them to exchange information about their current position, speed, direction, and other relevant data. In a VANET environment, vehicles can communicate with each other directly or through roadside infrastructure to enhance road safety, traffic efficiency, and other transportation-related applications. These networks rely on ad hoc communication, as the network topology changes rapidly due to the dynamic movement of vehicles.

VANETZA aims to enhance road safety and prevent accidents by facilitating the exchange of information between vehicles and the surrounding infrastructure. It utilizes the concept of Cooperative Intelligent Transport Systems (C-ITS) to enable vehicles to share relevant data in real time.

For our use case the data sharing is the most useful feature of Vanetza, more specifically we use the Cooperative Awareness Messages (CAMs) and Decentralized Environmental Notification Messages (DENMs). Furthermore, we only make use of On-Board Units (OBUs), not using Road Side Units (RSUs) since we aim to deploy this system to help combat wildfires in forests or remote areas, so we assume that there is no infrastructure in place.

Regarding the messages, CAMs allow us to continually update the coordinates, speed, and heading of the OBUs (vehicles), allowing us to create a real-time map of the units, which could be useful to make strategic decisions. The DENMs are used to spread an alert message in case a vehicle is suspected of being lost - either due to getting out of range of all other vehicles or due to an accident.

In future work, DENMs could be used to create a Rescue and Search party for lost vehicles or simple commands such as retreat. More types of messages could also be integrated, for example, to give commands such as a move to this place, making the system more complete and capable of handling more situations.

IPFS with SensorMesh

IPFS (InterPlanetary File System) is a decentralized peer-to-peer file-sharing protocol and network designed to create a distributed and permanent web. It aims to revolutionize the way we store, shares, and access files on the internet by providing a more resilient, efficient, and censorship-resistant system.

Peer-to-peer communication in the context of IPFS refers to the decentralized and direct interaction between nodes within the network. Unlike traditional client-server models, where data is retrieved from centralized servers, IPFS enables nodes to communicate directly with each other, forming a distributed network. This peer-to-peer architecture allows nodes to share and retrieve files without relying on intermediaries. Each node acts as both a client and a server, contributing resources and participating in the distribution of files. This approach enhances efficiency, scalability, and fault tolerance, as files are distributed across multiple nodes, the load is balanced, and the network becomes more resilient to failures and censorship. Peer-to-peer communication in IPFS also promotes privacy and security, as files are harder to track or censor due to their distributed nature.

One of the technologies that build on top of IPFS is OrbitDB. OrbitDB is a distributed peer-to-peer database built on IPFS (InterPlanetary File System). It provides a decentralized and secure way to store, query, and synchronize data across multiple nodes in a network. OrbitDB leverages the benefits of IPFS, such as peer-to-peer communication, content addressing, and distributed file storage, to create a distributed database system. One of the key aspects of OrbitDB that make it ideal for this communal use of IPFS private swarms is the capability of OrbitDB to allow for data replication and synchronization among multiple nodes in the network. Changes made to the database on one node can be propagated to other nodes, ensuring data consistency and availability across the network.

SensorMesh itself is an application built to be deployed on top of a running IPFS daemon with pub/sub capabilities. SensorMesh main capabilities rely on the ability to be able to connect to different data sources, like serial connection or MQTT channels, and redirect all of the collected data to an OrbitDB store.

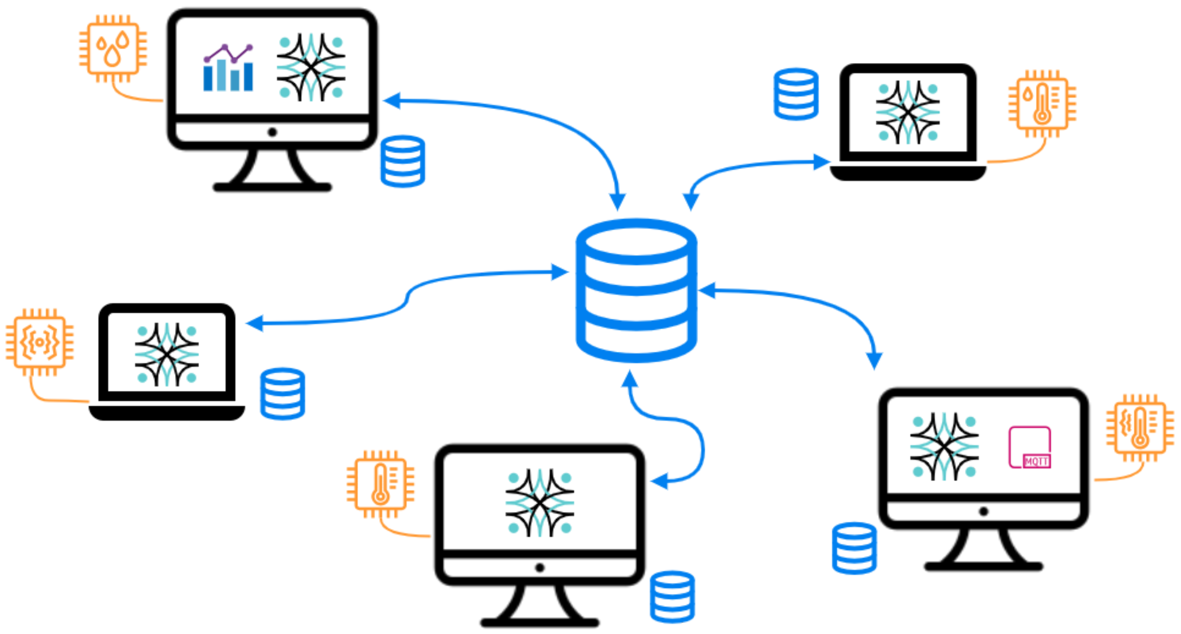

Figure 1 - SensorMesh deployment in a basic environment

The advantage of this is that we can imagine a scenario like the one exemplified in Figure 1, where multiple systems possess different sensors or are running different services, but by using SensorMesh, the knowledge is shared into a communal database, allowing for a generalized overview of the total environment.

Nodes

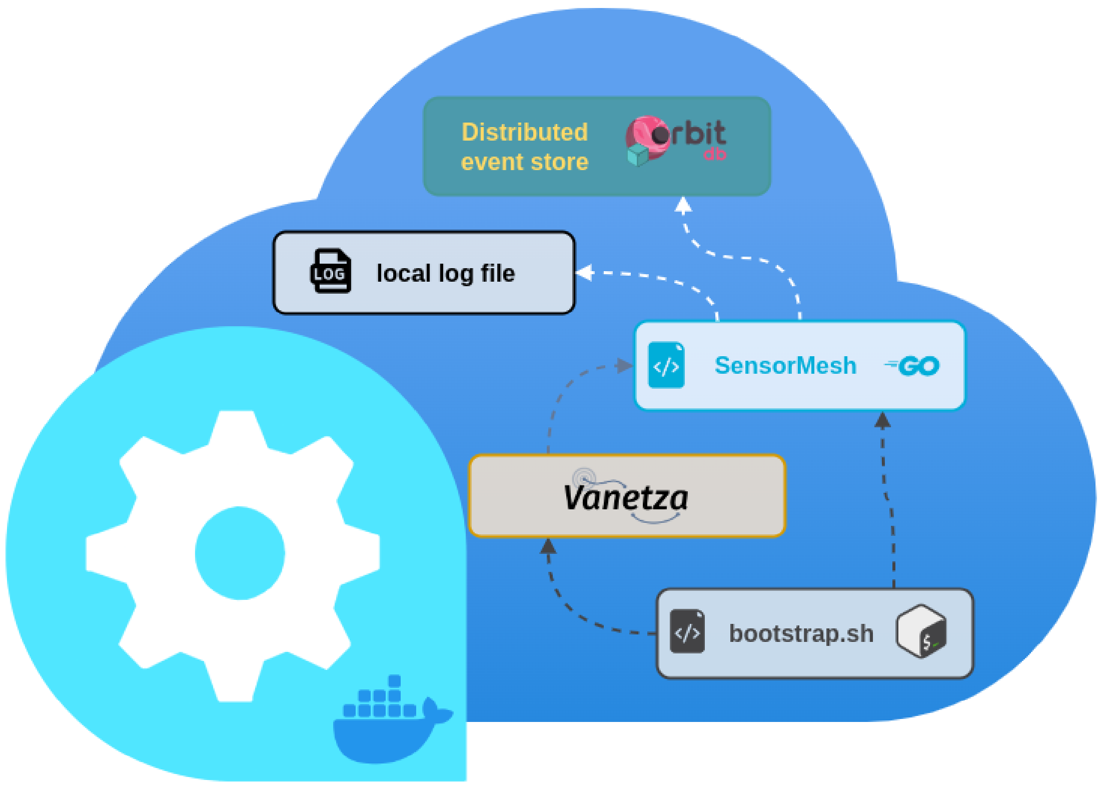

Figure 2 - Node architecture

For this system, we envisioned each vehicle to be an atomic node in the network. Each one of these vehicles would contain a computing unit that would be responsible for running a Vanetza and SensorMesh node.

On start-up, every node will receive an IPFS swarm key in order to configure itself for a private IPFS swarm, and the first node on the scene will be responsible for creating the OrbitDB store address to which all others will also connect. These first configurations of IPFS are done via a *shell* script and the store address is created by SensorMesh, which also creates the *log* file and will be responsible for managing data collection and publishing/subscribing via the IPFS.

Similarly, Vanetza starts on the boot-up. All nodes will send CAMs at 10Hz, following the standard. During the installation of Vanetza each node needs to be configured, such as the IP, type of vehicle, enable CAMs, etc. In a simulated environment this can be done by changing the docker file, on the test boards, since Vanetza was already installed, we simply had to change some of the configuration files accordingly.

Environment

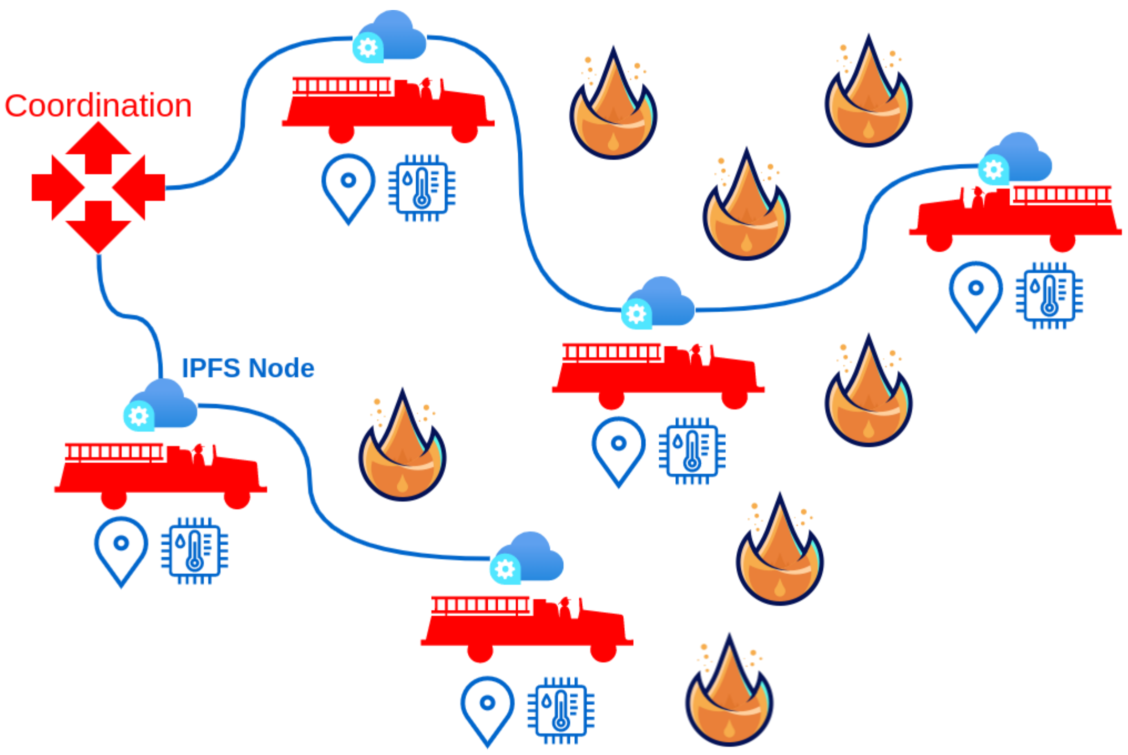

Figure 3 - Environment architecture

Once deployed, Figure 3 roughly illustrates how the nodes would organize themselves in the field. In order to cover large areas in a remote location, each node would only need to be able to reach at least one other node, and since they would all belong to the same IPFS Swarm, every piece of data that one would publish would them be propagated and replicated by every other node.

This type of connectivity would not only facilitate the coverage and introduction of new nodes upon vehicle arrival on scene but it would also allow verifying if any node went silent for an extended period of time, which could indicate that the vehicle had lost communication with the network.

With this, and by using Vanetza, we can double-check that no node is effectively receiving communication (CAMs) from the missing node, thus declaring it as lost and sending a DENM to spread the information of the lost node (ID, last known GPS position).

Implementation

Regarding the IPFS, a simplified approach to our implementation can be observed in Algorithm 1 and Algorithm 2.

Algorithms

Algorithm 1 - Publishing of content by SensorMesh
Input & Output - void

```
while True:
    content = []
    for sensor in sensors:
        if sensor["period"] <= (time.now() - sensor["last_visited"] ):
            content += read(sensor)
            publish(content)

    for channel in mqtt_channels:
        publish(orbitdb_store, channel["latest_message"])
```

Algorithm 2 - Subscription and logging of content by SensorMesh
Input & Output - void

```
logger = new logger ("~/.sensormesh/sensormesh.log")
while True:
    new_message = subscription(orbitdb_store).read()

    if new_message != logger.last_message:
        logger.append(new_message)

    for peer in orbitdb_store.peers():
        if peer.last_contact >= 5 min :
            vanetza.alert(peer)
```

Regarding Vanetza there are two main modules on the code. The frontend logic that creates the map, places the markers with correct colors, in "real-time". And the algorithms and logic that send CAMs and/or DENMs. It is relevant to notice that a node (OBU) always sends CAMs at 10 Hz even if it is also sending a DENM.

```
Algorithm 3 - FrontEnd Logic - defines colors of the markers in the map
Input: client, userdata, msg
Output: void
Initialisation:
if msg.topic == "vanetza/in/cam":           #sending CAMs
    if index == id_denm:
            latest_coordinates[index] = {
                (…),
                "color": "purple"              # Purple =  also sending denm at same time
            }
    elif index == 0:
            latest_coordinates[index] = {
                (…),
                "color": "red"                 # no denm, Base is red colored
            }
    elif color_obus == "blue":
            latest_coordinates[index] = {
                (…),
                "color": "blue"                # default non base is blue
            }
    elif color_obus == "yellow":
            latest_coordinates[index] = {
                (…),
                "color": "yellow"           # receiving denm

            }
        (...)
    else:
            latest_coordinates[index] = {
            (…),
            "color": "blue"  # default
    }

elif msg.topic == "vanetza/alert":       #msg is an alert from ipfs!
            (...)
            "color": "black"            # Black = dead/lost
            }
```

The algorithm below belongs to method publish_coordinates(), and it defines the color for the previous algorithm, as well as the ID for the DENM sender. It then proceeds to loop all ids and send a call to the method to publish a CAM or a CAM and DENM.

Algorithm 4 - Choose color and call method for CAM/DENM - publish_coordinates()
Input: void
Output: void - call adequate methods
Initialisation:

```
1.  if c==100:                           #after 10 seconds restart
2.      lost_id = None
3.      id_denm = None
4.      c=0
5.      color_obus = "blue"
6.  if lost_id != None:                  #define which one sends DENM, default base id
7.      id_denm = 0
8.      c += 1
9.      color_obus = "yellow"
10. if id_denm == lost_id and lost_id != None: #base is the lost, so choose one to send denm
11.     id_denm = 1
12.     c += 1
13.     color_obus = "yellow"
14. for index, client in enumerate(clients):   #for loop to send CAM/DENM
15.     if index != id_denm:
16.         gps_iterate(client, index)           #send CAMs
17.     else:
18.         lostInCombat(client, index)          #id is the one chosen to send DENM
```

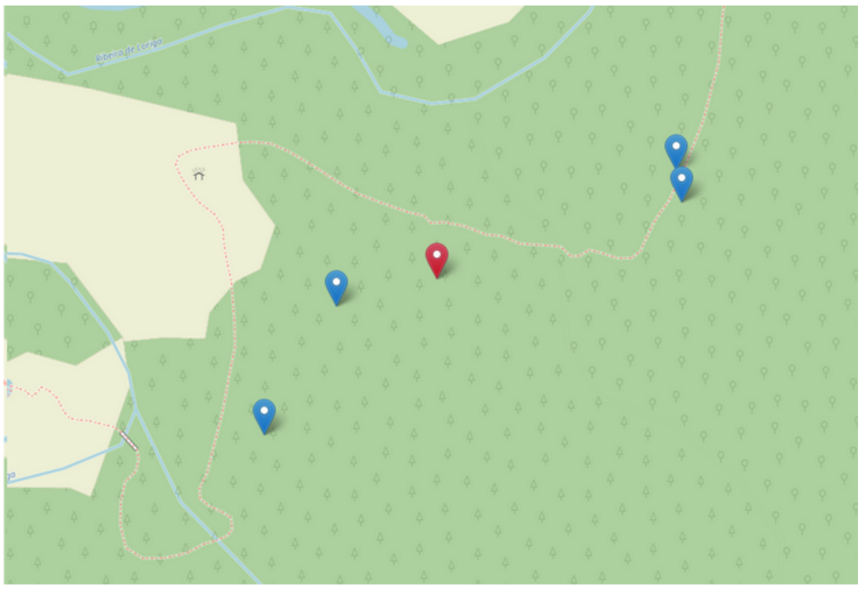

Figure 4 - Node placement in map visualization

Since we want to define some parameters of the CAMs of our OBUs we created a method gps_iterate() that will import a CAM template and then change some fields according to the OBU. Namely, the stationID, the GPS, and the speed. We defined the speed as a constant of 15 km/h ~ 4.16 m/s. To iterate the GPS we have several CSV files with coordinates that will make a path that we drew on a map. The location chosen is near Cabeça, a village from Guarda. Cabeça - Aldeia Natal

Algorithm 5 - Sending CAM
Input: client, index
Output: void - publish CAM

```
Initialisation:
    1.  current_line = line_save[index]
    2.  if current_line >= len(coords):
    3.          current_line = 0
    4.          line_save[index] = 0
    5.          print("Restarted the file for index: " + str(index))
    6.          print(time.ctime())
    7.  ## "Main" part
    8.  line = coords[current_line]
    9.  latitude, longitude = line.strip().split(",")
    10. with open("in_cam.json") as cam_file:
    11.     cam = json.load(cam_file)
    12. cam["stationID"] = index                  # Define the station ID
    13. cam["longitude"] = float(longitude) # Define the GPS coordinates
    14. cam["latitude"] = float(latitude)
    15. cam["speed"] = 4.16  # m/s = 15 km/h Define the speed
    16. client.publish("vanetza/in/cam", json.dumps(cam))
```

Algorithm 6 is the one responsible for sending DENMs. It not only sends CAMs, as in the previous Algorithm 5, but also sends a DENM at the same time. It is important to notice that besides the originatingStationID all other fields contain information about the lost OBU. This information comes from the IPFS part of the project.

Algorithm 6 - Sending DENM
Input: client, index
Output: void - publish DENM & CAM

```
    1.  (...) #sends CAM, equal to Alg. 5
    2.  #-------- DENM ---------
    3.  with open("examples/in_denm.json") as f:   #get template
    4.      denm = json.load(f)
    5.  denm["originatingStationID"] = index      # Define the station ID
    6.  denm["sequenceNumber"] = lost_id          # Define id of the lost OBU
    7.  denm["longitude"] = lost_gps[1]           # Define the GPS coordinates of the lost
    8.  denm["latitude"] = lost_gps[0]
    9.  denm["speed"] = 4.16  # m/s = 15 km/h   # Define the speed of the lost
    10. client.publish("vanetza/in/denm", json.dumps(denm))
```

Additionally, we created five different Python scripts, all of them identified in the beginning their goal. *Main.py* and *simul.py* are both used for simulation, with Docker. The only difference is that *Main.py* can be run without SensorMesh since it has conditions to launch DENMs, while *simul.py* needs SensorMesh to detect a lost OBU and launch a DENM. On the other hand, *test.py* should be used with real hardware. It is designed to launch DENMs with SensorMesh. To use SensorMesh *alerter.py* should be running, this python publishes the alert in the correct topic. Finally, *denm.py* is a test python that can simulate an alert being published by SensorMesh.

Demo

Our demo is available at the following link: https://youtu.be/8ublZULy3P8

Results

As you may see on the demo above, we achieved the intended purpose for this project we designed a system that does not rely on existing infrastructure but is self-sufficient and self-organizing. This was possible thanks to integration between IPFS and Vanetza. IPFS is responsible for saving all data on a database, and detecting if a node is lost, then it alerts the Vanetza, using MQTT. Vanetza is responsible for the communication between all nodes using CAMs, it is also checking the MQTT and if an alert is received from the IPFS it chooses one node to send a DENM to the others. We were able to not only implement these two technologies but also integrate them for the same use case!

Conclusion

In conclusion, our research on IPFS (InterPlanetary File System) and Vanetza (Vehicular Ad-hoc Network for Zero-delay Applications) has also provided us with a solid understanding of these technologies. IPFS is a decentralized and distributed file system that allows for secure and efficient content addressing and sharing. It ensures data integrity and immutability by utilizing cryptographic hashes. On the other hand, Vanetza focuses on enabling low-latency and reliable communication in vehicular networks, supporting various intelligent transportation systems.

The integration of IPFS into Vanetza presents exciting possibilities for enhancing data sharing, network resilience, and privacy in VANETs. By leveraging IPFS's decentralized file storage and retrieval mechanisms, Vanetza can benefit from improved data availability and integrity. This integration has the potential to advance the capabilities of VANETs and foster innovation in intelligent transportation systems.

Appendices

- GitHub Repository - https://github.com/davidjosearaujo/fire-mesh
- Youtube Video Demo - https://youtu.be/8ublZULy3P8