Emergency services platform for smart cities

Jay Lohokare[1], Reshul Dani[2*], Sumedh Sontakke[3^], Ameya Apte[4*], Rishabh Sahni[5*]*
College of Engineering, Pune
[*]Department of Computer and IT, [^]Department of Electrical Engineering
[1]*lohokarejs13.comp@coep.ac.in,* [2]*reshulsd13.comp@coep.ac.in,* [3]*sontakkesa15.elec@coep.ac.in,* [4]*apteaa15.comp@coep.ac.in,* [5]*rishabs14.comp@coep.ac.in*

***Abstract*—The Indian 'smart cities' have burgeoning population consuming the intelligent systems running over the city infrastructure and services. One critical set of such services is that of the emergency services involving Ambulances, Police and Firefighters. The services provided by these officials are often time critical, which makes their easy and quick availability a must have attribute. This paper proposes a solution to make such services available to those in need ensuring minimum response time. The proposed solution is based on capturing live location of the emergency services. This will enable the citizen in need to connect to the nearest available officials. The primary contribution of this paper is that it provides a scalable solution that can be extrapolated to entire city where the number of users using this system will be in millions. Every emergency service official will have an internet and Global Positioning System (GPS) enabled device (Smartphone or low cost hardware module) that will keep streaming live location data to servers. User when in need can get contact details of the nearest available official, which will minimize time involved in the official reaching the user. Security of the location data also has been handled in the proposed solution. The existing location tracking systems will not be able to handle the tremendous data traffic coming from entire city population. This paper proposes a back-end architecture that will be future ready to handle such huge number of users.**

Keywords*—*Emergency services, GPS, Internet of Things, Smart city, Big data

I. Introduction

With growing population of cities, the emergency services like Police, firefighters, ambulances face huge stress. These services are very crucial and their availability is backbone of any city. The current systems that allows citizen to access these services are centralized. All officials report to one central control unit where they are allocated tasks. This process of reporting to central control unit causes a lot of delays. Users have to call the helpline numbers to report emergencies. The control unit then contacts officials near the user to handle the emergency. This process of involving an intermediate control unit causes delays.

Instead of contacting a central control unit, citizen will be at benefit if they can directly contact officials near them when in emergency. This will get rid of intermediate communication delays. With improved connectivity and easy availability of smart phones, this feature is very feasible to implement. Also, the use of smartphone applications will help the emergency services capture the context of the users like locations. This will enable the emergency service officials reach to the user quickly, already prepared to meet requirements. The proposed solution aggregates all emergency services on one platform. This will make it convenient to the citizen to access all emergency services from single platform.

Using the platform, a citizen of a city will have access to emergency services at his fingertips. When in emergency, just a click of button will connect the user with appropriate authority. Instead of having to call the Ambulance/firefighters/police, the user will click on the type of emergency, to directly inform the nearest official to be available for service.

Collecting live location of emergency service officials will also make it easy to keep track of activities of every officials in the city, thus making the system more transparent.

II. Existing Systems

The existing solutions involve the paradigm of a central control unit which acts like a central point of contact for users and the emergency systems. Figure 1 shows the overview of existing systems.

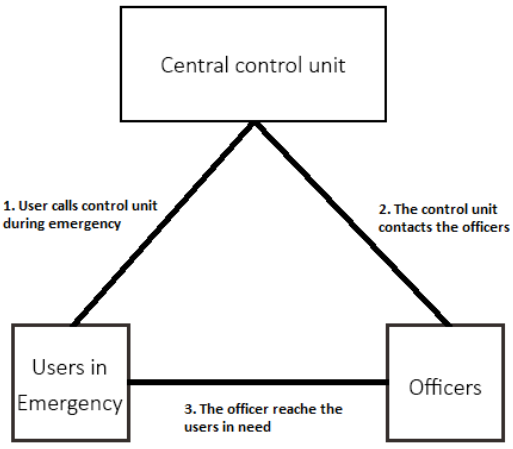

Fig. 1. Overview of the proposed solution

The citizen when in emergency, contact such central unit via through emergency contacts like 100 (Indian Police). These then communicate the emergency with appropriate officers in areas near to the user.

978-1-5090-6255-3/17/$31.00 ©2017 IEEE

The proposed solution removes the need of a central control unit. The system enables users to directly communicate with the nearby officers so as to reduce the communication time.

The existing location tracking systems lack a light-weight backend. A typical existing solution involves data transfer involving Hypertext transfer protocol (HTTP) running over a central server [1] which lacks scalability features. The data transfer from the end IoT modules is not efficient and results to consumption of a lot of internet bandwidth. The method of transferring location data between users involves communication media like SMS, REST APIs, and email for data transfer. These methods are highly unreliable and have a huge latency in data transfer which won't be useful in case of emergency situations.

The emergency request needs to be sent real time and the backend should connect officers with the user in emergency with minimum latency. The proposed solution eliminates points of high latency in the existing systems and thus brings in a factor of reliability.

III. PROPOSED SOLUTION

The emergency service officers using the proposed system will have smartphone application or a location tracking hardware on their vehicles. This application/hardware will keep streaming live location data of that officer to the system servers. The citizen (End users for the system) will have an application installed on their smartphones. When in emergency, they will click a button in the application/widget which will trigger an emergency request. The emergency request will fetch the user's live location and contact details and relay them to all nearby officers related to that emergency. The officers will get contact details and live location of the user, thus immediately letting them help the user. This will eliminate the delays involved in need for the user to call the police helpline when in emergency.

The proposed solution is unique because of its extremely scalable nature and implementation. The system uses big data technologies for data processing and a very light weight messaging system for enabling data communication. Thus, the proposed solution is ready for meeting the needs of big cities in country like India where huge population usually results to system failures at peak system demand times.

IV. OVERVIEW OF THE SCALABLE BACKEND

The 'smart cities' are witness tremendous population in recent years. The platform that connects these citizen to the emergency services needs to be able to handle huge data traffic in order to meet the latency and throughput requirements. Hence, such platform needs to be scalable and light weight. The existing location tracking solutions traditionally use HTTP protocol to communicate location data with servers. This has to be replaced with a better and light weight protocol. Using Message Queue Telemetry Transport (MQTT) [2] protocol will result to a flexible solution that reduces the internet bandwidth consumption at the data collection end. MQTT has proved to require minimum internet bandwidth even in bad network conditions [3,4]. Using MQTT enables use of distributed backend due to possibility of clustering MQTT brokers. Such MQTT clusters ensure fault tolerant and highly available systems.

Figure 2 shows the overview of the proposed solution. The solution has 2 primary part – First one is collecting location data of officials via MQTT and sending it to MongoDB cluster. The second part is where user triggers REST API with his/her location. The user's location is compared to locations in the database and the nearest official's contact details will be given to the user.

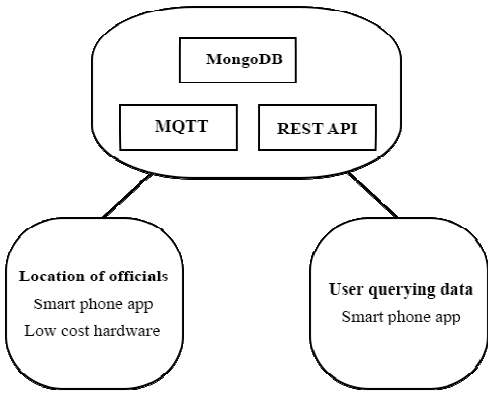

Fig. 2. Overview of the proposed solution

V. COLLECTION OF LOCATION DATA FROM OFFICIALS

Collecting real time location data from every single official in city is a major issue to tackle. This can be resolved by having a smartphone application that servers as location collecting end point.

The officials will login to this application using credentials set by the governing bodies (For example, every police official will get username and passwords that are pre-created by the police department). To ensure privacy for the official using the application, there is a function to set off sharing location when that official is not on duty. This ensures that the location of that official is not made available to others when the officer is not on duty.

Alternative to use of smartphone applications for ambulances and firefighters is use of a low cost hardware solution. This hardware will involve a GPS sensor and Internet module (GPRS). The hardware module will be fixed on the ambulance/fire-truck. Alternatively, the drivers of the ambulance/fire-truck can have smartphone application as in case of police.

A low cost hardware can be constructed using systems like raspberry Pi zero or Arduino Mega/Intel Galileo [5] combined with SIM900A GPRS GSM module. The GPRS module can obtain approximate geo-location using network triangulation, however a GPS module can be used to increase accuracy (Which will increase the system costs). These system will have only one drawback that they will have to be re-coded to re-configure the MQTT credentials.

VI. BACKEND ARCHITECTURE

The location data collected from emergency service officials needs to be collected from the source reliable and efficiently. The system should remain available even when there is heavy load on the servers in case of huge request traffic. Figure 3 shows the proposed architecture of the backend system.

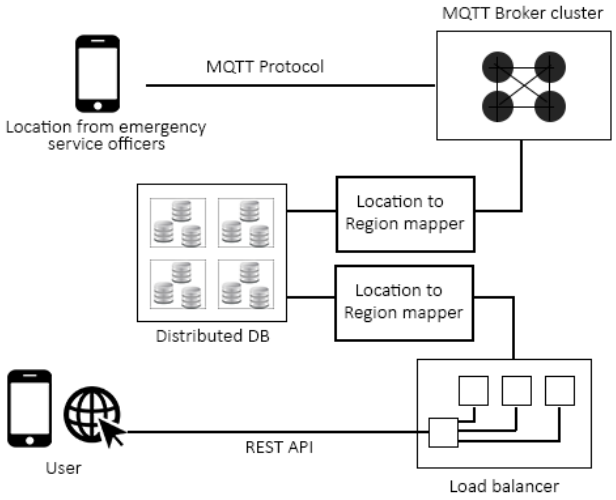

Fig. 3. Solution architecture

The solution requires use of light weight protocol so as to minimize load of sending heavy messages to servers, and to reduce internet costs. There are many protocols that support this light weight data communication (like XMPP, CoAP, DDS, and MQTT) [6]. With detailed study of protocols under constrained internet connectivity, it was found out that MQTT provides best connectivity parameters (25% better message delivery than the 2nd best protocol DDS) with Quality of service (QOS) 1. With QOS 2, DDS provide better performance (13% better message delivery than MQTT). However, the required scenario of data collection from emergency service officials doesn't require QoS 2.

QOS 0 = No message delivery guarantee

QOS 1 = Message delivered at-least once (Message redundancy might exist)

QOS 2 = Message delivered exactly once

The data collected from smartphones with officials/hardware is then sent to a clustered MQTT broker. MQTT cluster is achieved by maintaining MQTT topics and subscription lists across all the servers in the cluster.

MQTT then puts the collected data to MongoDB. MongoDB has distributed architecture and achieves high throughput through load balancing [7]. The MQTT brokers receive the location data from smartphone applications or the hardware module. This data is then put into MongoDB. The MQTT broker can be clustered by sharing a common Subscription list across the MQTT brokers.

MQTT is publish-subscribe protocol that enables message transfer through messages consisting of 2 parts – The message topic and the message payload. The server (also called MQTT broker) forwards these messages to all the devices who have 'subscribed' to that topic. By this way, MQTT enables a one to many data transfer. The primary issue faced while scaling up the MQTT servers is that a MQTT broker maintains list of all message topics and their mapping with devices subscribed to those topics. The proposed solution implements cluster of MQTT broker by making all MQTT brokers access same topic subscription list. The solution implemented involves use of Erlang based MQTT broker eMQTT [8]

The message received by MQTT broker will be put into MongoDB cluster. The reason to select MongoDB is that it supports clustering databases and thus is scalable. However, other databases like Apache Cassandra can also be used in place of MongoDB to implement the proposed solution. The clustering of MQTT and MongoDB servers ensures load distribution and fault tolerance. If one server of the cluster goes offline, the solution still will remain functional as other servers in the cluster will distribute the load of the dead server within themselves.

There are two ways to write the message received by MQTT to MongoDB. The first one involves editing MQTT broker to write any message received to the database. The other method involves creating a MQTT client subscribed to all topics and making it write all messages received to the database. Both these methods have their draw-backs. The first method will reduce the broker cluster performance as for every message received, the broker will have to write to a database. The second method will be a bottle-neck in the system as single MQTT client will be writing all messages being transferred in the system to a database. This client will crash if the number of messages transferred in the system exceed certain number. The proposed system implements writing to MongoDB using the first method. The precaution taken to prevent reduced broker performance is that the broker makes Asynchronous MongoDB write calls. This makes sure that the broker performance isn't affected much. Another alternative to tackle the reduced broker performance is to implement an in-memory database like 'REDIS' [9].

The server will receive the location data through MQTT and will store it in a database collection for that particular route number. This can be done by directly modifying the MQTT broker to write to database when data is received or by creating a MQTT client to do so. The MQTT broker writes the location received from emergency service officials to MongoDB. The format of document saved to MongoDB is the following:

```
{   "_id":"12345"
    "Latitude":123.456
    "Longitude": 123.456
    "name": "XYZ"
    "posting": "Inspector"
    "contact": +91777777777  }
```

The data saved is aggregated based on the type of service the official provides. For example, all the location data of a police officer will go into POLICE database. So when a citizen uses this platform to contact nearest policeman, the database queried will be the police database. Suppose a city has 40,000 policemen, the Policeman database will contain 40,000 location entries. When user queries data for nearest policeman, it will be highly inefficient to query all 40,000 entries to find the one nearest to the user.

As a solution to this inefficient way of querying data, the data saved into the database will be grouped by the region name corresponding to the latitude and longitude. Suppose a policeman is in region 'A', the location updates will be made to database 'policeman' and collection with name 'A'. Now, if the user is in region 'A', the query will search for locations of police in collection 'A'. The mapping of latitude to longitude to region can be done by using Google Maps API, or as done in the solution implemented, it can be done by having a database created that maps latitude longitude to regions.

When the user queries data, the region where the user is querying the data from will be recognized and then, the latitude longitude of the user will be compared with the location of all policemen from that region. The nearest policeman will be recognized and his contact details and location will be given to the user. Simultaneously, that policeman will also be sent notification with details of that user. This will ensure that the user can contact the policeman, and the policeman also already gets to know about user in need.

The user end of the proposed solution involves a smartphone application which queries the database for fetching appropriate information. This RESTFUL API can be easily scaled up by implementing a load balance like HAProxy [10].

VII. Data Access Points for Users

The users should be able to find the contact details of nearest emergency service officer on demand. The solution implemented allows this access through a website, smartphone application and SMS. The website and smartphone application query the data through RESTFUL APIs that can also be accessed through 3rd party applications (With prior authentication). To ensure authenticity of the requests the users make to the service, the user application will involve AADHAR based login. This will ensure accountability when any user makes emergency requests and will prevent users from making fake emergency requests.

The user will have to select the type of emergency he is in (Fire/Medical/Security). This will trigger appropriate REST API taking user's latitude longitude and finding appropriate officer's details for the user. When the REST API is called, the backend will first find the user's region by geo-coding. Then, the data from the region's MongoDB collection will be queried to find the officer nearest to that user. The user's location will be compared with that of officers in that region. The distance between the user and the officers will be calculated using the 'Haversine' formula:

La1: User latitude

Lo1: User longitude

La2: Officer's latitude

Lo2: Officer's longitude

Dlon: Lo2 – Lo1

Dlat: La2 – La1

R: Radius of earth

$$A = (\sin(Dlat/2))^2+\cos(La1)*\cos(La2)*(\sin(DLon/2))^2$$

$$C = 2 * \tan^{-1}(\text{sqrt}(A),\text{sqrt}(1-A))$$

$$\text{Distance} = R * C \qquad (1)$$

Using (1), the distance between the user and the officials in the particular region will be calculated. This calculation will run till officer within 1km is found. If there is no officer within 1km, the backend will select the officer with shortest distance from the user. Once the nearest officer is found, that officer will be notified on his application the details of the user in emergency (Including contact and location). Similarly, the user application automatically connect a call with that officer while showing details of that officer on the screen.

This functionality will be same for android and website (Except for automatically calling the officer which won't happen in case of website). The officer information can also be retrieved through SMS. There will be a SMS server expecting message data in a particular format. If user doesn't have internet, he can click on offline button on the app. When the user is in emergency, the application will send SMS to the SMS server. This server then extracts the user latitude and longitude to trigger the REST API. The response received is sent back to the application as an SMS.

The platform developed also allows other developers to use the REST APIs. The platform has an admin console where developers can login to register and get access tokens (Username and password for the REST API). This API requires the following parameter:

1. user_longitude (Long)
2. user_latitude (Long)
3. service_name (String : police/ambulance/fire
4. user_name
5. user_contact

The response received is of the following format:

1. officer_name
2. officer_latitude
3. officer_longitude
4. officer_contact

VIII. DATA TRANSFER DURING EMERGENCY

When the user triggers emergency request through the smartphone application, the backend connects the nearest officer with the user. There is an additional feature in the application which user can use to provide more information to the officer in real time. For example, when the user triggers medical emergency, the application connects the nearest ambulance with the user. Once this connection is established, the user application automatically calls the officer in that ambulance (Ambulance driver/Doctor). After the call is done, the application provides an interface to share more details with the officer. In-case of the ambulance example, the user can enter the medical conditions of the patient or any relevant data which can help the officer speed up the help process.

Similarly, once the ambulance reaches the user, IoT enabled sensors can be used to capture medical data related to the users [11]. This data can then be communicated to the hospital using the proposed backend to prepare the hospitals in advance.

IX. SECURITY

The proposed solution lacks detailed study of security protocols to be implemented for the scalable backend. Feasibility of implementing existing security systems [12] is one of the future scopes for this research paper. The primary factor to be addressed while implementing the security protocols is that the implementation should not hamper the light-weight nature and scalable backend performance. The implemented solution involves SSL/TLS security at MQTT connection stage. The MQTT brokers inherit this feature as a part of MQTT protocol. The additional security feature implemented is to prevent devices from sending data to servers for MQTT topics not belonging to that particular device. This means that if a device ‘A’ tries to send data for device ‘B’, the server will automatically disconnect that faulty device ‘A’ from the system.

X. RESULTS

The solution proposed in this paper has been implemented and tested for all features and functions. This system will help smart-city citizen during emergencies by letting them contact the concerned authorities with minimal steps. This can save the users from potential trouble caused by delays in communication with help. Leveraging internet availability in smart cities, this solution ensures convenience of the citizen. This system will help in management and proper allocation of emergency services in timely and efficient manner.

The system implemented was tested using Apache JMeter. The system with free tier AWS EC2 (1GB RAM, 30GB memory) was able to achieve ~25,000 concurrent MQTT connections to server. Clustering 2 MQTT brokers increased this number to 48,000 concurrent clients. Thus, using 2 free servers supports real time data streaming for around 50,000 emergency service officials. Running single node MQTT broker on AWS EC2 8GB RAM and 8 cores allowed 210,000 concurrent clients. This configuration supported 23,000 messages transfer per second (23,000 concurrent data requests). These results prove the horizontal scalability of the solution to meet the high number of users and data requests.

XI. CONCLUSION

The implemented solution provides a new, better and efficient way of accessing emergency services in a city. The solution brings all emergency services in the city to a single platform. The solution achieves light weight data transfer and high scalability due to use of clustered MQTT brokers. This solution is future ready and can easily scale up to meet the ever increasing population. Horizontal scalability is easily attainable as shown in the results. The system is thus a reliable one stop platform for communicating with the city officials when in emergency.

ACKNOWLEDGMENT

The proposed solution was implemented and demonstrated at Pune city’s ‘Digital Pune hackathon’. The executives of city governing bodies like Pune police commissioner, Pune municipal corporation head acknowledged and endorsed this platform. This solution won the hackathon owing to its unique feature and value addition to smart cities.

REFERENCES

[1] Sarbpreet, S. Tripathy and J. Mathew, "Design and evaluation of an IoT enabled secure multi-service Ambulance Tracking System," *2016 IEEE Region 10 Conference (TENCON)*, Singapore, 2016, pp. 2209-2214.

[2] MQTT , http://mqtt.org/documentation, mqtt.org

[3] Y. Chen and T. Kunz, "Performance evaluation of IoT protocols under a constrained wireless access network," *2016 International Conference on Selected Topics in Mobile & Wireless Networking (MoWNeT)*, Cairo, 2016, pp. 1-7

[4] S. Bandyopadhyay and A. Bhattacharyya, "Lightweight Internet protocols for web enablement of sensors using constrained gateway devices," *2013 International Conference on Computing, Networking and Communications (ICNC)*, San Diego, CA, 2013, pp. 334-340.

[5] O. A. Mohamad, R. T. Hameed and N. Ţăpuş, "Design and implementation of real time tracking system based on Arduino Intel Galileo," *2016 8th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, Ploiesti, Romania, 2016, pp. 1-6

[6] Vasileios Karagiannis, Periklis Chatzimisios, Francisco Vazquez-Gallego and Jesus Alonso-Zarate, “A Survey on Application Layer Protocols for the Internet of Things”, Transaction on IoT and Cloud computing, 2015

[7] MongoDB, https://docs.mongodb.com/

[8] EMQTT, http://emqtt.io/docs/v2/index.html

[9] REDIS, https://redis.io/

[10] HAProxy, http://www.haproxy.org/

[11] M. Hassanalieragh *et al.*, "Health Monitoring and Management Using Internet-of-Things (IoT) Sensing with Cloud-Based Processing: Opportunities and Challenges," *2015 IEEE International Conference on Services*

[12] Sarbpreet, S. Tripathy and J. Mathew, "Design and evaluation of an IoT enabled secure multi-service Ambulance Tracking System," *2016 IEEE Region 10 Conference (TENCON)*, Singapore, 2016, pp. 2209-2214.