IoTEventSim: an Event-Driven Simulator for Large-Scale IoT Applications in Cloud–Fog–Edge Architectures

Paulo Coelho[1], **Abadio de Paulo Silva**[1], **Rafael Pasquini**[1], **Rodrigo Miani**[1], **Pierre Sens**[2], **Luciana Arantes**[2]

[1]Universidade Federal de Uberlândia, FACOM – Minas Gerais – Brazil

[2]Sorbonne Université, CNRS, LIP6 – Paris – France

{paulocoelho,abadiops,rafael.pasquini,miani}@ufu.br,

{luciana.arantes,pierre.sens}@lip6.fr

Abstract. *Large-scale Internet of Things (IoT) applications increasingly span cloud–fog–edge infrastructures, where application behavior depends on dynamic network conditions, mobility, and device energy constraints. Evaluating such systems in real deployments is costly and difficult to reproduce, motivating simulation approaches that must balance realism and scalability. This paper presents IoTEventSim, an event-driven simulator designed for per-device modeling and runtime control in distributed IoT scenarios. In IoTEventSim, each device is modeled as an autonomous entity with explicit communication, battery, and location parameters, while transient and permanent events dynamically modify device behavior over time. The simulator natively supports MQTT and CoAP and provides a web-based interface for scenario management and monitoring. We evaluate IoTEventSim in geo-distributed cloud–fog–edge settings parameterized from FIT IoT-LAB measurements. Results show that the simulator reproduces the target latency and drop characteristics with high fidelity and scales to thousands of concurrently monitored devices without saturating CPU or network resources in the evaluated environment. These findings indicate that IoTEventSim is a practical environment for controlled, repeatable performance, robustness, and fault-experimentation studies of IoT applications under adverse and time-varying conditions.*

1. Introduction

The Internet of Things (IoT) has driven the deployment of large numbers of heterogeneous devices that continuously sense, process, and exchange data in domains such as smart cities, industry, and healthcare [Fabri et al. 2025]. As deployments grow, IoT applications increasingly rely on distributed cloud–fog–edge infrastructures to meet latency and bandwidth constraints, while still benefiting from the scalability of cloud resources [Atlam et al. 2018]. At the same time, this distribution makes experimentation substantially harder: the behavior observed by applications depends not only on device logic but also on dynamic network conditions, mobility, intermittent connectivity, and energy depletion.

Evaluating such systems in real testbeds is often impractical due to cost, operational complexity, and limited reproducibility, especially when experiments require

thousands of devices or must include adverse conditions (e.g., failures and network instabilities) [Almutairi et al. 2024]. Hence, simulation becomes an essential approach to explore configurations, validate assumptions, and perform controlled stress tests. However, faithfully simulating cloud–fog–edge IoT environments poses two competing requirements: (i) enough realism to capture per-device dynamics and transient disruptions, and (ii) enough scalability to support large populations and long-running workloads.

In this context, event-driven simulation is a key enabler. By modeling changes as discrete events that update device and network parameters over time, an event-driven approach avoids unnecessary per-timestep computation and facilitates large-scale experiments while preserving fine-grained control over each device's behavior. This is particularly useful for representing transient and permanent phenomena such as mobility, link degradation, packet loss bursts, battery drain, and device failures, which are central to cloud–fog–edge IoT scenarios.

Despite the availability of several IoT simulators, most existing tools focus primarily on infrastructure-level modeling, such as network communication, resource allocation, or application placement across cloud—fog—edge infrastructures. In these environments, IoT devices are typically represented as abstract entities or workload generators, which prevents explicit modeling of device-level dynamics such as sensing behavior, battery depletion, mobility, intermittent connectivity, and transient network disruptions. As a result, these simulators provide limited support for studying how individual device behavior evolves over time and affects system-level performance.

This limitation becomes particularly critical in large-scale and highly dynamic IoT deployments, such as disaster monitoring, industrial sensing systems, and geo-distributed infrastructures. Consequently, there is a need for simulation environments that combine large-scale experimentation with fine-grained, per-device runtime control, enabling researchers to dynamically modify device parameters during execution and observe the resulting system behavior in a controlled and reproducible manner.

In this work, we propose IoTEventSim, an event-driven simulator for IoT applications in distributed cloud–fog–edge architectures. IoTEventSim focuses on per-device modeling and runtime interaction: devices can be individually configured and observed throughout the simulation, and events can be injected to alter their behavior on demand, enabling controlled robustness testing and fault experimentation. The simulator supports the modeling of network instabilities, energy constraints, and workload variations, as well as a web-based interface for visual monitoring and scenario management.

The main contributions of this work are: (i) the proposal of IoTEventSim, a flexible and easy-to-use simulator for cloud–fog–edge IoT scenarios; (ii) native support for the MQTT and CoAP protocols; (iii) a per-device event-driven architecture that enables large-scale experiments with dynamic conditions; and (iv) the availability of the tool as open-source software for the scientific community, with source code publicly accessible on GitHub[1].

The rest of the paper is organized as follows. Section 2 presents the system model. Section 3 describes the IoTEventSim simulator. Section 4 presents the performance evaluation. Section 5 discusses related work. Finally, Section 6 concludes the paper.

[1] `https://github.com/paulo-coelho/iot-sim`

2. Model and Definitions

This section presents the system model and definitions related to the scope of this work.

2.1. System Model

IoTEventSim models an Internet of Things (IoT) ecosystem composed of a finite set of geographically distributed devices organized as a cloud–fog–edge hierarchy.

Each device is modeled as an autonomous networked entity with local state and an explicit service interface. Devices may differ in sensing behavior, communication parameters, energy characteristics, and geographic location. In IoTEventSim, each device is instantiated as an independent CoAP server, enabling direct request/response interactions as well as runtime event injection on a per-device basis; this design preserves the asynchronous and failure-prone access patterns typical of geographically distributed deployments.

Each device d_i is characterized by the following set of attributes:

$$d_i = \langle id_i, endpoint_i, path_i, loc_i, S_i(t), N_i(t), B_i(t) \rangle$$

- id_i is the unique identifier of the device;
- $endpoint_i = (host_i, port_i)$ is the network address where the device is available;
- $path_i$ represents the resource exposed by the device via CoAP;
- $loc_i = (lat_i, lon_i)$ defines the geographic location of the device;
- $S_i(t)$ represents the value from the sensor data generation function of the device at instant t, such as temperature readings;
- $N_i(t)$ represents the communication behavior perceived by the application at instant t, e.g., packet loss rate and latency, as detailed in §2.4;
- $B_i(t)$ corresponds to the battery level of the device at instant t.

2.2. Energy Model

Each device has an energy state defined by $B_i(t) \in [0, B_i^{max}]$. In IoTEventSim, energy consumption is described as a mapping of two event types that decrease the battery over time: an *idle-discharge* event, which models baseline drain while the device is available, and a *transmit-discharge* event, which models the additional energy spent when the device transmits data (e.g., when responding to a request or publishing a message).

Let B_i^{max} be the battery capacity (in mAh), r_i^{idle} the idle discharge rate (in mAh/min), and q_i^{tx} the energy cost per transmission (in mAh/tx). Given a time interval Δt (in minutes) with n_{tx} transmissions, the consumed energy (in mAh) is computed as:

$$\Delta B_i = \underbrace{r_i^{idle}\,\Delta t}_{\text{idle discharge}} + \underbrace{q_i^{tx}\,n_{tx}}_{\text{transmit discharge}} .$$

Therefore, the battery update rule is:

$$B_i(t + \Delta t) = \max\left(0,\ B_i(t) - \Delta B_i\right).$$

Example. Assume $B_i^{max} = 1000\,\text{mAh}$, $B_i(t) = 800\,\text{mAh}$, $r_i^{idle} = 10/60\,\text{mAh/min}$ (i.e., $10\,\text{mAh/h}$), $\Delta t = 30\,\text{min}$, $q_i^{tx} = 0.2\,\text{mAh/tx}$, and $n_{tx} = 30$ transmissions in the interval. Then, $\Delta B_i = (10/60) \cdot 30 + 0.2 \cdot 30 = 5 + 6 = 11\,\text{mAh}$, and $B_i(t + \Delta t) = 789\,\text{mAh}$.

When $B_i(t) = 0$, the device enters a permanent failure state and stops responding to requests. This model enables the representation of battery depletion, shutdowns, and irreversible failures.

2.3. Event Model

The simulator is event-driven: an event e is defined as a temporary or permanent modification to a subset of device parameters.

$$e = \langle T_e, \Delta_e, \tau_e \rangle$$

- $T_e \in \{\texttt{transient}, \texttt{permanent}\}$ defines the type of the event;
- Δ_e represents the variations applied to the device parameters;
- τ_e represents the duration of the event (when applicable).

Events may affect different aspects of the device, such as communication parameters, energy consumption, and geographic position.

2.4. Network Model

To model heterogeneous wide-area conditions without simulating packets, IoTEventSim represents network delay through a *latency profile*. For each device (or device–gateway link), the profile is derived from the empirical latency cumulative distribution function (CDF) and summarized by a set of percentiles. The resulting representation is an array of latency intervals, each associated with the probability mass between two consecutive percentiles.

Formally, let $F(\ell)$ be the latency CDF and let $Q(p) = F^{-1}(p)$ be its quantile function. Given a percentile set $\mathcal{P} = \{p_0, p_1, \ldots, p_k\}$ with $0 = p_0 < p_1 < \cdots < p_k = 100$, we build k intervals $I_j = [Q(p_{j-1}), Q(p_j)]$ with associated probability $w_j = (p_j - p_{j-1})/100$, for $j \in \{1, \ldots, k\}$.

In addition to delay, we consider an independent *drop mechanism*. Each device (or link) is parameterized by a drop probability $p_i^{drop} \in [0, 1]$ that represents the probability of a message being lost due to transient interference, congestion, or outages. For each outgoing message, the simulator draws $u \sim \mathcal{U}(0, 1)$ and drops the message if $u < p_i^{drop}$. This abstraction allows experiments to combine burst or high-latency conditions with packet loss, without the overhead of packet-level simulation. Section 3.1 shows how we implemented such mechanisms.

2.5. Architecture and Communication

The system architecture follows a cloud–fog–edge hierarchy. The edge layer comprises IoT devices, the fog layer comprises gateways that aggregate data from the edge, and the cloud layer provides centralized storage, monitoring, and analysis [Kuchuk and Malokhvii 2024].

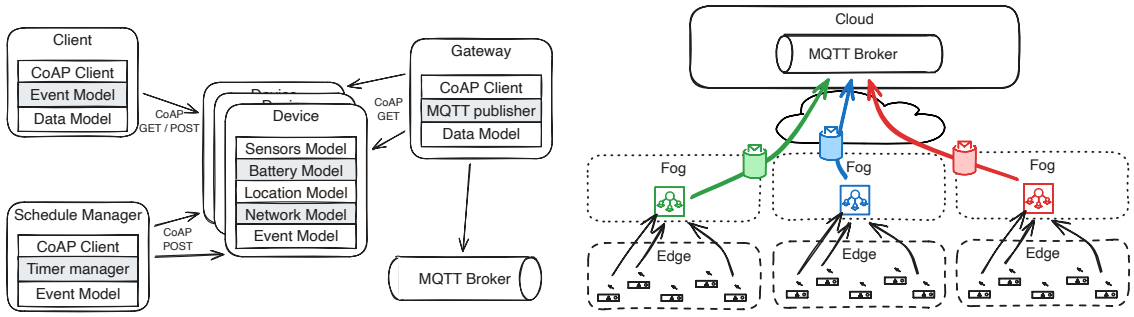
At the application layer, IoTEventSim supports MQTT and CoAP, which are widely used in IoT due to their low overhead and suitability for constrained devices [Almheiri and Maamar 2021, Seoane et al. 2021]. MQTT implements a publish–subscribe model, while CoAP follows a RESTful request–response model over UDP.

3. IoTEventSim

IoTEventSim is an event-driven simulator for Internet of Things applications deployed over cloud–fog–edge architectures. The simulator integrates (i) autonomous devices that can be addressed individually at runtime, (ii) a per-device event mechanism for time-varying reconfiguration, and (iii) fog-layer gateways responsible for periodic data collection, aggregation, and MQTT-based publishing, enabling monitoring and consumption at the cloud layer.

The edge layer is composed of simulated IoT devices that expose a CoAP service interface, enabling direct request/response interaction and event injection during execution. The fog layer is represented by gateways that periodically query edge devices via CoAP, aggregate the resulting readings, and publish the aggregated stream to an MQTT broker using a topic hierarchy. The cloud layer hosts the MQTT broker and consumes the published data through the monitoring application, without requiring direct access to edge devices.

Because each device is modeled as an independent entity and the execution is event-driven, IoTEventSim avoids a fixed global timestep and centralized per-round processing; consequently, scalability is primarily bounded by the host machine resources (CPU, memory, and I/O), unlike classical simulators whose global scheduling overhead grows with the number of nodes. Figure 1a illustrates the overall architecture while Figure 1b shows a typical deployment scenario.



(a) Simulator architecture. **(b) Typical deployment.**

Figure 1. Simulator architecture and typical deployment.

3.1. IoT Devices

Each simulated device hosts a CoAP server and exposes a set of immutable and mutable attributes. Immutable attributes define the device identity and service interface (unique identifier, CoAP endpoint, and resource path). Mutable attributes can be updated at runtime through events to emulate dynamic behaviors. In the current implementation, they include:

- the device *position*, represented by its longitude and latitude;
- the *energy state*, represented by current battery level (percentage), idle discharge rate (per minute), and energy cost per transmission;
- the *network behavior*, represented by packet drop ratio and a probabilistic latency profile (latency intervals with associated probabilities); and

- the *sensing behavior*, currently represented by a temperature value range used to generate readings.

Devices are initialized from a JSON configuration file that specifies initial values for all attributes. During execution, subsets of mutable attributes can be updated at any time using a CoAP POST request carrying an event description encoded as JSON.

Sensor data generation occurs within the device based on its current configuration. Upon each CoAP GET request, the simulator (i) generates a reading according to the sensing model, (ii) applies the current network conditions (loss and latency), and (iii) updates the energy state associated with request processing and transmission. Communication instabilities are captured through packet loss and probabilistic latency profiles, which can be updated dynamically through events.

In the current prototype, readings correspond to random temperature values sampled within the configured interval. The implementation is modular and can be extended with additional sensor types and value generation models.

To expose battery evolution as a percentage, we derive the corresponding discharge percentages for the device from the model in §2.2:

$$p_i^{idle} = 100 \cdot \frac{r_i^{idle}}{B_i^{max}}, \qquad p_i^{tx} = 100 \cdot \frac{q_i^{tx}}{B_i^{max}},$$

In the simulator configuration, we express discharge rates directly as percentage drops relative to B_i^{max}: an idle discharge rate (in %/min) and a transmit discharge rate (in %/tx). For the same example in §2.2, the idle discharge rate is $100 \cdot ((10/60)/1000) \approx 0.0167$ %/min (equivalently, 1 %/h), and the transmit discharge rate is $100 \cdot (0.2/1000) = 0.02$ %/tx. Idle and transmit discharge rates have initial configured values per device and can be dynamically changed via events.

During simulation, each time a message is sent, IoTEventSim first applies a probabilistic drop decision using the link/device drop probability $p_i^{drop} \in [0, 1]$ (§ 2.4). If the message is not dropped, the simulator samples the delivery delay by selecting an interval I_j with probability w_j and then drawing a uniformly distributed latency value inside the chosen interval (§ 2.4). This two-stage procedure (drop, then delay) is the mechanism used by IoTEventSim to reproduce the combined effect of loss and variable latency observed in wide-area IoT deployments, while keeping execution lightweight. Listing 1 illustrates this percentile-based latency profile format as an array of $(\ell_{min}, \ell_{max}, w)$ tuples.

3.2. Event Management

IoTEventSim supports *per-device* dynamic behavior changes through events, classified as *permanent* (values persist until replaced) or *transient* (values revert after the specified duration). Events are submitted via CoAP POST and applied progressively by an asynchronous routine over the configured transition interval; CoAP GET requests only expose the current device state for monitoring. Listing 1 exemplifies a transient event that updates a device position and network parameters linearly during the transition duration. Events can be submitted individually to devices by the client in Figure 1a.

IoTEventSim also provides a *Schedule Manager* to coordinate events across multiple devices: users submit schedule items containing a device endpoint, an application

time, and an event payload; the manager orders items by time and dispatches the corresponding CoAP POST requests. This mechanism enables more complex and repeatable scenarios than ad hoc per-device event injection, such as (i) synchronized disruptions (e.g., simultaneous failures or regional connectivity blackouts), (ii) staged incident timelines (e.g., progressive degradation followed by recovery waves), (iii) correlated mobility patterns across groups of devices, and (iv) large-scale what-if analyses in which the same schedule is replayed while varying only the baseline configuration (e.g., gateway placement, battery sizes, or network profiles).

Listing 1. Example transient event JSON submitted via CoAP POST.

```
{
  "event_name": "Moving",
  "event_type": "transient",
  "transition_duration_s": 100,
  "coordinate": {
    "latitude": 42.0,
    "longitude": -76.0
  },
  "drop_percentage": 20,
  "delay_profiles": [
    {
      "probability": 100,
      "min": 1.5,
      "max": 3.5
    }
  ]
}
```

As schedules are explicit artifacts, they can be versioned and shared, improving experimental traceability and supporting fair comparisons across simulator runs and alternative system designs.

3.3. Integration and Monitoring

Gateways can query devices via CoAP GET requests and publish aggregated readings via MQTT, enabling external applications to consume the simulated data stream. Gateways also export collected device readings to CSV files, which can be directly used as input datasets for incident prediction models. In addition to the provided web-based monitoring client (Figure 2), any integrated application can “talk” to the simulated environment by interacting with these standard interfaces (e.g., subscribe to MQTT topics and/or inject CoAP events), allowing arbitrarily complex closed-loop experimentation such as online analytics, adaptive control, anomaly detection, and digital-twin orchestration.

4. Performance Evaluation

4.1. Simulation Setup

In our experimental setup, we emulate a geo-distributed cloud–fog–edge deployment composed of three independent regions. Each region includes an edge cluster with 1000 simulated IoT devices and a fog gateway responsible for periodic data collection and aggregation. Gateways publish aggregated data to the cloud layer through an MQTT broker, enabling centralized consumption and monitoring.

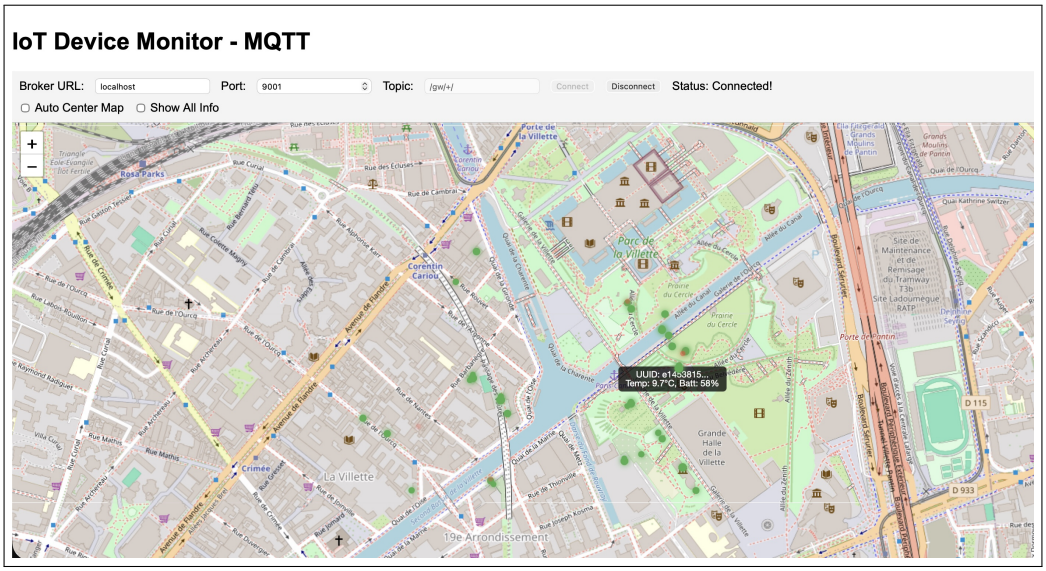


Figure 2. Web interface used to monitor the simulation.

The experiments pursue two objectives: (i) evaluating scalability as the number of concurrently active devices increases and (ii) validating that device behavior remains consistent with the expected dynamics derived from the initial configuration and injected event sequence (e.g., mobility, network impairment, and energy depletion). Specifically, we verify that the sensor readings, availability, and communication characteristics observed by gateways and the monitoring client match the parameter updates applied over time.

All experiments were conducted on virtual machines deployed across a cluster of eight Dell PowerEdge R710 servers interconnected by 1 Gbps Ethernet. The cluster provides an aggregate of 640 GB of RAM and 224 CPU cores. Each virtual machine was configured with 4 vCPUs, with 32 GB assigned to nodes running IoT devices and 4 GB assigned to the MQTT broker and IoT gateways.

To emulate heterogeneous wide-area network conditions among regions and between fog and cloud components, end-to-end latencies are injected at runtime using the *latency-setter* tool. The tool enforces configurable delay profiles in the underlying network stack, enabling repeatable experiments under controlled latency distributions. The latency-setter source code is publicly available on GitHub[2].

Table 1. Regional mean RTT matrix ($ms \pm \sigma$)

	Lille	**Saclay**	**Strasbourg**
Grenoble	39.56 ± 13.62	36.11 ± 15.39	38.70 ± 26.02
Lille		28.19 ± 8.28	41.80 ± 21.86
Saclay			35.99 ± 20.43

The simulation is based on four FIT IoT-LAB regions [Adjih et al. 2015]—Grenoble, Lille, Saclay, and Strasbourg—with the first three hosting devices and gateways, and Strasbourg hosting the MQTT broker, in a deployment analogous to Figure 1b.

[2]`https://github.com/paulo-coelho/latency-setter`

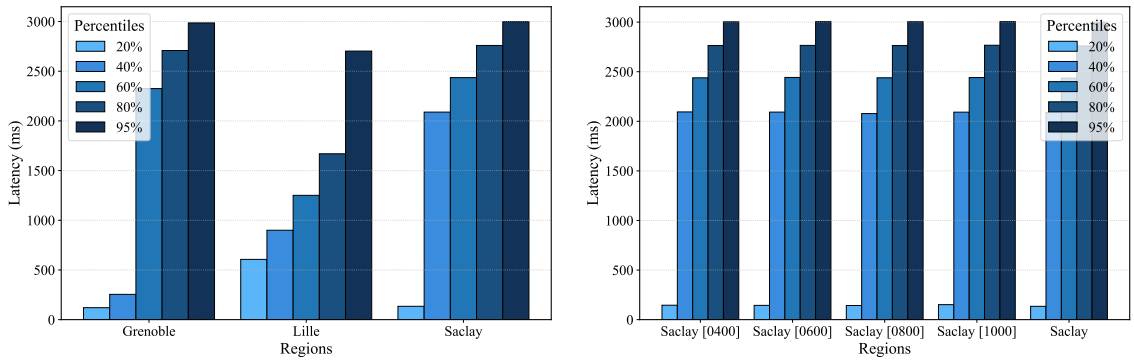
Round-trip times (RTT) were measured in a 24-hour experiment using 30-second ping intervals between each pair of regions. Table 1 reports the resulting values.

4.2. Simulation Accuracy

To evaluate simulation accuracy with respect to the configured parameters, we executed a 24-hour experiment in IoT-LAB [Adjih et al. 2015] with 10 physical devices distributed across three regions.

Figure 3a shows the latency distribution observed. These measurements were then used to parameterize the simulated scenario, in which each region included 1000 simulated devices configured according to the corresponding IoT-LAB latency profile.

Figure 3b shows that the latency distribution of simulated devices configured with the Saclay profile is consistent with the real measurements, even at a scale of 1000 devices.



(a) Latency distribution for devices in IoT-LAB.

(b) Latency distribution for simulated devices.

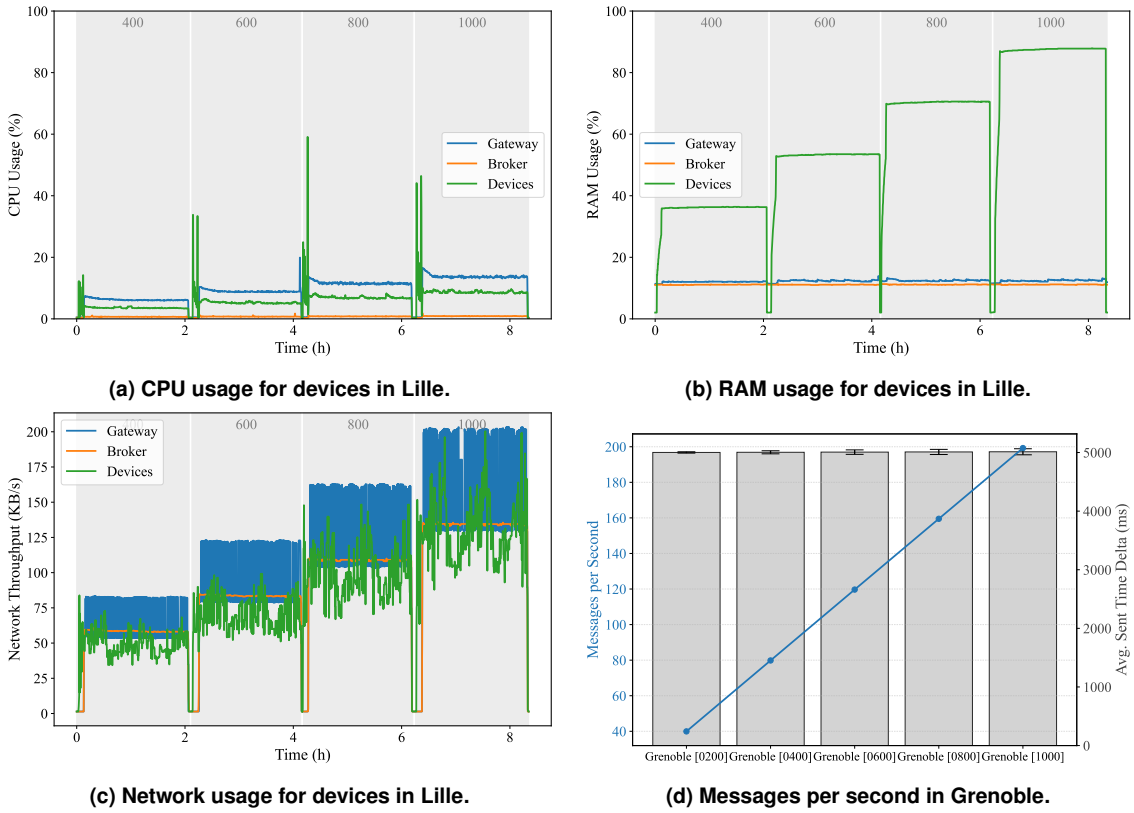
Figure 3. Latency percentiles for real and simulated devices.

The measured drop rates for Grenoble, Lille, and Saclay were 0.1%, 0.1%, and 11.1%, respectively. In the simulated scenario with 1000 devices per region, we observed drop rates of 0.11%, 0.12%, and 11.13%, indicating close agreement with the target behavior.

4.3. Scalability

The scalability experiments were conducted with an increasing number of devices, from 400 to 1000 per region (i.e., from 1200 to 3000 devices overall). Each configuration was executed for approximately 2 hours. A total of seven virtual machines were used: two per region (one for devices and one for the gateway) and one for the MQTT broker. Because results were similar across regions for the same scale, we present one representative region in each graph for clarity.

Figure 4a shows that, even with 1000 devices in Lille, CPU utilization remains moderate, indicating that no component becomes CPU-bound at this scale. In contrast, because each device is instantiated as an independent process, RAM consumption scales approximately linearly with the number of devices, as shown in Figure 4b. This behavior is expected from the process-per-device design and indicates that scalability is primarily constrained by available memory resources.



(a) CPU usage for devices in Lille.

(b) RAM usage for devices in Lille.

(c) Network usage for devices in Lille.

(d) Messages per second in Grenoble.

Figure 4. Performance with increasing number of devices.

A similar trend is observed for network bandwidth. Figure 4c shows that, even with 1000 devices, link utilization remains well below saturation. Finally, Figure 4d highlights two additional aspects. First, gateway throughput (messages per second) increases approximately linearly with the number of devices, reinforcing that the evaluated components are not saturated in this range. Second, the configured gateway probing interval of 5000 ms is preserved even with 1000 devices. The whiskers at the top of the bars represent the 99th percentile and confirm low timing variability.

4.4. Events

To evaluate the effectiveness of the event-based mechanism, we executed an experiment with 10 devices configured with an idle battery depletion rate of 0.001% per minute and an additional 0.03% depletion per request. For all devices, the baseline drop rate was set to 0%, and latency was configured in the 1.0–1.1 ms range with 100% probability.

The Schedule Manager was configured to inject a 10-minute transient event into 5 devices after 11 minutes of execution. During the event window, the affected devices were reconfigured with 100% probability for a latency of 3–3.1 ms, 41% drop rate, and 1% idle battery depletion per minute.

Figure 5 presents the behavior observed during the 31-minute execution. The dashed curves (battery level) exhibit near-linear decay for unaffected devices, whereas affected devices show an additional 10% drop during the 10-minute event interval (highlighted in gray).

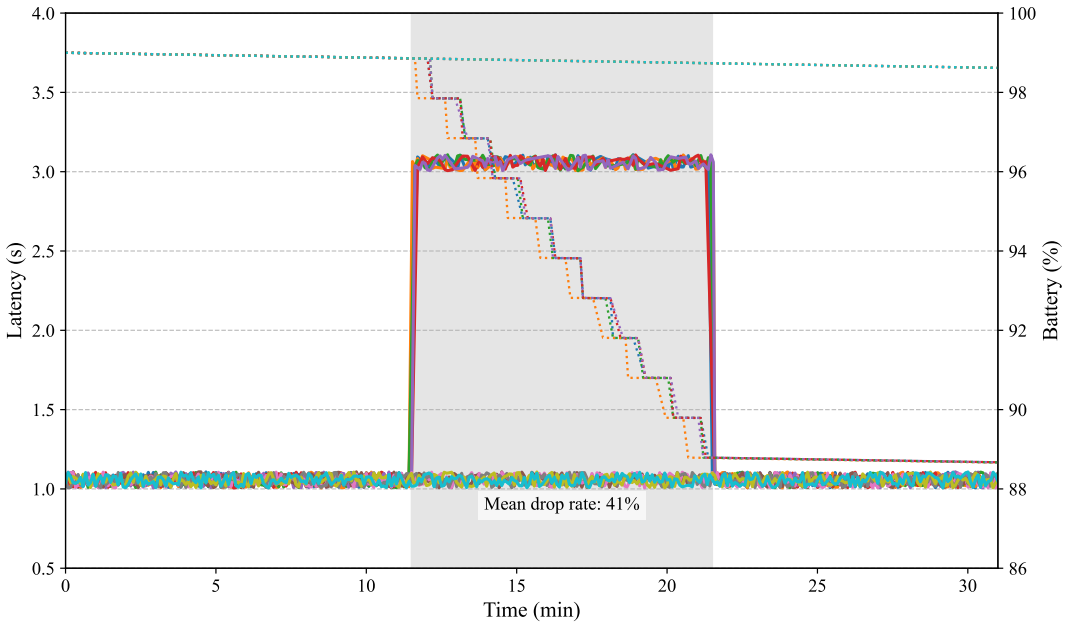


Figure 5. Transient-event experiment over a 31-minute execution window.

The latency increase is also visible in the solid curves for the affected half of the device set. Finally, the transient-event semantics restore the original latency, drop rate, and idle battery depletion values after the event ends.

5. Related Work

Existing cloud–fog–edge IoT simulators have advanced infrastructure and protocol evaluation, but they usually provide limited support for controlled robustness testing centered on per-device fault dynamics over time. Our target gap is the lack of environments that combine large-scale execution with explicit, runtime, device-level fault experimentation (e.g., transient and permanent disruptions) and direct observation of resulting behavior.

iFogSim [Gupta et al. 2017] is a simulator built on top of CloudSim [Calheiros et al. 2011] to model IoT environments in cloud–fog–edge architectures, focusing on module placement policies and resource management strategies. Applications are represented as processing graphs distributed among sensors, fog nodes, and the cloud. The tool evaluates simulated scenarios inspired by latency-sensitive applications, such as interactive games and distributed surveillance systems, analyzing metrics including end-to-end delay, energy consumption, and network utilization. Communication among components is modeled in an abstract manner, without explicitly representing application-layer protocols such as MQTT or CoAP. iFogSim also provides a graphical user interface for defining the topology and configuring experiments.

IoTSim-Edge [Jha et al. 2020] is a simulator for evaluating applications in cloud–fog–edge architectures, developed on top of CloudSim. The framework models IoT devices, edge nodes, and cloud infrastructure, incorporating mobility and energy consumption models, with experiments conducted entirely in a simulation environment using abstract representations of devices and protocols. The simulator represents different application-layer protocols at a logical level, including CoAP, XMPP, AMQP, and MQTT.

Its evaluation considers three scenarios inspired by real-world applications: healthcare monitoring with body sensors processed at the edge, smart building environments with sensors transmitting data to gateways and the cloud, and vehicular scenarios in which mobile vehicles dynamically connect to roadside units. In all scenarios, the impact of protocol selection and system load is analyzed using metrics such as end-to-end delay and energy consumption.

IoTFogSim [Pereira et al. 2021] is an event-driven simulator for applications in cloud–fog–edge architectures, developed in Python and based on the Parallel and Distributed Simulation (PADS) paradigm. The tool models IoT devices, fog nodes, and cloud infrastructure, enabling the evaluation of algorithms and protocols in scenarios with a large number of devices. The simulator provides a graphical interface for node configuration and executes experiments in a simulation environment through an MQTT-based application in which varying numbers of nodes exchange messages and generate events. In this context, performance metrics such as CPU and memory usage are analyzed as the number of nodes and events increases, with results compared to those of FogNetSim++ [Qayyum et al. 2018].

Dockemu [Portabales and Nores 2018] integrates the ns-3 network simulator with Docker containers to execute real applications in simulated environments. While ns-3 models the communication infrastructure—simulating link conditions and IoT technologies like LTE and 6LoWPAN—Dockemu lacks internal models for IoT sensors or data generation. Instead, each node is a container running real software (e.g., CoAP or MQTT). Message exchange depends entirely on these applications and the simulated network, making Dockemu a hybrid simulation-emulation framework focused on evaluating communication performance and protocol behavior.

[Kirsche and Schnurbusch 2014] proposes a simulation model of the IEEE 802.15.4 standard for the OMNeT++/INET framework, aiming to extend support for Internet of Things scenarios and wireless sensor networks. The work extends OMNeT++ through the modeling of the physical and MAC layers, enabling the representation of the communication stack typical of IoT environments and its integration with protocols such as 6LoWPAN. This approach allows the evaluation of communication behavior among nodes under different network configurations and operating conditions. Similarly to IoTEventSim, the proposal relies on a network simulator for controlled experimentation, with OMNeT++ modeling communication interactions among devices.

[Shahrokhi and Ahmadi 2023] uses the COOJA simulator, integrated with the Contiki operating system, to evaluate the energy consumption of application-layer protocols in Internet of Things environments, including MQTT, MQTT-SN, CoAP, and HTTP. In this context, each simulation node corresponds to a real hardware platform executing native Contiki code and communicating through a simulated network, enabling the analysis of protocol behavior under different operating conditions. The approach allows the comparison of application protocols under equivalent hardware and network conditions, supporting the identification of more energy-efficient solutions. Simulation at the hardware and operating system level, with the execution of real device code, enables detailed observation of consumption and performance aspects, contributing to the prior evaluation of strategies before deployment in real environments.

Overall, the analyzed simulators focus on communication infrastructure, protocols, or computational performance in IoT environments. In contrast, IoTEventSim explicitly models individual device behavior, including sensor data generation, energy consumption, and dynamic events that modify their states, while enabling real-time monitoring and controlled analysis of adverse scenarios in cloud—fog—edge architectures at scales of thousands of monitored devices. Table 2 summarizes the main aspects that distinguish IoTEventSim from related work.

Table 2. Related work comparison.

Simulator	IoT Protocol	Device Config.	Sensor Data	Network Model	Battery Model	Position Model	Event-driven	Dynamic events	CFE Support	Devices	Interface
[Gupta et al. 2017]	✗	†	†	✓	✓	✗	✓	✗	✓	64	GUI
[Jha et al. 2020]	✓	†	†	✓	✓	✓	✓	✗	✓	50	GUI
[Pereira et al. 2021]	✓	†	†	✓	✗	✗	✓	✗	✓	100	GUI
[Portabales and Nores 2018]	✓	✗	✗	✓	✗	✗	✗	✗	✗	06	✗
[Kirsche and Schnurbusch 2014]	✗	✗	✗	✓	✗	✗	✓	✗	✗	—	GUI
[Shahrokhi and Ahmadi 2023]	✓	✓	✓	✓	✓	✓	✗	✗	✗	01	GUI
IoTEventSim (this work)	✓	✓	†	✓	✓	✓	✓	✓	✓	3000+	Web

Legend: ✓= supported; †= partially supported; ✗= not supported.
Dynamic events: transient and permanent events. CFE: Cloud–Fog–Edge.

6. Conclusion

In this paper, we presented IoTEventSim, an event-driven simulator for large-scale IoT applications in cloud–fog–edge environments. The proposed approach combines per-device modeling, runtime event injection, and native support for MQTT and CoAP to represent heterogeneous and dynamic operating conditions. By explicitly modeling communication behavior, energy depletion, and device-level state transitions, IoTEventSim enables controlled experimentation under realistic adverse scenarios while maintaining a practical level of abstraction for large deployments.

The experimental evaluation indicates that the simulator reproduces target latency and drop behaviors with high fidelity and scales to thousands of concurrently monitored devices without saturating CPU or network resources in the evaluated setup. These results support the use of IoTEventSim as a practical environment for performance analysis, robustness testing, and fault experimentation before real-world deployment. As future work, we plan to extend the event model with periodic event schedules, broaden gateway-level configuration options (e.g., timeouts and queue policies), and support dynamic reassignment of devices across gateways to better represent adaptive edge infrastructures.

Acknowledgments

This work was partially supported by the STIC-AmSud Program (France–South America Cooperation in Information and Communication Science and Technology), funded by CAPES under grant 88881.985518/2024-01 (project ITERATION-D).

References

Adjih, C., Baccelli, E., Fleury, E., Harter, G., Mitton, N., Noel, T., Pissard-Gibollet, R., Saint-Marcel, F., Schreiner, G., Vandaele, J., et al. (2015). FIT IoT-LAB: A large scale open experimental IoT testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 459–464. IEEE.

Almheiri, A. and Maamar, Z. (2021). Iot protocols – mqtt versus coap. In *Proceedings of the 4th International Conference on Networking, Information Systems & Security*, NISS '21, New York, NY, USA. Association for Computing Machinery.

Almutairi, R., Bergami, G., and Morgan, G. (2024). Advancements and challenges in IoT simulators: A comprehensive review. *Sensors*, 24(5):1511.

Atlam, H. F., Walters, R. J., and Wills, G. B. (2018). Fog computing and the internet of things: A review. *Big Data and Cognitive Computing*, 2(2).

Calheiros, R., Ranjan, R., Beloglazov, A., De Rose, C., and Buyya, R. (2011). Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software Practice and Experience*, 41:23–50.

Fabri, E., Lima, M., Rodrigues, M., Silva, S., and Rangel, M. (2025). A revoluÇÃo da internet das coisas (iot): Conectando o mundo fÍsico e digital. *Revista Missioneira*, 27:173–183.

Gupta, H., Vahid Dastjerdi, A., Ghosh, S. K., and Buyya, R. (2017). ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296.

Jha, D. N., Alwasel, K., Alshoshan, A., Huang, X., Naha, R. K., Battula, S. K., Garg, S., Puthal, D., James, P., Zomaya, A., et al. (2020). Iotsim-edge: a simulation framework for modeling the behavior of internet of things and edge computing environments. *Software: Practice and Experience*, 50(6):844–867.

Kirsche, M. and Schnurbusch, M. (2014). A new ieee 802.15.4 simulation model for omnet++ / inet. *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques (SIMUTools 2014)*.

Kuchuk, H. and Malokhvii, E. (2024). Integration of iot with cloud, fog, and edge computing: A review. *Advanced Information Systems*, 8:65–78.

Pereira, R. S., Prazeres, C. V. S., Barbosa, M. T. M., Barros, E. B. C., and Peixoto, M. L. M. (2021). Iotfogsim: Um simulador orientado a eventos para avaliaçao de aplicaçoes baseadas em iot-fog-cloud. In *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*, pages 25–32. SBC.

Portabales, A. R. and Nores, M. L. (2018). Dockemu: Extension of a scalable network simulation framework based on docker and ns3 to cover iot scenarios. In *International Conference on Simulation and Modeling Methodologies, Technologies and Applications*.

Qayyum, T., Malik, A., Khattak, M., and Khan, S. (2018). Fognetsim++: A toolkit for modeling and simulation of distributed fog environment. *IEEE Access*, PP:1–1.

Seoane, V. et al. (2021). Performance evaluation of coap and mqtt in iot systems. *Computer Networks*, page 108338.

Shahrokhi, A. and Ahmadi, M. (2023). Power evaluation of IoT application layer protocols. In *2023 7th International Conference on Internet of Things and Applications (IoT)*, pages 1–7. IEEE.