EDICT: simulation of edge interactions across IoT-enhanced environments

Houssam Hajj Hassan, Georgios Bouloukakis, Ajay Kattepur, Denis Conan, Djamel Belaïd

EDICT: Simulation of Edge Interactions across IoT-enhanced Environments

Houssam Hajj Hassan[1], Georgios Bouloukakis[1], Ajay Kattepur[2], Denis Conan[1], Djamel Belaïd[1]
[1]*SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, France*
{houssam.hajj_hassan, georgios.bouloukakis, denis.conan, djamel.belaid}@telecom-sudparis.eu
[2]*Ericsson AI Research, India*, ajay.kattepur@ericsson.com

***Abstract*—This paper presents EDICT, a tool for simulating Edge interactions in IoT-enhanced environments. Recently, ML and AI-based techniques have gained prominence to solve IoT-related challenges. However, such models require large and diverse datasets to perform well. Finding real-world datasets that capture the performance of IoT systems is a challenging task due to the cost of deploying devices and instrumenting environments, as well as privacy/security concerns. This task becomes more challenging when datasets for specific situations (e.g., overloaded system, emergency scenarios) are needed. EDICT enables IoT systems designers to evaluate the performance of their IoT systems at design time. EDICT is capable of generating performance metrics datasets for specific instances of IoT-enhanced environments under different configuration parameters. To support runtime adaptation of smart environments, EDICT enables rapid performance prediction using ML techniques.**

***Index Terms*—Simulation, Smart Environments, QoS, Data Exchange**

I. Introduction

With the advent of Internet of Things (IoT) devices and supporting technologies, spaces (e.g., buildings, homes) are becoming smarter and interconnected. Edge-based infrastructures of today's sensorized environments include: IoT devices to sense physical phenomena or receive actuation commands; software components to process raw data and provide semantically enriched data; message brokers to exchange data; Edge servers as hosting machines; and the networking infrastructure. IoT applications operating over such Edge-based infrastructures provide services to improve people's daily activities, life quality, and public safety. Such applications are composed of different Quality of Service (QoS) requirements such as end-to-end latency bounds, throughput, and tolerated message loss rates. IoT systems designers have to tune the data exchange infrastructure to ensure that the QoS requirements of applications are satisfied, while proactively addressing dynamic situations that may require additional resources. Currently, designing such a distributed IoT system is a manual "by-experience" process that is error-prone and time consuming.

Commercial and open-source simulation tools [1], [2] have been developed to facilitate such tasks prior to system deployment. Network emulators [3] can be used as well to emulate networking events and evaluate the performance of networking infrastructures. Existing simulators usually provide graphical/command line interfaces, or scripts to create a virtual representation of the IoT system and run simulations. IoT designers have to spend considerable effort in learning how to use a simulator for representing an IoT system. In addition, it is often hard to define the desired output (e.g., performance measurements, energy consumption). Finally, creating variations of an IoT system that represents multiple situations, often requires complex simulation deployments. Hence, there is a need for a simulation tool that enables IoT designers to quickly simulate IoT systems designed based on standard IoT representations and data exchange architectures.

This paper presents EDICT, a simulation tool for evaluating the performance of Edge interactions in IoT-enhanced environments. EDICT abstracts the hardware and network implementation details, as well as the application-layer interactions of an IoT system as a *queueing network* [4] (also called *generic QoS model*). First, EDICT leverages the standard NGSI-LD (*Next Generation Service Interfaces-Linked Data*) protocol specification [5] to represent systems deployed in IoT-enhanced environments. Second, Edge interactions are represented based on the IoT-suitable publish/subscribe interaction paradigm [6]. These are given as input to the EDICT generic QoS model to be instantiated for simulating and evaluating the performance of Edge interactions. Multiple QoS model instances can be used to represent different situations (e.g., number of devices/applications), or QoS configuration parameters (e.g., applying priorities and resource allocation policies) can be applied to provide designers with a performance metrics dataset that can be used for design time system tuning. In addition, to support runtime adaptation of IoT systems, EDICT provides designers with a QoS prediction component that rapidly generates performance metrics when changes in the IoT system occur. The key contributions of this paper are:

- A standard representation of IoT data exchange in smart environments using an extended NGSI-LD model and a publish/subscribe architecture (§IV).
- A simulation tool for generating multiple customizable datasets containing performance metrics of Edge interactions in distributed IoT-enhanced environments (§IV).
- A QoS prediction mechanism for rapid runtime adaptation (§V).

In §II, we compare EDICT against existing IoT simulators. An overview of the EDICT architecture is presented in §III. EDICT's usefulness for design time system tuning and runtime adaptation is presented in §V. We conclude the paper with a

look towards future extensions of EDICT in §VI.

II. Related Work

This section provides a comparative summary of EDICT against a number of existing IoT simulators. In particular, based on Table I, we compare the IoT layer abstraction that the tools provide, the scope, simulation domain, and application domain for which the tools are intended to be used, and the input and output format of the simulators. These formats determine the usability of the tools and how easy it is to integrate the simulators in automated system tuning solutions. An extensive evaluation of related work is provided in [11].

While the presented tools are powerful enough for simulating IoT systems, we identify a number of challenges. IoT designers have to learn and get used to the specifics of the simulator they choose to use. Tools like CupCarbon [1] require users to learn a new scripting language to program sensor nodes. This requires spending considerable time learning how to use the tool to get the desired output, hence hindering the **usability** of the tool. In contrast, as shown in § III, EDICT users only need to provide the NGSI-LD description of their system. EDICT then automatically simulates Edge interactions in the provided environment and outputs simulation results as a dataset. In addition, using existing tools for simulating the same environment with **multiple configuration parameters** usually requires creating a new simulation with new instances of IoT devices and applications. This can be time consuming especially when IoT designers need to evaluate the performance of their systems in different situations. EDICT provides simulation results per subscription for multiple configuration parameters of the IoT system in one iteration and as a `CSV`-based performance metrics dataset (§IV-C, §V-A). This makes it easy to integrate EDICT in **automated system tuning** approaches, especially with the recent advances in ML and AI techniques that drive autonomous IoT systems. Existing simulators provide results visually ([7]–[10]) or provide them in files that require further processing [3], thus adding complexity for interpreting and analyzing simulation results. Finally, existing tools do not provide support for **runtime adaptation** of IoT systems. If changes occur in the Edge infrastructure, IoT designers have to re-run the simulations; this process is time consuming and cannot be done at runtime. A distinguishing feature of EDICT is that it provides a *QoS prediction* component that can be integrated in a runtime adaptation approach for timely readaptation of IoT systems.

III. EDICT Overview

Fig. 1 shows the high-level architecture of EDICT. To represent characteristics of smart environments, we leverage the NGSI-LD specification [5]. In particular, EDICT uses existing data models for the standard representation of smart environments as NGSI-LD *entities* that are further enhanced with IoT aspects based on our proposed application categories and QoS requirements. More details about the proposed NGSI-LD representation are presented in §IV-A. The *IoT-enhanced NGSI-Model* can be instantiated to represent the interactions

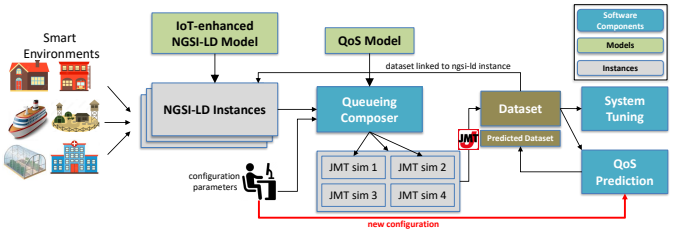

Fig. 1: EDICT Architecture

of smart environments to be simulated. NGSI-LD instances of smart environments and their interactions are represented using JSON-LD notation, a JSON-based serialization format.

To simulate interactions between IoT-enhanced NGSI-LD entities, we rely on Queueing Networks [4]. In particular, we define a *generic QoS Model* that consists of series of queues to model IoT interactions in smart environments. Such queueing networks can be composed on-the-fly depending on the corresponding smart environment instance and the ongoing situation (e.g., applications activated for emergency response)—more details in § IV. Finally, the EDICT end-user can configure simulations by applying parameters such as priorities, dropping rates, routing policies, network allocation policies, that can improve the end-to-end performance (i.e., latency and throughput) of IoT interactions. Depending on the given NGSI-LD instance and the configuration parameters, the *Queueing Composer* instantiates the composed code artifacts that, when executed, simulate end-to-end IoT interactions (from the IoT sensor to the IoT application) using the open-source queueing simulator *Java Modeling Tools* (JMT) [12].

IV. IoT System Representation

To build a tool for simulating IoT interactions at the Edge of any smart environment requires relying on a generic architecture to represent: (i) the environment and its IoT capabilities; and (ii) the interactions between IoT entities. Related generic models are presented in this section.

A. IoT Domain Model

NGSI-LD is a data model for the standard representation of smart environments that provides an API for publishing, querying, and subscribing to context data. Existing models are focused on describing the functional semantics of IoT systems and not QoS semantics such as requirements of applications and categories. In addition, entities that represent observations and the processing of raw data are missing. We extend the NGSI-LD representation of an IoT system by introducing Entities and Relationships as shown in Fig. 2. First, an existing `Device` entity is extended by adding attributes and relationships with new entities to represent the general function of IoT devices. In addition, we create a new `SmartEnvironment` entity that can either extend existing NGSI-LD models (e.g., Building) or define a new environment. IoT devices deployed in the smart environment can either be sensors that sense physical characteristics and generate `Observations` or `ActuationCommands`, or actuators that receive `ActuationCommands`. An `Observation`

Tool	IoT Layer Abstraction	Scope	Input	Automated System Tuning	Simulation Domain	Support for Readaptation
DPWSim [2]	Fog/Cloud	Deployment of web services on IoT devices	GUI	Through graphs	Generic	No
iFogSim [7]	Fog/Edge	Resource management / application scheduling	GUI / API calls / JSON	Through graphs	Generic	No
IoTSim [8]	Cloud	Big data processes	API calls	Through graphs	Generic	No
IoTNetSim [9]	Cloud/Edge/Network	Simulation of IoT services	API calls	Through graphs	Generic	No
IoTSim-Edge [10]	Edge	Mobility modeling / resource provisioning	GUI / JSON Configuration	Through graphs	Healthcare / Transportation	No
CupCarbon [1]	Network	Distributed algorithms / Environmental scenarios	GUI	Through CSV files	Smart City	No
NS-3 [3]	Network	Discrete network event simulation	API calls	Through PCAP / ASCII files	Generic	No
EDICT	Edge interactions over pub/sub	Simulation of Edge interactions in IoT environments	GUI / NGSI-LD models	Through CSV files	Generic	Yes

TABLE I: Comparison of Existing IoT Simulation Tools

represents a quantitative measurement in a specific physical space. For example, a temperature sensor deployed in `room324` would generate the observation `"temperature in room324"`. Thus, an application can easily receive the necessary data by specifying the type of data needed with the name/identifier of the space.

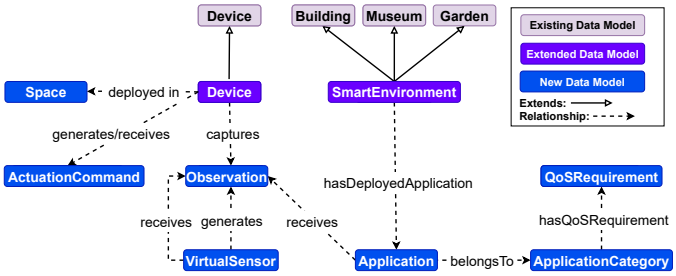

Fig. 2: NGSI-LD Data Model for IoT-enhanced Environments

We also create a new Entity type `VirtualSensor`, which represents software components that process raw IoT data to produce high-level measurements. Unlike physical sensors, virtual sensors take as input one or more `Observation`, process the received data, and output a new type of `Observation`. Finally, to represent different categories and QoS semantics of applications, we classify them into `ApplicationCategories` with each category specifying `QoSRequirements` (e.g., end-to-end latency, energy consumption, drop rates) that have to be met. The NGSI-LD model presented in Fig. 2 can be instantiated to describe an IoT system in diverse smart environments. Our extended NGSI-LD data models along with examples of smart environments can be found in [11] and at https://github.com/SAMSGBLab/edict--datamodels.

B. IoT Data Exchange Model

Existing IoT deployments consist of devices employing network access protocols (e.g., Z-Wave) for data collection in IoT gateways, as well as application-layer protocols (e.g., CoAP, MQTT) that forward data to message brokers (e.g., EMQx) for data processing and dissemination. EDICT relies on the publish/subscribe paradigm [6] to represent IoT interactions at the Edge of smart environments, where IoT devices and applications interact via a message broker. We develop a generic QoS model [11] that abstracts the underlying network, different application categories, and application instances that interact via the message broker with the IoT devices. The broker manages all the traffic at the Edge, and for this reason, message handling strategies are applied at the broker.

As depicted in Fig. 3A, ***IoT devices*** act as publishers that produce data related to the environment sensed; these data are encapsulated into messages that are tagged with a ***topic*** name for routing. Messages of a topic can be captured from both IoT devices and virtual sensors, while topics are characterized from the observation type (e.g., temperature data), the space in which a device is deployed (e.g., room 2065), the average message size and the message frequency. Such information can be found in the NGSI-LD instance describing a smart environment. Virtual sensors, actuators, and applications *subscribe* to receive relevant messages using topic-based subscription filters. Virtual sensors receive input from one or more sources of messages, process the received messages (e.g., using AI-based algorithms), and generate output messages that are sent back to the message broker. Note that such processed messages can be received by applications or other virtual sensors. IoT applications can be characterized by heterogeneous QoS requirements and they can be grouped into application categories (RT: real-time, ST: streaming, TS: transactional, AN: analytics, EM: emergency) depending on their QoS requirements, as specified in [13]. A message broker system is responsible for forwarding data from IoT devices to the corresponding application recipients. IoT systems designers can configure the data exchange infrastructure according to their needs. For instance, they can assign priorities to some topics via a *Data Flow Management* component, or/and specify a network resources allocation policy (e.g., max-min policy [14]) via a *Network Resource Management* component. Thus, IoT systems designers can use different configurations to tune the system and ensure the desired QoS performance at runtime. A formal representation of IoT date exchange and our *QoS Model* (Fig. 3B) is available in [11].

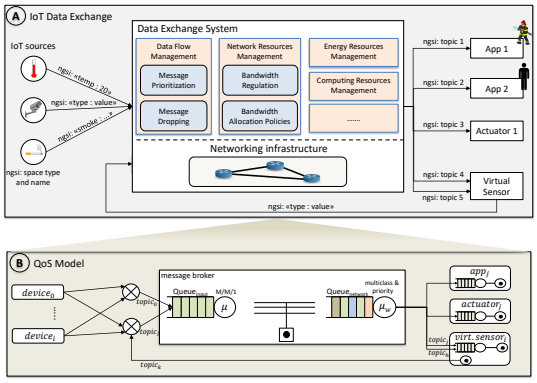

Fig. 3: IoT Data Exchange Model

C. IoT-aware Queueing Composition

The generic model defined in §IV-B provides all the basic elements to compose a queueing network, as well as the necessary configuration parameters used by application designers (network policies, priorities, etc.) to parameterize the composed queueing network. For each situation $s \in S$ and configuration parameters $p \in P$, the function $f(s,p)$ instantiates the QoS model to compose the queueing network rep-

IoT-enhanced Environment Properties				*QoS Requirements*		
categories	**# apps**	**# subs.**	**bandwidth**	**resp. time**	**throughput**	**drop rate**
AN	6	21	650 MB /s	best effort	best effort	best effort
RT	9	17		<400 ms	≥384 KB / s	0%
TS	6	12		<4 s	-	0%
ST	9	10		<2 s	≥384 KB / s	<2%
Total	30	60	650 MB/s			

TABLE II: Experimental Setup

resenting the environment under situation s and configuration parameters p. Depending on the IoT environment and deployed devices/applications, EDICT provides the basic queueing network, defined as a ***default network***, which models the default performance of the IoT data exchange system. Then, EDICT offers different configuration parameters (priorities, dropping, network policies) to model the system's performance.

To enable such dynamic queueing network composition, we leverage the JMT simulator [12]. JMT is an open-source suite of applications that offer a comprehensive framework for system modelling with analytical and simulation techniques, and performance evaluation. While JMT JSIMgraph provides a graphical user interface to design queueing models, we use JMT's API to dynamically compose and run the simulations. As depicted in Fig. 1, we design a *Queueing Composer* that instantiates the corresponding JMT queueing representation of an IoT environment's situation and parameterization. The composer takes as input (i) the NGSI-LD instance of an IoT-enhanced environment (see §IV-A), and (ii) the generic QoS model (see §IV-B). Through calls to the JMT Library, the composer then creates and simulates the queueing network that represents the IoT-enhanced environment. The dataset generated by EDICT can be leveraged by automated approaches for system tuning. For example, the PlanIoT [13] framework relies on performance metrics datasets to automatically manage IoT data flows using AI planning. Details about the queueing network composition process are presented in [11].

V. EDICT Evaluation

This section presents the evaluation of EDICT by (i) demonstrating how IoT designers can generate a dataset that captures Edge interactions in IoT-enhanced environments to automatically tune their systems (§V-A), and (ii) showing how EDICT's QoS prediction component enables rapid adaptation when changes happen to the Edge infrastructure (§V-B). Implementation details related to the EDICT prototype can be found in [11]. The EDICT code is publicly available on https://github.com/SAMSGBLab/edict.

A. Design-time System Tuning

We demonstrate the utility of EDICT by simulating IoT interactions in a smart office building. Table II presents details related to the IoT system deployed. We rely on the work presented in [15] to set the values for the message sizes and publishing frequencies for the devices. We use an ETSI standard [16] to compose the QoS requirements for the application categories. However, IoT designers can use other existing standards or define their own requirements.

Using the setup displayed in Table II and the NGSI-LD model presented in §IV-A, we create the corresponding NGSI-LD instance of the smart building and feed it to EDICT to generate the performance metrics dataset that captures the Edge interactions in the IoT system. The simulation results show an 82% utilization of the system. Fig. 4 shows the average end-to-end latency for each application category for some of the configurations applied to the system: (i) the default configuration, where all network resources are used to transmit all data without setting any priorities or drop rates, (ii) using the max-min network allocation policy [14], (iii) prioritizing time-sensitive applications based on the QoS requirements (RT applications), and (iv) setting a dropping rate of 2% for loss-tolerant applications based on the QoS requirements (AN and ST applications). Running simulations with multiple configuration parameters allows IoT designers to know which configuration is best suited for their needs. For instance, based on the results of Fig. 4, IoT designers can consider that they have to prioritize flows belonging to RT applications (prioritize RT) to satisfy all applications' QoS requirements. The dataset generated by EDICT can be used by automated system tuning approaches to save IoT designers the tedious effort needed to manually test different configurations after deployment.

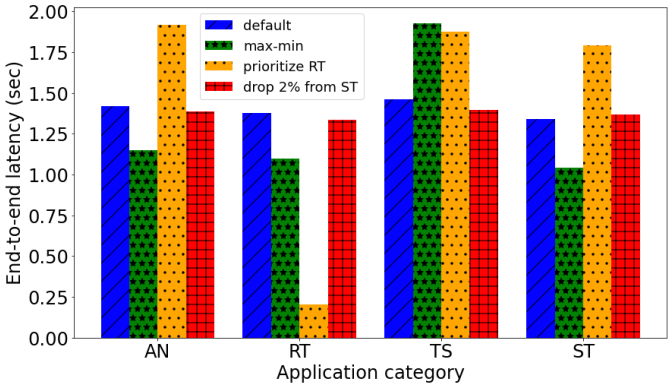

Fig. 4: End-to-End Latency per Application Category

B. Enabling Rapid Runtime Adaptation

To enable runtime adaptation in smart environments, IoT designers need to know the performance of their systems on-the-fly when changes in the Edge infrastructure occur. EDICT simulations can take up to 5 minutes to converge; this time is too long to perform rapid performance analysis and runtime adaptation. EDICT supports runtime adaptation through an ML-based QoS prediction component that quickly provides insights into the performance of the IoT system when subscriptions or configuration parameters change.

We validate EDICT's QoS prediction mechanism by (i) considering a changing number of subscriptions, and (ii) considering changing configuration parameters. In this section, we consider the following configurations: prioritizing one of the four application categories, prioritizing applications based on their QoS requirements (giving the highest priority to RT, ST, TS, and then AN in this order), and applying a drop rate of 2% for loss-tolerant applications (AN and ST). For a congested system, we apply more aggressive drop rates (5% and 10%). For the sake of comparison, we test four QoS prediction models that EDICT uses to support rapid runtime adaptation: (i) the KNN algorithm, (ii) the Linear Regression algorithm,

	Prediction when adding subscriptions				*Prediction when reconfiguring*			
Dataset size	**KNN**	**LR**	**DT**	**DW**	**KNN**	**LR**	**DT**	**DW**
220	0.035	0.036	0.031	0.035	0.0203	0.021	0.022	0.024
440	0.069	0.191	0.072	0.205	0.235	0.195	0.233	0.162
660	0.101	0.648	0.110	0.706	0.090	0.27	0.066	0.118
880	0.199	2.882	0.193	3.165	0.247	3.973	0.062	1.424
1100	0.195	3.487	0.204	3.875	0.346	7.975	0.202	3.027

TABLE III: Comparison of RMSE (sec) of QoS prediction

(iii) Decision Trees, and (iv) AWS's DataWig library [17], which is a deep learning-based framework to impute missing values in datasets. We compare the RMSE (Root Mean Square Error) and the prediction time needed for the four models.

We start first by evaluating EDICT's predictions when new subscriptions are added. We consider different dataset sizes that have an increasing number of subscriptions. For each dataset, the number of samples is equal to the number of subscriptions times the number of configurations. For example, when simulating a system with 20 subscriptions under 11 configurations, the number of samples is 220. We test EDICT's prediction mechanism on 5 datasets. For each iteration, we train the models on the dataset generated by EDICT, and then use the models to predict the end-to-end latency for new subscriptions that are added to the IoT system. We validate the predicted values by using EDICT to simulate the IoT system with the new subscriptions. Table III shows the RMSE for the four ML models. We notice that KNN ($0.195s$) and Decision Trees ($0.204s$) have the best performance even when the size of the dataset increases (1100). In contrast, the RMSE of the Linear Regression model ($3.487s$) and DataWig ($3.875s$) increases significantly as the dataset size increases (1100). This is caused by the non-linear nature of queueing delays that cannot be captured by these models. In terms of prediction time, Decision Trees achieve the best performance: they can predict metrics for new subscriptions in less than $4ms$. The linear regression model also has a low prediction time of $20ms$—albeit with a much higher error. Even though KNN's prediction time is significantly lower than DataWig's for small datasets, both models tend to have a similar prediction time of $2s$ when the dataset size increases.

Next, we test how well the four models perform when predicting the performance of the IoT system under new configuration parameters. Similarly to the approach above, we test EDICT's predictions on 5 datasets. However, instead of using the models to predict the metrics values when we add new subscriptions, we test how well the models can predict the metrics values for existing subscriptions under different configuration parameters (e.g., applying different drop rates). This task is challenging since the models have to predict values for configurations not seen during training. Again, as Table III shows, KNN ($0.346s$) and Decision Trees ($0.202s$) perform better than DataWig ($3.027s$) and the Linear Regression model ($7.975s$). We notice that the execution time for predicting metrics under new configurations is higher than that for predicting new subscriptions for all models. Decision Trees can predict metrics in $18ms$ and a Linear Regression model takes about $10ms$. KNN and DataWig have a much higher prediction time above $3s$. This is due to the lazy nature of KNN and the fact that DataWig relies on complex neural networks that have a higher prediction time than other models.

VI. Conclusion and Future Work

This paper presents EDICT, a simulation tool for evaluating the performance of Edge interactions in smart environments. EDICT leverages the NGSI-LD information model to represent data exchange in smart environments. We also present a generic QoS model that can be instantiated to create queueing networks that represent the smart environment instances. These queueing networks are simulated to provide a metrics dataset that evaluates the performance of data exchange for multiple situations and configuration parameters of an IoT-enhanced environment. The output dataset can be integrated into an automated system tuning approach by IoT systems designers. To support runtime adaptation, EDICT provides a QoS prediction mechanism that allows designers to get the performance of their systems on-the-fly when changes in subscriptions or configurations occur. Our future work includes adding performance metrics such as the energy consumption of devices. We shall also extend EDICT to support more input types (e.g., RDF), and investigate more QoS prediction techniques to allow faster and more accurate predictions.

References

[1] K. Mehdi *et al.*, "Cupcarbon: A multi-agent and discrete event wireless sensor network design and simulation tool," in *SIMUtools*, Mar. 2014.

[2] S. Han *et al.*, "DPWSim: A simulation toolkit for IoT applications using devices profile for web services," in *WFIoT*, 2014, pp. 544–547.

[3] G. Riley *et al.*, "The ns-3 network simulator," in *Modeling and tools for network simulation*. Springer, 2010, pp. 15–34.

[4] D. Gross *et al.*, *Fundamentals of queueing theory, 4th Ed.* John Wiley & Sons, 2008.

[5] "Context Information Management (CIM) NGSI-LD API V1.4.2," https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.04.02_60/gs_cim009v010402p.pdf, 04 2021.

[6] G. Bouloukakis *et al.*, "PrioDeX: a Data Exchange Middleware for Efficient Event Prioritization in SDN-based IoT systems," *ACM TIOT*, 2021.

[7] H. Gupta *et al.*, "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*, vol. 47, no. 9, pp. 1275–1296, 2017.

[8] X. Zeng *et al.*, "IOTSim: A simulator for analysing IoT applications," *Elsevier Journal of Systems Architecture*, vol. 72, pp. 93–107, 2017.

[9] M. Salama *et al.*, "IoTNetSim: A modelling and simulation platform for end-to-end IoT services and networking," in *UCC*, 2019, pp. 251–261.

[10] D. Jha *et al.*, "IoTSim-Edge: a simulation framework for modeling the behavior of Internet of Things and edge computing environments," *Software: Practice and Experience*, vol. 50, no. 6, pp. 844–867, 2020.

[11] H. Hajj Hassan *et al.*, "EDICT: Simulation of Edge Interactions across IoT-enhanced Environments," Télécom SudParis, Tech. Rep., 2023. [Online]. Available: https://hal.science/hal-04078497

[12] M. Bertoli *et al.*, "JMT: performance engineering tools for system modeling," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, pp. 10–15, 2009.

[13] H. Hajj Hassan *et al.*, "PlanIoT: A Framework for Adaptive Data Flow Management in IoT-enhanced Spaces," in *SEAMS*, 2023.

[14] D. Pan *et al.*, "Max-Min Fair Bandwidth Allocation Algorithms for Packet Switches," in *IPDPS*, 2007, pp. 1–10.

[15] R. Kumar *et al.*, "IoT Network Traffic Classification Using Machine Learning Algorithms: An Experimental Analysis," *IEEE IoT Journal*, vol. 9, no. 2, 2022.

[16] ETSI, "Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); LTE; Services and service capabilities," 3GPP TS 22.105 V15.0.0, Jul. 2018.

[17] F. Biessmann *et al.*, "DataWig: Missing Value Imputation for Tables." *J. Mach. Learn. Res.*, vol. 20, no. 175, 2019.