# City-Scale Real-Time Transit Tracking & ETA System: System Design Report

## 1. Introduction and Architectural Vision

### 1.1 Executive Summary

The implementation of a city-scale real-time transit tracking and Estimated Time of Arrival (ETA) system represents a critical infrastructure challenge in the domain of Intelligent Transportation Systems (ITS). As urban density increases and the demand for reliable public mobility grows, transit agencies face the dual pressure of operational efficiency and passenger satisfaction. This report delineates a comprehensive architectural blueprint for a system capable of ingesting high-frequency telemetry from a fleet of 20,000+ transit vehicles, processing this data in real-time to resolve geospatial ambiguities, calculating dynamic arrival predictions, and disseminating standardized GTFS-Realtime feeds to downstream consumers with sub-second latency.

The proposed architecture adopts a reactive, Event-Driven Architecture (EDA) centered on a log-centric backbone. This design choice is driven by the necessity to decouple high-velocity write workloads—generated by sensors reporting at 1Hz frequencies—from the high-variability read patterns of millions of concurrent commuters.1 By leveraging a polyglot persistence strategy, the system places data in stores optimized for specific access patterns: in-memory geospatial indexes for instantaneous "near me" queries, wide-column stores for immutable time-series history, and relational databases for static topology management.2 Furthermore, the system integrates advanced machine learning models for ETA prediction, moving beyond naive distance-based calculations to incorporate historical traffic patterns, dwell time variance, and real-time congestion metrics.4

This document serves as the definitive technical reference for the system, covering requirements engineering, component selection, data modeling, security protocols, and operational resilience strategies. It is intended for systems architects, engineering leads, and stakeholders involved in the deployment of mission-critical urban transit infrastructure.

### 1.2 System Context and Evolution

Historically, Automatic Vehicle Location (AVL) systems relied on polling-based architectures where central servers would query vehicles for their status. This model suffers from significant scalability limits and high latency, often resulting in "ghost bus" phenomena where vehicle positions on user maps lag behind reality by 30–60 seconds.6 Modern expectations, driven by consumer apps like Uber and Citymapper, demand real-time fluidity.4 The shift to IoT-native protocols such as MQTT and the adoption of streaming platforms like Apache Kafka allow for a push-based model where latency is minimized to the network transit time plus processing overhead, typically under three seconds.7

The system operates within a complex ecosystem. It interfaces upstream with On-Board Units (OBUs) via cellular networks (4G/5G/LoRaWAN) and downstream with public APIs, digital signage controllers, and third-party data aggregators (Google Maps, Apple Maps) via GTFS-Realtime feeds.8 The architecture must essentially function as a high-throughput transformation engine, converting noisy, raw GPS sensor data into clean, semantic transit events (e.g., "Vehicle 1024 arrived at Stop B").10

## 2. Requirements Engineering

Designing for city-scale operations requires a rigorous definition of requirements to handle the "Thundering Herd" scenarios typical of rush hours, where both data ingestion rates and user query volumes peak simultaneously.

### 2.1 Functional Requirements (FR)

The functional requirements define the specific behaviors and functions the system must support.

#### FR-01: Telemetry Ingestion

The system must support the concurrent connection of up to 50,000 transit vehicles (buses, trams, trains). Each vehicle will transmit telemetry packets containing at minimum: global unique identifier (vehicle\_id), WGS84 coordinates (latitude, longitude), speed, bearing, and timestamp.11 The ingestion layer must support industry-standard NMEA 0183 sentences as well as structured JSON or Protobuf payloads.12 To accommodate legacy hardware, the system must implement adapter patterns to normalize varying data formats into a canonical internal schema.10

#### FR-02: Real-Time Map Matching

Raw GPS data is inherently noisy due to urban canyon effects, multipath signals, and atmospheric interference. The system must map-match raw coordinates to the underlying road network graph defined in the GTFS static dataset.13 This process must distinguish between topologically adjacent but semantically distinct paths, such as a highway overpass versus the service road beneath it, ensuring that vehicles are not placed on incorrect routes.14

#### FR-03: Dynamic ETA Calculation

The system must calculate Estimated Times of Arrival (ETAs) for all remaining stops on a vehicle's current trip. This calculation must not rely solely on static schedule data. Instead, it must utilize a hybrid algorithm that considers the vehicle's current location, historical travel times for the specific time of day and day of week, and real-time traffic conditions derived from fleet-wide speeds.15 The prediction engine must update these estimates with every new position report.17

#### FR-04: GTFS-Realtime Dissemination

The system must expose standard GTFS-Realtime feeds, specifically TripUpdate, VehiclePosition, and Alert entities. These feeds must be serialized using Protocol Buffers to ensure compatibility with the Google Transit ecosystem and minimize bandwidth usage for consumers.18 The system must support both full-dataset snapshots and differential updates if required by specific client contracts.20

#### FR-05: Service Alert Integration

Operators must have the ability to inject service alerts (e.g., "Detour due to construction") into the system. These alerts must be propagated immediately to the ETA engine to adjust predictions and to the client-facing APIs to inform passengers.9 The association of alerts with specific routes, trips, or stops must follow the FeedEntity linkage defined in the GTFS-RT spec.9

### 2.2 Non-Functional Requirements (NFR)

The non-functional requirements dictate the system's operational qualities and constraints.

#### NFR-01: Latency

The end-to-end latency—defined as the time elapsed from a GPS event occurring at the vehicle to that event being reflected in the API response—must be strictly controlled. For a real-time experience, the target is **< 3 seconds** for the 95th percentile (p95) of events.21

* **Ingestion Latency:** < 100ms
* **Processing Latency (Map Matching + ETA):** < 500ms
* **Propagation Latency:** < 500ms
* **API Response Time:** < 100ms for read requests.22

#### NFR-02: Throughput and Scalability

The architecture must support a sustained ingestion rate of **50,000 events per second (EPS)**. This figure is derived from a fleet of 20,000 vehicles reporting at 1Hz, plus a safety margin for retransmissions and burst traffic.23 The processing layer must scale horizontally to handle this load without degradation. The database layer must support write throughputs commensurate with this ingestion rate while simultaneously serving read traffic.

#### NFR-03: Availability and Reliability

The system is critical infrastructure. It requires **99.99% availability** (approx. 52 minutes of downtime per year maximum). The architecture must be resilient to the failure of any single component or Availability Zone (AZ). It must employ self-healing mechanisms such as consumer group rebalancing in Kafka and automatic failover in database clusters.24

#### NFR-04: Data Freshness and Accuracy

Data presented to users must be fresh. If a vehicle has not reported a position within **90 seconds**, it should be flagged as "stale" or removed from the real-time predictions to prevent "ghost bus" scenarios.17 Map matching accuracy must exceed 95%, ensuring that vehicles are snapped to the correct road segment even in complex intersections.26

#### NFR-05: Security

The system must implement a Zero Trust security model. All vehicle connections must be authenticated via Mutual TLS (mTLS) to prevent unauthorized data injection or spoofing.27 API access must be secured via OAuth 2.0 and API keys, with strict rate limiting to prevent Denial of Service (DoS) attacks.29

### 2.3 Service Level Objectives (SLOs)

To quantify reliability and performance, the following Service Level Objectives are defined:

| **Metric** | **SLO Target** | **Measurement Method** |
| --- | --- | --- |
| **Ingestion Success Rate** | 99.99% | Percentage of HTTP/MQTT requests responding with 2xx/Ack. |
| **End-to-End Latency** | 95% < 3s | Time difference between event\_timestamp (sensor) and processing\_timestamp (DB commit). |
| **Map Matching Accuracy** | > 98% | Percentage of points successfully snapped to a valid graph edge. |
| **API Availability** | 99.95% | Percentage of API requests returning non-5xx errors.30 |
| **ETA Accuracy** | +/- 2 mins | Deviation between predicted arrival and actual arrival for 90% of trips.31 |

## 3. High-Level Architecture Design

The architecture follows the **Event Sourcing** and **Command Query Responsibility Segregation (CQRS)** patterns. This separation enables independent scaling of the write path (ingestion/processing) and the read path (APIs/Dashboards), which is essential given the asymmetric load profiles: write load is constant and predictable based on fleet size, while read load is highly volatile and spike-prone.32

### 3.1 Architectural Components Overview

The system is composed of five distinct layers, connected by a unified streaming backbone:

1. **Ingestion Layer (Edge):** Handles connectivity with mobile assets. It terminates SSL, authenticates devices, and normalizes payloads into a canonical Protobuf format.
2. **Streaming Backbone (Data Plane):** A distributed log (Apache Kafka) that acts as the single source of truth for all events. It buffers data and decouples producers from consumers.34
3. **Processing Layer (Compute):** Stateful stream processors (Apache Flink) that consume raw events, maintain the state of every vehicle and trip, and execute business logic (Map Matching, ETA).
4. **Persistence Layer (Storage):** A polyglot data layer comprising:
   * **Redis:** For low-latency real-time state and geospatial queries.
   * **ScyllaDB:** For high-throughput write-heavy telemetry archival.
   * **PostgreSQL/PostGIS:** For static schedule data and complex spatial relations.
5. **Service Layer (API):** Microservices that expose data to the outside world, adhering to the GTFS-Realtime specification.

### 3.2 Protocol Decisions and Data Flow

The data flow is unidirectional and reactive:

1. **Vehicle $\rightarrow$ Gateway:** The vehicle sends a telemetry packet via MQTT/TLS.
2. **Gateway $\rightarrow$ Kafka:** The gateway validates the packet and publishes it to the telemetry.raw topic.
3. **Kafka $\rightarrow$ Flink:** The Map Matching job consumes the raw point, snaps it to the road graph, and produces to telemetry.matched.
4. **Flink $\rightarrow$ Flink:** The ETA job consumes the matched point, calculates predictions, and emits trip.updates.
5. **Flink $\rightarrow$ Storage:** Sinks write the final state to Redis (for current view) and ScyllaDB (for history).
6. **API $\rightarrow$ User:** The user requests vehicle locations; the API fetches from Redis and returns the response.

### 3.3 Technology Stack Selection Justification

| **Component** | **Technology** | **Rationale** |
| --- | --- | --- |
| **Language** | **Go (Golang)** | Selected for the Ingestion Gateway and API services due to its high concurrency primitives (goroutines), low memory footprint, and efficient execution model compared to Java, which is critical for handling 50k+ concurrent connections.35 |
| **Stream Processing** | **Apache Flink (Java)** | Flink provides superior state management and "exactly-once" processing semantics compared to Spark Streaming. While Go is used for services, the maturity of Java libraries for Flink and Kafka dictates its use in the processing layer.37 |
| **Message Bus** | **Apache Kafka** | Chosen over Pulsar for its widespread industry adoption, mature ecosystem, and proven ability to handle massive throughput with low latency (5ms at p99). Kafka's log-centric storage is ideal for the replayability required to debug ETA algorithms.39 |
| **Hot Storage** | **Redis (Geospatial)** | Provides built-in geospatial indexing (GEOADD, GEORADIUS) with O(log(N)) complexity, enabling sub-millisecond retrieval of "buses near me".41 |
| **Warm Storage** | **ScyllaDB** | A drop-in replacement for Cassandra written in C++. It offers significantly higher throughput per node and lower tail latency by avoiding JVM garbage collection pauses, which is crucial for write-heavy telemetry workloads.2 |

## 4. Ingestion Layer: Handling the Flood

The Ingestion Layer is the system's front door. Its primary responsibilities are connectivity management, protocol translation, and security enforcement.

### 4.1 Protocol Strategy: MQTT vs. HTTP

For high-frequency telemetry, **MQTT 5.0** is selected as the primary protocol, with HTTP/2 as a fallback.

* **Bandwidth Efficiency:** MQTT's binary header overhead is minimal (2 bytes minimum) compared to HTTP's verbose headers. For a fleet of 20,000 vehicles, this reduction translates to gigabytes of data saved daily, significantly lowering cellular operational costs.7
* **Connection Lifecycle:** MQTT supports "Keep-Alive" and "Last Will and Testament" (LWT). If a vehicle enters a tunnel and the TCP connection drops, the broker detects the timeout and can immediately publish a "Connection Lost" event. HTTP is stateless and requires polling or timeout logic on the server side to detect disconnection.
* **QoS Levels:** We utilize **QoS 1 (At Least Once)** to ensure that location updates are delivered even if the network is flaky. The application layer handles deduplication via unique message IDs.7

### 4.2 Gateway Architecture

The Ingestion Gateway is a cluster of stateless Go microservices behind a Network Load Balancer (NLB).

* **mTLS Termination:** The gateway terminates the TLS connection, validating the client certificate against the internal CA.27
* **Payload Normalization:** It accepts various input formats (e.g., JSON, proprietary binary, NMEA sentences) and converts them into a standardized internal Protobuf schema (TelemetryEvent). This decouples the internal system from device-specific formats.10
* **Rate Limiting:** To protect the message bus, the gateway implements a token bucket rate limiter per vehicle\_id, rejecting bursts exceeding the configured threshold (e.g., > 5 Hz).29

### 4.3 Adapting NMEA Data

Many legacy transit vehicles output data in the NMEA 0183 format (e.g., $GPRMC sentences). The gateway implements a parsing logic to extract relevant fields:

* $GPRMC contains time, latitude, longitude, speed, and track angle.
* The gateway parses these strings, converts coordinates from Degrees-Minutes (DMM) to Decimal Degrees (DD), and serializes the result to the Protobuf schema.12

## 5. The Data Plane: Event Streaming and Processing

The core of the system is the data processing pipeline, responsible for turning raw coordinates into actionable transit information.

### 5.1 Kafka Topic Taxonomy and Partitioning

Effective Kafka usage requires a thoughtful topic design. We separate data stages into distinct topics to allow for inspection and replay.

| **Topic** | **Partition Key** | **Retention** | **Purpose** |
| --- | --- | --- | --- |
| telemetry.raw | vehicle\_id | 7 Days | Validated, normalized ingress data. Partitioning by vehicle\_id guarantees time-ordering for each vehicle.34 |
| telemetry.matched | vehicle\_id | 3 Days | Map-matched locations with segment IDs and snapped coordinates. |
| trip.updates | route\_id | 1 Day | Calculated ETAs and delays. Partitioning by route\_id groups all vehicles on the same line. |
| service.alerts | alert\_id | Compacted | Manual or automated system alerts (e.g., detours). |

**Partitioning Strategy:** We use vehicle\_id as the partition key for telemetry topics. This ensures that all updates from a single bus are processed by the same consumer instance in Flink, enabling the use of local state (e.g., "previous location") without external database lookups, maximizing performance.34

### 5.2 Stream Processing with Apache Flink

Apache Flink is utilized for its advanced stateful processing capabilities. The processing pipeline consists of two primary jobs:

#### Job 1: Map Matching and Trip Assignment

This job consumes telemetry.raw.

* **Map Matching:** It uses a Hidden Markov Model (HMM) or geometric projection to snap the GPS point to the road network. The static road network is loaded into Flink's managed state or queried from a side-loaded broadcast state (from PostGIS).14 The algorithm considers the vehicle's trajectory (bearing) and the connectivity of road segments to rule out impossible jumps (e.g., moving from a bridge to a river road instantly).
* **Trip Assignment:** The job correlates the matched location with the GTFS schedule. It determines which "Trip" the vehicle is currently serving. If the vehicle is geographically active but has no assigned trip, it is flagged as ADDED or unscheduled.17

#### Job 2: ETA Calculation Engine

This job consumes telemetry.matched.

* **State:** It maintains the current status of every active trip in RocksDB (Flink's state backend).
* **Calculation:** For every update, it recalculates the arrival time for all downstream stops. It merges the vehicle's current delay (schedule deviation) with the typical travel time for the remaining segments.
* **Output:** It emits TripUpdate events to the trip.updates topic, which conform to the GTFS-RT structure (stop sequences, arrival/departure times).45

### 5.3 Handling Out-of-Order Data

In mobile networks, data often arrives out of order. Flink's **Event Time** processing and **Watermarks** are critical here. We define a watermark strategy (e.g., BoundedOutOfOrderness(10 seconds)). If a packet arrives 5 seconds late, it is processed correctly based on its timestamp. If it arrives 15 seconds late, it is considered "late" and can be discarded or sent to a side-output for analysis, ensuring that the real-time state doesn't jitter backward in time.37

## 6. The Persistence Layer: Polyglot Database Design

A single database technology cannot meet the diverse requirements of geospatial queries, time-series archiving, and relational schedule management. We employ a polyglot persistence strategy.

### 6.1 Real-Time State: Redis (Geospatial)

Redis serves as the "Hot" data store, powering the user-facing APIs.

* **Data Structure:** We use the GEO data type.
  + GEOADD transit:locations <lon> <lat> <vehicle\_id>: Stores the latest location.
  + GEORADIUS transit:locations <user\_lon> <user\_lat> 500 m: Returns all vehicles within 500 meters of a user.
  + HASH vehicle:{id}: Stores metadata like current\_trip\_id, occupancy\_status, next\_stop\_id.
* **Performance:** Redis operations are atomic and in-memory. This allows the API to serve "near me" queries in sub-millisecond time, which is impossible with disk-based databases under heavy load.41 The use of Geohashing internally allows for efficient proximity searches.

### 6.2 Historical Telemetry: ScyllaDB

ScyllaDB is used for the "Warm" and "Cold" storage of telemetry. It is optimized for high-throughput writes.

* **Schema Design:** The schema is designed to optimize for "Query by Vehicle by Time".  
  SQL  
  CREATE TABLE telemetry\_history (  
   vehicle\_id text,  
   bucket\_day text,  
   timestamp timestamp,  
   lat double,  
   lon double,  
   speed double,  
   trip\_id text,  
   PRIMARY KEY ((vehicle\_id, bucket\_day), timestamp)  
  ) WITH CLUSTERING ORDER BY (timestamp DESC)  
   AND compaction = {'class': 'TimeWindowCompactionStrategy',...};
* **Partitioning:** The partition key is (vehicle\_id, bucket\_day). This ensures that a single partition doesn't grow indefinitely. The TimeWindowCompactionStrategy is selected to optimize for time-series workloads where data is written sequentially and rarely updated, reducing write amplification and disk I/O.2
* **Comparison:** Unlike PostgreSQL, which would struggle with table bloat and vacuuming at 1.2 billion rows per day (20k vehicles \* 60 secs \* 60 mins \* 24 hours), ScyllaDB's LSM-tree architecture handles massive ingestion rates effortlessly.23

### 6.3 Static Schedule: PostgreSQL + PostGIS

PostgreSQL manages the relational complexity of the GTFS static data.

* **Data:** Routes, Stops, Trips, Calendars, Shapes.
* **Spatial Operations:** PostGIS is used for complex spatial joins that don't need to be real-time. For example, "Find all stops within Polygon X" or "Calculate the total length of Route Y".3
* **Partitioning:** Tables like stop\_times (which can be huge) are partitioned by feed\_version\_id to allow for easy swapping of schedule versions.49

## 7. The Intelligence Layer: ETA Algorithms

The accuracy of the ETA is the primary metric of success for passengers. The system implements a tiered approach to ETA calculation.

### 7.1 Tier 1: Schedule Deviation (Baseline)

This is the simplest method. The system compares the vehicle's current location to the scheduled time at the last passed stop.

* $Delay = T\_{actual} - T\_{scheduled}$.
* $ETA\_{next\\_stop} = T\_{scheduled\\_next} + Delay$.
* *Pros:* Computationally cheap. *Cons:* Assumes constant delay, ignores traffic changes ahead.

### 7.2 Tier 2: Segment-Based Historical Profiling

The road network is divided into "segments" (edges between stops). The system maintains a historical profile of travel times for each segment, bucketed by 15-minute intervals and day of week (e.g., Monday 08:00-08:15).

* $ETA = \sum (\text{HistoricalTravelTime}(Segment\_i))$.
* This accounts for recurring traffic patterns (rush hour).15

### 7.3 Tier 3: Real-Time Machine Learning (Advanced)

For maximum accuracy, an ML model (e.g., XGBoost or LSTM) is employed.

* **Features:** Current delay, time of day, weather conditions (via external API), real-time traffic speed (from fleet aggregates), and headway (distance to the bus ahead).
* **Training:** The model is trained offline using the historical data stored in ScyllaDB.
* **Inference:** The Flink job calls the model (served via ONNX runtime or a sidecar) to generate predictions. This approach allows the system to predict "recovery" (catching up schedule) or "compounding delay" (getting stuck further).5

## 8. Service Layer: API and Dissemination

The output of the system must be consumable by various stakeholders. The API Gateway orchestrates this access.

### 8.1 GTFS-Realtime Feeds

The core output is the binary Protobuf feeds.

* **Snapshotting:** Instead of generating the feed on-the-fly for every request (which burns CPU), a background worker generates the vehicle\_positions.pb and trip\_updates.pb files every 5 seconds and uploads them to an S3 bucket / CDN edge.
* **Access:** Clients request the file via HTTP GET. The CDN serves the file, providing massive read scalability. Last-Modified headers are used to allow clients to skip downloading if the data hasn't changed.50

### 8.2 Developer API (REST/GraphQL)

For app developers needing specific data (e.g., "Get ETA for Stop 123"), a REST API is provided.

* **Endpoints:**
  + GET /vehicles?route=1: Returns live positions for a route.
  + GET /stops/{id}/predictions: Returns upcoming arrivals.
* **Rate Limiting:** To prevent abuse, we implement a **Token Bucket** algorithm backed by Redis. Authenticated users get higher limits (e.g., 1000 req/min) while anonymous users are throttled (20 req/min).29 The API returns standard headers: X-RateLimit-Limit, X-RateLimit-Remaining.

## 9. Security and Identity

Security is woven into every layer, from the edge to the cloud.

### 9.1 Threat Model: GPS Spoofing

A significant threat in transit systems is GPS spoofing, where malicious actors transmit fake GPS signals to disrupt operations or create phantom vehicles.

* **Detection:** The Ingestion Gateway implements physics-based anomaly detection.
  + *Speed Constraints:* If a bus moves faster than its maximum mechanical speed (e.g., > 120 km/h) or teleports effectively between reports, the data is rejected.
  + *Tunneling:* If the GPS signal strength is perfect while the map shows the vehicle is in a tunnel, it is suspicious.51
  + *Consensus:* Comparing the GPS location with cellular tower triangulation data (if available from the modem) provides a secondary check.

### 9.2 Mutual TLS (mTLS) for IoT

To ensure that only authorized vehicles can write to the system, **mTLS** is enforced.

* **Provisioning:** During vehicle maintenance, a unique client certificate is generated and securely stored in the OBU's Hardware Security Module (HSM) or Trusted Platform Module (TPM).
* **Validation:** The Ingestion Gateway validates the certificate chain against the private CA. If a certificate is compromised, it is added to a Certificate Revocation List (CRL) or OCSP responder to block access immediately.27

### 9.3 API Security

* **OAuth 2.0:** Administrative access (e.g., creating Service Alerts) uses the OAuth 2.0 Authorization Code flow with strict scopes (alerts:write, gtfs:read).
* **Device Flow:** For embedded signage at bus stops (which have no keyboard), the **OAuth 2.0 Device Flow** is used to authorize the device securely without embedding permanent credentials in the firmware.53

## 10. Operational Resilience and Conclusion

### 10.1 Disaster Recovery (DR)

The system is designed for resilience.

* **RPO (Recovery Point Objective):** The target is near-zero for real-time data (since it flows constantly) and < 1 hour for historical archives.
* **RTO (Recovery Time Objective):** < 4 hours.
* **Strategy:** An Active-Passive deployment across two regions. Kafka MirrorMaker 2 replicates the telemetry.raw topic to the secondary region. If the primary region fails, the Flink cluster in the secondary region is spun up to consume the replicated topic, and the API Gateway DNS is flipped.55

### 10.2 Observability

* **Distributed Tracing:** OpenTelemetry is used to trace the path of a datapoint from the Gateway $\rightarrow$ Kafka $\rightarrow$ Flink $\rightarrow$ Redis. This allows engineers to pinpoint exactly where latency is introduced.57
* **Metrics:** Prometheus scrapes metrics from all components. Key alerts include "Consumer Lag > 10s" (indicating processing is falling behind) and "Ingestion Error Rate > 1%".59

### 10.3 Conclusion

The "City-Scale Real-Time Transit Tracking & ETA System" described herein provides a robust, scalable, and secure foundation for modern urban mobility. By decoupling ingestion from processing through an Event-Driven Architecture, utilizing a polyglot persistence layer to match data storage to access patterns, and implementing rigorous security and operational standards, this system is capable of handling the rigorous demands of a smart city environment. It transforms chaotic, high-velocity sensor data into ordered, predictable, and valuable passenger information, ultimately improving the transit experience for millions of users.

*End of Report*

#### Works cited

1. The Power of Event-Driven Architecture: How Netflix and Uber Handle Billions of Events Daily - Ankur Verma, accessed on November 30, 2025, <https://developerport.medium.com/the-power-of-event-driven-architecture-how-netflix-and-uber-handle-billions-of-events-daily-0a2d09d7308c>
2. ScyllaDB vs. Apache Cassandra, accessed on November 30, 2025, <https://www.scylladb.com/compare/scylladb-vs-apache-cassandra/>
3. PostgreSQL + PostGIS + TimescaleDB - storage for transport monitoring systems (Ivan Muratov) | PgConf.Russia 2019, accessed on November 30, 2025, <https://pgconf.ru/en/talk/1588614>
4. Frequently Asked Questions - Citymapper, accessed on November 30, 2025, <https://citymapper.com/news/141/frequently-asked-questions>
5. Reinforcement Learning for Modeling Marketplace Balance | Uber Blog, accessed on November 30, 2025, <https://www.uber.com/blog/reinforcement-learning-for-modeling-marketplace-balance/>
6. Building a Real-Time Bus Tracking Data Display System - ISCAP Conference, accessed on November 30, 2025, <https://iscap.us/proceedings/conisar/2011/pdf/1824.pdf>
7. Track Transportation Assets in Real Time with Apache Kafka and Kafka Streams, accessed on November 30, 2025, <https://www.confluent.io/blog/real-time-asset-tracking-software-with-kafka-elasticsearch-kibana/>
8. Prototype of a System for Tracking Transit Service Based on IoV, ITS, and Machine Learning, accessed on November 30, 2025, <https://www.mdpi.com/2032-6653/14/9/261>
9. GTFS Realtime Reference, accessed on November 30, 2025, <https://gtfs.org/documentation/realtime/reference/>
10. Bus GPSsing - Casey Watts, accessed on November 30, 2025, <https://www.caseywatts.com/blog/bus-gpssing/>
11. Software Architecture for New Mobility Services - Telemetry Data - Grape Up, accessed on November 30, 2025, <https://grapeup.com/blog/how-to-build-software-architecture-for-new-mobility-services-gathering-telemetry-data/>
12. NMEAconverter :: OpenCPN - GitHub Pages, accessed on November 30, 2025, <https://opencpn-manuals.github.io/main/nmea_converter/index.html>
13. Map-matching Algorithm for Large Databases | The Journal of Navigation | Cambridge Core, accessed on November 30, 2025, <https://www.cambridge.org/core/journals/journal-of-navigation/article/mapmatching-algorithm-for-large-databases/1145FEDB112A53F18A5A80551D2C8956>
14. A map matching algorithm based on modified hidden Markov model considering time series dependency over larger time span - NIH, accessed on November 30, 2025, <https://pmc.ncbi.nlm.nih.gov/articles/PMC10637971/>
15. Estimation of arrival times (ETA) - ContPark, accessed on November 30, 2025, <https://contpark.com/estimation-of-arrival-times-eta/>
16. What makes a great ETA calculation? - Shippeo, accessed on November 30, 2025, <https://www.shippeo.com/blog/what-makes-a-great-eta>
17. GTFS Realtime Best Practices, accessed on November 30, 2025, <https://gtfs.org/documentation/realtime/realtime-best-practices/>
18. GTFS Realtime Bindings, accessed on November 30, 2025, <https://gtfs.org/documentation/realtime/language-bindings/overview/>
19. GTFS Realtime Overview - Transit - Google for Developers, accessed on November 30, 2025, <https://developers.google.com/transit/gtfs-realtime>
20. GTFS Realtime - Vehicle Positions - Metro Train - Victorian Government Data Directory, accessed on November 30, 2025, <https://discover.data.vic.gov.au/dataset/gtfs-realtime/resource/e2158e21-e6cb-4611-919f-90117b36a610>
21. Design a Real-Time Traffic Analysis System: From Zero to Hero | Coudo AI Blog, accessed on November 30, 2025, <https://www.coudo.ai/blog/design-a-real-time-traffic-analysis-system-from-zero-to-hero>
22. Identifying and Addressing API Latency Issues - Odown, accessed on November 30, 2025, <https://odown.com/blog/api-latency/>
23. What's the best database to store large amounts of GPS tracking data? : r/geospatial - Reddit, accessed on November 30, 2025, <https://www.reddit.com/r/geospatial/comments/1bo42bm/whats_the_best_database_to_store_large_amounts_of/>
24. Building a Real-Time Architecture: 8 Key Considerations - RisingWave, accessed on November 30, 2025, <https://risingwave.com/blog/building-a-real-time-architecture-8-key-considerations/>
25. Achieving relentless Kafka reliability at scale with the Streaming Platform | Datadog, accessed on November 30, 2025, <https://www.datadoghq.com/blog/engineering/streaming-platform-kafka-custom-abstractions/>
26. DMM: Fast Map Matching for Cellular Data - Get Started with OpenScholar - University of California, Merced, accessed on November 30, 2025, <https://sites.ucmerced.edu/files/wdu/files/20mobicom-dmm.pdf>
27. Transportation IoT Security Solutions - DigiCert, accessed on November 30, 2025, <https://www.digicert.com/iot/iot-device-manager/iot-of-transportation>
28. Identity onboarding and lifecycle management - AWS Documentation, accessed on November 30, 2025, <https://docs.aws.amazon.com/whitepapers/latest/designing-next-generation-vehicle-communication-aws-iot/identity-onboarding-and-lifecycle-management.html>
29. 10 Best Practices for API Rate Limiting in 2025 | Zuplo Learning Center, accessed on November 30, 2025, <https://zuplo.com/learning-center/10-best-practices-for-api-rate-limiting-in-2025>
30. SRE SLOs: Defining SLAs, SLIs, and SLOs in SRE | NetApp, accessed on November 30, 2025, <https://www.netapp.com/learn/cvo-blg-sre-slos-defining-slas-slis-and-slos-in-sre/>
31. Estimated Time of Arrival: Accuracy, Methods and AI Tools - Techstack, accessed on November 30, 2025, <https://tech-stack.com/blog/estimated-time-of-arrival/>
32. CQRS Pattern - Azure Architecture Center | Microsoft Learn, accessed on November 30, 2025, <https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs>
33. Understanding Event Sourcing and CQRS Pattern | Mia-Platform, accessed on November 30, 2025, <https://mia-platform.eu/blog/understanding-event-sourcing-and-cqrs-pattern/>
34. Documentation - Apache Kafka, accessed on November 30, 2025, <https://kafka.apache.org/documentation/>
35. Java outperforms Go in Apache Beam: Performance insights with Spark and Flink runners, accessed on November 30, 2025, <https://www.besidethepark.com/blog/big-data-batch-processing-with-apache-beam>
36. Using Golang Instead of Java for Fast Processing: A Performance Comparison - Medium, accessed on November 30, 2025, <https://medium.com/@kalpit.sharma/using-golang-instead-of-java-for-fast-processing-a-performance-comparison-69b63f411de8>
37. Flink vs Kafka Streams: A Complete Comparison - Confluent, accessed on November 30, 2025, <https://www.confluent.io/blog/apache-flink-apache-kafka-streams-comparison-guideline-users/>
38. What is it better java over GoLang? - Reddit, accessed on November 30, 2025, <https://www.reddit.com/r/java/comments/1framad/what_is_it_better_java_over_golang/>
39. Kafka vs. Pulsar vs. RabbitMQ: Performance, Architecture, and Features Compared, accessed on November 30, 2025, <https://www.confluent.io/kafka-vs-pulsar/>
40. Kafka vs Pulsar: Choosing the Right Stream Processing Platform - RisingWave, accessed on November 30, 2025, <https://risingwave.com/blog/kafka-vs-pulsar-choosing-the-right-stream-processing-platform/>
41. Geospatial queries in Redis - Memurai, accessed on November 30, 2025, <https://www.memurai.com/blog/geospatial-queries-in-redis>
42. Which solution should I pick for scalable frequent read / write of public geolocation data over HTTP? - Stack Overflow, accessed on November 30, 2025, <https://stackoverflow.com/questions/59525903/which-solution-should-i-pick-for-scalable-frequent-read-write-of-public-geoloc>
43. AWS c3.2xlarge Performance Benchmark - ScyllaDB, accessed on November 30, 2025, <https://www.scylladb.com/product/benchmarks/aws-c3-2xlarge-benchmark/>
44. Real-Time Scan-to-Map Matching Localization System Based on Lightweight Pre-Built Occupancy High-Definition Map - MDPI, accessed on November 30, 2025, <https://www.mdpi.com/2072-4292/15/3/595>
45. Protobuf - General Transit Feed Specification, accessed on November 30, 2025, <https://gtfs.org/documentation/realtime/proto/>
46. Apache Flink™ vs Apache Kafka™ Streams vs Apache Spark™ Structured Streaming — Comparing Stream Processing Engines - Onehouse.ai, accessed on November 30, 2025, <https://www.onehouse.ai/blog/apache-spark-structured-streaming-vs-apache-flink-vs-apache-kafka-streams-comparing-stream-processing-engines>
47. TimescaleDB for Vehicle Telematics - IIoT - TigerData Community Forum, accessed on November 30, 2025, <https://forum.tigerdata.com/forum/t/timescaledb-for-vehicle-telematics/153>
48. PostGIS Performance Showdown: Geometry vs. Geography | by Jacob Baskin | Coord, accessed on November 30, 2025, <https://medium.com/coord/postgis-performance-showdown-geometry-vs-geography-ec99967da4f0>
49. Chapter 4. Data Management - PostGIS, accessed on November 30, 2025, <https://postgis.net/docs/using_postgis_dbmanagement.html>
50. Best Practices | V3 API - MBTA, accessed on November 30, 2025, <https://www.mbta.com/developers/v3-api/best-practices>
51. GPS Spoofing Attack Detection in Autonomous Vehicles Using Adaptive DBSCAN - arXiv, accessed on November 30, 2025, <https://arxiv.org/html/2510.10766v1>
52. GPS Spoofing Attack Detection on Intersection Movement Assist using One-Class Classification, accessed on November 30, 2025, <https://www.ndss-symposium.org/wp-content/uploads/2023/02/vehiclesec2023-23038-paper.pdf>
53. OAuth 2.0 Device Flow for IoT Integration - Salesforce Help, accessed on November 30, 2025, <https://help.salesforce.com/s/articleView?id=xcloud.remoteaccess_oauth_device_flow.htm&language=en_US&type=5>
54. OAuth2 Device Flow-proxy Using Free Azure Components - Xebia, accessed on November 30, 2025, <https://xebia.com/blog/oauth2-device-flow-proxy-using-free-azure-components/>
55. RTO (Recovery Time Objective) and RPO (Recovery Point Objective) | Explore - Commvault, accessed on November 30, 2025, <https://www.commvault.com/explore/rto-rpo>
56. Disaster Recovery Planning and Data Backup for Information Systems and Services | Office of the VPIT-CIO | University of Michigan, accessed on November 30, 2025, <https://it.umich.edu/information-technology-policies/general-policies/DS-12>
57. Importance of Traces in Event-Driven Architecture and Benefits of the OpenTelemetry Operator | by Taffarel de Lima Oliveira | Medium, accessed on November 30, 2025, <https://medium.com/@taffarelo3/importance-of-traces-in-event-driven-architecture-and-benefits-of-the-opentelemetry-operator-771b73276aa8>
58. Distributed Tracing and OpenTelemetry for Event-Driven Integration - Solace, accessed on November 30, 2025, <https://solace.com/blog/what-is-distributed-tracing-and-how-does-opentelemetry-work/>
59. Kafka Consumer Lag: Causes, Impacts, and Solutions - Groundcover, accessed on November 30, 2025, <https://www.groundcover.com/blog/kafka-slow-consumer>
60. Step-by-Step Guide to Monitoring Kafka Consumer Lag - RisingWave, accessed on November 30, 2025, <https://risingwave.com/blog/step-by-step-guide-to-monitoring-kafka-consumer-lag/>