# Easily deployable Ad-hoc Network in case of disasters

**Dasari Surya Sai Venkatesh (IMT2017012), Srishti Adil (MT2019115)**

## Introduction:

Ad hoc networks have been proposed as an appealing communication technology to deal with the unexpected conditions emerging during and/or after the occurrence of a disaster. Communication between victims and crew members involved in rescue operations is crucial in order to alleviate the disastrous consequences and save lives. In such situations of disaster, regular communication systems might fail, and thus, people are unable to communicate with the disaster management team **(DMT)**, even though they have a phone with them. The goal of this project is to design a communication system that will enable these people to communicate with DMT using their phones. Presentation Slides can be found here.

## Goal:

To make a network of Arduino based nodes that can be easily deployable in times of network blackout (disasters). Using this network, people can connect to a temporary network (using their Smartphone), through which they can send their current location and their condition to the DMT (as shown in Figure 1).

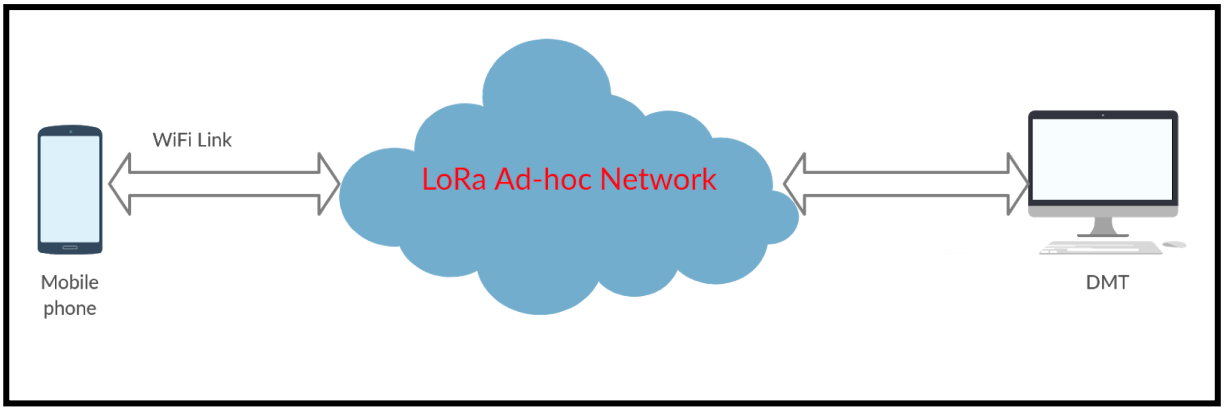


Figure 1: Connectivity of mobile phones to the Disaster Management Team through LoRa Network

## Proposed Model:

Our proposed system consists mainly of two layers;

- Layer1: Communication between nodes
- Layer2: Communication between a node and a phone.

In Layer1, we decided to go with an **ad-hoc** network. Ad-hoc networks have characteristics that are desirable for our system, such as it does not rely on a pre-existing infrastructure and it is a decentralized type of network (Mesh topology).
Thus, we decided to use LoRa modules for communication in layer1. LoRa modules have favorable features like high range, low power consumption, and very low bit rate for transmission. As our system is only for SOS signals (and not designed for heavy loads), thus low bit rate does not bother us.

In Layer2, we decided to go with WiFi modules. WiFi modules are present in almost every phone nowadays, and Arduino based WiFi modules to have great on-board processing and storage capabilities. They also have high data rates and are reliable, which suits our required condition for this project setup.

## Challenges:

- The major challenge faced by us was selecting and implementing a routing algorithm. After a little bit of research, we have decided on the distance vector routing algorithm. The distance vector algorithm is efficient and suitable for transmissions of low load. This algorithm also takes care of node discovery problems.
- Another challenge in the project is: the LoRa module always sends a message as a broadcast, so nodes will not know to whom the message is intended. To overcome this, we have to define message structure (which includes to and from address), and also to keep the messages uniform.
- One more challenge faced is, testing our implementation of the routing algorithm. LoRa modules have a very high communication range (approx 2-3km). So we cannot test various scenarios unless we set up nodes in locations that are far from each other. We also have a solution to this problem which we will see in the latter part of this report.
- If we visualize the connection between nodes as a graph. Our next challenge is a cut vertex failure. This kind of failure will result in loss of communication with a set of other vertices (nodes).
- If the resulting distribution of the nodes is concentrated in one area, it would lead to one more challenge, as it will result in the spatial redundancy of devices in the concentrated area.

## System Model:

The system consists of various components, including

- The LoRa Arduino Access Point (node), which typically establishes the network between various devices (be it of DMT or the mobile phones of victims).
- The DMT (Disaster Management Team) who will have access to the queen node (which is essentially LoRa Arduino Access Point).
- The mobile phones of victims for connection with the LoRa nodes (using WiFi modules).

The below figure (Figure 2) describes the overall picture of the communication structure of our system. The red color line indicates WiFi communication between mobile devices & LoRa nodes whereas the black lines indicate LoRa based communication between nodes.

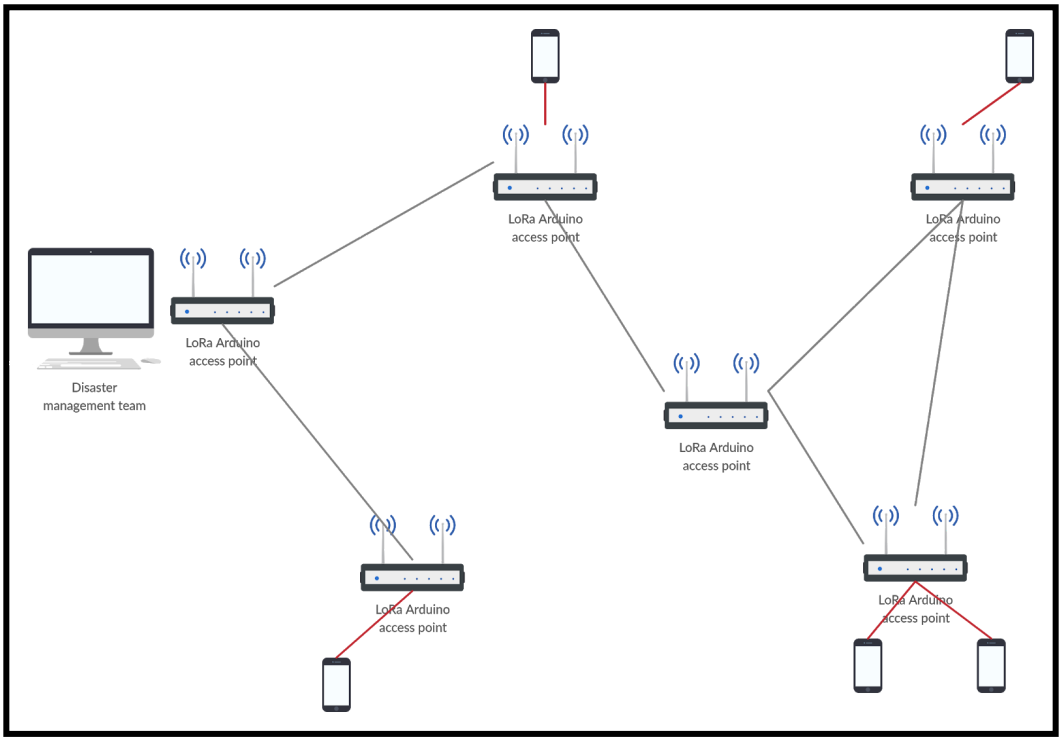


Figure 2: The System Model

## Hardware Setup:

The hardware used in the project are as follows:

- Arduino: Arduinos have low power consumption and this feature is most important for the project. As, when we deploy this system in emergency situations, it is expected to run for a long time. If a node stops working due to battery issues, and if it was the cut vertex, it will hamper a large section of node/system connectivity. Also, people might not be able to communicate because the node in their vicinity has stopped working. Arduinos are small and compact and are very easy to use. On the other hand, raspberry pi also has a high computational capability, and it also supports WiFi and Bluetooth modules (required by the project), but Arduino was chosen because of its low power consumption in comparison with raspberry pi.
- We chose the Dragino LoRa shield for Arduino in this project as it provides an easy interface between Ardunio and LoRa shields.
- Wifi module ‘ESP8266’ is chosen for the project as it is a generic module and also because a lot of resources are available on their usage techniques.
- Batteries: any generic battery pack should suffice.

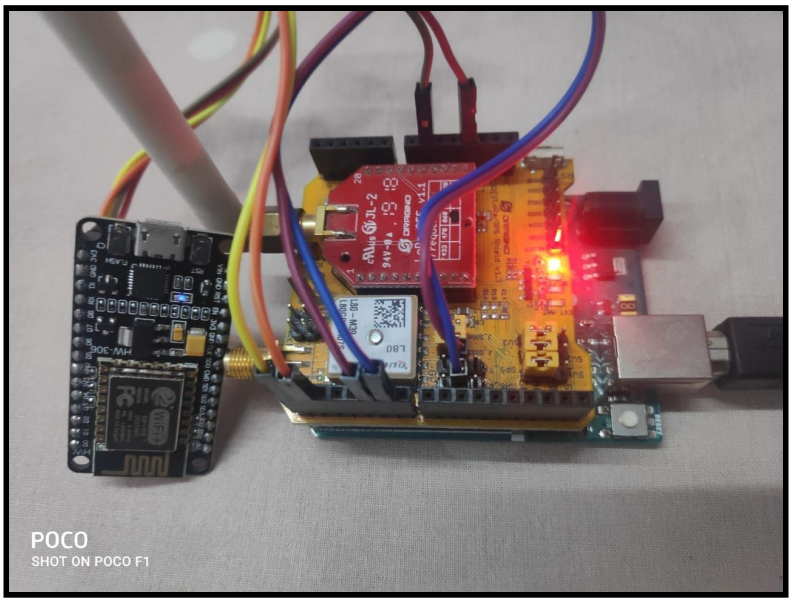


Figure 3: Hardware Setup including LoRa module

## Software Setup:

### a. Arduino:

We have written the code for Arduino in the C programming language. For interfacing between Arduino and LoRa shields, we are using the RH_RF95 library, which can be downloaded from here.

We have defined a few constants in our code. For example:

- node_id: Each LoRa node is identified with a unique ID.
- network_id: LoRa sends messages in a broadcast fashion, so when the nodes communicate with other nodes, and if they have the same network_id, then we can identify them and keep the messages. Also, using this we can ignore the messages sent from another system.
- Message type: We have 4 types of messages (distance vector, data, acknowledgement and intermediate node message) and thus, we initialised 4 integer constants to differentiate between these messages.
    - int const distance_vector_type = 1;
    - int const data_type = 2;
    - int const ack_type = 3;
    - int const intermediate_msg = 4;

We have also defined different message structures for each message type so that we can identify the message type by looking at its structure.

- Distance Vector (broadcast message, mentioned in the code as 254):
  Structure : network_id + type + from + to + vector
- Acknowledgment message:
  Structure : network_id + type + from + to + actuall_to + dummy_vector

- Data message:
  Structure : network_id + type + from + to + actuall_to + Message

We have also created some functions (block of codes which perform a specific task) explained as follows:

- recv_message: It basically consists of a complex combination of if-else statements to determine whether the received message is intended for the node or not. If yes, it copies the message into global variables, from which other functions can access these messages. It also returns the type of the message received.
- send_message: This function takes the message and encapsulates it according to the message type. It then converts the encapsulated message to an array of uint8_t, and then sends the message.
- action_center: It takes the message-type as input. And performs tasks according to the received message. The main tasks are:
  - If it receives a distance-vector: Then the action center will update its own distance vector (it calls the function 'update_distance_vector'). If its distance vector is modified then it will again broadcast the modified distance vector.
  - If it receives the data: then it will print the message content on the serial monitor.
  - If it receives an intermediate message: It will find the next hop from the distance vector table. Then it will send that message to the next hop.
- update_distance_vector: This function takes the received distance vector and updates its vector according to the 'distance vector algorithm'.

## Distance-Vector Algorithm:

We need to connect the nodes (essentially LoRa Arduino Access Points) for network setup, thus an efficient distributed routing protocol is required. While selecting a protocol which will be suitable for our project, we need to focus on the efficiency and feasibility of the algorithm. Therefore, we decided to go with the Distance Vector routing protocol as it fulfills the above-mentioned requirements. Distance Vector Routing is a dynamic routing algorithm and it measures the distance by the number of routers a packet has to pass, one router counts as one hop. It works in the following steps:

Step 1: Each router prepares its routing table by using the information about:

- All the routers present in the network
- Distance to its neighboring routers

Step 2: Each router updates its routing table as follows:

- Each router exchanges its distance vector with its neighboring routers.
- Each router prepares a new routing table using the distance vectors it has obtained from its neighbors.
- This step is repeated for (n-2) times if there are n routers in the network.
- After this, routing tables converge/become stable.

The algorithm keeps on repeating periodically and never stops so that the shortest path can be updated in case any link goes down or topology changes.

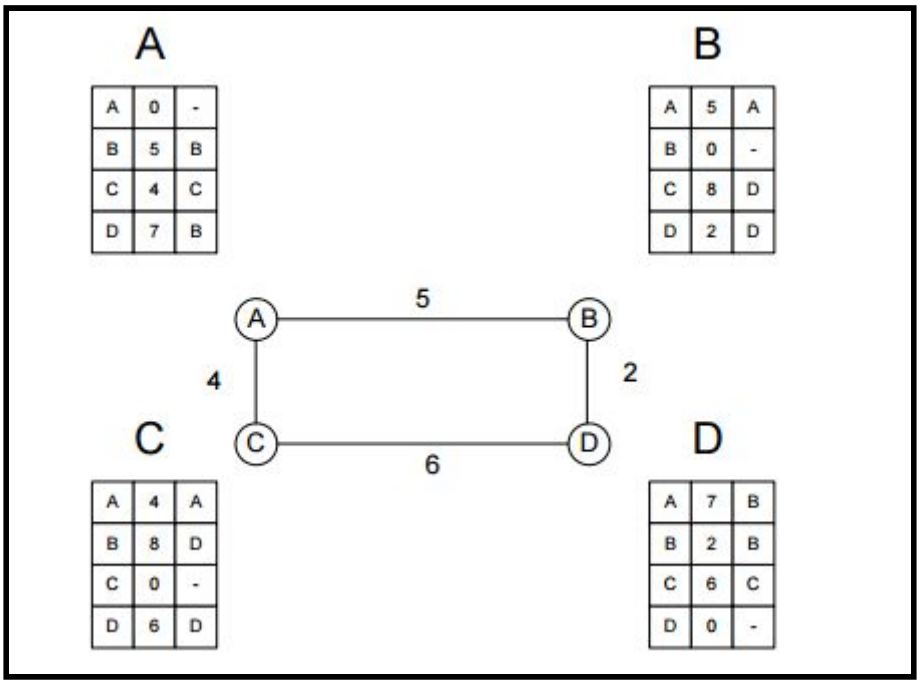


Figure 4: Distance Vector values in different nodes

## b. ESP32 (development board):

In this project, ESP32 is used to act as an access point and host an SOS webpage. This SOS webpage is used to collect names and addresses from the users. The access point is a captive portal, which means that whenever a user connects to an AP (access point), the user's device automatically redirects them to the SOS webpage.

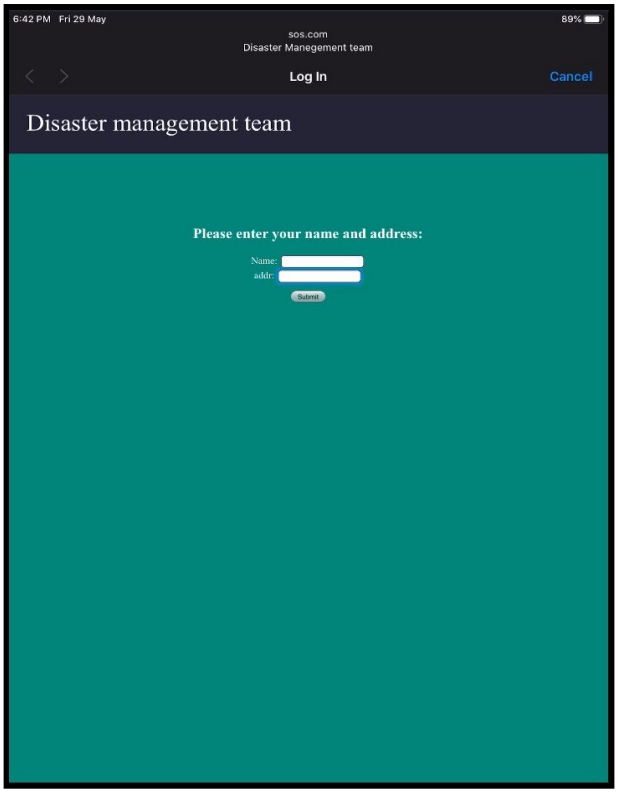


Figure 5: SOS Webpage automatically opened when connected to node's Wifi

## Testing Setup:

The following steps were performed to check/test our implementation of distance vector routing algorithms.

- The ‘straight line mesh configuration’ was selected for testing.
- Then, we initialized the nodes with appropriate distance vectors.
- Next, we created one dummy node which can send messages according to our input. Using this node we did send some specific messages to a node to check its behavior according to the received message. For example,
    - If we send a distance-vector then it has to update it.
    - If we send a message then we have to check whether it is hopping or not.
    - We also checked whether the message is taking the shortest path or not.

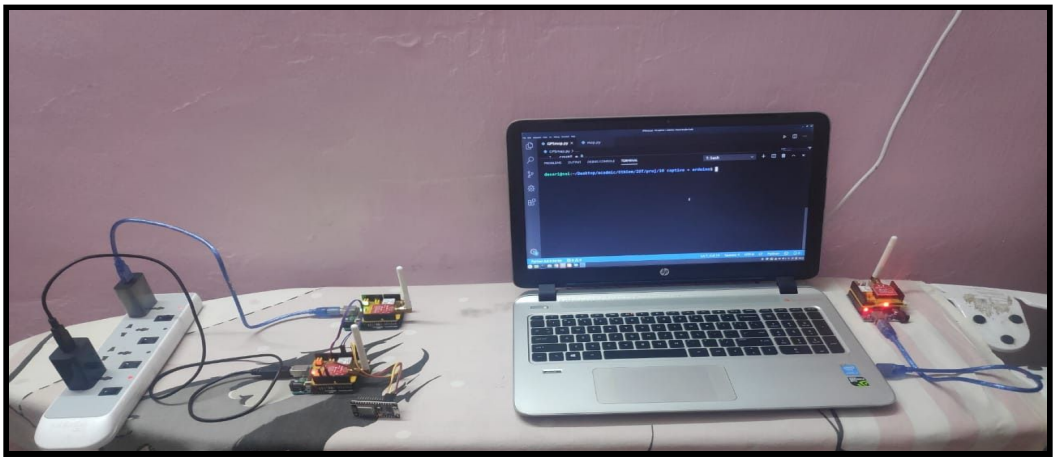

Figure 6: Project Setup with 3 nodes

## Results:

We have successfully performed the following objectives in a respective manner:

1. **Point-point communication of nodes** (LoRa to LoRa communication)
   We performed the hardware and software setup and established the connection between the various components.

2. **Sending a vector to another node**
   We implemented the basic ‘send’ and ‘receive’ using the distance-vector routing algorithm. We tested this feature by uploading the code in 2 Arduinos and then sending a message containing a distance vector to the node after receiving a message from another node.

3. **Defining different types of messages**
   We defined several types of messages required for the project (as discussed in the ‘Software Setup’ section). We incorporated the message types in Arduino devices by defining functions related to each specific message type.

4. **Implementing the routing protocol**
   We implemented the distance vector routing protocol, so that, if a node receives a distance vector from another node, it updates the current distance-vector and then broadcasts it to other nodes.

5. **Testing of the routing algorithm**
   We successfully implemented the message hopping using a dummy node, in the straight-line configuration. It was done by following the steps of Testing Setup (as discussed in the 'Testing Setup' section).
   Figure 5 depicts the one message hop between the nodes and can be explained as follows:
   - The top left section is the output screen for the dummy node which initiates the communication. It sends the data to the intermediate node (top right section of the window).
   - The top right section is the output screen of the intermediate node which receives the message from the dummy node and then broadcasts it to the final node (bottom section of the window).
   - The bottom section of the window is the output screen of the final node which flashes the message received from the intermediate node.

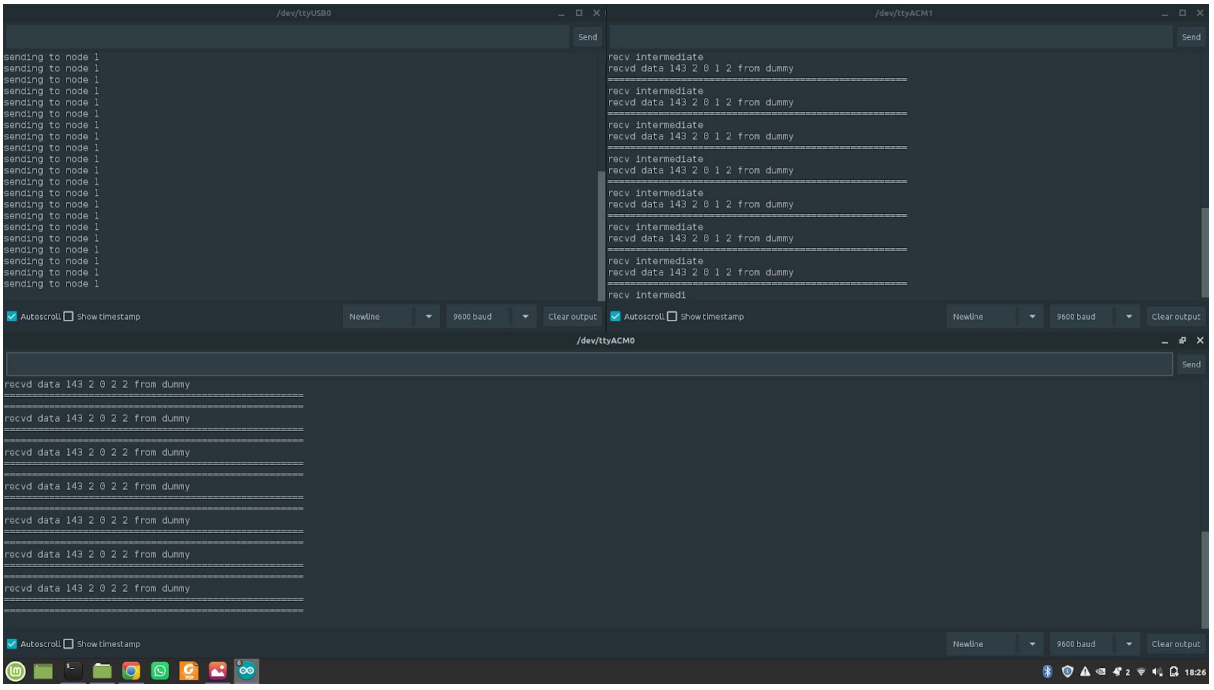


Figure 7: Output screen for message hopping

6. **Accessing GPS Coordinates and Locating them**
   We introduced the Wifi module to the existing system, which will essentially create a local hotspot. The system is designed in a way that, whenever the WiFi access point is accessed and connected by a person through their phone, it automatically redirects them to the SOS webpage (Figure 5). In the SOS page, the name and address of the person is required to be filled. As soon as the person fills the details

and submits the form (by clicking the submit button in SOS webpage), the message along with GPS coordinates is sent to the queen node (the node which is controlled and accessible by Disaster Management Team) through message hopping technique. A python program overviews this whole process, it communicates with the DMS node via serial port and receives the messages, and then extracts GPS info from each message. The DMT thus receives the name and location of the individual along with the extracted coordinates (Figure 8).

```
dasari@sai:~/Desktop/acadmic/6thSem/IOT/proj/10 captive + arduino$ python3 GPSmap.py
['Arduino LoRa TX Test']
['LoRa radio init OK!']
['Set Freq to: 865.00']
['recvd data 143 2 0 2 2 ## ', ' sai venkatesh ', ' phoolbagh, vizinagaram']
position =
latitude =18.126922
longitude =83.422076
```

Figure 8: Coordinates extracted by DMT through SOS page

The coordinates are then fed into another python program which plots the location into an HTML file (Figure 9). Thus an approximate location of the person can be known and traced through the map.

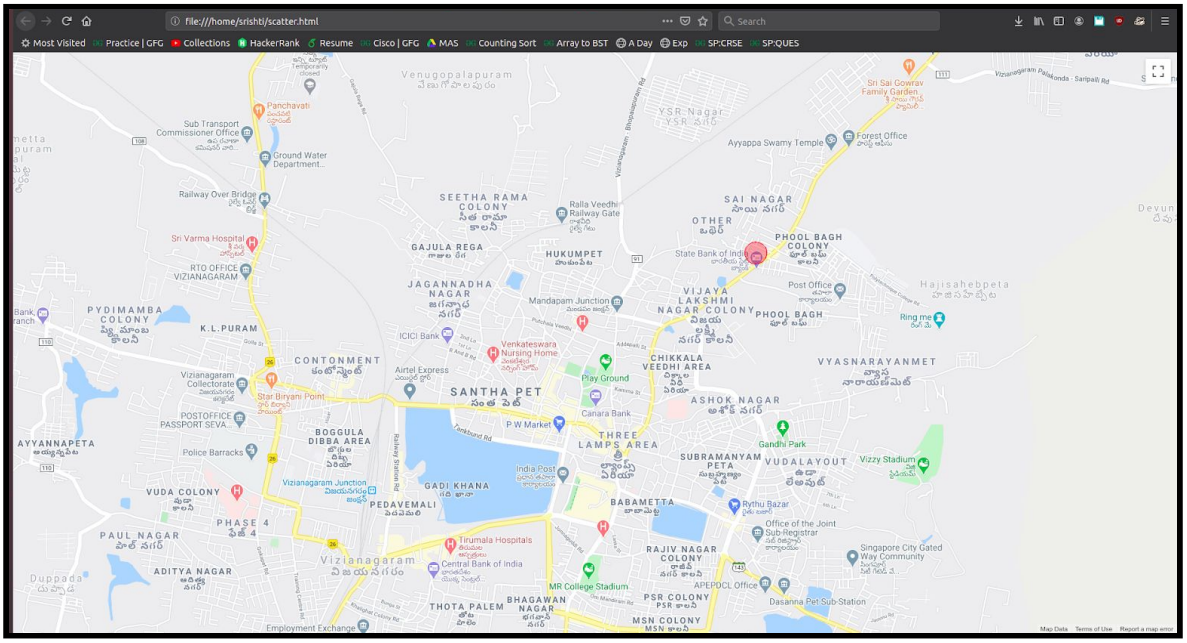


Figure 9: The ‘RED’ mark shows the location plotted in the Google Maps

## Issues:

- The GPS module on the LoRa+GPS shield takes a variable time for calibration (5mins to 25 mins). Due to this, a node takes some extra time before it can start functioning.

## Future Works:

- In the future versions, we can aim to make a better SOS webpage which can take more information from people in need about the essential services which are required (like food, clothes and more).
- In the present version, we take GPS coordinates from the nodes. To improve it, we can take GPS coordinates directly from the phone itself in the future versions.

## Conclusions:

Ad-hoc networks can be a very suitable technology to cope with the hostile and decentralized conditions during and after a natural or man-made disaster. Because of the ubiquity of mobile phones, new possibilities regarding the usage of them in disaster scenarios have arisen. Mobile phones are usually provided with Wi-Fi, Bluetooth, or 3G or 4G radio interfaces and equipped with GPS receivers for self-positioning. All these characteristics are beneficial for enabling device-to-device communications. Thus, the proposed system, when implemented, may help save many lives and reduce the number of casualties in a disaster as it is easy to deploy and does not require much planning for getting the system up and running.

## References:


1. A Survey on Multihop Ad Hoc Networks for Disaster Response Scenarios - DG Reina, M. Askalani, SL Toral, F. Barrero, E. Asimakopoulou, N. Bessis, 2015
2. Load balancing ad hoc on-demand multipath distance vector (LBAOMDV) routing protocol
3. A Survey on Rapidly Deployable Solutions for Post-disaster Networks
4. Arduino LoRa Communication - Transmitter & Receiver Setup for sending Temperature and Humidity Data
5. Handshake Three Way with Arduino · GitHub
6. Distance Vector Routing Algorithm | Example
7. https://www.instructables.com/id/Class-Automation-System-Using-NodeMCU/