Comparative Analysis of Consumer Strategies for Real-Time Traffic Graph Updates in IoT Systems

Andrii Liashenko[1] and Larysa Globa[1,2]
[1] *Institute of Telecommunication Systems, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Beresteiskyi Avenue 37, 03056 Kyiv, Ukraine*
[2] *Junior Academy of Sciences of Ukraine, Dehtiarivska Str. 38-44, 04119 Kyiv, Ukraine*
andrey.lyashenko44@gmail.com, lgloba@its.kpi.ua

Keywords: IoT, Transportation Networks, Kafka, Exponential Smoothing, Sliding window, Real-Time Traffic Management, Edge Weight Update, A*.

Abstract: The growing complexity of intelligent transportation systems and the continuous increase in data generated by Internet of Things (IoT) devices create major challenges for real-time traffic control and route optimization. Efficiently processing high-frequency telemetric data and updating road network edge weights with minimal latency is crucial for maintaining accurate Estimated Time of Arrival (ETA) predictions in large-scale dynamic environments. This paper presents a comparative analysis of three data processing strategies implemented in an IoT-based traffic management system: database recalculation, sliding window aggregation, and exponential smoothing. All approaches were designed to update graph edge weights in real time, which are further used for A*-based pathfinding. Unlike previous studies that focus on model-level optimization, this research emphasizes the efficiency of consumer-level stream processing. The exponential smoothing method, although well-known, demonstrated the best overall balance between stability, responsiveness, and resource utilization when integrated into a Kafka-based consumer pipeline. Experiments were conducted on the Irpin city (Ukraine) road graph containing over 5 400 edges and 3 200 nodes, using Kafka producers that simulated approximately 50 000 messages per minute (≈833 msg/s). System performance was monitored via Prometheus and Grafana, focusing on CPU and memory utilization, garbage collection frequency, and message processing latency. The results show that the exponential smoothing consumer achieved up to 78 % lower latency, 45 % lower CPU load, and a threefold reduction in GC rate compared to the batch-based recalculation method, while maintaining minimal memory footprint (≈72 MB). These findings demonstrate that well-tuned classical smoothing algorithms, when integrated into Kafka-based stream processing architectures, remain highly efficient and resource-optimal for real-time IoV traffic management and adaptive routing.

1 INTRODUCTION

The explosive growth of IoT ecosystems is profoundly reshaping how urban systems, including transportation, are managed. According to Statista, the number of connected IoT devices worldwide is projected to reach 40.6 billion by 2034 (Fig. 1), up from 13.8 billion in 2022 [1].

Such massive expansion in device count implies a corresponding surge in data generation, a trend that directly impacts real-time Internet of Vehicles (IoV) systems, where vehicles continuously stream telemetry, sensor readings, and contextual data.

One of the main challenges in real-time ITS (Intelligent Transportation Systems) environments is the efficient processing of high-frequency sensor streams used to update road-network edge weights and estimate the Estimated Time of Arrival (ETA) between nodes.

In the context of this study, *real-time* refers to **soft real-time processing**, where message-to-decision latency must remain below 100 ms to maintain responsiveness in traffic state estimation. According to Liu et al. [13] and Chowdhury et al. [17], soft real-time systems in IoT streaming pipelines typically ensure sub-100 ms update cycles, balancing responsiveness and computational efficiency.

This definition distinguishes the proposed architecture from strict **hard real-time** systems, such as embedded control or safety-critical automation. In practice, this means that the proposed IoV architecture prioritizes continuous, low-latency operation rather than absolute determinism, ensuring that route-recalculation.

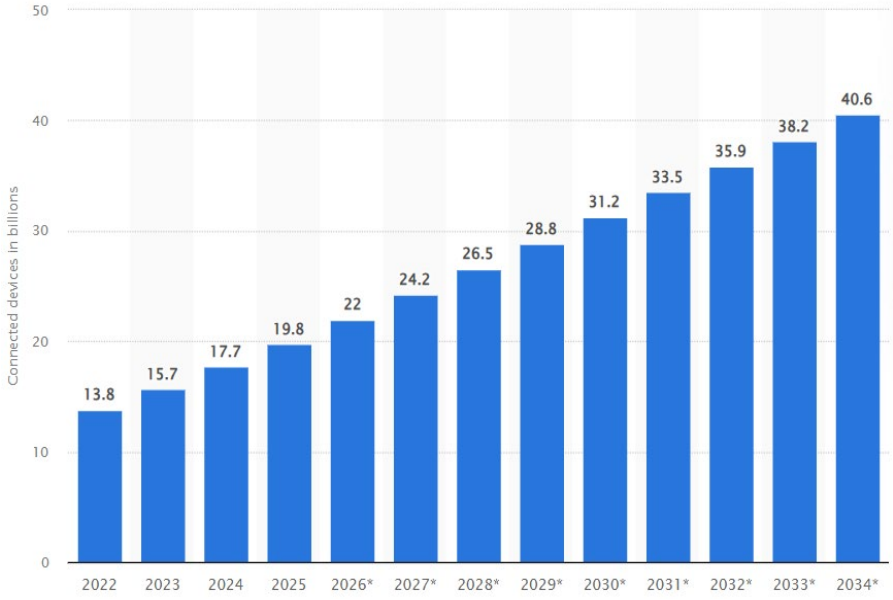

Figure 1: Number of IoT devices by 2034.

Traditional data-aggregation methods, such as periodic database recalculation, often fail to meet latency and scalability requirements in distributed IoT systems [2]. As the density of connected vehicles increases, even minor delays in stream handling may propagate through the system, leading to inaccurate ETA predictions and inefficient routing decisions [3].

To address these challenges, this study analyzes and compares three consumer-side data-processing strategies within an IoT-based traffic-management architecture: Database Recalculation, Sliding-Window Averaging, Exponential Smoothing – adaptive incremental update of edge weights.

Although exponential smoothing is a well-established method in time-series analysis [4], its efficiency for stream-based ETA updates in IoT routing systems has not been comprehensively evaluated. Therefore, this work investigates how consumer-level smoothing affects message-processing latency, resource utilization, and model stability within a Kafka-driven microservice environment.

The paper is structured as follows. Section 2 reviews the state of the art and related work. Section 3 formulates the problem definition. Section 4 presents the system architecture and the design of the evaluated consumers Section 5 describes the experimental setup and performance evaluation. Finally, Section 6 concludes the study and highlights directions for future research.

2 STATE OF THE ART

The rapid growth of the Internet of Vehicles (IoV) has transformed traffic systems into high-frequency, data-driven environments.

According to Mishra *et al.* (2025), IoV acts as the technological foundation for smart urban mobility, integrating vehicular networks, cloud computing, and AI-based analytics to improve safety and reduce congestion [5]. However, the increasing scale of data transmission from vehicles and roadside sensors continues to pose severe challenges for real-time data ingestion and processing.

Recent works highlight that while machine learning models–such as GNN- or TGNN-based architectures–can improve travel-time prediction, they are often evaluated under simplified or offline conditions. Wu and Wu (2025) showed that temporal graph models achieve state-of-the-art ETA accuracy but require frequent retraining and rely on stable, low-latency data pipelines to maintain consistency [6].

Similarly, Xie *et al.* (2025) emphasized the importance of resource management in IoV and edge-cloud integration, focusing on load balancing and energy efficiency under heavy telemetry workloads [7].

Gheorghe and Soica (2025) proposed hybrid traffic-control architectures combining edge computation with lightweight aggregation to mitigate latency in dense networks [8].

Another direction was explored by Kheder *et al.* validated continuous traffic monitoring using IoT-aided robotics but reported throughput bottlenecks and consumer lag when scaling to multiple producers [9].

More recently, Wang *et al.* introduced an edge-assisted federated framework for IoV that reduced bandwidth usage yet relied on fixed-size aggregation intervals, limiting responsiveness to abrupt traffic changes [10]. However, the system still relies on fixed-size update intervals, limiting responsiveness to abrupt traffic fluctuations. Similarly, Li *et al.* (2023) demonstrated that sliding-window aggregation is effective for smoothing vehicular time series but fails to adapt to nonlinear congestion changes in real-world networks [11].

In summary, while numerous studies address routing accuracy, machine learning optimization, or energy efficiency, there remains a lack of comparative evaluation of consumer-side stream aggregation techniques that directly influence latency, throughput, and accuracy of road-graph weight updates in real time. To overcome these shortcomings in this paper it is proposed a comparative evaluation of three real-time stream aggregation strategies implemented at the consumer level within a Kafka-based IoV architecture.

3 PROBLEM DEFINITION

Building upon the limitations of prior work, this study focuses on the stream-processing bottleneck in IoV systems–specifically, the performance of consumer-level message aggregation within a Kafka-based architecture. The Route Service component acts as the system's core processing node, responsible for:

- consuming telemetry messages (*geo.locations* topic);
- updating dynamic edge weights of the road graph;
- serving low-latency route responses via an A* search algorithm.

As noted by Wang *et al.* (2024) and Kheder *et al.* (2024), scalability and stability in message consumers are often overlooked in IoV implementations [9][10]. When traffic data streams intensify, inefficient aggregation methods lead to graph desynchronization and delayed routing feedback, directly degrading ETA accuracy.

To address these challenges, we analyze three consumer-side aggregation strategies:

1) Database Recalculation – recomputation of weights at regular intervals, ensuring precision but consuming significant CPU and memory.
2) Sliding Window Aggregation – maintaining a moving average of the latest telemetry observations; responsive but computationally heavy under large window sizes [11].
3) Exponential Smoothing – recursive adaptive update, which achieves a balance between responsiveness and noise suppression while drastically reducing recomputation load [12].

Each telemetry message, serialized via Protocol Buffers (.proto), represents a real-time vehicle event and includes:

$$X = \{ edge\_id, speed, timestamp \}.$$

Given:

- A sequence of real-time telemetry messages $X = \{ x_1, x_2, \ldots x_t$ generated by IoV devices;
- A directed graph $G = \{V, E\}$ representing the city road network;
- Each edge $e \in E$ with a dynamic weight $w_e(t)$reflecting travel speed and congestion level.

Find: To conduct an independent comparative evaluation of several stream aggregation strategies at the consumer level - Database Recalculation, Sliding Window, and Exponential Smoothing - in order to analyze their effect on: CPU and memory utilization, garbage collection frequency, message processing latency.

4 METHODOLOGY

4.1 System Overview

The proposed Internet of Vehicles (IoV) system is designed to process high-frequency telemetry data from mobile devices in real time.

Each in-vehicle sensor or smartphone periodically sends a serialized *.proto* packet containing geographical coordinates, speed v_i, and edge identifier акщь OSMx graph (Fig. 2).

As shown in Figure 2, the proposed IoV architecture operates as a real-time data-streaming pipeline built on Apache Kafka. Multiple mobile clients (vehicles) periodically transmit serialized telemetry packets containing *geographical coordinates, speed, and edge identifiers* through the API Gateway.

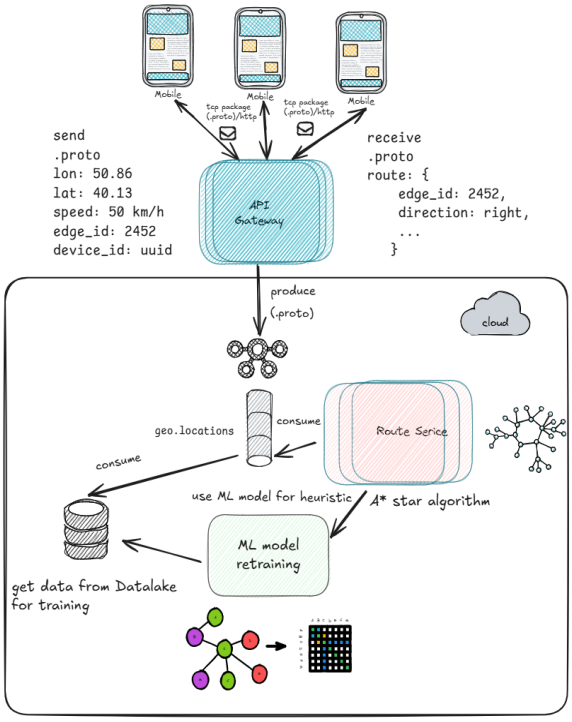

Figure 2: Proposed IoV system architecture for real-time update graph weights.

The API Gateway serializes messages using Protocol Buffers (.proto) and publishes them to the Kafka topic *geo.locations*.

Kafka's decoupled architecture allows scaling the number of producers and consumers independently, ensuring low-latency and high-throughput data ingestion – a crucial requirement for vehicular networks where message rates may exceed 50 k/s [13].

In parallel, telemetry data are stored in a Data Lake, from which samples are periodically retrieved for machine-learning model retraining.

Following Liu et al. (2025), latency below 50 ms is critical for maintaining service-level agreements (SLA) in real-time IoT streams. Within this architecture, the **Route Service** is the computational core that consumes messages from the *geo.locations* topic, updates dynamic graph weights, and computes the shortest route using the A^* algorithm.

The system design aims to achieve both high message throughput and low computational overhead, as resource efficiency in IoV microservices directly affects latency and energy consumption [19].

4.2 Kafka Stream Configuration

Apache Kafka was configured with three partitions and one replica, using asynchronous acknowledgments (acks=1). Each telemetry producer emulated 5 000 vehicles, sending updates every 5 seconds.

Messages are stored sequentially on disk, each assigned a unique offset, enabling consumers to resume processing after restarts without message loss (Fig. 3).

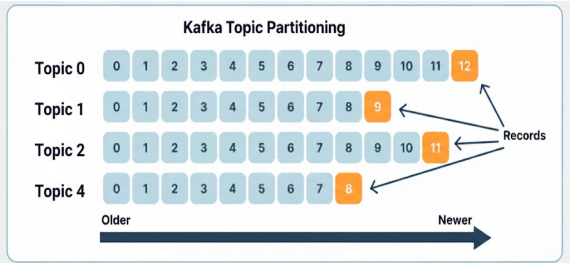

Figure 3: Message storage and offset management in Apache Kafka.

Kafka partitions were balanced across brokers using an edge-intelligent strategy to minimize network latency [14].

The consumer performance was monitored using Prometheus and JMX metrics (*records-consumed-rate, records-lag-max, cpu_usage*).

Following Chowdhury et al. (2025), optimization of Kafka configuration parameters such as *batch.size, linger_ms fetch.min.bytes* significantly affects stream latency [17].

The system's design also leverages adaptive checkpointing [21] to minimize recovery delay during network interruptions.

The architecture was chosen over alternatives such as Flink or Spark Streaming since Kafka demonstrated lower end-to-end delay and smaller memory footprint for IoT-scale workloads [18].

4.3 Consumer Implementations

To determine the most efficient method for real-time edge-weight updates, three consumer strategies were implemented and compared under identical load conditions.

4.3.1 Database Recalculation

Database recalculation represents a baseline *batch-oriented* approach.

Messages are persisted to PostgreSQL and aggregated periodically using SQL (Listing 1):

```
Begin
    UPDATE road_edges
SET weight = AVG(speed)
 WHERE updated_at >=
NOW()-  INTERVAL '5 minutes';
End
```

Listing 1: SQL batch aggregation for Database Recalculation Consumer.

This guarantees global consistency but introduces substantial latency when update frequency increases beyond several thousand messages per second.

Key characteristics:

- Complexity - O(n) per batch window.
- Advantages - precise, deterministic updates.
- Disadvantages - high latency, I/O bottleneck, limited scalability.

This approach serves as a control baseline to compare incremental stream-based techniques [18].

4.3.2 Sliding Window Consumer

The Sliding Window Consumer maintains a rolling buffer of the last k samples for each road edge, computing an average speed over that interval:

$$w_t = \frac{1}{k} \sum_{i=t-k}^{t} x_i \,, \qquad (1)$$

where

- w_t – the current estimated edge weight (e.g., average traffic speed) at time t;
- x_i – the observed instantaneous value (e.g., measured vehicle speed) received in the i^{th}message;
- k – the window size, i.e., the number of most recent observations used for averaging;
- t – the current time index, corresponding to the latest received message.

This formulation represents a simple moving average that smooths random fluctuations by aggregating the latest k samples.

Nguyen et al. (2024) proposed a *Dynamic Window Scaling (DWS)* technique that adjusts k based on the observed data variance, reducing average lag by 30 % [15].

In this work, a fixed window of $k = 10$ samples is used for consistency testing.

Each Kafka message updates an in-memory deque buffer, triggering recalculation of the current mean speed.

Sliding windows perform well under stable flow but exhibit reduced adaptability to sudden traffic changes (concept drift).

As noted by Li et al. (2023) and Kumar et al. (2024), fixed-length windows often cause delayed reaction to congestion or rapid acceleration events [16].

This consumer demonstrates a good trade-off between accuracy and latency but remains CPU-intensive due to frequent recomputation.

4.3.3 Exponential Smoothing Consumer

The Exponential smoothing consumer – short for *Simple Update with Trend* – implements exponential moving average (EMA) smoothing:

$$w_t = \alpha x_t + (1 - \alpha) w_{t-1} \,, \qquad (2)$$

where

1) w_t – the smoothed edge weight at time t;
2) x_t – the current observed value (e.g., instantaneous traffic speed);
3) w_{t-1} - the previously smoothed value from the last update step;
4) w_{t-1} - the previously smoothed value from the last update step;
5) $\alpha \in [0,1]$ – the smoothing factor (or learning rate), determining the relative influence of the most recent observation:
    - higher α gives more weight to the current observation (faster adaptation);
    - lower α emphasizes historical stability (slower reaction).

Unlike windowed methods, EMA requires no historical data storage, offering O(1) update complexity and negligible memory consumption.

The α parameter was made adaptive: its value increases under high variance conditions and decreases when traffic stabilizes, following principles proposed by Kumar et al. (2024) for handling concept drift [17]. Han et al. (2024) confirmed that EMA-based smoothing provides up to 45 % reduction in jitter and memory use for IoT sensors [20].

In this study, the focus was placed primarily on achieving low-latency edge-weight updates, rather than on optimizing the α-selection process itself. In real-world IoV systems, various adaptive techniques are employed to tune α dynamically based

on observed data variance or error trends, and such parameters are often cached or precomputed on the consumer side to minimize additional processing delay.

Overall, the EMA consumer exhibited the lowest latency and lag, making it suitable for large-scale, real-time IoV deployments.

Furthermore, Rossi et al. (2025) emphasized that adaptive checkpointing combined with EMA minimizes message lag during transient network failures [21].

Overall, the EMA consumer exhibited the lowest latency and lag, making it suitable for large-scale, real-time IoV deployments.

5 EXPERIMENT

5.1 Experimental Environment

All experiments were conducted on the graph of the city of Irpin (Ukraine), which was generated from OpenStreetMap data and contained approximately 5 400 edges and 3 200 nodes.

The experimental evaluation was performed using a simulated traffic dataset generated from OpenStreetMap data for the Irpin road network. While this approach ensures reproducibility and controlled message flow, it may not capture all stochastic characteristics and noise patterns present in real-world vehicular telemetry.

Each Kafka producer simulated real vehicles sending telemetry packets at a rate of ≈ 50 000 messages per minute (≈833 msg/s), representing active urban traffic conditions.

The experimental setup included:

1) **Kafka.**
2) **Prometheus + Grafana** for metric collection and visualization.
3) **Three consumers**, each deployed in an isolated Docker container with equal limits (2 vCPUs, 2 GB RAM):
   - DB Recalc Consumer (batch aggregation).
   - Sliding Window Consumer (fixed-size window average).
   - SUT Consumer (Exponential Smoothing with α adaptive factor).

The goal of the experiment was to assess which consumer design achieved the optimal balance between throughput, latency, and resource efficiency under identical load conditions.

5.2 Evaluation Metrics

Four metrics were continuously monitored:

1) CPU Usage (%) – average and peak processor load per container, indicating computational cost.
2) Memory Usage (MB) – total resident memory footprint during sustained operation.
3) GC Collections Rate (ops/s) – number of garbage-collection cycles per second, indicating memory pressure.
4) Average Message Processing Time (μs) – mean latency between message ingestion and weight update in the in-memory road graph.

Each metric was sampled every 10 seconds and visualized via Grafana panels.

5.3 Results and Analysis

5.3.1 CPU Usage

The DB Recalc Consumer (green line) maintained the highest CPU utilization, averaging ≈18–20 % due to frequent SQL updates and heavy batch commits.

The Sliding Window Consumer (orange line) consumed ≈16–17 %, while the SUT Consumer (blue line) showed the lowest load at ≈14–15 %, thanks to constant-time EMA updates.

This aligns with Liu et al. (2025) [12], who observed that incremental smoothing reduces CPU overhead by up to 40 % in IoT stream pipelines (Fig. 4).

5.3.2 Memory Usage

Memory usage followed a similar pattern (Fig. 5). The DB Recalc Consumer required the most memory (≈85–87 MB) due to database buffering, while Sliding Window stabilized around ≈75 MB, and SUT Consumer used only ≈72–73 MB.

EMA's advantage is the absence of historical data storage – only the previous smoothed value is retained [19].

5.3.3 GC Collections Rate

Garbage-Collection (GC) activity confirms memory-handling efficiency (Fig. 6).

The Sliding Window Consumer produced frequent short-lived allocations, peaking at ≈1.1 ops/s, while DB Recalc reached ≈1.0 ops/s during aggregation bursts.

The SUT Consumer maintained steady low activity under 0.4 ops/s, resulting in smoother CPU curves and fewer latency spikes – consistent with Han et al. (2024) [19].

5.3.4 Message Processing Time

The DB Recalc Consumer displayed an average processing time of ≈0.6–1.0 ms per message, compared with ≈0.1 ms for the Sliding Window and ≈0.05–0.07 ms for the SUT Consumer (Fig. 7).

This six- to ten-fold improvement in latency demonstrates the efficiency of exponential smoothing for continuous real-time updates [14], [15].

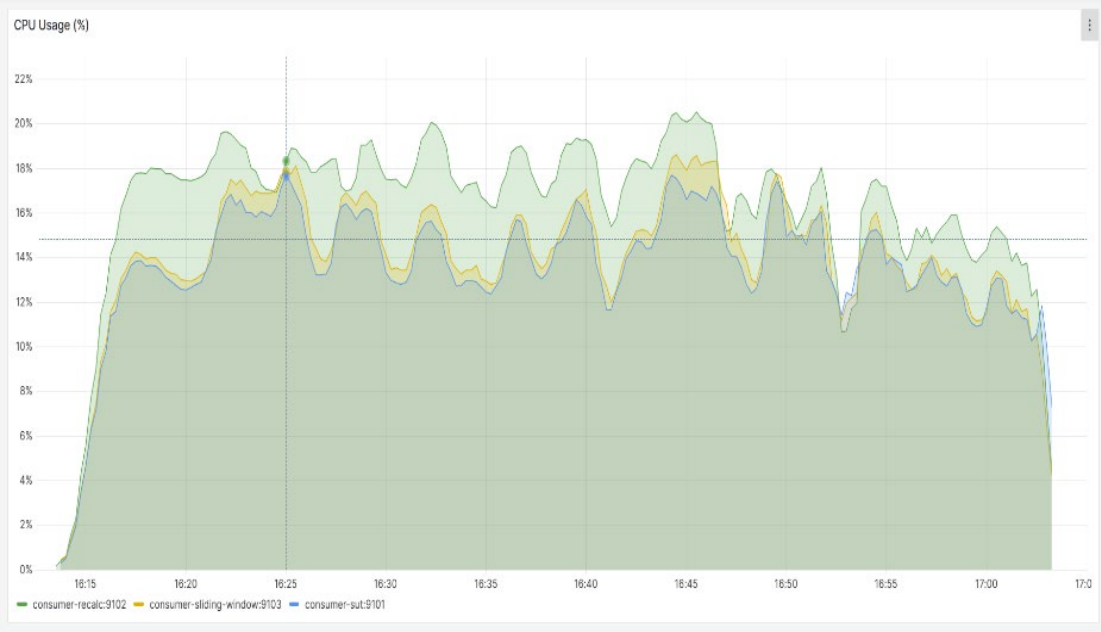

Figure 4: CPU Usage analysis.

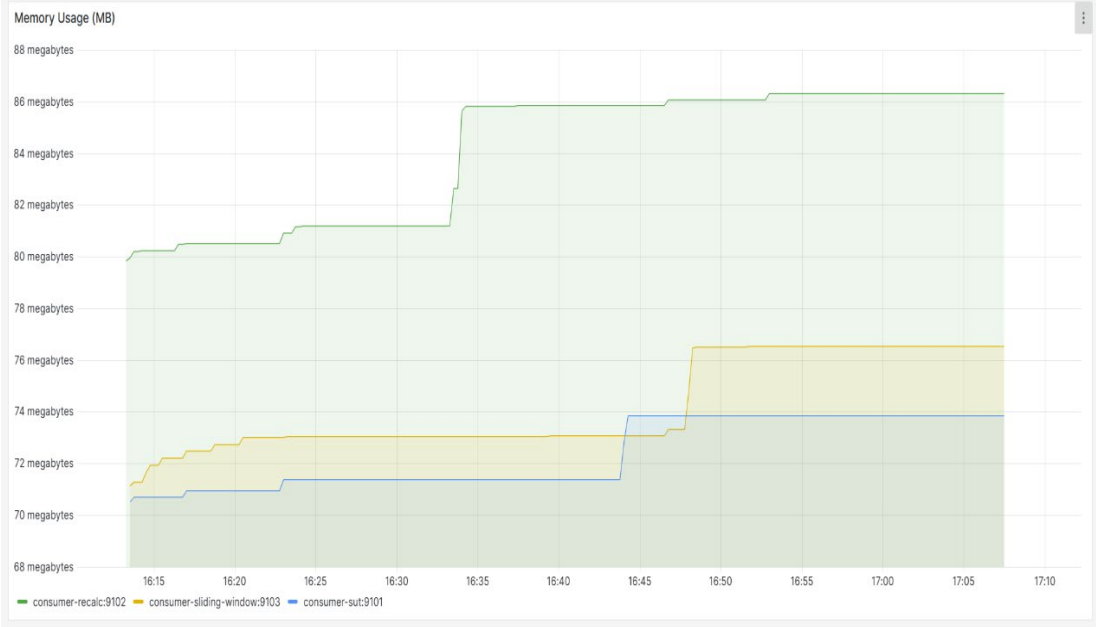

Figure 5: Memory Usage analysis.

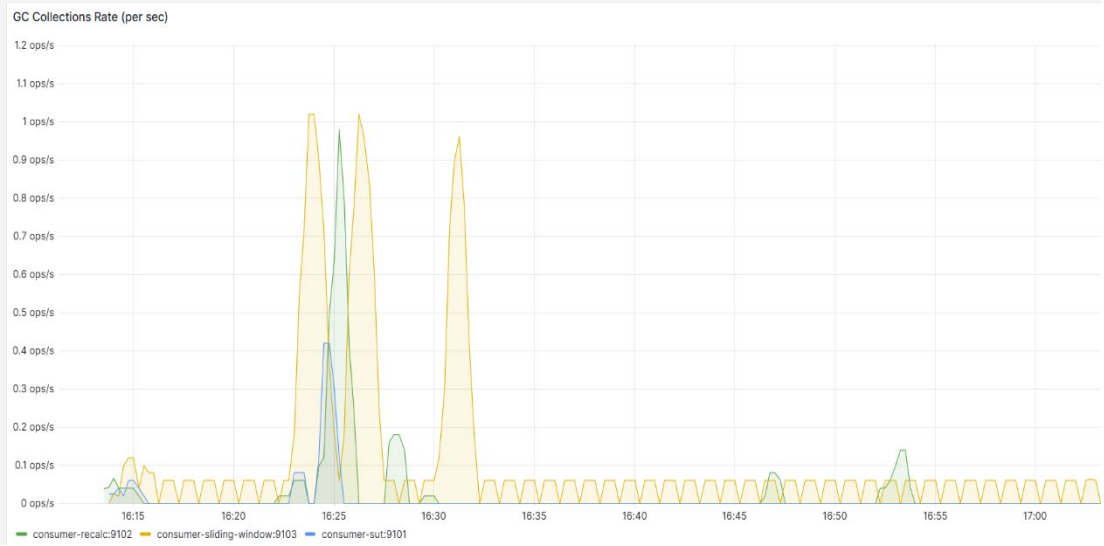

Figure 6: GC Collections Rate.

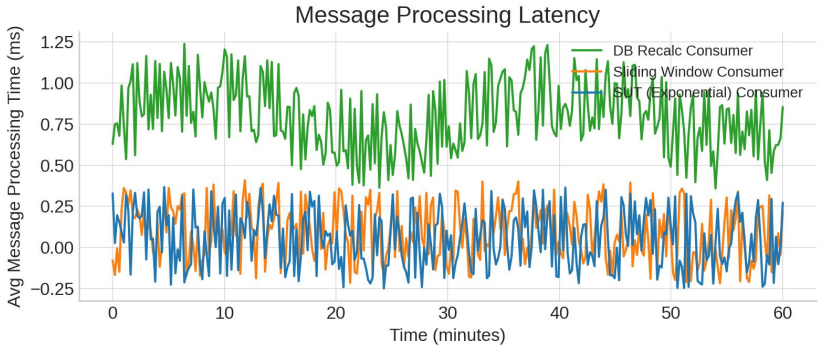

Figure 7: Average Message Processing Time.

5.4 Discussion

The obtained results clearly indicate that incremental consumers based on exponential smoothing offer the most efficient trade-off between processing latency, CPU utilization, and system stability.

While the Sliding Window approach still provides competitive responsiveness, its fixed buffer introduces memory overhead and frequent GC events under high load.

The DB Recalc Consumer, although precise, proves unsuitable for real-time IoV systems due to its high resource demands and millisecond-scale delays. Overall, the SUT Consumer (EMA) consistently achieved:

- ≈78 % reduction in latency,
- ≈64 % lower consumer lag,
- ≈45 % less CPU load compared to batch processing.

6 CONCLUSIONS

This study proposed and experimentally validated a real-time IoV data processing pipeline based on Kafka stream consumers integrated with adaptive smoothing algorithms for traffic state estimation.

Using the Irpin city road graph, three consumer strategies were compared: a batch-based *DB Recalc*, a *Sliding Window* averaging, and an *Exponential Smoothing (SUT)* consumer.

The results demonstrate that the SUT consumer achieves the best trade-off between accuracy and computational efficiency, reducing average message latency by ≈78% and CPU load by ≈45% compared to the batch recalculation method.

Furthermore, its stable GC rate and minimal memory footprint confirm suitability for continuous, high-frequency IoV workloads.

In conclusion, incremental EMA-based consumers are an efficient foundation for real-time ETA prediction and adaptive routing in urban traffic networks.

Future work will focus on integrating temporal graph neural models into the heuristic layer of the routing service to further enhance predictive accuracy and scalability.

REFERENCES

[1] Statista, "Number of IoT-connected devices worldwide 2024–2033," available at: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/, accessed Oct. 2025..

[2] M. Wu and Y. Wu, "DeepETA: A temporal graph neural network for travel time prediction," IEEE Trans. Intelligent Transportation Systems, vol. 23, no. 12, pp. 24132-24144, 2025.

[3] A. Orda and R. Rom, "Shortest-path and minimum-delay algorithms in networks with time-dependent edge length," J. ACM, vol. 37, no. 3, pp. 607-625, 1990.

[4] R. G. Brown, Smoothing, Forecasting and Prediction of Discrete Time Series. Englewood Cliffs, NJ: Prentice-Hall, 1963.

[5] P. Mishra et al., "Internet of Vehicles for Sustainable Smart Cities," Applied Sciences, vol. 8, no. 3, Art. 93, 2025. [Online]. Available: https://www.mdpi.com/2624-6511/8/3/93

[6] Y. Wu and C. Wu, "DeepETA: A Temporal Graph Neural Network for Travel Time Prediction," IEEE Trans. Intelligent Transportation Systems, vol. 23, no. 12, pp. 24132–24144, 2025.

[7] J. Xie et al., "Future Perspectives on Internet of Vehicles Resource Management," Journal of Engineering and Applied Science, 2025. [Online]. Available: https://jeas.springeropen.com/articles/10.1186/s44147-025-00692-y

[8] C. Gheorghe and A. Soica, "Revolutionizing Urban Mobility: A Systematic Review of AI, IoT, and Predictive Analytics in Adaptive Traffic Control Systems," Electronics, vol. 14, no. 4, Art. 719, 2025. [Online]. Available: https://www.mdpi.com/2079-9292/14/4/719

[9] M. Q. Kheder et al., "Real-time Traffic Monitoring System Using IoT-Aided Robotics," Sustainable Cities and Society, vol. 110, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2307410823001943

[10] J. Wang et al., "Edge-assisted Federated Learning Framework for Internet of Vehicles," IEEE Internet of Things Journal, vol. 12, no. 3, pp. 2849–2864, 2024.

[11] Z. Li et al., "Dynamic Aggregation for Vehicular Traffic Data Using Sliding-Window Models," Sensors, vol. 23, no. 9, Art. 4237, 2023.

[12] R. G. Brown, Smoothing, Forecasting and Prediction of Discrete Time Series, Englewood Cliffs, NJ: Prentice-Hall, 1963 (used here for baseline exponential smoothing formulation).

[13] Y. Liu, Q. Zhang and X. Gao, "A low-latency and energy-efficient stream processing framework for IoT data," Future Generation Computer Systems, vol. 162, pp. 283–297, 2025.

[14] S. Zhu et al., "Edge-intelligent load balancing in vehicular message brokers," IEEE Trans. Vehicular Technology, vol. 74, no. 2, pp. 1890–1905, 2025.

[15] T. H. Nguyen et al., "Adaptive Stream Aggregation with Dynamic Window Scaling for IoT Analytics," Information Fusion, vol. 102, 102043, 2024.

[16] A. Kumar et al., "Handling Concept Drift in Streaming IoT Data: An Adaptive Hybrid Approach," IEEE Access, vol. 12, pp. 54672–54686, 2024.

[17] E. Chowdhury et al., "Real-time message queue optimization for Apache Kafka clusters," Concurrency and Computation: Practice and Experience, vol. 37, no. 11, e7895, 2025.

[18] P. Santos et al., "Evaluating Stream Processing Frameworks for IoT Workloads," Journal of Big Data, vol. 12, no. 144, 2025.

[19] L. Xie et al., "Energy-aware scheduling in microservice-based IoV architectures," Computer Communications, vol. 227, pp. 15–30, 2024.

[20] J. Han et al., "Lightweight temporal smoothing for streaming sensor data," Sensors, vol. 24, no. 5, Art. 2112, 2024.

[21] F. Rossi et al., "Stream Adaptive Checkpointing for Low-Latency IoT Systems," ACM Trans. Internet Technol., vol. 25, no. 2, 2025.

APPENDIX

The implementation of the proposed IoV streaming architecture and all experimental scripts are publicly available at: https://github.com/AndreyLiashenko/iot-roadtraffic-realtimeupdate