ParkGrid: Smart-City Parking Occupancy and Illegal Parking Monitoring Platform

Lameya Islam and Jawayria Hashmi
Software Engineering for Internet of Things
University of L'Aquila

1 Introduction

Rapid urbanization and the increasing number of private vehicles have intensified the demand for efficient parking management systems. Poor parking utilization, illegal parking behavior, and lack of real-time visibility significantly contribute to traffic congestion, fuel wastage, and environmental pollution in modern cities. As a result, smart-city initiatives increasingly rely on Internet of Things (IoT) technologies to monitor, analyze, and optimize urban infrastructure.

This project presents **ParkGrid**, a smart-city parking occupancy and illegal parking monitoring platform. ParkGrid provides real-time monitoring of parking slots across multiple zones, detects overstaying vehicles, computes congestion indicators, and generates alerts to assist city administrators in traffic management and enforcement. The system is designed as a complete end-to-end IoT solution, covering data generation, communication, processing, storage, visualization, and alerting.

2 System Overview

ParkGrid follows the complete IoT data lifecycle:

$$\textit{Sensing} \rightarrow \textit{Communication} \rightarrow \textit{Processing} \rightarrow \textit{Storage} \rightarrow \textit{Visualization} \rightarrow \textit{Alerting}$$

Each parking slot is modeled as a simulated IoT sensor that periodically publishes its current status. These sensor updates are transmitted through a lightweight messaging protocol, processed in real time to derive higher-level metrics, stored in a time-series database, visualized through interactive dashboards, and used to trigger alerts when abnormal conditions are detected.

3 System Architecture

3.1 Sensor Layer

The sensor layer consists of simulated parking sensors implemented in Python. Each simulated sensor represents an individual parking slot and publishes periodic status updates with stochastic behavior to emulate real-world vehicle arrivals and departures.

Each sensor message contains the following attributes:

- `slot_id`: Identifier of the parking slot
- `zone`: Identifier of the parking zone (e.g., center, residential, university)
- `occupied`: Binary value indicating whether the slot is occupied (0 or 1)

- `parking_duration`: Accumulated parking time in minutes
- `timestamp`: Local (Rome) Timezone

The publishing interval is configurable, allowing realistic simulation of real-time smart parking sensors deployed in urban environments.

3.2 Communication Layer

Communication between the sensor layer and the processing layer is implemented using the MQTT protocol. MQTT is a lightweight publish–subscribe messaging protocol designed for low-bandwidth and low-latency IoT applications.

ParkGrid uses a hierarchical topic structure:

```
/parking/<zone>/<slot_id>/status
/parking/alerts
```

This topic design enables scalable sensor deployment and logical separation of data streams by zone and slot.

3.3 Processing and Middleware Layer

Node-RED is used as the middleware layer for real-time data processing. Incoming MQTT messages are parsed and processed using Node-RED flows and function nodes.

The middleware performs the following tasks:

- Parsing incoming JSON sensor messages
- Maintaining an in-memory state of parking slots per zone
- Computing zone-level occupancy percentages
- Detecting illegal parking through overstay thresholds
- Generating structured alert messages
- Forwarding processed data to storage and alerting services

Zone occupancy is computed by tracking the number of occupied slots relative to the total number of known slots in each zone. Overstay detection is performed by comparing the parking duration of occupied slots against a configurable time threshold.

3.4 Data Storage Layer

Processed parking data is stored in InfluxDB, a time-series database optimized for high-frequency, timestamped IoT data.

All data is stored under the measurement `parking_status`. The schema includes both identifying fields and numeric metrics:

Fields:

- `slot_uid`: Synthetic unique identifier per slot
- `slot_id`
- `zone`
- `occupied`

- `parking_duration`
- `overstay_flag`
- `zone_occupancy`

This schema supports historical analysis, real-time querying, and aggregation for visualization.

3.5 Visualization Layer

Grafana is used to visualize both real-time and historical parking data. Interactive dashboards provide insights into parking usage and congestion patterns.

The dashboards include:

- Zone occupancy time-series plots
- Overstay event time-series
- Live slot status tables

Dashboards automatically refresh at short intervals and allow filtering by time range to support operational monitoring.

3.6 Alerting Layer

ParkGrid generates alerts under the following conditions:

1. **Illegal parking**: A vehicle remains parked longer than the allowed duration
2. **Zone congestion**: Zone occupancy exceeds a configurable threshold

Alerts are published to MQTT topics and delivered via Telegram notifications. Each alert includes contextual information such as zone, slot identifier (when applicable), alert type, descriptive message, and timestamp.

4 Technologies Used

Component	**Technology**
Sensor Simulation	Python
Communication Protocol	MQTT (Eclipse Mosquitto)
Middleware	Node-RED
Database	InfluxDB
Visualization	Grafana
Alerting	MQTT, Telegram
Deployment	Docker Compose

Table 1: Technologies used in ParkGrid

5 Methodology

The ParkGrid platform was implemented following a modular, data-driven IoT development methodology aligned with real-time stream processing principles.

1. **Sensor Simulation and Data Generation**
   Parking sensors were simulated using a Python-based generator. Each parking slot was modeled as an independent state machine with probabilistic arrival and departure behavior. At fixed intervals, each simulated sensor published structured JSON messages containing occupancy status, parking duration, zone information, and timestamps.

2. **Message Transport via MQTT**
   Sensor data was transmitted using the MQTT publish–subscribe protocol. A topic hierarchy based on parking zone and slot identifier was adopted to enable scalable routing and selective subscription. MQTT ensured low-overhead, real-time data delivery suitable for high-frequency IoT telemetry.

3. **Stream Processing and State Management**
   Node-RED function nodes were used to process incoming MQTT messages in real time. The middleware maintained an in-memory state for each parking zone, tracking the latest occupancy status of all known slots. This state was used to compute derived metrics such as zone occupancy percentage and to evaluate rule-based conditions for illegal parking detection.

4. **Event Detection and Alert Generation**
   Illegal parking events were detected by comparing the parking duration of occupied slots against a configurable threshold. Zone congestion events were detected by evaluating zone occupancy against a predefined percentage limit. Alerts were generated as structured event objects and routed to external notification channels.

5. **Time-Series Data Persistence**
   Processed sensor data and derived metrics were persisted in InfluxDB using a time-series data model. Each data point was stored with a timestamp and associated fields, enabling efficient temporal queries, aggregation, and historical analysis.

6. **Visualization and Analytics**
   Grafana dashboards were developed to visualize real-time and historical parking metrics. Flux queries were used to aggregate, pivot, and summarize time-series data for zone-level statistics and per-slot status tables. Dashboards were configured for periodic refresh to support live monitoring.

7. **Containerized Deployment**
   All system components, including the simulator, MQTT broker, Node-RED, InfluxDB, and Grafana, were deployed using Docker Compose. This ensured environment consistency, simplified dependency management, and enabled rapid system setup and portability across different machines.

6 Results and Discussion

ParkGrid successfully delivers real-time insights into parking occupancy and illegal parking behavior. The system accurately reflects zone congestion levels, detects overstaying vehicles, and provides actionable alerts. The modular architecture allows easy scalability by adding new zones or slots without architectural changes.

7 Compliance with IoT Requirements

The project satisfies all functional and non-functional requirements specified in the IoT system development guidelines. These include simulated sensing, MQTT-based communication, middleware processing, time-series data storage, interactive visualization, real-time alerting, scalability, and containerized deployment.

8 Limitations and Future Work

The current implementation relies on simulated sensors and rule-based detection. Future work may include integration of real parking sensors, machine learning-based demand prediction, GIS-based visualizations, and enhanced security mechanisms such as authentication and encrypted communication.

9 Conclusion

ParkGrid demonstrates a complete end-to-end IoT system for smart-city parking management. By integrating sensing, communication, processing, storage, visualization, and alerting, the platform provides a practical and extensible solution for monitoring parking occupancy and detecting illegal parking behavior in modern urban environments.