Grenze International Journal of Engineering and Technology, Jul Issue

Real-Time Location Tracking System for Predetermined Routes

Aashish Nehete[1], Disha Gandhi[2], Jeet Mehta[3], Prasenjit Bhavathankar[4] and Dayanand Ambawade[5]
[1-5]Sardar Patel Institute of Technology, Mumbai, INDIA
Email: aashish.nehete@spit.ac.in, {gandhidisha0, jeetmehta1620}@gmail.com, {p_bhavathankar, dd_ambawade}@spit.ac.in

***Abstract*—For large organisations with a huge number of vehicles in their fleet, it is crucial for the controller to be able to track and manage their fleet in the most optimised manner and also make this facility available to many users. Using real-time location system, a software solution can be developed for fleet management that will report the location of vehicles in a fleet directly to the fleet manager and the users in real time. Moreover, these vehicles follow the same path repeatedly and as such generate redundant data by storing the latitude and longitude values each time; a naive solution to this problem. A new solution has been presented in this paper that will not only reduce this redundancy in storage but also reduce network usage. Using an appropriate control, a 55.8% compression in the average message size was achieved along with drastic reduction in the total number of messages being sent over the network.**

***Index Terms*— Location tracking, fleet management system, checkpoint, GPS, real-time, routes, public transport, MQTT, Kafka, SSE.**

I. INTRODUCTION

With the development of technology, the transport sector is rapidly developing, making travel easy and convenient for all. A single button on the phone helps you in arranging your way to travel, and the app features give you an added incentive. Increasing traffic in rising and developing cities has led to a high demand for automation to solve the problem of vehicle tracking. With the exponential rise of smartphones this is done by a real time location tracking system [3]. The user can locate the desired vehicle on a map and accordingly plan their schedule. The traditional methods [3], [7] involve sending long latitude and longitude values which increase the packet size of the message along with also being a potential privacy issue. The increase in the size of data packet increases the network usage and load on the central server, directly affecting the efficiency of the system.

The system aims to solve this problem by implementing a system designed for all vehicles that follow a predetermined path repeatedly. Applications range from vehicles serving logistical needs in any industry to public transport. It is easy to understand that a vehicle following the same path repeatedly will tend to generate similar data every time the process is repeated using a traditional system [2]. It is possible to eliminate this redundant data like latitude and longitude values, and only store relevant data like the time at which a vehicle is at a given location.

Especially in the public transport sector, such a system can not only impact the administrators but also any interested users who are able to track their desired vehicle, bus or train using only a smartphone. Using this knowledge, a user can make an informed decision regarding their travel [9]. Unfortunately, with such a large audience, it is necessary to make appropriate provisions of system scalability and durability. For exactly this

Grenze ID: 01.GIJET.7.2.12

reason, various technologies like MQTT, Apache Kafka and Server Sent Events are leveraged in the system.
The paper is structured as follows. Beginning with a short overview of related work followed by the architecture of the system. Then on to discussing the components involved and their working in a sequential manner. The paper is then concluded with implementation results obtained by comparing the proposed system with a control.

II. Related Work

Outlining a framework for IoT applications based on GPS, [1] mentions three important properties: 1. Emphasis on real-time; 2. Ability to handle data in large amounts; 3. Ensuring privacy and unauthorised access. Objects in motion generate location data which is generally represented by a coordinate system comprising of a latitude and a longitude value where each of these are 32-bit float values. If the update interval is too frequent, the continuous transmission of these values along with auxiliary data from a node to a central server will result in a load on the network bandwidth as well as the data stores. Moreover, as highlighted by [12] and [1], any unauthorised access to this data will result in the leak of exact locations of the object being tracked. [12] use a combination of reference points to obfuscate the actual location. A fleet management system can track the location as well as the telemetry data as demonstrated by [4]. It is possible to create low cost systems capable of tracking real-time locations that can be accessed remotely as exhibited by [5]. [6] illustrates how a combination of GPS and GSM, along with other sensors for auxiliary data, can be used as nodes on an IoT application to achieve remote tracking purposes. Apart from GPS, other location techniques can also be used as demonstrated by [9] includes implementing a bus tracking system inside a campus.
To create a scalable system it was important to select communication technology appropriately. Kafka was found to be better than competitors like RabbitMQ and ActiveMQ [10]. Furthermore, the lightweight MQTT protocol delivers the shortest latency [11] for IoT applications. Also server sent events is simplex communication technique based on the HTTP protocol.

III. Proposed System

In this section, the proposed tracking system to locate a vehicle is described. An introduction to the concept is stated followed by a list of components involved. Finally, a series of steps illustrate the working of the entire system.

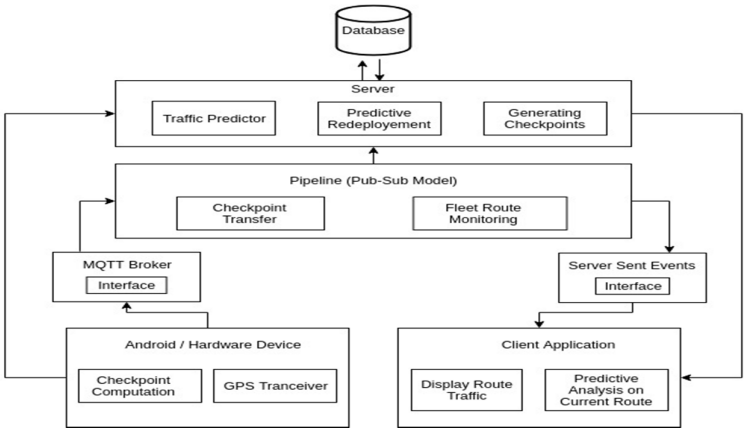

Figure 1. Architecture of the Location Tracking System

A. Basis: Checkpoint Theory

The nature of fleet management systems or public transport is such that the route of a vehicle is predetermined and repeated. It is assumed that both parties: the vehicle and remote user, can be made aware of this route beforehand. In such a scenario, it will be redundant to transmit the exact location values from the vehicle to a remote user every time. Alternatively, this redundancy can be avoided by using checkpoints along the predetermined route. The route is divided into parts, thus converting a single route into a list of checkpoints. These checkpoints and their associated coordinate values are communicated to the end users as well as the vehicle being tracked in advance. A geofence [8] of set radius is created around all checkpoints, effectively triggering an event whenever the vehicle enters any such geofence. As soon as the event is triggered, a message

containing the route information and checkpoint number is sent to the end user. Using prior knowledge as mentioned above, the client is sufficiently able to decode the location coordinates from the received checkpoint number.
This reduces the redundancy introduced in the system by two ways:

1. It reduces the size of data packet by using a checkpoint value which is an integer instead of huge latitude and longitude values.
2. The update interval at the remote end is based on predefined distance intervals rather than time intervals. For example, a system which updates the remote end after 1000 ms will send redundant data to the server if the vehicle is stationary.

B. Components

Fig. 1 shows the schematic diagram of the proposed system. Separation of various components not only improve privacy of the user but also contribute towards scalability and durability. These components are as follows:

1) Client Application: An Android application will consist of a map enabled to display the real-time location of the vehicle being tracked to the user. The application receives messages from the pipeline using Server Sent Events. Fur- thermore, data of all required routes and checkpoints will be made available locally in a database. Sample data is presented in the following format:

```
{
  "_id": "5c52f4c66cc73b10ef0b0cc7",
  "name": "Route1",
  "checkpoints":
    { "id": 1, "latitude": 19.068128, "longitude": 72.848526 },
    { "id": 2, "latitude": 19.067638, "longitude": 72.847028 },
    ...
}
```

The route id field is used to uniquely identify the route and using the checkpoint field, it is possible to retrieve the latitude and longitude values corresponding to a specific checkpoint number. After retrieval, this location is displayed on the map. If the route or checkpoint is not found locally, a query is sent to the remote server to fetch the required details. Thus, the local database acts as a cache for the checkpoint values.

2) Android/Hardware Device: To enable tracking of the vehicle, a device equipped with GPS transceivers and a communication module like GSM, ethernet or WiFi that can leverage TCP/IP protocol is required to be attached with the vehicle. Fortunately, a smartphone satisfies both these requirements. Hence by creating an Android application it is possible to access the current location of the vehicle as well as communicate over the internet. The device also stores the necessary route and checkpoints data locally. If the data is not available, it can be fetched from a remote server initially. Once the route is identified, a geofence is created around all the checkpoints belonging to that route. Essentially, geofence is a circular area defined by a set radius with the checkpoint at centre. Afterwards, a trigger is activated whenever the vehicle enters into any geofence. On activation, the corresponding checkpoint number is identified and along with the route id, are encoded into JSON and transmitted via the MQTT protocol to a remote MQTT Broker. Below is a sample message that is transmitted in this step.

```
{ "route_id": "5c52f4c66cc73b10ef0b0cc7", "checkpoint": 3 }
```

3) Pipeline: Kafka is a distributed streaming platform based on a publish-subscribe model. It acts as a pipeline between the vehicle and the end user, and a distributed store for the events being transmitted. Additionally, the pipeline has no knowledge of actual locations being tracked as it only deals with the route id and checkpoint numbers, making it unaware of the exact coordinate values. Even if unauthorised access is gained to this pipeline, it would be impossible to determine the exact location thus ensuring privacy in the system.
4) MQTT Broker: The tracking device sends a message using the lightweight MQTT protocol which is based on top of the TCP/IP protocol. Mosquitto is a message broker used to receive these messages remotely. Using the Kafka Connector API, this broker is transformed into a data source for the Kafka broker thus enabling all incoming messages to directly flow into the pipeline.
5) Server Sent Events: Server Sent Events is a uni-directional communication protocol based on HTTP. The user receiving the real-time location of the vehicle follows a one-way communication. In such a scenario, it is beneficial to use a simplex communication technique like SSE. A NodeJS application is used to consume messages from the Kafka broker and feed it into any open connections from the end user.

6) Server: A central server is responsible for generating and storing the routes and checkpoints. A REST API in NodeJS exposes CRUD operations on the data to all authorised users. Hence, all tracking devices and client applications can use this API to query for route and checkpoint data on-demand. Moreover, this server is independent of the pipeline and can be deployed independently to ensure privacy and scalability.

7) Database: The server uses a MongoDB database to store route and checkpoint data.

C. Working

1. The tracking device on a vehicle requests a list of checkpoints for its specific route.
2. A geofence is created around all checkpoints.
3. As soon as the object triggers the geofence, it identifies the corresponding checkpoint and sends it along with the route_id to the MQTT Broker.
4. The message is passed on to the Kafka pipeline using the Kafka Connector API.
5. The SSE interface is subscribed to the pipeline and receives the message.
6. On receipt, the SSE interface pushes the message to all running client application who are interested in the specific route.
7. The client applications also request a list of checkpoints for the specific route and creates a local copy.
8. The client application on arrival of a message, decodes the message to identify the checkpoint. Using the local copy of checkpoints, the exact coordinate values are determined.
9. The location is displayed on a map to the user.
10. Steps 3-9 are repeated for all the checkpoints on the route.

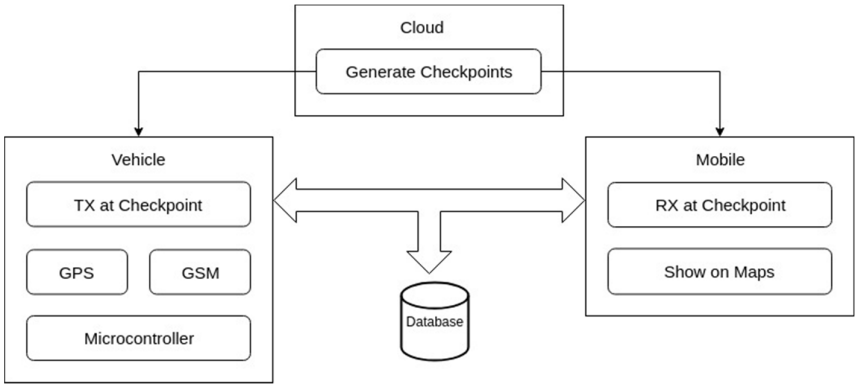

Figure 2. Flow of message in the system

IV. RESULTS

TABLE I. MESSAGES USING PROPOSED APPLICATION

Sr No	Message	Bytes
1	{”route id”:”5c52f4c66cc73b10ef0b0cc7”, 54 ”checkpoint”:1}	54
2	{”route id”:”5c52f4c66cc73b10ef0b0cc7”, 54 ”checkpoint”:2}	54
3	{”route id”:”5c52f4c66cc73b10ef0b0cc7”, 54 ”checkpoint”:3}	54
….	….	….
25	{”route id”:”5c52f4c66cc73b10ef0b0cc7”, 54 ”checkpoint”:25}	55

TABLE II. MESSAGES USING CONTROL APPLICATION

Sr No	Message	Bytes
1	{"latitude":19.06815276332509, "longitude":72.84854496435365, "route_id":"5c52f4c66cc73b10ef0b0cc7"}	98
2	{"latitude":19.0681159897221, "longitude":72.84842059261777, "route_id":"5c52f4c66cc73b10ef0b0cc7"}	97
3	{"latitude":19.068079327936598, "longitude":72.848296599059, "route_id":"5c52f4c66cc73b10ef0b0cc7"}	97
….	….	….
25	{"latitude":19.124826399568803, "longitude":72.83836606889963, "route_id":"5c52f4c66cc73b10ef0b0cc7"}	99

These results are based on readings taken in Sardar Patel Institute of Technology, Mumbai. The complete details of the experiment are further explained.

Results of the proposed system can be acquired using simulation studies. For the purpose of comparison, two different applications were created on the Android platform. One implements the checkpoint algorithm, while the other is a control which uses default functionality provided by the Android platform to receive location updates. The simulation was performed on a 10 kilometer stretch of road in Mumbai. In the proposed application 25 checkpoints were used; roughly one checkpoint per 400 meters. Table I tabulates part of the messages that are being transmitted using the proposed algorithm. On the other hand, Table II tabulates part of the messages being transmitted using the control application.

The results can be quantised using 2 metrics:

Number of Messages While using the proposed algorithm, it is evident that the number of messages will always be equal to the number of checkpoints created. In the test case, that value is 25. However for the control application, location updates were received with the minimum possible frequency available on the Android platform. It was found that 712 messages were transmitted by the control application. Therefore,

$$N_{proposed} = 25 \qquad (1)$$

$$N_{control} = 712 \qquad (2)$$

TABLE III. ACCURACY FOR DIFFERENT CHECKPOINTS

Checkpoints	Accuracy (meters)	Total Size (bytes)
25	400	1366
50	200	2741
100	100	5492
200	50	11092

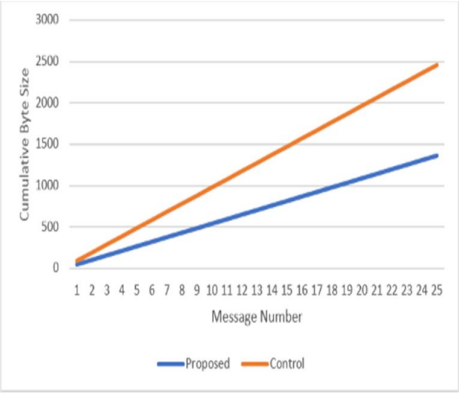

Figure 3. Message Number vs Cumulative byte size

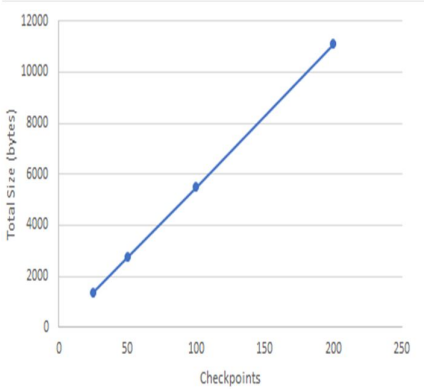

Figure 4. Checkpoints vs Total size in bytes

The control application sends almost 28 times more messages than the proposed system. Part of this drastic difference also lies in the fact that the control application continues sending messages even when the vehicle is stationary.

Average Message Size The size of a message is determined by the number of bytes required by the JSON encoded message. Table I and Table II also contain the bytes of each message. Average message size for proposed algorithm,

$$\textit{(total size in bytes)/(number of messages)} = 1365/25 = 54.6 \textit{ bytes} \qquad (3)$$

And for the control application,

$$\textit{(total size in bytes)/(number of messages)} = 69669/712 = 97.849 \textit{ bytes} \qquad (4)$$

Thus the compression achieved per message is given by,

$$(Avg_{proposed})/(Avg_{control}) = 54.6/97.849 = 0.558 \quad (5)$$

This gives us a compression rate of 55.8% per message.

Fig. 3 is a graph which plots the cumulative byte size against the message number for both the proposed and control applications plotted for the first 25 messages. It is evident from the graph that the control application diverges higher than the proposed application.

Consequently, the accuracy of location for the application is directly proportional to the number of checkpoints. In the above example, 25 equally spaced checkpoints were placed along a 10km route. Hence it can be said that the accuracy of this system is 400 meters. The following table tabulates the accuracy as well as the total number of bytes required to send all messages.

It can be inferred from Table III that better accuracy requires a compromise on the number of data packets as well as the total size of the data being transferred. From Fig. 4 it is evident that the relationship between checkpoints and total size is linear. Thus an optimum number of checkpoints must be decided based on the requirements. Parameters like the speed of vehicle or user requirements should be considered while making such a decision.

IV. Conclusion

As described above in the graphs, the system can not only track real-time location of vehicles but also ensure scalability and low bandwidth usage for the same. The proposed checkpoint theory has been successfully implemented. This revised system provides less redundancy, low latency and more security when compared with traditional systems. The system can be modified for various use cases like Commercial City Buses, Airplanes, Shipping Lanes, Government Service Vehicles, etc. Further additions to this app would include analytics performed on the prefetched data.

Future Scope

A concept of dividing a single route into granular checkpoints has been discussed in this paper. With the advent of the information age and rise of ubiquitous computing, it is necessary that efficient services are made available to the public using technology at hand. Generation of large amounts of data from IoT devices asks for larger data storage capacity. Although easily found, data storage should be further optimised not only to keep redundancy in check, but also streamline the following analysis carried out on the data. Armed with the location data and timestamp, time-series analysis can be used to determine estimated time of arrivals at each of those checkpoints. Furthermore, in a public transport system like buses, the stops can be added as checkpoints to achieve ETA results similarly.

Apart from the obvious applications containing predetermined routes, a personal use case can leverage this concept as well. Oftentimes while travelling, the destination, and maybe, the route is also predetermined. If someone intends to share their location remotely, the checkpoints can be similarly implemented with all parties involved. Although, unexpected changes in the route need to be handled appropriately in such cases.

References

[1] J. Lakshmi, S. K. Nandy, A. Gupta, P. K. Akulakrishna, and V. A. Kachore, “Cloud Framework for IoT associated GPS location based applications,” Cloud Systems Lab, SERC, IISc, Bangalore-12, India, Oct. 2015.

[2] H. D. Pham, M. Drieberg and C. C. Nguyen, "Development of vehicle tracking system using GPS and GSM modem," 2013 IEEE Conference on Open Systems (ICOS), 2013, pp. 89-94, doi: 10.1109/ICOS.2013.6735054.

[3] S. Lee, G. Tewolde and J. Kwon, "Design and implementation of vehicle tracking system using GPS/GSM/GPRS technology and smartphone application," 2014 IEEE World Forum on Internet of Things (WF-IoT), 2014, pp. 353-358, doi: 10.1109/WF-IoT.2014.6803187.

[4] A. Goel and V. Gruhn, "A Fleet Monitoring System for Advanced Tracking of Commercial Vehicles," 2006 IEEE International Conference on Systems, Man and Cybernetics, 2006, pp. 2517-2522, doi: 10.1109/ICSMC.2006.385242.

[5] G. Kiran Kumar, A. Mallikarjuna Prasad, “Public Transportation Mangement Service using GPS-GSM”, International Journal of Research in Computer and Communication Technology, IJRCCT, ISSN-2278-5841, Vol-1, Issue -3, Aug -2012.

[6] D. Jose, S. Prasad, and V. G. Sridhar, “Intelligent Vehicle Monitoring Using Global Positioning System and Cloud Computing,” Procedia Computer Science, vol. 50, pp. 440–446, 2015, doi: 10.1016/j.procs.2015.04.012.

[7] K. Maurya, M. Singh, and N. Jain, “Real time vehicle tracking system using GSM and GPS technology-an anti-theft tracking system,” International Journal of Electronics and Computer Science Engineering, vol. 1, Jun. 2012.

[8] D. Suganthi, J. Paul S, J. Shamil, Patel, Dhruva G, and U. Student, “Vehicle tracking with geo fencing on android platform,” International Journal of Engineering Science and Technology, vol. 8, Art. no. 4, 2018.
[9] E. C.-W. Lau, “Simple bus tracking system,” Journal of Advanced Computer Science and Technology Research, vol. 3, Art. no. 1, 2013.
[10] J. Kreps, N. Narkhede, and J. Rao, “Kafka: A distributed messaging system for log processing,” 2011, vol. 11, pp. 1–7.
[11] Z. B. Babovic, J. Protic and V. Milutinovic, "Web Performance Evaluation for Internet of Things Applications," in IEEE Access, vol. 4, pp. 6974-6992, 2016, doi: 10.1109/ACCESS.2016.2615181.
[12] M. K. Jones and D. C. Hicks, “System and method for enciphering and communicating vehicle tracking information,” 2002.