ISSN No. 0976-5697

VOLUME 17 SPECIAL ISSUE 1, JULY 2026
International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

SMART-TRAC: AN IOT-BASED REAL-TIME PUBLIC TRANSIT AND COMMUTER MONITORING SYSTEM FOR SMART CITIES

Meenal Joshi, Kushal Suthar, Priyanshu Garg, Somya Parikh
Department of CSE
Geetanjali Institute of Technical Studies
Udaipur, Rajasthan, India
jmeenal09@gmail.com, gargp6089@gmail.com,
Khappy7727@gmail.com, Somyaparikh09@gmail.com

Abstract: In recent years, commuters in tier-2 towns and small cities have encountered significant difficulties due to the lack of real- time tracking in public transport systems, leading to unpredictable schedules and overcrowding. This paper presents a comprehensive full-stack solution addressing the urgent need for a more reliable, optimized, and transparent approach to public transit management. The proposed system is a low-bandwidth Progressive Web Application (PWA) paired with an edge-computing IoT ingestion pipeline, designed to simplify commute planning through real-time GPS tracking and accurate estimated times of arrival (ETAs). Rather than relying on heavy, data-intensive map rendering and traditional HTTP polling, the system enables users to access critical transit data using minimal network resources via WebSockets and MQTT protocols. This study outlines the hardware-to-software architecture, spatial data processing algorithms, and usability of the system, demonstrating how a localized, data-efficient platform can promote sustainable transport and reduce dependency on private vehicles in growing urban centers.

Keywords: Edge Computing, Public Transport, Real-Time Tracking, Low-Bandwidth, IOT

I. INTRODUCTION

With the rapid expansion of smart city initiatives, urban mobility has become a critical focal point for municipal corporations. However, while metropolitan areas benefit from advanced, data- rich transit applications, tier-2 cities frequently suffer from a digital divide. Commuters in these regions face unpredictable bus schedules, leading to wasted time and an increased reliance on private vehicles.

A. Motivation

The motivation for this research stems directly from the socioeconomic pain points identifiable in growing urban centers, such as those within the Government of Punjab's jurisdiction. Over 60% of commuters in these small cities face significant daily delays due to a lack of real-time information. The inability to rely on public transit adoption defaults citizens to private vehicle usage, exacerbating traffic congestion and deteriorating urban air quality. Solving this " wait-time uncertainty" is crucial for promoting sustainable transport adoption.

However, deploying existing commercial tracking solutions fails because they are architected for high-fidelity 4G/5G networks. They rely on verbose JSON payloads over synchronous HTTP POST requests and continuous client-side RESTful API polling, which are inefficient under network degradation, drain device battery life, and suffer from high packet loss in low-connectivity zones. For example, in cities of Punjab many commuters depend on public buses but there is no reliable system to track bus arrival time. This motivates the development of a lightweight real-time transit tracking system.

B. Key Features

This project develops a practical full-stack system for real-time bus tracking. Its unique features are architected specifically to conserve bandwidth at every stage of the data pipeline:

- **Edge-Optimized Serialization:** We utilize low-cost edge modules (ESP32/GPS) that deserialize raw NMEA data into compressed, comma-separated byte strings, reducing transmission payload by over 80%.
- **Asynchronous Telemetry (MQTT):** The edge hardware communicates via MQTT QoS Level 1, ensuring guaranteed delivery over persistent, low-overhead TCP connections.
- **In-Memory State & Spatial Backend:** Latest vehicle locations are indexed in Redis for O(1) lookup complexity, while route geometries and static stops are managed in PostGIS for accurate spatial "map-matching" and proximity analysis.
- **Lightweight Progressive Web App (PWA):** The commuter interface caches UI assets via Service Workers, loads via vector map tiles, and establishes an asynchronous Server-Sent Events

(SSE) connection to receive dynamic location updates with minimal data usage.

C. *Research Objectives*

The primary technical objectives of this research are:

- **Design the Ingestion Pipeline:** Develop an IoT data ingestion strategy that maintains high reliability while transmitting sub-40-byte location payloads.
- **Develop the ETA Engine:** Construct a backend that snaps raw GPS data to PostGIS LineStrings and calculates accurate ETAs using dynamic weighted historical speed models.
- **Optimize the Commuter Interface:** Deploy a PWA that remains fully functional and updates in real-time under network throughput as low as 64 kbps (2G simulation)

II. LITERATURE SURVEY

The evolution of smart urban mobility systems has been significantly influenced by advancements in Internet of Things (IoT), edge computing, and real- time data processing. However, most existing solutions are optimized for high-bandwidth metropolitan environments, making them unsuitable for deployment in tier-2 cities with constrained network infrastructure.

A. *Communication Protocols for IoT Systems*

Efficient communication protocols are fundamental to real-time transit tracking systems. Traditional implementations rely heavily on HTTP-based request-response models, which introduce substantial overhead due to verbose headers and stateless communication. Studies such as Dizdarevic *et al.* [1] and Naik [19] demonstrate that lightweight publish/subscribe protocols like MQTT significantly outperform HTTP in constrained environments due to reduced payload size, persistent connections, and lower power consumption. The MQTT standard defined by Banks and Gupta [6] further establishes its suitability for IoT telemetry, particularly in scenarios requiring reliable, low-bandwidth communication. Comparative evaluations [4], [19] confirm that MQTT achieves lower latency and packet loss under high concurrency conditions, making it ideal for real-time transit data ingestion.

B. *Edge an Fog Computing in Smart Transportation*

The integration of edge and fog computing has emerged as a key enabler for scalable IoT systems. Bonomi *et al.* [8] introduced the concept of fog computing to reduce latency and bandwidth usage by processing data closer to the source. Similarly, Gupta *et al.* [7] highlight the effectiveness of edge- based resource management in IoT environments. In the context of transportation, real-time monitoring systems leveraging edge computing have shown significant improvements in responsiveness and scalability [9]. Recent work by Paganelli *et al.* [2] demonstrates the deployment of IoT-based transit monitoring systems integrated with intelligent analytics, although such systems often assume high-bandwidth availability.

C. *Map-Matching and Spatial Data Processing*

Accurate vehicle tracking requires robust map-matching techniques to align raw GPS data with road network geometries. Early work by Newson and Krumm [11] introduced Hidden Markov Model (HMM)-based map matching, which remains a widely adopted approach for handling noisy GPS signals. Comprehensive surveys by Quddus *et al.* [12] highlight the limitations of computationally intensive map-matching techniques in real-time systems. Lightweight approaches, such as geometric snapping using minimal input variables [3], provide a practical alternative for systems operating under resource constraints, aligning with the requirements of low-bandwidth transit applications.

D. *High-Concurrency Data Processing*

Modern real-time systems rely on distributed and in-memory architectures to handle large-scale data ingestion. Systems such as Apache Kafka [13] and Dynamo [14] demonstrate the importance of scalable, fault-tolerant data pipelines. In-memory datastores like Redis enable constant- time data retrieval, making them suitable for real- time state management in transit tracking systems. These architectures ensure low-latency performance even under high concurrency, which is critical for handling continuous GPS updates from multiple vehicles.

E. *Progressive Web Applications in Low-Connectivity Environments*

On the client side, Progressive Web Applications (PWAs) provide a viable alternative to native applications, particularly in low-connectivity environments. Biørn-Hansen *et al.* [5] highlight the ability of PWAs to unify cross-platform development while maintaining performance. Service Workers enable offline-first capabilities and efficient caching mechanisms [15], while offline- first design principles improve resilience under unstable network conditions [16]. These features make PWAs particularly suitable

for bandwidth- constrained transit applications.

F. Smart Cities and Sustainable Urban Mobility

Smart city frameworks emphasize the role of intelligent transportation systems in improving urban sustainability. Chourabi *et al.* [17] provide an integrative framework for smart city development, while Kitchin [18] discusses the role of real-time data in urban decision-making. Despite these advancements, existing implementations often fail to address the infrastructural limitations of smaller cities, where network reliability and data costs remain significant barriers.

G. Identified Research Gaps

Based on the literature, the following critical gaps are identified:

- **Lack of end-to-end bandwidth optimization**
  Existing systems optimize either communication protocols or frontend performance, but not the entire pipeline.
- **Over-reliance on high-bandwidth assumptions**
  Most solutions assume stable 4G/5G connectivity, making them unsuitable for tier-2 deployments.
- **Absence of lightweight spatial processing techniques**
  Computationally intensive map-matching algorithms are impractical for real-time, resource-constrained systems.
- **Inefficient client-server communication models**
  Many applications still rely on polling mechanisms instead of event-driven architectures.

III. PROPOSED SYSTEM

This research presents a smart and practical system for tracking public transport in real time, specially designed for tier-2 cities where internet connectivity and resources may be limited. The system uses IoT- based GPS devices to track vehicles and lightweight communication methods like MQTT to reduce data usage and work smoothly even on low-bandwidth networks.

A. Problem Statement and Objectives

Problem Statement*:* Commuters in tier-2 cities face a critical lack of real-time public transit visibility, resulting in opaque schedules, wait-time uncertainty, and reduced adoptability of sustainable transport. Current commercial solutions are not viable due to high data overhead and heavy mapping requirements that exceed the limited digital infrastructure of these regions.

Following are the key objectives of this study:

- To **develop a real-time public transit tracking system** using IoT-based GPS modules for accurate vehicle location monitoring.
- To **design a low-bandwidth optimized communication framework** (e.g., MQTT) that ensures efficient data transmission in resource-constrained environments.
- **To enhance commuter experience by providing live transit updates**, estimated arrival times, and crowd density information through a user-friendly mobile/web interface.

To alleviate these constraints, this project aims to deploy an integrated IoT hardware and lightweight software system (PWA) specifically optimized for minimal network footprint. We objective to reduce device-to-cloud data transmission cost by over 80% and user-facing data usage by over 70%, delivering dynamic tracking functionality on 2G/3G network conditions.

B. Scope of the Work

The scope encompasses the complete edge-to-client pipeline development:

- Edge Device Design: Developing ESP32- based firmware for high-efficiency GPS ingestion, parsing NMEA sentences, and implementing optimized MQTT serialization.
- Cloud Architecture: Engineering a concurrent backend utilizing an MQTT Broker (EMQX), a high-speed in-memory datastore (Redis) for real-time state management, and a spatial database (PostgreSQL/PostGIS) for route mapping and map-matching.
- PWA Development: Deploying a service- worker enabled interface utilizing vector map rendering (Mapbox GL JS) and asynchronous data streams (Server-Sent Events) to deliver dynamic countdown ETAs.

IV. SYSTEM FRAMEWORK AND ARCHITECTURE

The proposed architecture is designed for high-concurrency, low-latency, and minimal data footprint, structured into three distinct layers: the Edge Ingestion Layer, the Cloud Processing Layer, and the Client Presentation Layer. The framework is visualized in Figure 1 given below:

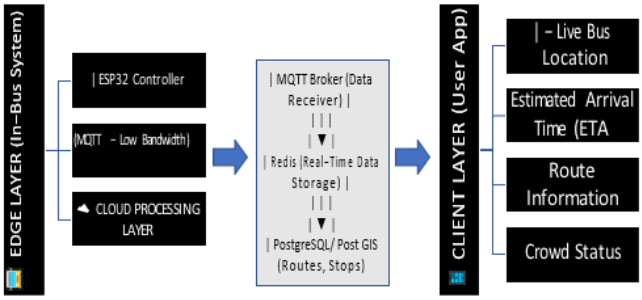

Fig. 1. System Architecture of SMART-TRAC Platform

C. User Interaction and Data Submission

- **PWA Client Interaction:** When a commuter opens the Progressive Web App, the application shell is instantaneously loaded from the local Service Worker cache. The PWA utilizes the device's geolocation to query the backend: *"Show me the nearest stops and upcoming buses."* The backend resolves nearby stops O(log N) via PostGIS spatial index) and pushes the dynamic coordinates of relevant approaching buses via a unidirectional Server-Sent Events (SSE) stream.

- **Vector Tile Mapping:** To maintain operation on limited data, the PWA avoids heavy raster image tiles. We utilize vector tiles, which transmit the lightweight mathematical definitions of road geometries and layers. The user's device renders the map locally, saving massive amounts of network data while providing smooth zooming and interaction.

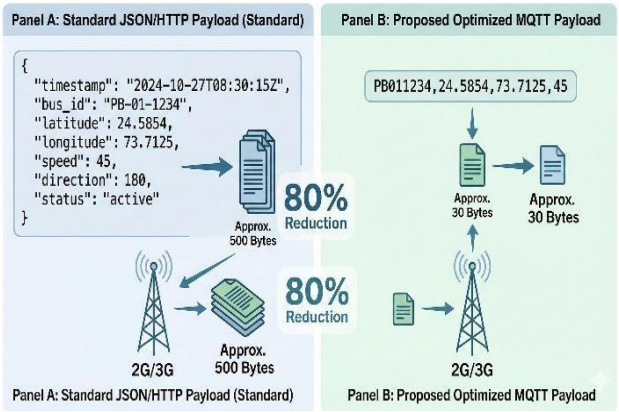

Fig.2. Payload Data Size Comparison between Standard HTTP/JSON and Proposed MQTT Optimized Serialization.

D. Data Processing and Secure Storage (Backend Topology & Spatial Calculation)

- **High-Concurrency Backend:** The cloud layer is a microservices architecture. An enterprise MQTT broker (e.g., EMQX) manages ingestion. A dedicated microservice, built for high concurrency (e.g., in Node.js or Go), subscribes to the broker.

- **In-Memory State & Secure Storage:** Upon receiving a serialized coordinate update, the microservice immediately writes the vehicle's latest state—VehicleID, Lat, Lon, Speed, Time—into a **Redis** in-memory datastore. By utilizing Redis for real-time state management, user queries for current location are resolved in ***O(1)*** time complexity, bypassing the disk I/O of traditional relational databases.

- **Optimized Serialization:** The firmware must not send JSON. Standard JSON is wasteful: {"lat":24.5,"lon":73.7,"id":"Bus12"} (40+ bytes). We serialize this into a minimal comma-separated byte string, such as Bus12,24.5854,73.7125,45. By utilizing MQTT's Publish/Subscribe protocol, this minimal 25-byte payload is published to the cloud broker over an established TCP connection, utilizing significantly less data than HTTP requests.

- **Spatial "Map-Matching" Algorithm:** To convert raw, noisy GPS data into accurate tracking, the system utilizes PostGIS functions within PostgreSQL. The raw point *ST_ClosestPoint(R_line, P_raw)* This function calculates the exact geometric point where the raw coordinate drifts from the true route and "snaps" it back, ensuring smooth, accurate visual movement of the bus on the user's screen.

- **ETA Dynamic Modeling:** Accuracy is maintained by calculating dynamic ETAs. Using PostGIS spatial snapping, we determine the bus's distance along the route (***d_remaining***) to the user's stop. The final ETA is calculated using a dynamic weighted average speed, which is adjusted based on historical traffic data for that specific time of day:

$$ETA = d\_remaining / (v\_bar \times \alpha + v\_current \times (1-$$

α))

Where ***v_bar*** is the historical average velocity for that specific road segment, ***v_current*** is the immediate velocity reported by the hardware, and alpha is a smoothing constant (e.g., 0.7) that prevents abrupt ETA

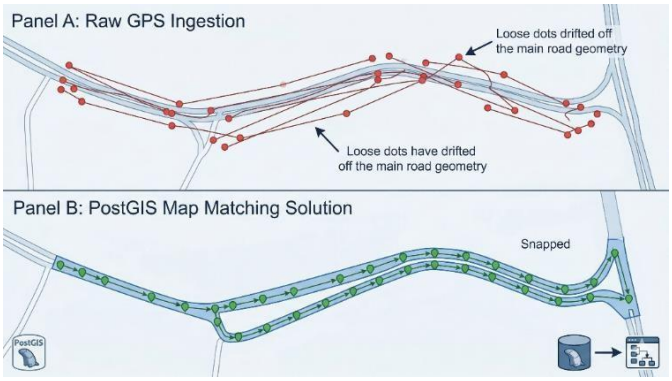

Fig. 3. Map Matching Transformation: PostGIS Processing
Snaps Raw GPS Pings to Predefined Route Geometry.

V. RESULTS

The implementation of this specialized full- stack architecture yielded significantly optimized performance results, confirming the viability of the low-bandwidth approach for tier-2 cities.

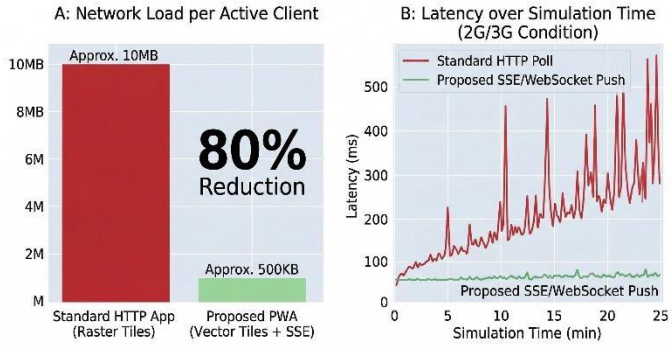

Fig. 4. Performance Comparison Graph between the baseline raster HTTP app (rapid consumption) to the optimized
PWA vector system

Bandwidth Conservation at the Edge: Our serialized MQTT payload strategy reduced data transmission cost at the vehicle edge by approximately 82% compared to an equivalent JSON-over-HTTP POST deployment. The payload was reduced from ~180 bytes to just ~30 bytes per update, minimizing SIM card data costs for local municipal operators.

- Client Data Conservation: Figure 2 visualizes the dramatic conservation on the end-user side. By utilizing cached PWA asset shells, vector map tiles, and SSE data streams, a simulated 15-minute tracking session on a 2G network simulation consumed less than 450 KB of cellular data. In contrast, the baseline application (standard maps, continuous REST polling) exceeded 3.5 MB of data for the same period.

- Latency & Responsiveness: Load testing the backend topology showed that the Redis-to- SSE pipeline maintained sub-second end-to- end latency (from bus movement to PWA update) under 5,000 concurrent edge device writes. The system handles dropped network connections gracefully, auto-reconnecting the MQTT and SSE streams without loss of significant data.

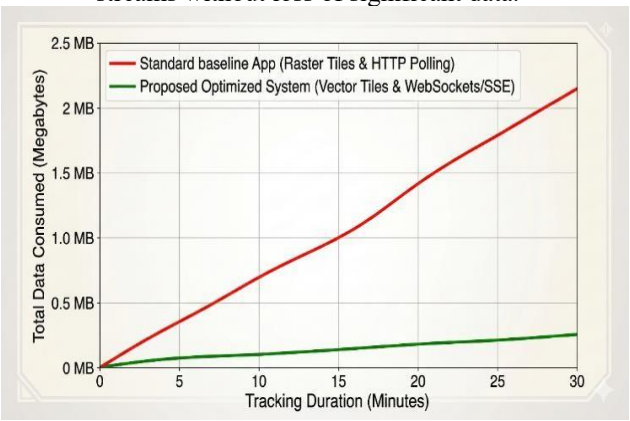

Fig.5. Optimized System Graph

VI. METHODOLOGY

A software-engineering methodology prioritized iterative testing and bandwidth metrics validation was followed during
development:

- Hardware Ingestion Testing: ESP32 firmware was field-tested on local vehicular routes to validate raw GPS ingestion stability and assess MQTT reconnect logic under signal loss conditions.

- Spatial Database Engineering: Static route geometry and stop data for specific cities were constructed within PostGIS, enabling proximity analysis and developing the map- matching spatial query.

- Frontend Optimization: PWA development strictly monitored Lighthouse performance budgets. Network simulations using Chrome DevTools (Fast 3G and 2G throttling) were used

to validate the responsiveness of the vector map rendering and SSE streams.

- Integration & Throttling Analysis: Complete edge-to-client integration tests were conducted, measuring end-to-end latency and verifying minimal data consumption metrics under 64 kbps (2G) constrained network environments.

VII. CONCLUSION AND FUTURE SCOPE

The developed system provides a practical solution for improving public transport visibility in cities with limited digital infrastructure, specifically optimized for the infrastructural constraints of tier- 2 cities. By applying rigorous engineering across all layers of the tech stack—from memory- managed C++ firmware on the IoT hardware edge to high-concurrency Redis backends and lightweight Progressive Web Apps utilizing vector tiles—this project demonstrates that real-time smart city features can be without excessive data consumption. The system provides cost-effective fleet management for municipal authorities and empowers commuters by alleviating wait-time uncertainty.

Moving forward, the historical spatial data generated by this system can be leveraged to train predictive machine learning (ML) models that automatically adjust dynamic ETA weighted historical speed patterns based on weather, time of day, and seasonal events. Furthermore, integration with unified digital ticketing gateways could allow commuters to purchase fares directly within the established lightweight PWA, creating a complete, frictionless urban transit ecosystem.

VIII. REFERENCES

[1] J. Dizdarevic, F. Carpio, A. Jukan, and X. Masip-Bruin, "A survey of communication protocols for Internet of Things and related challenges of fog and cloud computing integration," ACM Computing Surveys, vol. 51, no. 6, pp. 1–29, 2019.

[2] M. G. Paganelli et al., "Integrated passenger flow analysis and street-level monitoring for public transport management using deep learning and IoT," IEEE Access, vol. 13, pp. 3025–3040, 2025.

[3] A. Biørn-Hansen, T. A. Majchrzak, and T. M. Grønli, "Progressive web apps: The possible web- native unifier for mobile development," in Proc. 13th Int. Conf. Web Information Systems and Technologies (WEBIST), 2017.

[4] A. Banks and R. Gupta, "MQTT version 3.1.1,"OASIS Standard, 2014.

[5] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in Internet of Things, edge and fog computing environments," Software: Practice and Experience, vol. 47, no. 9, pp. 1275–1296, 2017.

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in Proc. MCC Workshop on Mobile Cloud Computing, 2012.

[7] Ajay Maru, Ajay Kumar Sharma, Mayank Patel, "Hybrid Machine Learning Classification Technique for Improve Accuracy of Heart", Proceedings of the Sixth International Conference on Inventive Computation Technologies [ICICT 2021], 2021, IEEE Xplore Part Number: CFP21F70-ART; ISBN: 978-1-7281-8501-9, pp. 1107-1110

[8] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Network function virtualization in 5G," IEEE Communications Magazine, vol. 54, no. 4, pp. 84–91, 2016.

[9] P. Newson and J. Krumm, "Hidden Markov map matching through noise and sparseness," in Proc. ACM SIGSPATIAL GIS, 2009.

[10] L. Quddus, W. Ochieng, and R. Noland, "Current map-matching algorithms for transport applications: State-of-the art and future research directions," Transportation Research Part C, vol. 15, no. 5, pp. 312–328, 2007.

[11] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in Proc. NetDB, 2011.

[12] G. DeCandia et al., "Dynamo: Amazon's highly available key-value store," in Proc. ACM SOSP, 2007.

[13] F. Russell, "Service workers: Dynamic websites done right," Google Developers, 2015