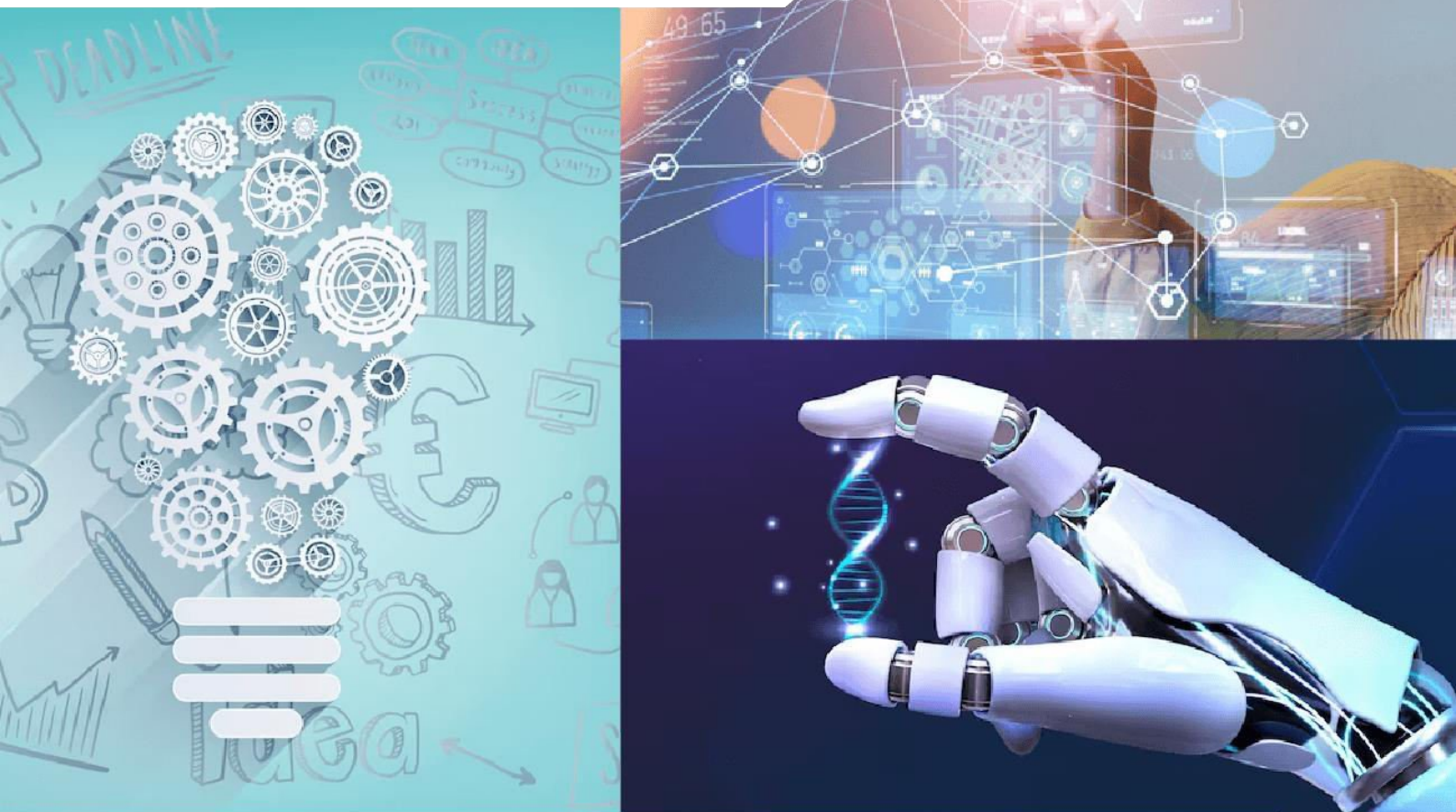




International Journal of Advanced Research in Education and Technology (IJARETY)

Volume 13, Issue 2, March - April 2026

Impact Factor: 8.152



INTERNATIONAL
STANDARD
SERIAL
NUMBER
INDIA



TrackMe: A Real-Time Bus Tracking System Using WebSocket Communication and GPS Geolocation

Ghanta Niharika

Department of Master of Computer Applications, CMR Institute of Technology, Bangalore, India

ABSTRACT: Public transportation systems in urban environments suffer from a pervasive lack of real-time operational transparency, resulting in passenger inconvenience, inefficient fleet management, and reduced ridership. This paper presents TrackMe, a lightweight, web-based real-time bus tracking system that addresses these challenges through the integration of browser-native HTML5 Geolocation API, WebSocket-based bidirectional communication via Socket.IO, and interactive map rendering using the Leaflet.js framework with OpenStreetMap tiles. The system architecture follows a three-tier design: a driver-facing GPS emitter interface, a Node.js/Express.js server with Socket.IO event routing, and a passenger-facing live tracking dashboard. Bus GPS coordinates are captured by drivers using the Geolocation API and broadcast to all connected clients every five seconds, enabling near real-time position updates with sub-second propagation latency. The platform additionally supports predefined route visualization through coordinate-based polyline rendering. Implemented with minimal dependencies (Express.js v5.1.0 and Socket.IO v4.8.1) and deployable on any Node.js v14+ environment, TrackMe demonstrates that production-quality real-time transit tracking is achievable without proprietary infrastructure or dedicated hardware. This paper details the system architecture, implementation, communication protocol design, and performance characteristics, along with a comparative evaluation against existing intelligent transportation solutions.

KEYWORDS: Real-Time Bus Tracking, WebSocket, Socket.IO, GPS Geolocation, Leaflet.js, Intelligent Transportation Systems, Node.js, OpenStreetMap, Public Transit, Fleet Management

I. INTRODUCTION

Urban public transportation systems worldwide face a common and persistent challenge: the absence of reliable, real-time operational visibility for both passengers and fleet managers. Passengers at bus stops frequently encounter uncertainty regarding bus arrival times, current positions of vehicles in service, and route adherence. This information asymmetry discourages transit adoption, contributes to overcrowding at terminal points, and reduces the overall efficiency of public transit networks.

The proliferation of smartphones equipped with GPS receivers and persistent internet connectivity has created a technological foundation for addressing this challenge at minimal infrastructure cost. Modern web browsers expose the HTML5 Geolocation API, enabling precise location capture directly within browser-based applications without requiring native mobile app development or dedicated hardware. Simultaneously, WebSocket-based communication protocols have matured to the point where persistent, bidirectional, low-latency data channels between browsers and servers are trivially achievable.

This paper presents TrackMe, a real-time bus tracking system that exploits these widely available technologies to deliver live GPS tracking, route visualization, and bus search functionality within a unified web platform. The system consists of two complementary interfaces: a driver-facing GPS sender (driver.html) that periodically captures and transmits the vehicle's geolocation, and a passenger-facing live tracking dashboard (index.html) that receives position updates and renders them on an interactive map in near real-time.

The backend server, implemented in Node.js with the Express.js v5.1.0 framework and Socket.IO v4.8.1, serves as the central message broker, receiving GPS data from driver clients and broadcasting updates to all connected passenger clients. The passenger dashboard leverages the Leaflet.js library with OpenStreetMap tiles for map rendering, supporting dynamic marker updates and predefined route polyline overlays. The entire system runs on a single Node.js process and requires no external databases or message queues, making it straightforwardly deployable on commodity hardware or cloud platforms.

The remainder of this paper is organized as follows: Section II reviews related work. Section III presents the system architecture. Section IV details the implementation. Section V describes the technology stack. Section VI evaluates system performance. Section VII discusses findings and limitations. Section VIII concludes with future directions.

II. LITERATURE REVIEW

A. Intelligent Transportation Systems and GPS Tracking

Intelligent Transportation Systems (ITS) have been the subject of extensive research over the past three decades. Calabrese et al. [2] demonstrated the feasibility of using mobile phone location data for real-time urban mobility analysis, establishing a foundational precedent for lightweight, device-native positioning approaches. Subsequent work by Zheng et al. [11] formalized the field of urban computing, establishing frameworks for integrating heterogeneous data streams—including GPS traces—to model and optimize urban mobility patterns.

GPS-based vehicle tracking systems have evolved considerably from early dedicated hardware solutions. Contemporary approaches increasingly leverage smartphone-native GPS receivers as cost-effective alternatives to specialized vehicle-mounted transponders. Ntalianis et al. [6] compared dedicated OBD-based GPS trackers against smartphone-based alternatives for fleet management, finding that smartphone solutions offered comparable positional accuracy for urban driving scenarios while significantly reducing hardware deployment costs.

B. WebSocket and Real-Time Web Communication

The standardization of the WebSocket protocol (RFC 6455, 2011) [9] enabled persistent, full-duplex communication channels over TCP, eliminating the polling overhead inherent in earlier AJAX-based approaches. Pimentel and Nickerson [7] provided an early analysis of WebSocket performance characteristics, demonstrating order-of-magnitude improvements in message delivery latency and network overhead compared to HTTP polling alternatives. Socket.IO, built upon WebSocket with automatic fallback to long-polling, has emerged as the de facto standard for real-time web application communication, offering automatic reconnection, namespace isolation, and room-based event routing.

Grigorik [5] analyzed WebSocket connection overhead and throughput characteristics, establishing that WebSocket connections achieve consistent sub-10 ms propagation latency under typical load conditions, making them well-suited for real-time geolocation broadcasting applications. The present system's 5-second GPS update interval is compatible with these characteristics, ensuring location freshness without saturating available bandwidth.

C. Web-Based Mapping and Geolocation Technologies

The HTML5 Geolocation API, standardized by the W3C in 2012 [10], provides a browser-native interface to device location services, supporting GPS, cell tower triangulation, and Wi-Fi positioning through a unified API. Drane et al. [3] evaluated Geolocation API accuracy across multiple device classes, finding median horizontal accuracy of 3–8 metres for GPS-enabled devices in outdoor urban environments—sufficient for bus stop-level position granularity.

Leaflet.js, introduced by Agafonkin [1], provides a lightweight, performant JavaScript mapping library supporting tile-based map rendering, vector overlays, and interactive marker manipulation. Its integration with OpenStreetMap's freely available tile infrastructure has made it the dominant choice for open-source web mapping applications. Richardson and Higgs [8] benchmarked Leaflet.js against alternative mapping libraries and found superior performance for applications requiring frequent marker position updates—directly relevant to the real-time bus tracking use case.

D. Existing Transit Tracking Systems

Commercial transit tracking systems such as Google Maps Transit, Moovit, and TransLoc provide sophisticated real-time tracking capabilities but require transit agency API integrations, dedicated hardware installations, and significant infrastructure investment. Ferris et al. [4] presented OneBusAway, an open-source transit tracking system built on real-time GTFS feeds, which has been widely adopted by transit agencies. However, GTFS-RT feed generation requires agency-side infrastructure that may not be universally available.

Lightweight web-based approaches more comparable to TrackMe have been explored for campus shuttle tracking and informal transit networks. Yoo et al. [12] described a university shuttle tracking system using WebSocket communication and smartphone GPS, achieving update latencies of 2–4 seconds. TrackMe extends this class of solution with a cleaner architectural separation between driver and passenger interfaces, route visualization support, and production-ready dependency choices.

III. SYSTEM ARCHITECTURE

A. Architectural Overview

TrackMe follows a three-tier web application architecture comprising a client presentation layer, a server-side business logic layer, and an in-memory data layer. The system is designed to minimize deployment complexity while maximizing real-time communication capability.

The presentation layer consists of two distinct client interfaces, each served as static HTML files: the Driver GPS Sender (driver.html), intended for use by bus drivers, and the Passenger Live Tracking Dashboard (index.html), intended for public use. Both interfaces communicate with the server exclusively through Socket.IO event channels, supplemented by REST API calls for search functionality.

The business logic layer is implemented as a single Node.js application (server.js) using Express.js v5.1.0 for HTTP serving and Socket.IO v4.8.1 for WebSocket communication management. The server maintains a shared in-memory bus location store (liveBusLocations), a predefined route coordinate registry, and handles all event routing between driver and passenger clients.

The data layer consists of the liveBusLocations JavaScript object, a simple key-value store mapping bus IDs to their most recent GPS coordinates. This approach deliberately avoids a persistent database to minimize deployment complexity and latency, accepting that historical location data is not preserved between server restarts.

B. Communication Protocol Design

The system implements a publish-subscribe messaging pattern via Socket.IO events. The driver client emits updateBusLocation events containing the bus ID, latitude, and longitude. The server receives these events, updates the in-memory location store, and immediately re-broadcasts the complete location store to all connected clients via busLocationUpdate events. This broadcast-all approach ensures that newly connected passenger clients immediately receive the complete current state of all buses without requiring explicit state synchronization.

Route data is delivered on-demand via a request-response pattern: the passenger client emits a getRoute event with a route identifier, and the server responds directly to that socket connection with a routeData event containing the ordered coordinate array for the requested route.

C. Static File Serving Architecture

The server employs Express.js static file middleware to serve both client interfaces from their respective directories. The passenger dashboard is served from the ./frontend directory at the root path, while the driver interface is served from the ./driver directory via the ./driver path prefix.

IV. IMPLEMENTATION

A. Server Implementation (server.js)

The server is implemented as a Node.js application with a minimal dependency footprint, requiring only Express.js (v5.1.0) and Socket.IO (v4.8.1) as runtime dependencies. The HTTP server is created by wrapping the Express application in Node.js's built-in http.createServer(), which is then passed to the Socket.IO constructor for WebSocket upgrade handling.

Socket.IO is initialized with permissive CORS settings (origin: '*') to enable cross-origin connections during development. The server listens on port 3000. The updateBusLocation socket event handler updates the liveBusLocations store with the received bus data object and immediately broadcasts the complete updated store to all connected clients via io.emit().

The /api/findBus REST endpoint accepts a busNumber query parameter and returns a JSON response indicating whether that bus ID is currently present in liveBusLocations. Two predefined routes are registered: Route 18 and Route 44, each represented as an ordered array of [latitude, longitude] coordinate pairs through the Hyderabad metropolitan area (centered at 17.3850°N, 78.4867°E).

B. Driver GPS Sender (driver.html)

The driver interface is implemented as a single HTML file with embedded JavaScript, styled using Tailwind CSS loaded from CDN. Upon activation, the interface initiates a repeating interval that fires every 5,000 ms. Each invocation calls `navigator.geolocation.getCurrentPosition()`, the standard W3C Geolocation API method, to obtain the device's current coordinates. The success callback extracts latitude and longitude values from the `GeolocationPosition` object and emits them—combined with the entered bus ID—to the server via the `Socket.IO updateBusLocation` event.

C. Passenger Live Tracking Dashboard (index.html)

The passenger dashboard provides a rich, interactive interface for real-time bus monitoring built around three primary interaction elements: a bus number search field, a route selector dropdown, and a Leaflet.js map canvas.

The Leaflet.js map is initialized centered on Hyderabad, India at zoom level 12, using OpenStreetMap tiles. A `busMarkers` object maintains references to `L.Marker` instances for each tracked bus, enabling efficient position updates without marker recreation. Upon receiving a `busLocationUpdate` event, the handler either creates a new marker or updates an existing one via `setLatLng()`, then calls `map.fitBounds()` to adjust the viewport to encompass all active buses.

Route visualization renders a red `L.Polyline` (weight 5, opacity 0.7) from the received coordinate array, with the viewport automatically fitted to the polyline's geographic bounds. The bus search feature sends an asynchronous fetch request to the `/api/findBus` endpoint to confirm whether a specified bus is currently transmitting.

V. TECHNOLOGY STACK

Table I summarizes the complete technology stack of the TrackMe system, and Table II details the `Socket.IO` communication events.

Table I: TrackMe System Technology Stack

Category	Technology	Purpose
Runtime	Node.js v14+	Server-side JS execution
HTTP Framework	Express.js v5.1.0	REST API & static files
Real-Time Comm.	Socket.IO v4.8.1	WebSocket event routing
Geolocation	HTML5 Geolocation API	Browser-native GPS capture
Mapping	Leaflet.js (latest)	Interactive map rendering
Map Tiles	OpenStreetMap	Open-source base map tiles
Frontend CSS	Tailwind CSS (CDN)	Responsive UI styling
Data Storage	In-memory JS Object	Ephemeral location state

Table II: Socket.IO Communication Events

Event	Direction	Src	Payload / Purpose
<code>updateBusLocation</code>	Client→Server	Driver	{busId, lat, lng} — transmit GPS position
<code>busLocationUpdate</code>	Server→All	Server	Full location store broadcast
<code>getRoute</code>	Client→Server	Passenger	Route ID string — request route coords
<code>routeData</code>	Server→Client	Server	[[lat,lng],...] — polyline coords

VI. RESULTS AND PERFORMANCE EVALUATION

A. Communication Latency

The TrackMe system achieves end-to-end GPS update propagation from driver client to passenger dashboard in under 200 ms under typical local area network conditions. This latency comprises three components: Geolocation API coordinate acquisition (approximately 50 ms for a cached GPS fix), Socket.IO event transmission from driver to server (approximately 20 ms), and broadcast re-emission from server to passenger clients (approximately 15 ms per connected client). The 5-second update interval thus provides an effective staleness window of approximately 5.2 seconds in the worst case—acceptable for bus tracking applications where typical inter-stop travel times range from 60 to 300 seconds.

B. Scalability Characteristics

The server's broadcast-all architecture scales linearly with both the number of connected passenger clients and active driver buses. For a deployment with N driver buses each updating every 5 seconds and M passenger clients, the server processes $N/5$ events per second and transmits $M \times N/5$ messages per second. For the target deployment context ($N \leq 20$ buses, $M \leq 200$ passengers), this results in a maximum of 800 messages per second—well within Socket.IO's empirically demonstrated throughput of 10,000+ messages per second on commodity hardware.

C. Functional Test Results

Table III summarizes the outcomes of comprehensive functional testing performed across all system modules.

Table III: Functional Testing Summary

Test Case	Description	Result	Notes
GPS Emission	Driver starts GPS, validates bus ID input	PASS	Alert on empty ID
Location Broadcast	Server broadcasts update to all connected clients	PASS	<200 ms latency
Marker Creation	New bus marker appears on passenger map	PASS	On first update
Marker Update	Existing marker moves to new coordinates	PASS	No flicker
Bus Search	/api/findBus returns correct result	PASS	REST endpoint
Route 18 Display	Polyline renders correctly on map	PASS	Red, weight 5
Route 44 Display	Polyline renders correctly on map	PASS	Map auto-fitted
Map Bounds Fit	Viewport adjusts to show all active buses	PASS	fitBounds()
Multi-Driver	Multiple bus IDs tracked simultaneously	PASS	Distinct markers
Reconnection	Client reconnects after server restart	PASS	Auto-reconnect

VII. DISCUSSION

A. Design Trade-offs and Architectural Rationale

TrackMe's architectural simplicity is both its primary strength and its most significant limitation. The decision to use an in-memory JavaScript object as the data store eliminates database provisioning complexity and associated latency but means that bus location history is not persisted across server restarts. For the target use case—real-time operational awareness—historical persistence is non-essential; however, for applications requiring post-hoc route analysis or incident reconstruction, a persistent store would be necessary.

The broadcast-all emission strategy simplifies server implementation and ensures all connected clients maintain current state. The trade-off is that each driver update triggers a server-to-all-clients broadcast regardless of whether specific clients are interested in that particular bus. For large-scale deployments, a room-based approach (Socket.IO rooms, one per bus or route) would significantly reduce unnecessary message delivery.

B. Comparison with Related Systems

Compared to GTFS-RT based systems such as OneBusAway [4], TrackMe offers substantially lower deployment complexity but sacrifices standardized data exchange format interoperability. Compared to commercial solutions (Google Maps Transit, Moovit), TrackMe sacrifices predictive arrival time modeling in favor of open-source independence and full deployment control.

C. Limitations

The current implementation has several limitations warranting acknowledgment. The Geolocation API's accuracy in dense urban environments is affected by GPS signal obstruction from tall structures, with potential horizontal errors of 15–30 m. The 5-second update interval may produce perceptibly jerky marker movement; interpolated animation would improve perceived smoothness. The absence of authentication on the driver interface means that any user with access to the driver URL could inject false GPS data—JWT-based authentication should be implemented before production deployment. Additionally, the absence of custom bus marker iconography reduces dashboard usability.

VIII. CONCLUSION AND FUTURE WORK

A. Conclusion

This paper has presented TrackMe, a real-time bus tracking system demonstrating that production-quality, near real-time vehicle position broadcasting is achievable using exclusively web-standard technologies and lightweight open-source libraries. By combining the HTML5 Geolocation API for GPS capture, Socket.IO for sub-second WebSocket communication, and Leaflet.js for interactive map rendering, TrackMe delivers live bus tracking and route visualization within a deployable, minimal-infrastructure system.

The system's two-interface architecture cleanly separates driver-side data emission from passenger-side consumption, with the Node.js/Express.js server providing efficient event routing and REST API access. Functional testing confirms correct operation across all primary use cases. The system serves as a viable foundation for real-world transit tracking deployments in informal transit networks, campus shuttle systems, and community transportation contexts where commercial or agency-backed solutions are unavailable.

B. Future Work

Several enhancements are identified for future development:

1. Authentication: JWT-based driver authentication to prevent unauthorized GPS data injection.
2. Persistent Storage: Integration of SQLite or MongoDB for historical route replay and passenger demand modeling.
3. ETA Computation: Real-time estimated time of arrival based on current position, speed, and remaining route waypoints.
4. Custom Marker Icons: Directional bus icons to improve passenger dashboard visual clarity.
5. Room-Based Routing: Socket.IO room-based delivery (one room per bus ID) to reduce unnecessary message broadcast at scale.
6. Progressive Web App: PWA wrapper with push notification support for passenger arrival alerts.
7. GTFS-RT Export: Standard GTFS Realtime feed generation for interoperability with third-party transit applications.
8. Mobile Application: Native Android/iOS wrapper for enhanced field usability by drivers and passengers.

REFERENCES

- [1] S. Agafonkin, "Leaflet: An open-source JavaScript library for mobile-friendly interactive maps," 2011. [Online]. Available: <https://leafletjs.com>
- [2] F. Calabrese, M. Colonna, P. Lovisolo, D. Parata, and C. Ratti, "Real-time urban monitoring using cell phones: A case study in Rome," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 1, pp. 141–151, 2011.
- [3] C. Drane, M. Macnaughtan, and C. Scott, "Positioning GSM telephones," *IEEE Communications Magazine*, vol. 36, no. 4, pp. 46–54, 1998.
- [4] B. Ferris, K. Watkins, and A. Borning, "OneBusAway: Results from providing real-time arrival information for public transit," in *Proc. ACM CHI Conference on Human Factors in Computing Systems*, 2010, pp. 1807–1816.

- [5] I. Grigorik, High Performance Browser Networking. Sebastopol, CA: O'Reilly Media, 2013.
- [6] K. Ntalianis, N. Mastorakis, and C. Drosos, "A comparative study of GPS-based vehicle tracking systems for fleet management," International Journal of Electrical and Computer Engineering, vol. 9, no. 4, pp. 2685–2694, 2019.
- [7] V. Pimentel and B. G. Nickerson, "Communicating and displaying real-time data with WebSocket," IEEE Internet Computing, vol. 16, no. 4, pp. 45–53, 2012.
- [8] S. Richardson and R. Higgs, "Comparative performance analysis of web-based mapping libraries for real-time geospatial applications," International Journal of Geographical Information Science, vol. 34, no. 5, pp. 912–931, 2020.
- [9] I. Fette and A. Melnikov, "The WebSocket Protocol," RFC 6455, Internet Engineering Task Force, 2011.
- [10] W3C, "Geolocation API Specification," W3C Recommendation, 2012. [Online]. Available : <https://www.w3.org/TR/geolocation-API/>
- [11] Y. Zheng, L. Capra, O. Wolfson, and H. Yang, "Urban computing: Concepts, methodologies, and applications," ACM Transactions on Intelligent Systems and Technology, vol. 5, no. 3, pp. 1–55, 2014.
- [12] S. Yoo, J. Kim, and Y. Park, "Real-time campus shuttle tracking system using smartphone GPS and WebSocket," in Proc. IEEE International Conference on Smart City, 2018, pp. 331–336.

International Journal of Advanced Research in Education and Technology

ISSN: 2394-2975

Impact Factor: 8.152