Benchmarking context-aware services for smart-transportation IoT data exchange

Efstratios Ntallaris, Georgios Bouloukakis, Kostas Magoutis

Benchmarking Context-Aware Services for Smart-Transportation IoT Data Exchange

Efstratios Ntallaris[1,3], Georgios Bouloukakis[2], Kostas Magoutis[1,3]
entallaris@ics.forth.gr,georgios.bouloukakis@telecom-sudparis.eu,magoutis@csd.uoc.gr
[1]Institute of Computer Science (ICS), Foundation for Research and Technology - Hellas (FORTH), Greece
[2]Télécom SudParis, Institut Polytechnique de Paris, France
[3]Computer Science Department, University of Crete, Greece

ABSTRACT

In the fast-growing realm of smart cities, integrating Internet of Things (IoT) devices into transportation systems is essential for improving efficiency and safety. Deploying these systems in real-world settings demands access to contextual data, and middleware systems to facilitate the exchange of both contextual and IoT data. Existing IoT-based data exchange systems such as Orion-LD, Stellio and Scorpio in the FIWARE space, support the modeling and representation of both context and IoT systems. This paper introduces a comprehensive testbed and a benchmarking platform designed to evaluate the performance of FIWARE context-aware brokers. The testbed incorporates real data from a real Bus Transportation Service in the city of Ioannina, Greece, as well as synthetic data enabling a realistic assessment of query and ingestion performance.

The results show that microservices-based architectures like Stellio and Scorpio scale better than traditional designs like Orion-LD under high loads, but all brokers perform similarly at low loads. Furthermore, temporal queries present challenges for IoT applications due to their high cost across all evaluated brokers. However, write-optimized data stores offer an advantage by improving ingestion speed. The paper emphasizes the importance of understanding and addressing the operational inefficiencies of context-aware brokers to improve IoT system performance. Overall, this work introduces a novel benchmarking platform for smart transportation systems, featuring a realistic testbed with both real and synthetic IoT datasets, as well as detailed experimental results that identify key performance bottlenecks and offer potential optimization strategies.

CCS CONCEPTS

• **Computing methodologies** → **Modeling methodologies**; • **Information systems** → **World Wide Web**; • **Computer systems organization** → *Embedded and cyber-physical systems.*

KEYWORDS

Smart Transportation Systems, IoT, Testbed, Benchmark, Context Brokers

ACM Reference Format:
Efstratios Ntallaris[1,3], Georgios Bouloukakis[2], Kostas Magoutis[1,3]. 2024. Benchmarking Context-Aware Services for Smart-Transportation IoT Data Exchange. In *14th International Conference on the Internet of Things (IoT 2024), November 19–22, 2024, Oulu, Finland*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3703790.3703802

IoT 2024, November 19–22, 2024, Oulu, Finland

ACM ISBN 979-8-4007-1285-2/24/11.
https://doi.org/10.1145/3703790.3703802

1 INTRODUCTION

In today's fast-evolving technological landscape, smart environments are enriched with Internet of Things (IoT) devices and applications that aim to enhance various aspects of our lives. These IoT-enhanced systems are key to developing sustainable, efficient cities by optimizing energy use, resource management, transportation, and more. The demand for smart transportation is particularly high due to the complexity of traffic systems and the risks posed by autonomous driving [32]. IoT plays a crucial role in this field by enabling real-time data collection, analysis, and decision-making across devices, vehicles, and infrastructure [31, 38]. However, managing the large volume, high velocity, and diverse nature of IoT data presents a significant challenge for traditional data management approaches due to the real-time processing requirements, as well as the dynamic nature of smart transportation systems [26].

Context-aware message brokers are state-of-the-art solutions for enabling dynamic and adaptive decision-making in smart city applications, including traffic management systems [25]. Such brokers serve as intermediate components between IoT devices and applications to collect, manage, and disseminate data along with context information via a standardized mechanism (APIs and protocols). Leveraging contextual information allows for the delivery of more relevant services to end-users, improving their overall experience. Context-aware brokers widely used in our study include Scorpio [13], Stellio [14], and Orion-LD [12]. All facilitate seamless integration and interoperability between IoT devices and applications by utilizing the Next Generation Service Interface - Linked Data (NGSI-LD) [10] protocol.

When designing and implementing an IoT system with context brokers, it is essential to account for various overheads, including database execution, architectural complexity, protocol communication, and scalability needs. Understanding such operational inefficiencies requires addressing various challenges in IoT-enhanced transportation systems. In particular, the frequent querying and updating of context information can lead to performance bottlenecks and

increased latency, which is even more pronounced when executing temporal queries. In addition, applications for transportation systems may require near real-time data, which can be difficult to deliver due to the inherent limitations of the underlying data storage and processing mechanisms.

Existing benchmarking solutions [5, 15, 22, 28] typically focus on limited aspects of IoT workloads, such as ingestion rates, streaming data processing, or time-series data performance. Moreover, existing FIWARE-based benchmarks [4, 20, 24] rely on synthetic data, they lack a focus on smart transportation systems and, they do not fully utilize the range of NGSI-LD queries.

This paper introduces a testbed that incorporates: (i) real-time data from the Bus Service of the city of Ioannina; (ii) real data from existing datasets one having traffic related data in Aarhus, Denmark, over six months [35]; and (iii) a dataset that is collected from 54 sensors deployed at the Intel Berkeley Research Lab [34]. We then present a benchmarking platform with a comprehensive set of standard spatial and temporal queries that represent all the functionalities related to fetching and storing data in IoT-enhanced transportation systems. This benchmark assesses the diverse functionalities associated with data retrieval and management. Given that context brokers leverage different databases for various purposes, the benchmark provides thorough comparative analysis and evaluation for the full range of context broker capabilities.

The key contributions of this work include:

- A Smart City Bus Benchmark (SCBenchmark) for transportation systems that enables the execution of test queries in diverse database systems.
- A testbed for a real Bus Transportation Service in the city of Ioannina, Greece, that leverages multiple context-aware message brokers for IoT data collection and querying.
- A comparative analysis and evaluation of FIWARE-based context brokers using the proposed SCBenchmark and a wide range of test queries.

Experimental results show that in most cases, the predominant source of overhead comes from Time-Series databases.

The remainder of this paper is organized as follows: Section 2 presents background information on NGSI-LD context-aware brokers discussing their architectures and types of underlying databases used. In Section 3, we introduce a Smart City Bus Benchmark (SCBenchmark) used to evaluate three open-source FIWARE context brokers using bus-transportation data collected from the city of Ioannina. Finally, Section 4 discusses related work, and Section 5 concludes the paper and describes future work.

2 BACKGROUND

This section provides an overview of the NGSI-LD protocol utilized in context-aware message brokers and a summary of the underlying database technologies of these brokers.

NSGI-LD Protocol. NGSI-LD [10] serves as the standard protocol for developing context-aware applications, particularly tailored for handling context information within smart cities initiatives, encompassing sectors such as healthcare and transportation. A key feature of NGSI-LD is its ability to support linked data and semantic interoperability, facilitating seamless information sharing and utilization across systems and organizations. End-users or applications can submit queries using HTTP REST requests compliant with the NGSI-LD standard. This work focuses on the optimization of temporal query execution in context-aware message brokers, and thus describes the steps necessary to execute a query in underlying NGSI-LD-based database systems.

Fig. 1 depicts an NGSI-LD request sent to the context broker in step 1. NGSI-LD queries are next translated into database queries (step 2), and then executed to retrieve the necessary information. Following the retrieval of data from the context-aware brokers (step 3), it undergoes transformation back into NGSI-LD format (step 4) before being delivered as contextual information to the user or application.

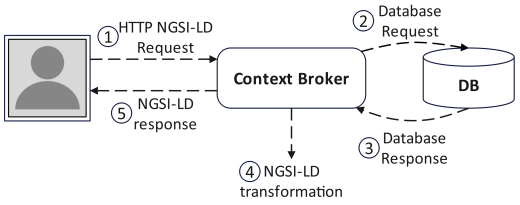


Figure 1: Query execution in NGSI-LD context brokers.

Context-aware Brokers. The selection of the underlying databases as well as the architectural design of context brokers can impact the performance of queries. State-of-the-art brokers rely on one or more databases to enhance performance. When a Context Broker receives a client request, it determines the most suitable underlying database to handle the query. For example, Orion-LD supports integration with Mintaka [11], a component from the FIWARE ecosystem which gives the ability to retrieve and store temporal data. Mintaka exploits the TimescaleDB [1], a relational database optimized to handle time-series workloads. Orion-LD utilizes MongoDB as its underlying database. Scorpio uses PostGIS [30], an extention of PostgreSQL that is optimized for geospatial data and queries. Stellio uses TimescaleDB to store temporal data, and PostGIS to store data related with geospatial data. Both Sellio and Scorpio use Apache Kafka [27] in their architecture for fault tolerance support. On the other hand, Orion-LD lacks of fault tolerance.

To ensure fault tolerance in Orion-LD, it is essential to introduce a scheduler, as well as a cluster of Orion-LD context brokers and MongoDB databases. Such a setup increases potential "writes" to disk and thus the performance overheads. Some brokers do not directly store the NGSI-LD entities in the underlying databases. Instead, they perform transformations to maintain backward compatibility with the rest of the system (e.g Orion-LD NGSI-v2). Therefore Step 4 of Fig. 1 is taking place in Orion-LD to transform an entity fetched from the database to NGSI-LD. Context brokers are in this way akin to simple polystore middleware [9].

3 THE SCBENCHMARK

This section introduces the ***Smart City Bus Benchmark (SCBenchmark)***, which includes: (i) a realistic testbed with context brokers for collecting real-time data of buses operating in the city of Ioannina; (ii) representative smart city bus queries to be used as test queries in SCBenchmark; and (iii) the SCBenchmark experimental evaluation. The SCBenchmark is available as open source at https://github.com/satrai-lab/scbenchmark

3.1 The SCB Testbed

To evaluate the feasibility of building IoT-enhanced applications for transportation systems, it is essential to realistically evaluate current context-aware message brokers. For this purpose, we design and implement the Smart City Bus (SCB) testbed. SCB leverages NGSI-LD for modeling static (e.g., bus stations) and dynamic IoT data (e.g., temperature over time), as well as its associated API [2] to provide a comprehensive experimental platform. Our goal is to examine the performance of context brokers using ***data querying and ingestion*** over three main NGSI-based context-brokers.

Smart transportation systems typically rely on the GTFS Standard [16], which facilitates the sharing of transportation schedules and related geographic data among transit agencies. However, GTFS has limited support for IoT-enhanced transportation systems. To demonstrate an IoT-enhanced transportation system realistically, we convert GTFS to NGSI-LD models and implement them using the Bus Service of Ioannina [7]. NGSI-LD enables us to create a data representation enriched with context and semantic information, facilitating deeper analysis and seamless integration with IoT devices and applications. Based on [6, 7], NGSI-LD modelling incorporates the *Bus Entity* that models information about the available areas within the bus, the IoT devices deployed on the bus, and the observations these devices monitor. Each IoT device is associated with every observation it tracks. Similarly, *Bus Station Entities* include context information about the areas within the bus station, the IoT devices deployed at the station, and the observations monitored by each IoT device.

This work utilizes all entities presented in [7] and associate them with specific IoT devices in areas shown in Fig. 2 for the bus service of Ioannina, ensuring that our testbed provides a realistic bus service environment. In particular, the bus service of Ioannina operates a fleet of 60 active buses. Fig. 2 shows a visualization of the bus Mercedes Citaro G (C 628.233-13), showcasing its bus areas and the corresponding sensor placements relative to the bus. For each bus, we track its GPS position along with 24 other IoT observations using various sensors. These include 8 seat sensors for occupancy and 1 sensor each for measuring pollution, humidity, temperature, fuel consumption, vibration, voltage, and current. This sensor distribution enables visualization that maps out and color-codes the areas where these sensors are located across buses, as depicted in Fig. 2.

The Bus Service of Ioannina operates with 428 active bus stations, each equipped with 5 sensors for data collection. These sensors include 3 Time Of Flight (ToF) sensors, 1 temperature sensor, and 1 humidity sensor. Fig. 3 presents a 2D visualization of a bus station, illustrating its layout and the placement of sensors relative to the station. In terms of data collection, we utilize a combination of real-world IoT data derived from real datasets and synthetic data.

Table 1: Sensor Observation Sources.

Sensor Observation	*Source*
Temperature	Replayed from Sensor [34]
Humidity	Replayed from Sensor [34]
Voltage	Replayed from Sensor [34]
Tire Pressure	Synthetic
Occupancy Seat	Synthetic
Fuel Consumption	Synthetic
Engine Temperature	Synthetic
Engine Vibrations	Synthetic
Time of Flight (ToF)	Synthetic
Real-Time Bus Locations	Ioannina city bus API

In particular, we utilize publicly available real-world traffic data from Aarhus, Denmark [35], which includes 21GB of detailed traffic information collected over six months, and 150MB sensor data (temperature, humidity, and voltage) from the Intel Berkeley Research Lab [34]. We integrate this publicly available real-world data into our testbed by combining them with synthetic data. For this, we introduce a generator that simulates dynamic traffic conditions and realistic sensor behavior. We use real-time traffic data, including average travel time, speed, and vehicle counts, derived from an existing dataset [35] within the CityPulse project [8]. Environmental data related to air pollution and geographic coordinates across urban areas is sourced from another dataset [29] from the same project. Meteorological measurements—humidity, pressure, temperature, wind direction, and speed—are obtained from yet another dataset [37], also part of the CityPulse project. To model bus operating conditions, we use parameters such as temperature, humidity, light, and voltage, as described in [34]. For the rest of the sensors shown in Figs. 2 and 3, we generate synthetic observations for buses and bus stations, including tire pressure, occupancy, fuel consumption, engine temperature/vibrations, and Time of Flight sensor data from the bus stations. These synthetic observations are designed to replicate realistic sensor behavior and safety guidelines. Finally, we obtain real-time bus location data through Restful APIs from the bus service of Ioannina.

The main objective of modeling bus context-aware entities and integrating IoT devices within Ioanninas's bus service for our testbed is to conduct a realistic evaluation of three distinct NGSI-LD context brokers: Stellio, Scorpio, and Orion-LD. The testbed assesses the effectiveness of each context broker in managing, querying, and utilizing data from the bus service ecosystem in three key aspects: ***query performance***, ***data ingestion*** and ***scalability***. This evaluation enables us to provide enhanced services and accurate analyses through a variety of *general*, *temporal*, and *geospatial queries* applicable in smart transportation systems. Fig. 4 illustrates our testbed, showcasing the Orion-LD, Scorpio, and Stellio context brokers along with their underlying databases. Mintaka is employed separately to facilitate temporal queries on Orion-LD, while a synthetic

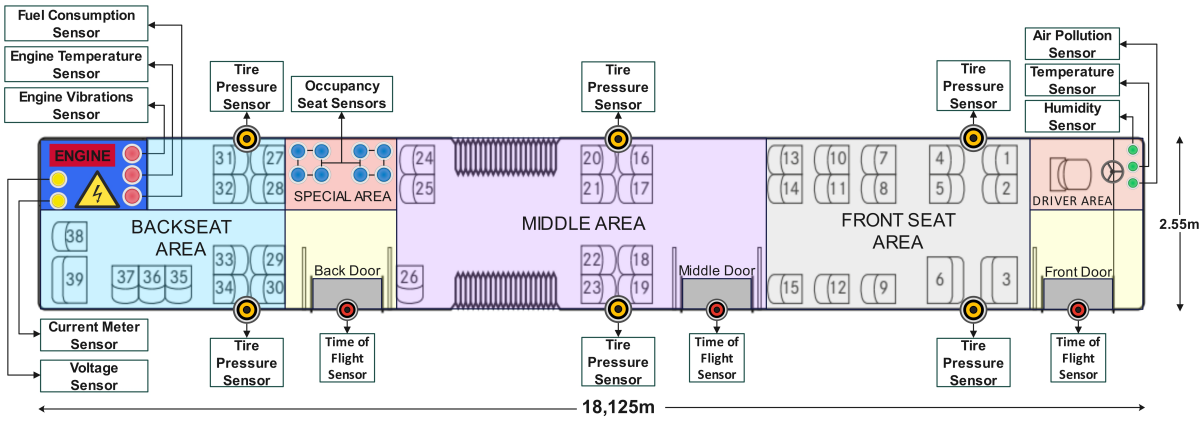


Figure 2: 2D representation of a Mercedes Citaro G (C 628.233-13).

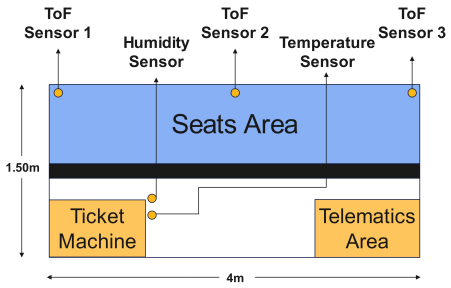


Figure 3: 2D representation of a Bus Station.

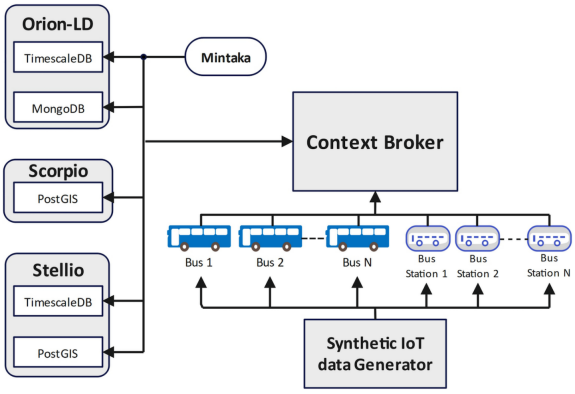


Figure 4: SCB Testbed Overview.

data generator generates observation values for buses and bus stations.

3.2 SCB Test Queries

To define representative *test queries* for the SCBenchmark and SCB testbed, we define the following smart city bus related applications: *Predictive Maintenance*, *Route Optimization*, and *Client Mobile Application*. Predictive Maintenance leverages IoT technology to monitor key components of the bus fleet in real-time. In particular. data of mechanical performance (Fuel Consumption, Engine Vibrations, Engine Temperature, Tire Pressure, Current meter) are collected to avoid unplanned downtime and minimize the repair costs. Route Optimizations utilizes IoT data to dynamically adjust bus routes based on real-time bus positions, passenger demand, and Occupancy sensors. The goal of this application is to improve the operational efficiency and reduce fuel consumption. Finally, the Client Mobile Application provides the ability to passengers to access bus schedules, routes and interactive features.

The above applications are responsible for enabling e-ticketing, real-time updates of bus locations, and interaction with the system (e.g the ability for disabled passengers to reserve a seat in advance, if available, ensuring they can travel with safety). In addition. they issue data requests to the context broker, which execute queries to the underlying databases based on data workloads.

Queries are categorized into 3 representative groups for SCBenchmark:

General Queries: Applications may request data related to the current state of entities via a context broker. When applications seek general data on the bus system such as bus locations, schedules, bus information, the following queries can be exploited:

Query 1: Bus Location Retrieval. It retrieves the current location of a specific bus identified by its unique resource name (URN) b_u.

Query 2: Temperature Threshold Analysis. It identifies buses where the engine temperature exceeds a predefined threshold T.

Query 3: Fuel Efficiency Ranking. It determines the top 10 buses with the highest fuel consumption.

Temporal Queries: Certain applications such as predictive maintenance and demand forecasting, rely on Machine Learning algorithms access to temporal data. These algorithms typically require access to observations that change over time periods. To access temporal data from applications, the following queries can be used:

Query 4: Aggregate Air Pollution Assessment. It calculates the average air pollution of all buses over a 12-hour period.

Query 5: Daily Bus Occupancy Analysis. It assesses the average occupancy rate of each bus for a specific day.

Table 2: Context Brokers and Underlying Databases.

Context Broker or Extension	Version	Underlying DB	DB Version
Orion-LD	1.3.0	MongoDB	3.6
Mintaka	0.5.4	TimescaleDB	TimescaleDB 1.7.5 PostgreSQL 12
Stellio	2.5.2	TimescaleDB PostGIS	PostgreSQL 14 TimescaleDB 2.11 PostGIS 3.3
Scorpio	4.1.0	PostGIS	PostgreSQL 15 PostGIS 3.3

Geospatial Queries: Applications can perform geospatial queries through context brokers by leveraging geographical information such as longitude and latitude coordinates. By using these queries, applications can search for data within specific geographical regions, enabling functionalities such as distance calculations to points or polygon intersection checks. Examples of representative geospatial queries are defined as follows:

Query 6: Proximity-Based Bus Search. It finds all buses located within a 5-kilometer radius of a given geographical point.

Query 7: Bus Stop Proximity Search. It locates all bus stops within a certain radius of a specified point.

For certain queries, such as Q3, Q5, and Q6, some context brokers (e.g., Orion-LD) do not support aggregation functionalities. Consequently, aggregation computations are conducted at the client side when data arrives for all context brokers.

3.3 SCBenchmark Setup

We now present our benchmark setup, utilizing the SCB testbed, aimed at assessing the capabilities of existing NGSI-LD based context brokers. This evaluation focuses on their efficiency in handling **generic**, **temporal**, and **geospatial queries** as well as their proficiency in **data ingestion**. SCBenchmark setup enables a comprehensive analysis of each context broker's performance, facilitating an informed selection of the most suitable platform for enhancing operational efficacy and data management within IoT domains.

In our evaluation, we distinguish between two types of hosts: the **context broker host**, where the context broker of interest is deployed and evaluated, and the **SCBenchmark host**, where we execute our benchmarking processes. Both the context broker and the SCBenchmark hosts are equipped with an Intel Xeon Bronze 3206R processor clocked at 1.90GHz with 32GB DDR4 memory, a Dell Micron 480GB SSD, and a Toshiba 2TB 7200RPM hard disk. Versions of underlying databases are shown in Table 2, and the cache size of each database is set to 2 GB. Despite cache size, default settings were used for all databases.

For this setup, the NGSI-LD and Mintaka API endpoints are utilized to retrieve data, as depicted in Fig. 4. Our goal is to evaluate the performance of each query described in Section 3.2 in a real-world scenario. Each query is executed 1000 times, with the execution time recorded on the client side for every instance. The average execution times are then calculated based on these 1000 runs. Before executing the queries described in Section 3.1, we conducted a preliminary data collection phase to process realistic IoT data from the utilized datasets, as well as to gather synthetic data using the tool described in Section 3.1.

This dataset is collected at a 10-second interval, focusing on two main aspects: (i) the active fleet of buses; and (ii) the bus stations of Ioannina. Data were collected over a week from more than 60 buses and 420 bus stations, resulting in approximately 204 GB for about 240 million records in temporal databases, and approximately 5.97 MB with around 6,121 records in the context databases. Queries are executed on different databases based on the context broker and the clients' request. For the Orion-LD context broker, general queries and geospatial queries run in MongoDB (Q1, Q2, Q3, Q6, Q7), while temporal queries run on TimescaleDB (Q4, Q5). Temporal queries run through Mintaka for Orion-LD as described in Section 2. For the Stellio context broker, general queries and geospatial queries run in PostGIS (Q1, Q2, Q3, Q6, Q7), while temporal queries run in TimescaleDB (Q4, Q5). For the Scorpio context broker, all queries run in PostGIS. We use Apache JMeter to generate the required parameters for each of the queries and the time ranges (for temporal queries) by following the Zipfian distribution.

3.4 SCBenchmark Results

We now assess the efficiency and scalability of various context brokers using the SCB benchmark and CSB testbed. We break up our evaluation into three categories: *Context-Broker Query*, *Ingestion Time* and *Scalability* evaluation. These performance metrics are important for understanding how different context brokers handle typical demands of real-world IoT applications, where rapid processing and management of context-aware data is crucial.

Note that the SCBenchamak results can be reproduced by following the instructions provided at https://github.com/satrai-lab/scbenchmark.

3.4.1 Context-Broker Queries. Fig. 5 presents a detailed analysis of the execution time for queries Q1 to Q7. This includes the time spent querying the database (**DB time**) and the time associated with additional processing overheads (**processing time**). Processing time for Scorpio and Stellio includes Apache Kafka processing delays and query processing delays on the client side. For Orion-LD, it includes of NGSI-LD transformations and query processing delays on the client side. The X-axis depicts labeled queries from Q1 to Q7 while the Y-axis shows the average execution time in milliseconds (ms) for each query on a logarithmic scale. In all queries, the processing time is under 50 ms, with database access being the dominant component of execution time. Because of the logarithmic scale, the processing time for Q4 and Q5 is not visible.

The analysis of query execution time reveals that all context brokers perform comparably with small differences: For generic queries (Q1-Q3), Orion-LD outperforms other context brokers, thanks to its efficient handling of NGSI-LD transformations and minimal additional processing requirements. Scorpio and Stellio exhibit slightly longer execution times, primarily due to the overhead introduced by their integration with Apache Kafka (flushing to disk) which adds processing delay.

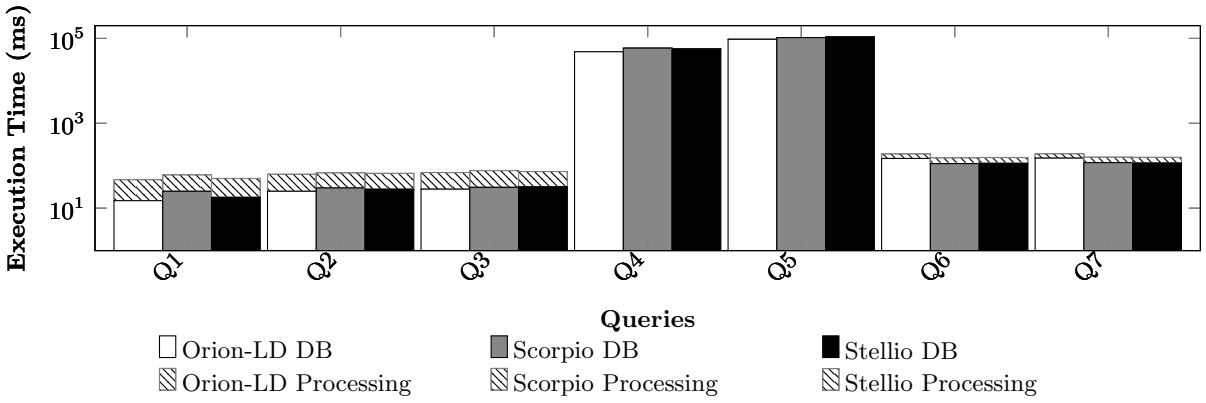


Figure 5: Execution time for queries Q1 to Q7 for Orion-LD, Scorpio, and Stellio, in log scale. Orion-LD uses MongoDB for all queries except for the temporal queries, Q4 and Q5, which are handled by Mintaka.

For temporal queries (Q4, Q5), context brokers often encounter significant delays primarily due to the increased processing load and complexities associated with the high volume of data being queried and analyzed. Temporal data, which have a time-series form with continuous and append-only growth, requires accessing large sequential data blocks, making these queries more demanding than point queries. Despite special handling provided by time-series databases like TimescaleDB, queries take significant time in all cases due to the extensive data processing required. The delays measured in these queries can be squarely attributed to the heavy time-series processing required here.

For geospatial queries (Q6, Q7), we observe that Scorpio and Stellio, which use PostgreSQL and PostGIS respectively, demonstrate faster query execution compared to MongoDB. The primary reasons for this performance difference are the indexing and query optimization capabilities of Scorpio and Stellio's underlying databases, particularly PostGIS, which is specifically designed for efficient handling of complex geospatial data through advanced indexing [23].

An interesting question given the cost of temporal queries is whether database caching may be used to speed them up. Our extensive empirical investigation shows that this is challenging to achieve, as temporal queries often have complex, time-varying access patterns, making it difficult in the general case to accurately predict and cache the relevant data. In addition, the dynamic and real-time nature of IoT data necessitates continuous updates, resulting in frequent cache invalidation and refreshing, which can drastically reduce cache efficiency [21]. Consequently, even if large amounts of memory were available for caching, the unpredictable access patterns and continuous data updates characteristic of IoT scenarios would limit the effectiveness of conventional database caching strategies. One exception is caching temporal data from IoT transportation systems near the application, utilizing knowledge of the application's data access patterns to implement intelligent caching or prefetching strategies tailored to its needs.

For instance, if an observation from the Engine Temperature sensor is requested for a specific time range, the cache may proactively fetch all the observations from the engine-related sensors, as it is likely that the Engine Vibrations Sensor observations and Fuel Consumption Sensor observations will also be required for the same time range, for predictive maintenance purposes. Such an application-specific caching and prefetching mechanism could be implemented in a case-by-case basis for different IoT application domains.

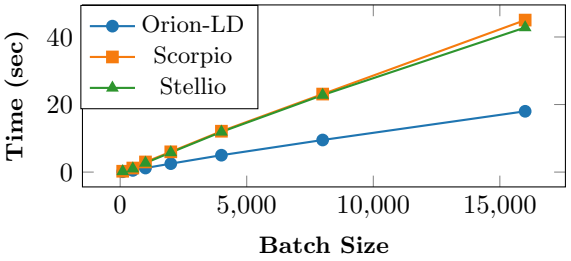
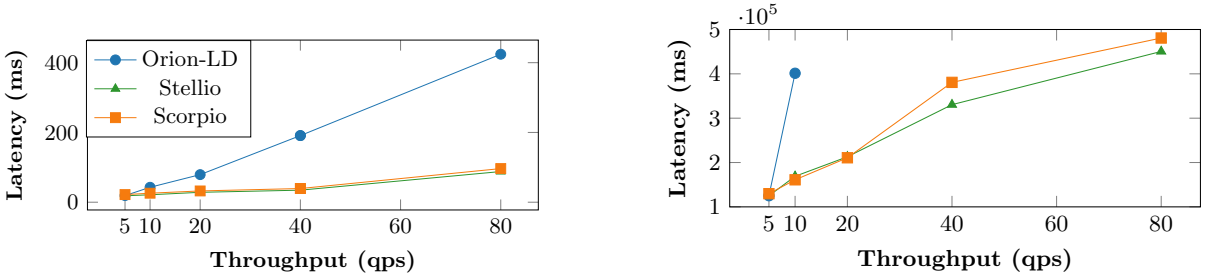


Figure 6: Context broker ingestion time.

In summary, we observe all evaluated context brokers to perform comparably with small differences. For generic queries, Orion-LD outperforms Scorpio and Stellio due to its more integrated architectural design and lower computational overhead. For temporal queries, Stellio and Scorpio exhibit similar execution times and outperform Orion-LD thanks to their more efficient underlying PostGIS database indexing compared to MongoDB's indexing. Regarding temporal queries, all of the evaluated context brokers face challenges due to the high data volumes associated with such queries, resulting in substantial performance bottlenecks across systems. The higher overheads observed for Scorpio and Stellio are primarily driven by the additional processing delays introduced by the use of Apache Kafka, which also persists data to disk for durability purposes. In contrast, Orion-LD's overheads are largely attributable to the transformations required for NGSI-LD compliance and the handling of large datasets.

3.4.2 Ingestion Time. We now evaluate how efficiently context brokers handle the ingestion of context-aware data. The effectiveness of data ingestion directly impacts the responsiveness of context brokers and the speed at which new



(a) Q1 Latency vs. Throughput.

(b) Q5 Latency vs. Throughput.

Figure 7: Latency vs. throughput comparison for queries Q1 and Q5.

data becomes available to consumers, thus influencing their overall performance.

Context brokers have the capability to ingest entities in *batches*. To assess their performance with different batch sizes and number of entities in each batch, we design an ingestion workload (referred to as the *batching workload*), which helps us understand the processing cost for each batch size for a specific broker. For this workload, we generate batch sizes ranging from 100 entities to 16,000 entities, with each NGSI-LD entity consisting of 1KB of information. Due to limitations of Orion-LD related to message size, we have increased the payload using the option `outReqMsgMaxSize` from 5MB to 15MB. We then measure the average time for the brokers to ingest batches of 100, 500, 1000, 2000, 4000, 8000, and 16,000 NGSI-LD entities.

Fig. 6 compares the batch ingestion performance of Orion-LD, Stellio, and Scorpio as the batch size and number of entities increase. The x-axis depicts batch size and the y-axis execution time in seconds. Our results show that the execution time increases for all context brokers as the batch size grows, which is anticipated as bigger batches carry more entities to be processed. Orion-LD ingests faster per batch compared to the other brokers. This occurs due to its underlying database technology, MongoDB, which employs Log-Structured Merge (LSM) Trees. LSM Trees are optimized for *write-heavy* applications, as they initially write data to a fast in-memory structure before periodically merging these writes to disk, reducing write amplification and enhancing write performance. Additionally, MongoDB's high write throughput and efficient batch write handling contribute to Orion-LD's strong ingestion capabilities. In contrast, Stellio and Scorpio leverage PostGIS for context data storage, which relies on B+ Trees. While B+ Trees are efficient for a range of database operations, particularly read-heavy and mixed read/write workloads, they may lack the ingestion speed of LSM Trees under high write-load conditions. This is due to the additional overhead involved in maintaining the balanced structure of B+ Trees during write operations, leading to increased write latency under heavy write loads.

3.4.3 Scalability. To assess the scalability of the context brokers, we create a high-throughput SCBenchmark host running JMeter to generate queries Q1 and Q5 asynchronously. These queries where chosen as the least and most resource-intensive queries in our set. We measure several key metrics, including the total duration of the experiment, the total number of requests served, and the latency of each request, from initiation by the SCBenchmark host to completion and return to the client. Starting with a baseline load of 5 queries per second, we incrementally increase the load to determine the saturation point at which the performance of the context brokers began to degrade. This allows us to evaluate how each context broker handles increasing workloads and identify their respective performance thresholds. To evaluate the scalability, we measure the total time, total number of requests and the latency of each query. From these values, we calculate queries/sec and average latency for requests.

To use optimal settings for concurrency where possible, we change the Orion-LD default setting of the reqMutexPolicy option from "all", which means that only one query can be processed at a time, to "none", which allows for concurrent processing of queries. Our evaluation shows that Scorpio and Stellio are already capable of processing multiple queries simultaneously, thus we are confident that we established a fair baseline for comparison.

Fig. 7 shows the throughput and latency for queries Q1 and Q5. The x-axis depicts the throughput as ***queries per second (qps)*** and the y-axis the ***average latency (ms)***. For query Q1, Orion-LD exhibits a latency of 19 ms at 5 qps, which increases to 43 ms at 10 qps, 79 ms at 20 qps, 191 ms at 40 qps and 424 ms at 80 qps. This significant increase in latency at 20 qps, indicates a struggle with higher throughput due to its architecture using threads and blocking I/O. Stellio showed a latency of 18 ms at 5 qps, increasing to 21 ms at 10 qps, 28 ms at 20 qps, 34 ms at 40 qps and 87 ms at 80 qps. Scorpio had a similar response to Stellio to increasing load, both performing better than Orion-LD. Stellio differentiates exhibiting more robust performance vs. Scorpio (with a similar architecture) particularly in the costlier Q5 query, highlighting a somewhat more efficient data processing pipeline.

For Q5, Orion-LD's latency is 125 ms at 5 qps, spiking to 401 ms at 10 qps. For 20-80 qps, queries execute for more than $5 * 10^5$ ms (8.33 mins), indicating a significant bottleneck. We consider this delay unreasonably long, especially compared to other systems and thus consider it as a timeout period (also for visualization purposes). The bottleneck is due to the limited parallelism available in Orion-LD towards the temporal database TimescaleDB. Scorpio's latency is 129 ms at 5 qps, 161 ms at 10 qps, 210 ms at 20 qps, 380 ms at

40 qps, and 480 ms at 80 qps, showing a more controlled increase compared to Orion-LD. Stellio maintained relatively lower latency indicating better scalability.

Overall, Stellio and Scorpio exhibit better scalability compared to Orion-LD, owing to their microservice-based architecture featuring asynchronous execution of queries.

4 RELATED WORK

This section reviews related work on the evaluation of IoT-enhanced transportation systems and FIWARE-based context brokers.

IoT Benchmarks. Benchmarks have been proposed to test a variety of IoT-related queries. IoTABench [5] utilizes synthetic smart-meter data generated by a Markov-chain data generator trained using real data. The benchmark issues six query types, including selection, aggregation, and projection, ordered by operations. TPCx-IoT [28] is based on electric power stations encompassing a variety of sensor types. It features workloads with concurrent reads and query operations executing 5 query operations per 10,000 sensor readings, employed to evaluate the ingestion and processing of streaming data from sensors. Smartbench [17] is specifically designed to evaluate the capabilities of multiple relational database management systems (RDBMSes) within IoT smart spaces, particularly focusing on real-time applications.

IoTDB-Benchmark [22] evaluates time-series databases by focusing on data retrieval queries, ingestion rate, and resource usage. It generates synthetic data within a given range and encompasses 10 types of queries, ranging from a single entity return to time range queries with value filters. CityBench [3] is developed to evaluate the performance of RDF stream processing engines within smart city applications, focusing on handling smart city data streams effectively. BigBench [15] extends the TCP-DS benchmark by incorporating semi-structured data from user clicks and unstructured data from online product reviews. OpenBenchmark [36] is a cloud-based service that facilitates repeatable and reproducible experimentation on supported testbeds, instrumenting firmware according to industry-relevant test scenarios, and collecting and processing experiment data.

While these benchmarks assess aspects of IoT systems with underlying databases, they do not specifically address smart transportation systems. Unlike previous works, our platform evaluates 3 context brokers across diverse query types, providing a comprehensive performance assessment.

FIWARE-based Benchmarks. The existing literature explores various benchmarking approaches for evaluating the performance and feasibility of FIWARE middlewares. Martinez et al. [24] examines the viability of using FIWARE context brokers for precision agriculture applications. This testbed uses simulated data to assess Orion's throughput and latency. Araujo et al. [4] presents a comprehensive performance evaluation of Orion-LD. Their evaluation focuses on horizontal and vertical scalability, identifying bottlenecks and proposing cost-efficient deployment strategies of smart city applications using CoAP and MQTT for data exchange through Orion-LD.

In contrast, our evaluation focuses on querying and batch ingestion of context/temporal NGSI-LD entities across three context brokers serving smart-transportation applications, incorporating synthetic data and real data from a city bus service.

Testbeds. The Martin Luther King (MLK) Smart Corridor [18] addresses the challenges posed by urbanization and explores how IoT and communication technologies can provide quantifiable insights into a city's infrastructure state. It emphasizes the complexity of data generated by smart cities and highlights the need for advanced data integration platforms that can support data collection, analysis, and storage.

Similarly, SmartSantander [33] presents a city-scale experimental facility for IoT research and experimentation, offering insights into the architecture and implementation of large-scale IoT testbeds within the context of a smart city. Datta et al. [19] address the lack of comprehensive testing environments for connected car services by proposing a novel testbed utilizing IoT and microservices to facilitate a full-stack, cloud-based platform for experimentation and development in the burgeoning field of connected vehicles.

Although the above works cover various IoT domains, none specifically target context-awareness in smart city applications. Our SCB testbed is designed to scale up for multiple buses, bus stations, observations, and devices, while utilizing context-aware semantics to enhance system efficiency.

5 CONCLUSION AND FUTURE WORK

This paper introduces SCBenchmark, a platform for evaluating the performance of context-aware brokers in IoT-enhanced smart transportation systems. SCBenchmark utilizes three widely used context-aware brokers: Orion-LD, Stellio, and Scorpio. The evaluation is carried out on the realistic SCB testbed, which integrates both real and synthetic data. The findings show that brokers perform similarly under low loads, with Orion-LD slightly outperforming the others in generic and geospatial queries. However, all brokers face significant challenges with temporal queries due to the complexity and high load of managing large volumes of time-series data. In terms of scalability, Stellio and Scorpio perform best, while Orion-LD struggles, especially with concurrent temporal queries. Additionally, write-optimized data stores offer a notable advantage in ingestion speed, as observed with Orion-LD.

Future research involves exploring strategies to optimize temporal query processing capabilities of context-aware brokers. The incorporation of a dedicated temporal cache, such as TSCache [21], as well as application-specific caching and prefetching policies on a case-by-case basis, represent promising approaches to enhancing performance. By addressing the performance bottlenecks associated with temporal queries and by continuing to build upon scalable architectural (such as asynchronous, microservices-based) styles, researchers can unlock the full potential of context-aware brokers in time-sensitive applications, ultimately enhancing the overall efficiency and reliability of IoT systems in smart transportation contexts.

6 ACKNOWLEDGMENTS

The authors thankfully acknowledge funding by the GreenInCities Horizon Europe (GA no. 101139730) project.

REFERENCES

[1] 2017. TimescaleDB : SQL made scalable for time-series data. https://api.semanticscholar.org/CorpusID:34446750

[2] NGSI-LD API Specification. https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.05.01_60/gs_CIM009v010501p.pdf.

[3] Muhammad Intizar Ali, Feng Gao, and Alessandra Mileo. 2015. Citybench: A configurable benchmark to evaluate rsp engines using smart city datasets. In *International semantic web conference*. Springer, 374–389.

[4] Victor Araujo, Karan Mitra, Saguna Saguna, and Christer Åhlund. 2019. Performance evaluation of FIWARE: A cloud-based IoT platform for smart cities. *J. Parallel and Distrib. Comput.* 132 (2019), 250–261. https://doi.org/10.1016/j.jpdc.2018.12.010

[5] Martin Arlitt, Manish Marwah, Gowtham Bellala, Amip Shah, Jeff Healey, and Ben Vandiver. 2015. Iotabench: an internet of things analytics benchmark. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. 133–144.

[6] Georgios Bouloukakis, Chrysostomos Zeginis, Nikolaos Papadakis, Kostas Magoutis, George Christodoulou, Chrysanthi Kosyfaki, Konstantinos Lampropoulos, and Nikos Mamoulis. 2023. SmartCityBus-A Platform for Smart Transportation Systems. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*. 1299–1300.

[7] Georgios Bouloukakis, Chrysostomos Zeginis, Nikolaos Papadakis, Panagiotis Zervakis, Dimitris Plexousakis, and Kostas Magoutis. 2023. Enabling IoT-enhanced Transportation Systems using the NGSI Protocol. In *Proceedings of the 12th International Conference on the Internet of Things* (Delft, Netherlands) *(IoT '22)*. Association for Computing Machinery, New York, NY, USA, 33–40. https://doi.org/10.1145/3567445.3567460

[8] CityPulse. 2024. *CityPulse Datasets.* http://iot.ee.surrey.ac.uk:8080/index.html Accessed: September 2024.

[9] Jennie Duggan, Aaron J Elmore, Michael Stonebraker, Magda Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stan Zdonik. 2015. The bigdawg polystore system. *ACM Sigmod Record* 44, 2 (2015), 11–16.

[10] ETSI. [n. d.]. *Context Information Management (CIM) NGSI-LD API V1.4.2.* ttps://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.04.02_60/gs_cim009v010402p.pdf

[11] FIWARE Foundation. 2024. *FIWARE Foundation.* https://github.com/FIWARE/mintaka

[12] FIWARE Foundation. 2024. Orion-LD Context Broker. https://github.com/FIWARE/context.Orion-LD. Accessed: September 2024.

[13] FIWARE Foundation. 2024. Scorpio Context Broker. https://github.com/ScorpioBroker/ScorpioBroker. Accessed: September 2024.

[14] FIWARE Foundation. 2024. Stellio Context Broker. https://github.com/stellio-hub/stellio-context-broker. Accessed: September 2024.

[15] Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. 2013. Bigbench: Towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*. 1197–1208.

[16] Google. [n. d.]. *GTFS Standard.* https://gtfs.org/

[17] Peeyush Gupta, Michael J Carey, Sharad Mehrotra, and oberto Yus. 2020. Smartbench: A benchmark for data management in smart spaces. *Proceedings of the VLDB Endowment* 13, 12 (2020), 1807–1820.

[18] Austin Harris, Jose Stovall, and Mina Sartipi. 2019. MLK Smart Corridor: An Urban Testbed for Smart City Applications. In *2019 IEEE International Conference on Big Data (Big Data)*. 3506–3511. https://doi.org/10.1109/BigData47090.2019.9006382

[19] Soumya Kanti Datta, Mohammad Irfan Khan, Lara Codeca, B. Denis, Jérôme Härri, and Christian Bonnet. 2018. IoT and Microservices Based Testbed for Connected Car Services. In *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*. 14–19. https://doi.org/10.1109/WoWMoM.2018.8449768

[20] Apostolos Lazidis, Konstantinos Tsakos, and Euripides Petrakis. 2022. Publish-Subscribe approaches for the IoT and the cloud: Functional and performance evaluation of open-source systems. *Internet of Things* 19 (05 2022), 100538. https://doi.org/10.1016/j.iot.2022.100538

[21] Jian Liu, Kefei Wang, and Feng Chen. 2021. TSCache: an efficient flash-based caching scheme for time-series data workloads. *Proc. VLDB Endow.* 14, 13 (sep 2021), 3253–3266. https://doi.org/10.14778/3484224.3484225

[22] Rui Liu and Jun Yuan. 2019. Benchmarking Time Series Databases with IoTDB-Benchmark for IoT Scenarios. arXiv:1901.08304 [cs.DB]

[23] Antonios Makris, Konstantinos Tserpes, Giannis Spiliopoulos, and Dimosthenis Anagnostopoulos. 2019. Performance Evaluation of MongoDB and PostgreSQL for spatio-temporal data.

[24] Ramón Martínez, Juan Ángel Pastor, Bárbara Álvarez, and Andrés Iborra. 2016. A Testbed to Evaluate the FIWARE-Based IoT Platform in the Domain of Precision Agriculture. *Sensors* 16, 11 (2016). https://doi.org/10.3390/s16111979

[25] Boris Moltchanov and Oscar Rodríguez Rocha. 2014. A context broker to enable future IoT applications and services. In *2014 6th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. 263–268. https://doi.org/10.1109/ICUMT.2014.7002113

[26] Olli Mämmelä, Pekka Karhula, and Jukka Mäkelä. 2019. Scalability Analysis of Data Transfer in Massive Internet of Things Applications. In *2019 IEEE Symposium on Computers and Communications (ISCC)*. 1–7. https://doi.org/10.1109/ISCC47284.2019.8969722

[27] Neha Narkhede, Gwen Shapira, and Todd Palino. 2017. *Kafka: The Definitive Guide Real-Time Data and Stream Processing at Scale* (1st ed.). O'Reilly Media, Inc.

[28] Meikel Poess, Raghunath Nambiar, Karthik Kulkarni, Chinmayi Narasimhadevara, Tilmann Rabl, and Hans-Arno Jacobsen. 2018. Analysis of tpcx-iot: The first industry standard benchmark for iot gateway systems. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1519–1530.

[29] Polution Dataset. 2024. *Polution Dataset.* http://iot.ee.surrey.ac.uk:8080/datasets.html#pollution

[30] PostGIS Database. 2024. *PostGIS Database.* https://postgis.net/

[31] P S Saarika, K. Sandhya, and T. Sudha. 2017. Smart transportation system using IoT. In *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)*. 1104–1107. https://doi.org/10.1109/SmartTechCon.2017.8358540

[32] Shormee Saha. 2019. A TRAFFIC MANAGEMENT SYSTEM APPROACH WITH THE IMPLEMENTATION OF ARTIFICIAL INTELLIGENCE ALGORITHMS. https://doi.org/10.33564/ijeast.2019.v04i06.001

[33] Luis Sanchez, Luis Muñoz, Jose Antonio Galache, Pablo Sotres, Juan R Santana, Veronica Gutierrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco, Evangelos Theodoridis, et al. 2014. SmartSantander: IoT experimentation over a smart city testbed. *Computer networks* 61 (2014), 217–238.

[34] Sensor Intel Berkeley Lab Dataset. 2024. *Sensor Intel Berkeley Lab Dataset.* https://db.csail.mit.edu/labdata/labdata.html

[35] Traffic Dataset. 2024. *Traffic Dataset.* http://iot.ee.surrey.ac.uk:8080/datasets.html#traffic

[36] Mališa Vučinić, Božidar Škrbić, Enis Kočan, Milica Pejanović-Djurišić, and Thomas Watteyne. 2019. OpenBenchmark: Repeatable and Reproducible Internet of Things Experimentation on Testbeds. https://doi.org/10.1109/infcomw.2019.8845160

[37] Weather Dataset. 2024. *Weather Dataset.* http://iot.ee.surrey.ac.uk:8080/datasets.html#weather

[38] Jun Zhang, Yichuan Wang, Shuyang Li, and Shuaiyi Shi. 2021. An Architecture for IoT-Enabled Smart Transportation Security System: A Geospatial Approach. *IEEE Internet of Things Journal* 8, 8 (2021), 6205–6213. https://doi.org/10.1109/JIOT.2020.3041386