Received: 30 May 2019 | Revised: 8 November 2019 | Accepted: 11 December 2019

DOI: 10.1002/spe.2787

SPECIAL ISSUE PAPER

IoTSim-Edge: A simulation framework for modeling the behavior of Internet of Things and edge computing environments

Devki Nandan Jha[1] | **Khaled Alwasel**[1] | **Areeb Alshoshan**[1] | **Xianghua Huang**[1] | **Ranesh Kumar Naha**[2] | **Sudheer Kumar Battula**[2] | **Saurabh Garg**[2] | **Deepak Puthal**[1] | **Philip James**[1] | **Albert Zomaya**[3] | **Schahram Dustdar**[4] | **Rajiv Ranjan**[1]

[1]Newcastle University, Newcastle upon Tyne, UK

[2]University of Tasmania, Hobart, Tasmania, Australia

[3]Sydney University, Sydney, New South Wales, Australia

[4]TU Wien, Vienna, Austria

Correspondence
Devki Nandan Jha, Newcastle University, Newcastle upon Tyne, UK.
Email: D.N.Jha2@newcastle.ac.uk

Funding information
Engineering and Physical Sciences Research Council, Privacy aware cloud ecosystems (PACE) and Sustainable urban power supply through intelligent control and enhanced restoration of AC/DC networks (SUPER), EP/R033293/1, EP/T021985/1

Summary

With the proliferation of Internet of Things (IoT) and edge computing paradigms, billions of IoT devices are being networked to support data-driven and real-time decision making across numerous application domains, including smart homes, smart transport, and smart buildings. These ubiquitously distributed IoT devices send the raw data to their respective edge device (eg, IoT gateways) or the cloud directly. The wide spectrum of possible application use cases make the design and networking of IoT and edge computing layers a very tedious process due to the: (i) complexity and heterogeneity of end-point networks (eg, Wi-Fi, 4G, and Bluetooth); (ii) heterogeneity of edge and IoT hardware resources and software stack; (iv) mobility of IoT devices; and (iii) the complex interplay between the IoT and edge layers. Unlike cloud computing, where researchers and developers seeking to test capacity planning, resource selection, network configuration, computation placement, and security management strategies had access to public cloud infrastructure (eg, Amazon and Azure), establishing an IoT and edge computing testbed that offers a high degree of verisimilitude is not only complex, costly, and resource-intensive but also time-intensive. Moreover, testing in real IoT and edge computing environments is not feasible due to the high cost and diverse domain knowledge required in order to reason about their diversity, scalability, and usability. To support performance testing and validation of IoT and edge computing configurations and algorithms at scale, simulation frameworks should be developed. Hence, this article proposes a novel simulator IoTSim-Edge, which captures the behavior of heterogeneous IoT and edge computing infrastructure and allows users to test their infrastructure and framework in an easy and configurable manner. IoTSim-Edge extends the capability of CloudSim to incorporate the different features of edge and IoT devices. The effectiveness of IoTSim-Edge is described using three test cases. Results show the varying capability of IoTSim-Edge in terms of application composition, battery-oriented modeling, heterogeneous protocols modeling, and mobility modeling along with the resources provisioning for IoT applications.

KEYWORDS
edge computing, Internet of Things, simulation, software

1 INTRODUCTION

Advances in Internet of Things (IoT) have a transformative impact on society and the environment through a multitude of application areas, including smart homes, smart agriculture, manufacturing, and healthcare. To achieve this, an ever-increasing number of heterogeneous IoT devices are continuously being networked to support real-time monitoring and actuation across different domains. Cisco predicts that 50 billion IoT devices are going to be connected by 2020 (https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf). Traditionally, the enormous amount of data, generally known as the big data, are sent to the cloud by IoT devices for further processing and analysis. However, the centralized processing in cloud is not suitable for numerous IoT applications due to the following reasons: (i) some applications require close coupling between request and response;[1] (ii) Delay incurred by the centralized cloud-based deployment is unacceptable for many latency-sensitive applications;[2] (iii) there is a higher chance of network failure and data loss, and (iv) sending all the data to cloud may drain the battery of the IoT device at a faster rate.[3] To address the above issues, numerous concepts have been proposed, which offer cloud-like resources near to the edge of the network. The most prominent of these are fog computing, mobile edge computing, and cloudlets.[4] However, all these systems are considered to be a part of *edge computing* system. Although some of the existing literature distinguishes between these technologies, following Reference 4, this article abstracts both to these, relatively new, paradigms of computing to be part of edge layer.

The introduction of edge computing addresses these issues by providing the computational capacity in a near proximity to the data generating devices. Smart edge devices such as Smart phone, Raspberry Pi, UDOO board, and so on support local processing and storage of data on a widespread but smaller scale. However, the constituent devices in edge computing are heterogeneous as each one may have specific architecture and follows particular protocols for communication. Unlike the cloud where the location of a datacenter (data center in the UK) is fixed, edge and IoT devices can be mobile and change location frequently. In addition to this, edge and IoT devices are powered by batteries, solar (or a combination of the two), or continuously connected to an external power supply as compared to cloud datacenter which is always connected to a stable power source. To exploit the advantages offered by edge computing, it is necessary to understand the features and capabilities of edge and IoT devices along with their composition in a proposed IoT data analysis framework. The diversity of underlying IoT and edge devices, data types and formats, communication mediums, application scope, functional complexity, and programming models makes the evaluation very challenging and time consuming.

Evaluating the framework in a real environment gives the best performance behavior; however, it is not always possible as most of the test frameworks are in development. Even if the infrastructure is available, it is very complex to perform the experiment as setting it up requires knowledge of all the associated IoT and edge devices, which is not intuitive. Running multiple experiments to test a desired framework requires reconfiguration of multiple devices, and changes to required parameters quickly become untenable due to the volume of changes needed. In addition, performing the experiment in the real environment is very expensive due to the setup and maintenance cost incurred. Since the real environment is dynamic, it is very hard to reproduce the same result for different iterations, which may lead to misinterpretation of the evaluation. All these challenges hinder the use of real environment for benchmarking the edge computing environments. To overcome this issue, another feasible alternative is the use of *simulators*. The *simulators* offer a window of opportunity for evaluating the proposed hypothesis (frameworks and policies) in a simple, controlled, and repeatable environment. A simulation environment must mimic the key complexity and heterogeneity of real networks and support multiple scenarios that affect real IoT deployments.

1.1 Challenges

Simulating and modeling a realistic IoT scenario is very challenging due to various reasons[5] such as (i) variety of IoT devices need to be combined with edge device and cloud to satisfy the requirements of an application; (ii) modeling networking graph between diverse type of IoT and edge computing device in an abstract manner can be very challenging; (iii) modeling data and control flow dependencies between IoT and edge layers to support diverse data analysis workflow structure is nontrivial; (iv) capacity planning across edge computing layer is challenging as it depends on various configuration parameters, including data volume, data velocity, upstream/downstream network bandwidth, to name a few; (v) the communication between IoT and edge device is very different from cloud datacenter communication, which are generally based on wired and/or wireless protocol. The connectivity between IoT and edge computing layers, as we discuss later in the article, can be diverse. Hence, they are very difficult to model in an abstract way while not losing the expressiveness, that is, lower level details related to protocol latency, impact of protocols on battery discharge rate of underlying IoT device, and so on; (vi) mobility is another important parameter of IoT devices as sensors embedded to many physical devices are moving. Since the range of edge device is limited, the movement of sensor may leads to handoff. Also, the data sent to an edge device for processing may not be in the current range of IoT device. Thus, for receiving the processed data, an edge-to-edge communication is required. Modeling the mobility and handoff for a large number of IoT devices with varying velocity is very challenging; (vii) dynamicity of IoT environment leads to addition and removal of IoT and edge devices very frequently. This may be caused by numerous factors, for example, device failure and network link failure. Modeling the scalability of IoT devices with heterogeneous features at a fast rate is very challenging; and (viii) since IoT environment is an emerging area, new applications might be developed in future. It is very important for a simulator to allow users to customize and extend the framework based on their requirements. Making a general simulator that allows easy customization is very challenging.

Different simulators are proposed in the literature. Simulators such as CloudSim[6] and GreenCloud[7] are specific for cloud environment; however, EdgeCloudSim[8] and iFogSim[9] are proposed for edge computing environment. However, to the best of our knowledge, we could not find any simulator that addresses all the above challenges.

1.2 | Contributions

This article aims to build a novel simulator, IoTSim-Edge, that allows users to evaluate the edge computing scenario in a easily configurable and customizable environment. IoTSim-Edge is built on the existing simulators, which tries to capture the complete behavior of IoT and edge computing infrastructure development and deployment. Especially it covers all the above discussed challenges in a seamless manner. In addition, the proposed simulator can be easily used to analyze various existing or futuristic IoT applications. In particular, the proposed simulator is able to model the following scenarios.

- New IoT application graph modeling abstraction that allows practitioners to define the data analytic operations and their mapping to different parts of the infrastructure (eg, IoT and edge) (see Section 4.1).
- Abstraction that supports modeling of heterogeneous IoT protocols along with their energy consumption profile. It allows practitioners to define the configuration of edge and IoT devices along with the specific protocols they support for networking. Details are presented in Section 4.2.
- Abstraction that supports modeling of mobile IoT devices (see Section 4.3). It also captures the effect of handoff caused by the movement of IoT devices. To maintain a consistent communication, IoTSim-Edge supports a cooperative edge-to-edge communication that transfers the processed data of the respective IoT device by one edge via another edge.

Outline. This article is organized as follows. Section 2 presents a brief background of IoT infrastructure environment and the architecture of edge computing. An illustration of the architecture of our proposed simulator, IoTSim-Edge, is given in Section 3. It also explains the implementation details of IoTSim-Edge. Three case studies evaluating IoTSim-Edge is presented in Section 4, while the recent related work comparing IoTSim-Edge with the existing simulator is given in Section 5. Section 6 concludes the article, giving some future work suggestions.

2 | IoT AND EDGE COMPUTING

This section discusses the background information of the IoT environment in terms of the modeling challenges. It also gives the general architecture of edge computing considered for modeling by the proposed simulator.

2.1 | IoT environment

IoT can be defined as the "sophisticated sensors and chips which are embedded in the physical things, objects and living beings surrounding us and connected through the Internet to monitor and control the connected things".[10] Numerous IoT applications improve our daily lifecycle in different vertical of domains such as smart homes, smart health care, industry 4.0, and disaster management. IoT functionality is delivered by six main elements, namely, sensing, identification, communication, computation, services, and semantics.[11] IoT devices sense the environment while scattered ubiquitously capturing the physical and environmental information. They are identified based on the application requirements and techniques employed to implement the applications. The computation process is distributed across IoT device, edge and cloud datacenter based on the desired functional, and quality of service (QoS) parameters of the application. To achieve this, data are sent from IoT device to edge and further from edge to cloud using different communication protocols. The computational result can be used to make some decisive operation to achieve the desired application process.

Consider a simple example of smart home that controls all the devices of a home and eases the life of the inhabitant. The IoT devices are sensors embedded to all the devices such as refrigerator, heater/cooler, light bulb, and car, while the edge devices are gateways and mobile phones. The smart home system uses private cloud datacenter resources. Home devices are connected to the gateway using light-weight protocol and bluetooth low energy (BLE) using constrained application protocol (CoAP) for data transmission. Mobile phone is connected to 4G, while gateways are connected to Wi-Fi for data transmission to private cloud. When the resident person leaves for office, the smart home system automatically switches off the light bulbs and heater/cooler. The system also checks the refrigerator for available egg or milk and sends a message to the person to bring those things. Based on the location information obtained from the person's phone and car, the system restarts the bulbs and adjusts the room temperature.

Modeling such a realistic IoT application requires a combination of sensors, actuators, and edge devices on a large scale with different operating environments. It is a complicated task due to the heterogeneous characteristics of IoT and edge devices and requires continuous optimization for resource provisioning, allocation, migration, and fault tolerance during the application processing. Again, the implementation is specific for only one application. Giving a generic model for IoT application that can be modeled for any IoT application requires a level of abstraction along with specific details for that application. The main challenges are in terms of modeling application composition, network protocols, mobility of IoT devices, and energy consumption in the form of battery drainage as discussed below.

2.1.1 | Application composition

An IoT application consists of a sequence of operations performed on sensed data. It can be represented in variety of ways; however, this article follows Reference 12 to represent an IoT application as a directed acyclic graph (DAG) of microelements (MELs). Each MEL is an abstract component of the application that represents the resources, services, and data altogether in the form of microservice, microdata, microcomputing, or microactuation.[12] Modeling an application as a DAG of MEL is very challenging as we need to encapsulate multiple components together. Also, the sequence of MEL is very important as it represents the data and control flow at the abstract level.

2.1.2 | Communication protocols

In the IoT environment, network connectivity and messaging protocols play a significant role in the communication. By the inherent characteristic of the IoT environment, it has a complex network interaction between different IoT environment components. Table 1 shows the common communication protocols employed by the IoT environment. Based on the distance, type of devices, and specified constraints, any of the given protocol can be utilized by the IoT and edge devices for data transfer. Mobile devices need to leverage different protocols as compared to static ones. Modeling these protocols within the application graph is very challenging.

In addition, various application-level messaging protocols are available to facilitate the data transfer from sensors to edge devices and further to cloud servers. Table 2 briefs few commonly used protocols for this purpose. Transferring data using any of these protocols affect the system performance in significantly different ways. A single messaging protocol will not be able to satisfy the requirements of complex IoT use case. Hence, it is required to share different protocols for different devices and at different layers. Modeling these scenarios with handshaking between various protocols is very challenging

TABLE 1 Overview of different communication protocols

Protocol	Data Rate	Distance	Power Efficiency	Reliablity	Cost	Service
BLE	Approx. 0.27 Mbps (M)	Approx 100m	L	H	L	Wearable devices and smart connected devices
Wi-Fi	Approx. 54 Mbps (H)	Approx. 50 m	M	M	L	Home IoT, office IoT, Smart cities
4G LTE H	Approx 12 Mbps (L)	Large	H	H	H	Agriculture, industries, transportation, fleets
Zigbee	Approx. 250 kbps (L)	Approx. 100 m	L	H	L	Indoor asset tracking home automation
Long Range (LoRa)	Approx. 50 kbps (L)	Several miles	L	H	M	Smart city, energy management, supply chain management
Sig Fox	Approx. 1 kbps (VL)	Several miles	VL	H	M	Environmental sensors, smart meters
NFC	Approx. 42 kbps (M)	Approx. 20 cm	VL	H	VL	Contactless payment transaction, local asset tracking

Note: L: low, M: medium, H: high, VS: very short, VL: very low.
Abbreviations: BLE, bluetooth low energy; IoT, Internet of Things.

TABLE 2 Overview of different application-level messaging protocols

App. layer Protocol	Restful	Trans. layer Protocol	QoS	Architecture	Security	Header Size (Bytes)	Sync.
CoAP	Yes	UDP	Yes	Pub-Sub, Req-Res	DTLS	4 (minutes)	async/sync
XMPP	No	TCP	No	Pub-Sub, Req-Res	SSL	—	async
MQTT	No	TCP	Yes	Pub-Sub	SSL	2	async/sync
AMQP	No	TCP	Yes	Pub-Sub	SSL	8	async
HTTP	Yes	TCP	No	Req-Res	SSL	—	async

Abbreviations: AMQP, advanced message queuing protocol; CoAP, constrained application protocol; MQTT, message queue telemetry transport; QoS, quality of service; SSL, secure sockets layer; TCA, transmissionm control protocol; XMPP, extensible messaging and presence protocol.

2.1.3 | Mobility of IoT devices

IoT devices embedded with cars or smart phones support mobility to assist users in more flexible ways. Considering the range of edge device as fixed, mobile IoT device can move from the range of one edge device to another causing handoff. The handoff may be hard or soft depending on the speed of IoT device and the signal range of edge device.[13] In order to simulate the mobility in a realistic way, we need a number of features, for example, IoT device speed and acceleration/deceleration, motion path, edge range intersection, and topological maps.[14] Incorporating all these features for the realistic mobility simulation is a very complex task because of a large number of data points with highly dependent characteristics and relationships. Moreover, the data transfer may fail at any time when the IoT device is moving from one location to another location due to the weak strength of the signal.

2.1.4 | Battery drainage

Most of the IoT devices are powered by battery that is limited and need to be recharged at particular period of time. It is very important for these devices to sustain the battery for longer duration especially for application where it is not easy to recharge, for example, sensor in river or at a disaster place (earthquake, landslip, etc.). Sustaining battery for longer time saves a lot of cost, which is vital for every application. Sending data at different rates and using different communication protocols causes the battery drainage at different rates. Therefore, it is very important to monitor the battery consumption in different case scenarios. Simulating the battery drainage of IoT device is very complex task because of the nonlinear effect while drainage of the battery and the fact that each device hardware supports different network connectivity and communication protocols. Hence, providing multilevel abstraction for energy characterization with fine accuracy and tight energy consumption bounds for different devices based on various factors such as hardware, sensing requests, and communication protocols, which is a complex task.

2.2 | Architecture of IoT-Edge computing

Figure 1 represents the architecture of IoT-Edge computing. The IoT infrastructure consists of mainly two components: sensing nodes and actuator nodes. Sensing nodes will collect the information of surroundings through sensors and send the information for processing and storage. Actuators will be activated based on the analysis of the data. The communication layer is responsible for data transfer to/from IoT devices, edge devices, and cloud. Different communication protocols are available for data transfer, as shown in Tables 1 and 2.

The next layer is for edge infrastructure, which consists of different types of edge devices such as Arduino and Raspberry Pi. These devices can be accessed transparently with the help of different types of virtualization and containerization mechanisms. It provides infrastructure for the deployment of the raw data generated by the sensing nodes. In many cases when the edge is enabled enough to process the data, it does not need to send the data to cloud for further processing. Finally, the result is sent back to the actuator for performing the particular action.

The application or services layer consists of different services that can be directly accessible to the users. These applications will be accessed via a subscription model. The example services are a smart home, smart-City, smart healthcare, and smart transportation.

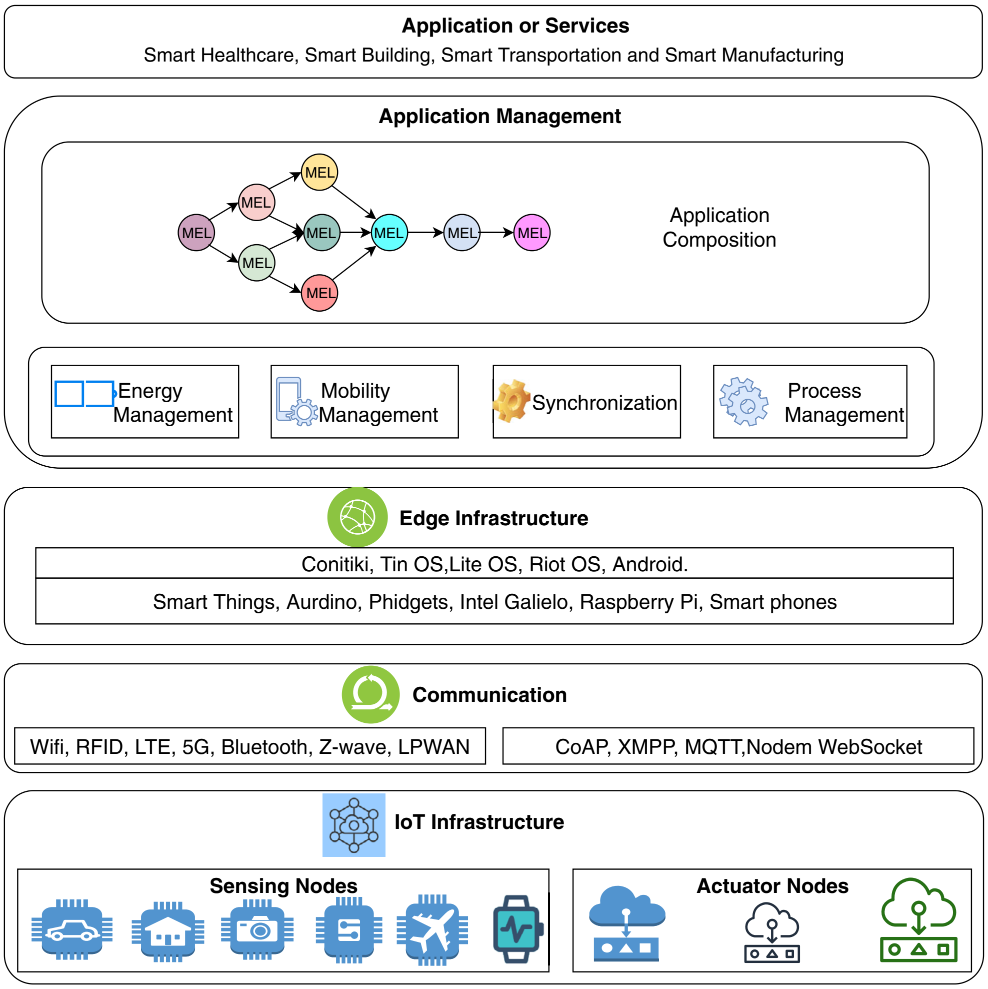

Application or Services
Smart Healthcare, Smart Building, Smart Transportation and Smart Manufacturing
Application Management
MEL
Application Composition
Energy Management
Mobility Management
Synchronization
Process Management
Edge Infrastructure
Conitiki, Tin OS,Lite OS, Riot OS, Android.
Smart Things, Aurdino, Phidgets, Intel Galielo, Raspberry Pi, Smart phones
Communication
Wifi, RFID, LTE, 5G, Bluetooth, Z-wave, LPWAN
CoAP, XMPP, MQTT,Nodem WebSocket
IoT Infrastructure
Sensing Nodes
Actuator Nodes

FIGURE 1 Architecture of Internet of Things (IoT)-Edge computing [Color figure can be viewed at wileyonlinelibrary.com]

The application management layer will manage the applications deployed in the edge environment. This layer is responsible for the application composition where an application is decomposed into a DAG of MELs, which abstracts both software and data.[15] Each MEL can be deployed across the edge or cloud datacenter. It also manages the QoS requirements along with load balancing and fault-tolerance handling. It offers other management services such as resource management, storage management, and device management. Thus, services provided by this layer will ensure the user's QoS requirement satisfaction.

Existing application deployment and scaling techniques developed for other distributed computing environments such as cloud or grid are not suitable for the new IoT-Edge environment. This is because of the diverse characteristics of smart devices along with the mobility feature and the modern application architecture that has a strict dependency and requires distributed processing. Depending on the application type, a variety of collaborations between IoT, edge, and cloud are required to achieve the desired QoS requirements. Hence, the development of new application deployment and scaling techniques is required. It is necessary to test and validate these techniques before actual deployment. However, testing these new techniques in the real environment with different conditions is very time-consuming and expensive. Moreover, due to the distributed ownership of the devices, testing requires multiple access mechanisms, which makes it even more complicated. Therefore, a simulation framework such as IoTSim-Edge simulator, which supports the deployment of an application to evaluate the performance of different techniques and scenarios under different conditions, is required. Moreover, evaluating the techniques under different scenarios and conditions can be done with minimal cost in the simulation environment.

3 | IoTSIM-EDGE ARCHITECTURE

The architecture of the proposed simulator consists of multiple layers, as shown in Figure 2. A brief description of each component is presented in this section.

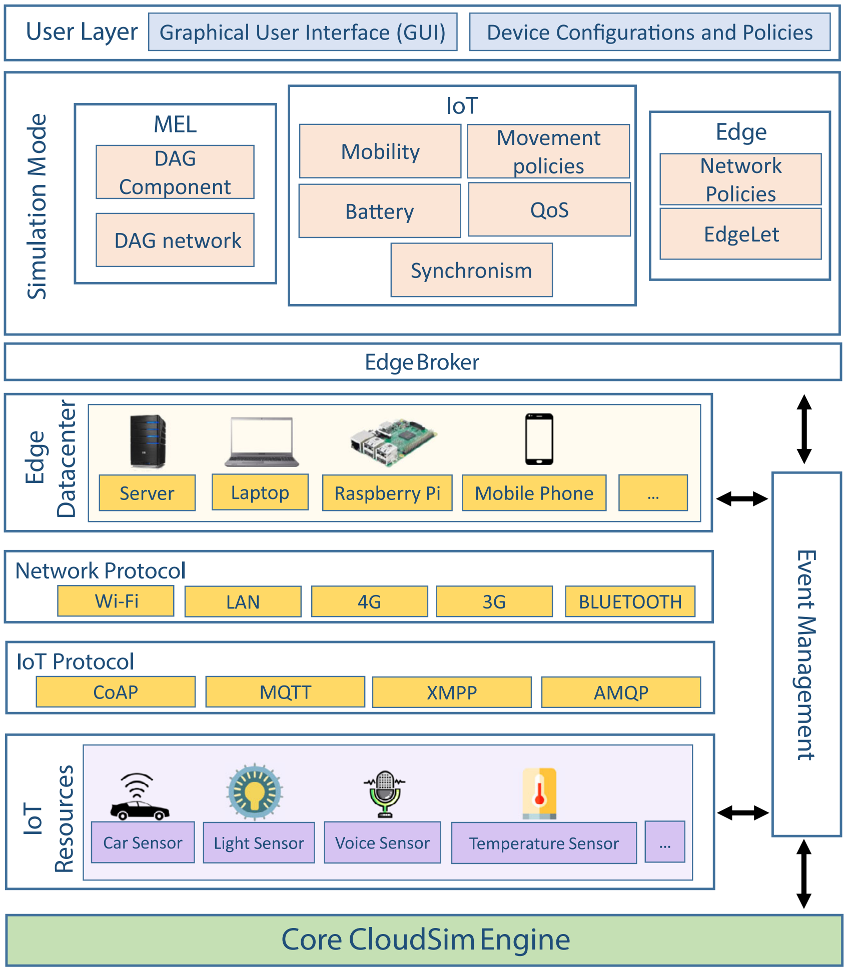


FIGURE 2 Architecture of IoTSim-Edge simulator [Color figure can be viewed at wileyonlinelibrary.com]

IoTSim-Edge is built on the top of CloudSim[6] simulation tool. CloudSim provides the underlying mechanism for handling communication among subscribed components (eg, broker, edge datacenter, and IoT resources) using an event management system. The core components of CloudSim are extended to represent the edge infrastructure in line with edge's features and characteristics. IoT resources layer contains different types of IoT devices (eg, car sensor and motion sensor), where each one has their own features and behaviors along with performing different operations of sensing and actuation. Sensors in the IoT device seamlessly generate data, while actuators are responsible for generating the response. Traditionally, cloud resources are used for processing IoT data. However, in edge-IoT approach, sensor data are processed in the edge datacenter for faster processing time. Edge datacenter consists of heterogeneous processing devices such as smartphone, laptop, Raspberry Pi, and single-server machine. Edge-IoT management layer coordinates processing by receiving user request from users' layer and process the requests using the Edge-IoT resources. Resource broker facilitates the deployment process. Users can provide input through a graphical user interface by mentioning different device configurations and policies. Edge-IoT management layer consists of several components such as EdgeLet, policies, mobility, battery, synchronism, QoS, network protocols, communication protocols, transport protocols, and security protocols.

3.1 | Design and implementation

For the implementation of IoTSim-Edge, we extended the existing classes of CloudSim, as shown in Figure 3, as well as defined numerous new classes in order to model realistic IoT and edge environments (see Figures 3 and 5). Any entity that extends SimEntity class can seamlessly send and receive events to other entities when required through the event management engine. Figure 4 shows a simplified behavior of the simulator's lifecycle as steps and actions.

For modeling an edge infrastructure, we designed and implemented numerous new classes. The main classes are EdgeDataCenter, EdgeBroker, EdgeDatacenterCharacteristics, EdgeDevice, MEL, and EdgeLet. The EdgeDataCenter class is responsible for establishing connection between edge and IoT devices based on the given IoT protocol (eg, CoAP) along with performing edge resource provisioning, scheduling policies, and monitoring edge processing. Furthermore, it is also responsible for EdgeDevice creation and EdgeLet submission, setting up a network connection between edge and IoT infrastructure, checking network availability, among others. The characteristics of EdgeDataCenter (eg, types of policies, types of IoT protocols, and types of IoT devices) are fed by EdgeDatacenterCharacteristic class. The EdgeBroker class acts on behalf of users in terms of establishing connection with edge and IoT devices, negotiating with resources, submitting

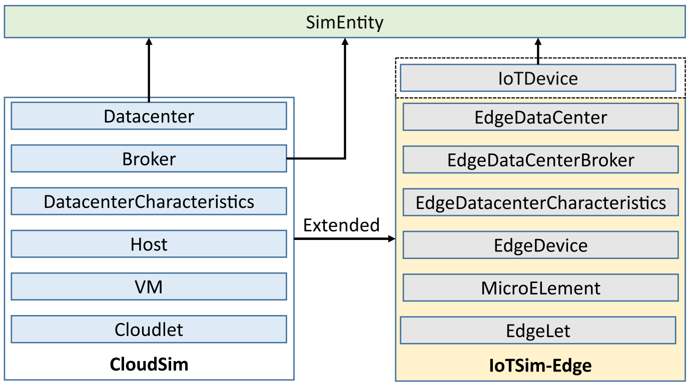


FIGURE 3 Used classes of ClousSim in IoTSim-Edge plus classes use event management system [Color figure can be viewed at wileyonlinelibrary.com]

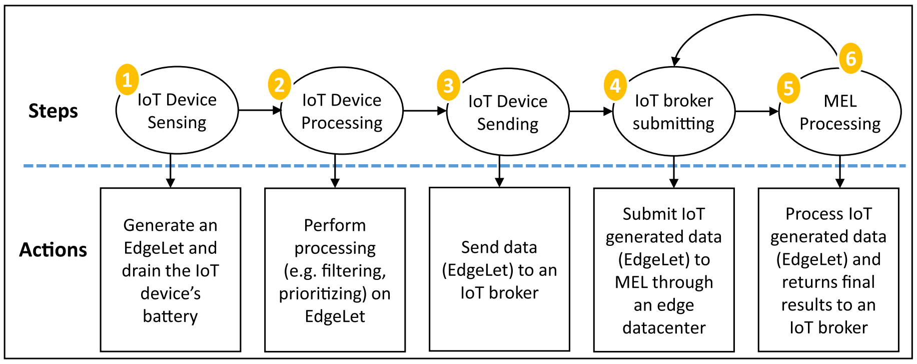

Steps
1 IoT Device Sensing
2 IoT Device Processing
3 IoT Device Sending
4 IoT broker submitting
5 MEL Processing
6
Actions
Generate an EdgeLet and drain the IoT device's battery
Perform processing (e.g. filtering, prioritizing) on EdgeLet
Send data (EdgeLet) to an IoT broker
Submit IoT generated data (EdgeLet) to MEL through an edge datacenter
Process IoT generated data (EdgeLet) and returns final results to an IoT broker

FIGURE 4 An overview of IoTSim-Edge steps and actions [Color figure can be viewed at wileyonlinelibrary.com]

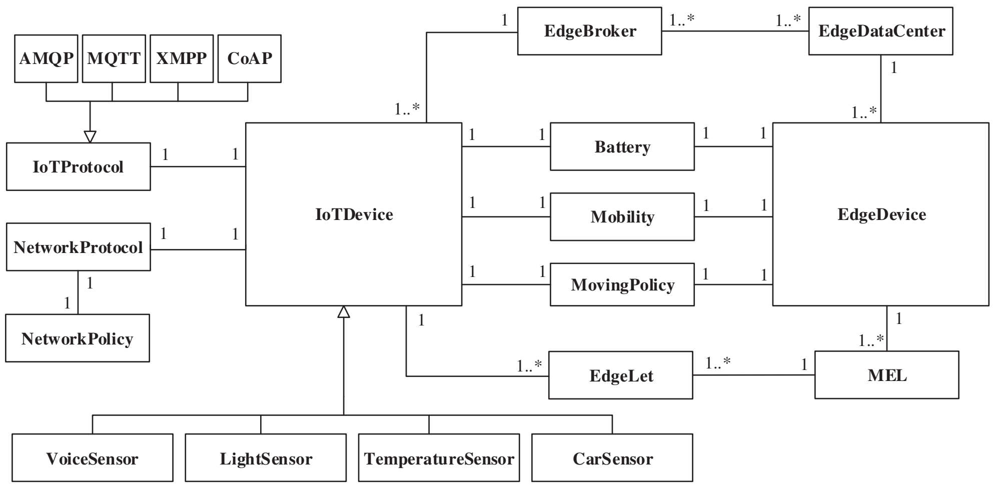


FIGURE 5 Class diagram of IoTSim-Edge simulator

IoT and edge requests, and receiving results. The EdgeDevice class presents the model of edge devices in terms of receiving and processing IoT generated data where the data processing is carried out in accordance with the given EdgeLet policy. The MEL class models the abstract operation performed on IoT data on either edge or Cloud datacenters. For the current implementation of MEL, we only consider it running on edge devices. The EdgeLet class model tasks need to be executed inside MEL. An edge device may contain a battery (eg, mobile phone) and be moving around; therefore, the battery and mobility classes are designed to enable the edge device to obtain such characteristics. The MovingPolicy class dictates the moving conditions and behaviors of edge devices.

Similarly, for modeling the IoT infrastructure, we developed many new classes, as shown in Figure 5. The IoTdevice class mimics the behavior of real IoT devices in terms of sensing, processing, mobility, data rate, and so on. Since IoT devices are often self-powered and moving, the battery and mobility classes exist to empower IoT devices with such characteristics, similar to edge devices as mentioned before. The movement policy of IoT devices is directed by the MovingPolicy class; it can be extended with new moving policies according to users' requirements (eg, velocity and location of cars' sensors). The NetworkProtocol class models the well-known communication protocols (eg, Wi-Fi and 4G LTE) in terms of speed rate; for example, Wi-Fi can transfer network packets at a speed of 200 Mbps, while 4G LTE at a speed of 150 Mbps. The speed rate of EdgeLet, in other words, the delay time to send EdgeLet to an edge datacenter, is obtained from the NetworkPolicy class. The IoTProtocol class models the features of IoT protocol with regard to QoS and battery consumption. As every IoT protocol (eg, CoAP and extensible messaging and presence protocol [XMPP]) has different processing techniques, each one is modeled in a way that everyone has different power consumption rates. More detailed description of each class is given as follows.

3.1.1 | EdgeDataCenter class

This class controls the core edge infrastructure. It intercepts all incoming events and performs different operations based on the payload of the event, such as resource provisioning and submitting EdgeLet requests to respective MEL(s). It obtains and monitors the capacity of edge devices and MELs along with capturing and reporting MEL processing status to the edge broker. The class is also modeled to support a location-aware mechanism for IoT and edge devices, which helps in establishing and terminating the connection between edge and IoT devices based on the range criteria. This class also supports a power-aware technique to track the battery lifetime of edge devices. Once the battery of an edge device is fully discharged, EdgeDataCenter will automatically detach the edge device and forward unserved requests to another available edge device.

3.1.2 | EdgeBroker class

This class is a users' proxy, in which it generates users' requests in accordance with their prescribed requirements. It has a range of duties to perform, such as submitting edge and MEL provisioning requests to edge datacenter, requesting IoT devices to generate and send data to their respective edge devices and receiving final processing results from MEL. This class supports a power-aware model for IoT devices. As EdgeBroker continuously tracks the battery consumption of IoT devices, it will disconnect the drained IoT devices from its available IoT device list.

3.1.3 | EdgeDevice class

This class behaves similar to a real edge device. It hosts several MELs and facilitates the procedure of CPU sharing mechanism via a given CPU sharing policy (eg, time-shared, space-shared). It is also connected to a specific number of IoT devices, which send their data for processing. It can easily be attached with a battery and power draining policy based on the case when it is battery-driven.

3.1.4 | IoTDevice class

This class models the core characteristics of IoT devices, particularly those that share and have in common such as generating and sending data. As every device has its own specifics (eg, protocols, data generation rate, and power consumption rate), the class is, therefore, extended to include the missing features of a receptive IoT device. In the current version of the simulator, there are four classes that extend the IoTDevice class: VoiceSensor, LightSensor, TemperatureSensor, and CarSensor, as shown in Figure 5. Any new required type of IoT device can easily extend the IoTDevice Class and implement the new features. In the real IoT environment, every IoT device is equipped with IoT protocol (eg, CoAP) and network protocol (eg, 4G LTE); therefore, this class contains similar characteristics by using IoTProtocol and NetworkProtocol classes, as discussed later.

3.1.5 | MEL class

This class represents one component of IoT application graph. It represents the main processing requirement of the application component. Based on the application requirement, setEdgeOperation method can be configured. The dependency between various components can be represented using upLink and downLink, which can be easily set to represent any complex application. It can be easily configured to represents any complex IoT application based on the user's requirement.

3.1.6 | EdgeLet class

This class models the IoT generated data and MEL processing data. Once IoT device establishes connection with its respective edge device(s), it generates IoT sensed data as required in a form of EdgeLet, which contains payload. The payload encapsulates required information to guide MEL on the processing stage, such as size of data to be processed, IoT device ID, and destination MEL ID. By using EdgeLet data size, the network delay required to send EdgeLet to its destination MEL can be computed taking into account the network transmission rate.

3.1.7 | Mobility class

This class contains the mobility model of IoT and edge devices. The mobility plays a vital role in an IoT-Edge ecosystem where maintaining a real-time connectivity status is of importance to properly evaluate the performance. The attributes of mobility model consist of range, velocity, location, and time interval. Each attribute, excluding time interval, has two separate values to represent the horizontal and vertical directions of IoT and edge devices. Figure 6 shows an example of how such attributes are used by an edge datacenter to track the location of IoT and edge devices. Algorithm 1 shows the pseudocode of a simple tracking technique implemented in the EdgeDataCenter class. Users can easily extend this class to implement their own mobility model.

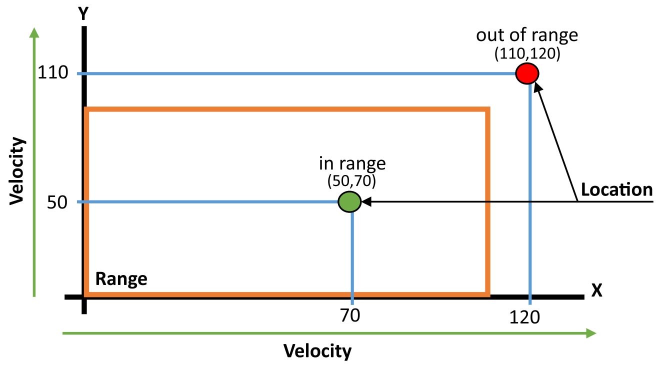


FIGURE 6 How an edge datacenter tracks the location of Internet of Things (IoT) and edge devices [Color figure can be viewed at wileyonlinelibrary.com]

Algorithm 1. Update and Track Location

```
Require: t: time, int: interval, $v_x$: velocity in X direction, $v_y$: velocity in Y direction, $rangeX_E$: X range of edge device E,
  $rangeY_E$: Y range of edge device E, $locationX_I$: X coordinate of the IoT device I, $locationY_I$: Y coordinate of the IoT
  device I
  $isOutRange \leftarrow false$
  $t_{new} \leftarrow t_{old} + int$
  $locationX_I^{new} \leftarrow locationX_I^{old} + v_x \times int$
  $locationY_I^{new} \leftarrow locationY_I^{old} + v_y \times int$
  if $locationX_I^{new} > rangeX_E \,||\, locationY_I^{new} > rangeY_E$ then
      $isOutRange \leftarrow true$
  else
      $sendInternalEvent(I, int, updateLocation)$
  end ifreturn $isOutRange$
```

3.1.8 | Battery class

This class models the battery characteristics of a portable IoT or edge device. It captures the behavior and consistently computes the lifetime of the battery. In a real IoT-Edge environment, it is very difficult to know the exact power consumption for every executed task in addition to the power consumed by the device's internal components. Therefore, this class simplifies the power consumption model as an inverse proportion relationship between draining rate and battery capacity as discussed in detail in Section 3.2.

3.1.9 | IoTProtocol class

This class models the well-known IoT application protocols. Every protocol has its own attributes and features such as power consumption rate. For our proposed simulator, the implemented classes consider four best candidates in IoT and edge computing: message queue telemetry transport (MQTT), advanced message queuing protocol (AMQP), XMPP, and CoAP;[16] however, it can be easily extended for any other considered protocol. QoS parameters are also associated with the IoT protocols, which controls the acknowledgement and response method for each protocol.

3.1.10 | NetworkProtocol class

This class presents the modeling of network protocols (eg, WiFi and 4G LTE). Implementing such models in the IoTSim-Edge framework is required to properly evaluate the performance of IoT-Edge applications. Each network type is designated with its relative network speed (eg, 200 Mbps for WiFi and 150 Mbps for 4G LTE). By modeling the transmission rate, transmission time can be obtained taking into account the EdgeLet size.

3.1.11 | Policies classes

These classes model the policies for three activities, namely, device movement, battery consumption, and network transmission. The device movement policy instructs edge datacenters to keep tracking of the movement and location of IoT and edge devices (see Algorithm 1). The battery consumption policy tracks and computes the remaining power capacity of IoT and edge devices. The network transmission policy computes the time taken to transfer data from IoT device to edge device. Such policies can be extended to derive new IoT-Edge designs and solutions. All these classes can be extended to implement and test different user policies.

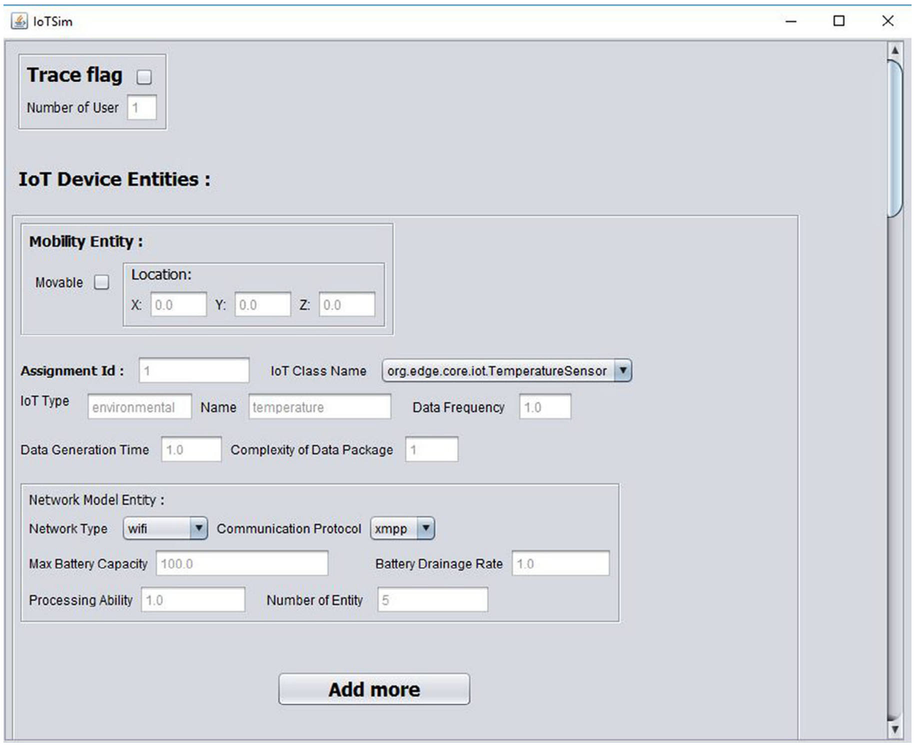
3.1.12 | UserInterface class

It provides the necessary methods to easily configure and test the IoT application development without knowing the details of the simulator. It allows a user to define all the parameters using the interactive interface that is converted into the desired configuration file. A snapshot of the user interface is shown in Figure 7.

3.2 | Calculation and event processing

As shown in the sequence diagram (see Figure 8), the simulation process takes place immediately after initializing the required IoT-Edge infrastructure, which is derived from the given configuration file. Figure 9 shows the test configuration file for IoT device. Once the required IoT devices and MELs (residing in edge hosts) are created, the edge broker will ask edge datacenters to establish connections between IoT devices and their respective MELs. As a result, each edge datacenter will establish requested connections and update the edge broker with the connection status for every request. Also, every edge datacenter will consistently track the location of IoT and edge devices in addition to the power consumption during the whole lifetime of the simulation process. Note that the battery consumption of every IoT devices is maintained by the edge broker.

Every time the edge broker receives a new connection establishment (ACK), it will notify the IoT device about the connection. IoT device will then start sensing and generating data in the form of EdgeLets, transfer the EdgeLets to



Trace flag ☐
Number of User: 1

IoT Device Entities :

Mobility Entity :
Movable ☐
Location: X: 0.0 Y: 0.0 Z: 0.0

Assignment Id : 1
IoT Class Name: org.edge.core.iot.TemperatureSensor
IoT Type: environmental
Name: temperature
Data Frequency: 1.0
Data Generation Time: 1.0
Complexity of Data Package: 1

Network Model Entity :
Network Type: wifi
Communication Protocol: xmpp
Max Battery Capacity: 100.0
Battery Drainage Rate: 1.0
Processing Ability: 1.0
Number of Entity: 5

FIGURE 7 Snapshot of user interface for IoTSim-Edge [Color figure can be viewed at wileyonlinelibrary.com]

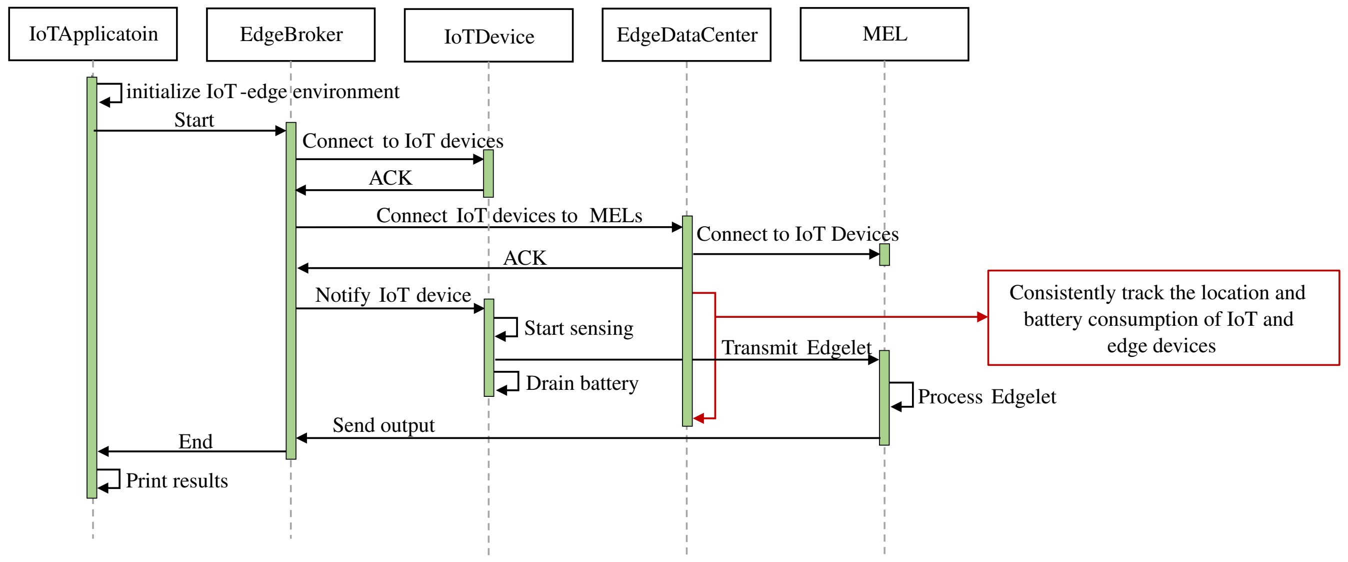


FIGURE 8 Workflow of IoTSim-Edge simulator [Color figure can be viewed at wileyonlinelibrary.com]

FIGURE 9 An example of Internet of Things (IoT) definitions in a JSON format

```
"ioTDeviceEntities": [
        {
            "mobilityEntity": {
                "movable": false,
                "location": {
                    "x": 0.0,
                    "y": 0.0,
                    "z": 0.0
                }
            },
            "assignmentId": 1,
            "ioTClassName": "org.edge.core.iot.TemperatureSensor",
            "iotType": "environmental",
            "name": "temperature",
            "data_frequency": 1.0,
            "dataGenerationTime": 1.0,
            "complexityOfDataPackage": 1,
            "networkModelEntity": {
                "networkType": "wifi",
                "communicationProtocol": "xmpp"
            },
            "max_battery_capacity": 100.0,
            "battery_drainage_rate": 1.0,
            "processingAbility": 1.0,
            "numberofEntity": 5
        }]
```

destination MELs, and drain its battery power according to the defined drainage rate. The IoT devices will keep repeating these actions until they reach their simulation ending time. Every time a MEL receives IoT generated data (EdgeLet), it will process the EdgeLet and returns final output to the edge broker.

The total processing time of an MEL depends on the data size to be processed. In our implementation, the processing is done in two steps as shown below. Given the data size DS, first, the data are converted into the number of million instructions (MI) based on the type of functionality performed, as shown in Equation 1. Again based on the shrinking factor, new EdgeLet is constructed with shrinked data size, which is transferred to other edge device. Along with this, a processing is performed on the data that takes $\text{Time}_{\text{proc}}$ time, as shown in Equation 2. Here, million instructions per second (MIPS) is the CPU capacity in MI per second. The total processing time, Total_Proc_Time$_{\mathrm{MEL}}$, is given by the maximum time required to either shrink or process the data, as given in Equation 3.

$$\mathrm{MI} = f(\mathrm{DS}), \tag{1}$$

$$\mathrm{Time}_{\mathrm{proc}} = \frac{\mathrm{MI}}{\mathrm{CPU\ Capacity\ (MIPS)}}, \tag{2}$$

$$\mathrm{Total\_Proc\_Time}_{\mathrm{MEL}} = \max(\mathrm{Time}_{\mathrm{shrink}}, \mathrm{Time}_{\mathrm{proc}}). \tag{3}$$

As a more detailed explanation, the updateBattery method will be called during the sensing process, which updates the battery. Battery consumption directly depends on the processing data size and the underlying transmission protocol. The new battery level, Bat$_{\mathrm{new}}$ is calculated, as given in Equation 4

$$\mathrm{Bat}_{\mathrm{new}} = \mathrm{Bat}_{\mathrm{old}} - \mathrm{Battery\_Consumption}_{\mathrm{total}}. \tag{4}$$

Battery_Consumption$_{\mathrm{total}}$ is the battery consumption that is calculated, as given in Equation 5

$$\mathrm{Battery\_Consumption}_{\mathrm{total}} = \mathrm{DS}[(1-\rho) \times \lambda_{\mathrm{proc}} + \rho \times \lambda_{\mathrm{comm}}], \tag{5}$$

where DS is the total data size, ρ is the shrinking factor, λ_{proc} is the drainage rate for processing, and λ_{comm} is the drainage rate for data transfer using specific protocol. Our assumption is that if the shrinking factor ρ, $1-\rho$ times DS, is processed and ρ times DS is sent. The remaining battery is calculated after every processing or transfer.

If the battery of IoT device is drained, the device will be shut down. If the updateBattery method finds some IoT device with available battery, the data generation event will be invoked, which generates data at the defined data rate (frequency).

The generated data are sent to edge device using some specific protocols. The network transmission time depends on the protocol used as every protocol has specific data packet size and data rate. The transmission time, Time$_{\mathrm{trans}}$ is calculated, as given in Equation 6

$$\mathrm{Time}_{\mathrm{trans}} = \frac{N_{\mathrm{packet}}}{\mathrm{data\ rate}_P}, \tag{6}$$

where N_{packet} is the total number of packet to be transmitted and data_rate$_P$ is the data rate for protocol P. Number of packets, N_{packet}, depend on the max packet size allowed by the particular protocol.

Every time the data are generated by the IoT device, it is ready to be uploaded to the edge. The next step is checking the edge device availability by the broker. If the specified edge device is not available, the broker will find a new edge device based on the edge device's availability, communication protocol, maximum acceptable number of IoT devices, and geographic location in the mobility model. If all the edge devices are disabled for any reason, the simulation will be stopped. If certain edge devices are still running, this IoT device cannot get connected with any running one and broker simply discards the request from IoT. Throughout the event processing, the connection header is transmitted with every event that sends or receives data, which can be referred to as an information package. For example, when the edge datacenter broker wants to know where the EdgeLet come from, it will access the connection header to get the target id.

When the edge device is available to the IoT device, the broker will transmit the data to edge datacenter and then the edge datacenter will check again to ensure nothing is wrong. If anything goes wrong, the data will be discarded. In the case of all above process running smoothly, the edge device will finally process the EdgeLet, which is an asynchronous process. After processing the EdgeLet, the edge device will send an event to edge datacenter telling that the EdgeLet has been processed, and edge datacenter will send an event asking broker whether that IoT device is still available to this edge device or not. If the IoT device is not in the range, the broker will search the whole network to find an edge device that is connected to the desired IoT device. This whole process is called *edge-to-edge communication*. After obtaining that edge device, the broker will indicate the edge device to transfer the returned data to another edge device, that is, having the connection with the desired IoT device. Presumably, this process does not cost any time in data transmission. Therefore, there is no delay or battery consumption during this process. On the other hand, if everything goes fine, the processed data will finally be passed to the IoT device, then this IoT device will actuate, update battery, and check availability. After all this, the whole process is repeated. The simulation process is kept running until batteries run out, and once all batteries ran out, the simulation process will be terminated.

4 | IoTSIM-EDGE EVALUATION

To evaluate the applicability of IoTSim-Edge, three case studies are modeled. The details of each case study are given below.

4.1 | Case 1: Healthcare system

Human activity recognition is beneficial for multiple purpose from maintaining fitness to monitoring healthcare . For example, medical staff can easily monitor the health condition of a patient and offer the necessary assistance whenever required.[17] Based on the human physical activity, one can easily compute the calorie consumption and follow a particular diet or exercise.[18] Emergence of smart IoT devices such as *Pebble Watch* or *Fitbit* makes it easier to monitor the activity. These smart devices create raw activity data and send to a smart phone using low-energy protocols, for example, BLE. The raw activity data are processed by some algorithms to provide information about the activity levels.

Following the work in Reference 19, in which the activity data are partitioned into multiple microoperations, we also realized the same scenario. To process the raw data using step count algorithm, we model each microoperation as an MEL that can be processed by underlying IoT or edge devices. Figure 10 shows the basic MEL graph for a simple healthcare scenario, which is required to be deployed on the edge computing environment.

To find an optimal deployment solution on the distributed IoT infrastructure, it is necessary to analyze different possible deployment plans. Since there is strict dependency among MELs, it is important to consider this while deployment. To implement this in our simulator, we considered two edge devices, in which one edge device has an embedded IoT device. The IoT device generates the data based on the defined data rate. Research shows that the battery drainage rate for transmission is higher than computation.[19]

Since we considered edge devices that are also battery-powered, based on the processing happening on the edge device, the battery drainage rate varies, which is analyzed in this work. However, performing more operations near to the IoT device will also increase the execution time, which is very important in many cases. The configuration of experiment is shown in Table 3. Based on the real scenario,[19] we assigned MIs to different MELs. Figure 11 shows the comparison of battery hours of edge to process the request with varying shrinking factor. Shrinking factor represents the processing happened on the edge device. The battery consumed by processing and communication is calculated, as given in Equation 4.

The result in Figure 11 validates the proof that performing the processing on the edge device with IoT embedded will lead to save more battery as compared to sending all the raw data to other edge devices. The result shows that processing 90% of operations on edge device increases the battery consumption by 266% as compared with performing only 10% of operation. Since we defined that edge E1 has small processing and storage capacity, all the processing cannot be performed there. Different types of analysis can be easily performed using this scenario. Users can also propose different algorithms to find a suitable deployment option based on multiple objectives in an easy way.

4.2 | Case 2: Smart building

Recently, smart building application that automates the lighting, heating, air-conditioning, air-quality, and so on has received much attention.[20] Different types of sensors (eg, temperature, humidity, motion, and air quality) deployed at

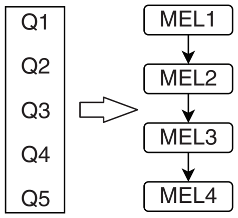


FIGURE 10 A simple microelement (MEL) graph for distributed deployment for healthcare query operation

TABLE 3 Configuration file for Case 1

IoT device		MEL		Edge device 1	
Location	0, 0, 0	id	1	Type	Raspberry Pi
Movable	False	RAM	10000 MB	Movable	False
Data frequency	1	BW	10 000 Mbps	Signal range	100 m
Data generation time	1	Shrinking factor	"Variable"	Max IoT device capacity	10 000
Network protocol	Bluetooth	Network protocol	Bluetooth	Max battery capacity	20 000 units
IoT Protocol	CoAP	Uplink	—	Battery drainage rate for processing	0.1 units/h
Max battery capacity	300 units	300 units	2	Battery drainage rate for transfer	0.6 units/h

Abbreviation: BW, Bandwidth; IoT, Internet of Things.

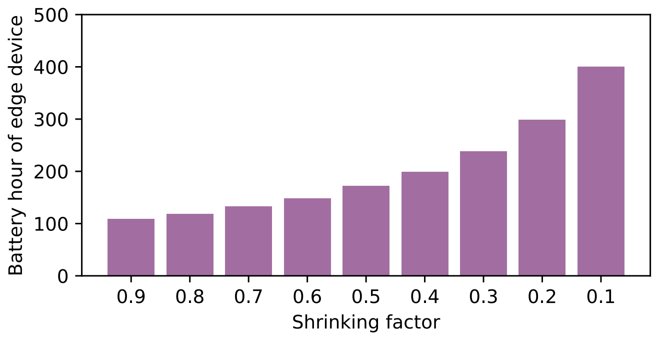


FIGURE 11 Simulation result for Case 1 showing the variation of battery hours for different scenarios [Color figure can be viewed at wileyonlinelibrary.com]

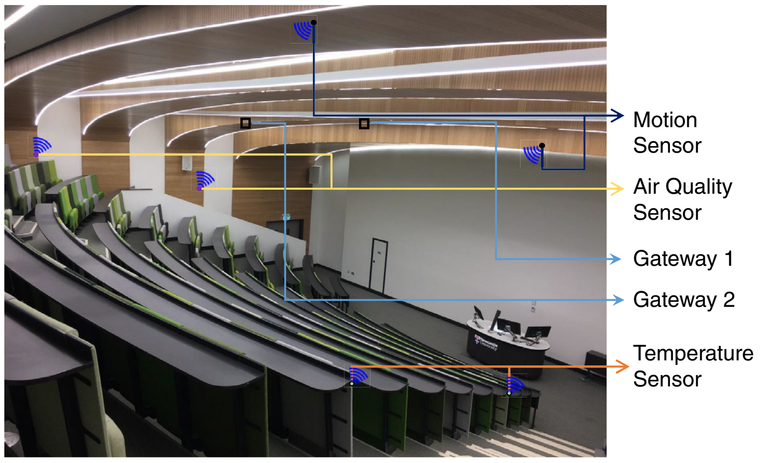


FIGURE 12 Smart building example (Lecture hall of Urban Sciences Building at Newcastle University) [Color figure can be viewed at wileyonlinelibrary.com]

specific sites monitor the building activity and send the data to the associated edge device (eg, raspberry pi or single personal computer) for processing and analysis. Figure 12 shows an example of smart building constructed at Newcastle University (https://www.ncl.ac.uk/computing/about/usb/). Edge device does some local processing and sends the data to cloud if further storage or some complex analytics are required. Multiple IoT devices sends their data to one edge device following one specific communication protocol. Features like latency directly depend on the packet size and data rate. Research shows that one protocol is better than other protocols in terms of packet size and data rate.[21] Again, since the IoT devices are battery operated, varying the data rate also changes the battery consumption. In the view of the fact that energy conservation is one of the important features,[22] recharging or replacing the battery at a frequent rate is not suggested. It is better to use a protocol that consumes lesser energy.

To analyze this case, we have simulated a smart building scenario with varying number of sensors following either CoAP or XMPP protocol. The configuration of IoT device is similar to the one discussed for Case 1 except the protocol for the executed IoT device is either CoAP or XMPP and the IoT type is environmental as it monitors the building

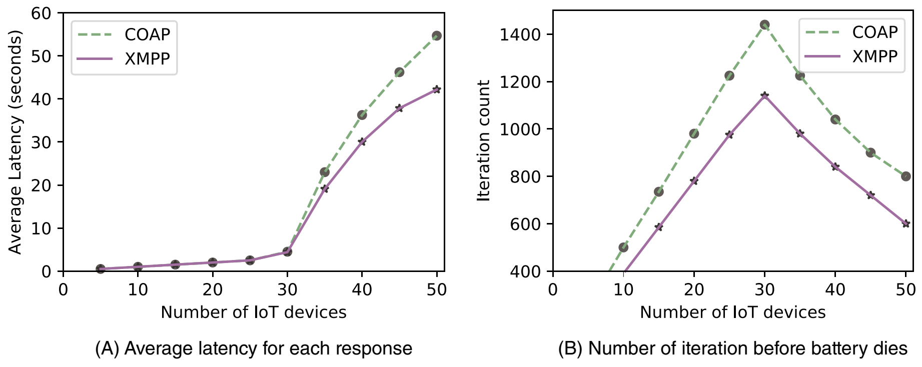


FIGURE 13 Simulation result for smart building. A, Average latency for each response. B, Number of iteration before battery dies [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 4 Battery running hours using CoAP and XMPP protocol

Protocol	Battery Hours
CoAP	149.73
XMPP	119.32

Abbreviations: CoAP, constrained application protocol; XMPP, extensible messaging and presence protocol.

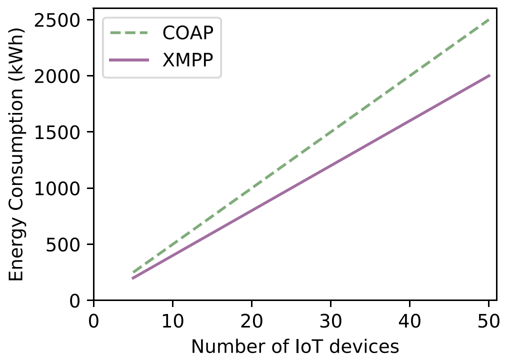


FIGURE 14 Average energy consumption by each edge device [Color figure can be viewed at wileyonlinelibrary.com]

environment. Also, the number of IoT device varies from 1 to 50. The configuration of edge device for Case 2 is the same as explained in Case 1. The simulation results are presented in Figure 13. Figure 13A shows the average latency for each response. The figure clearly shows that initially the latency increases slowly with an increasing number of sensors; however, after the edge device got saturated, the latency becomes very high as the resource-constrained edge device will take more time to process the request. The result also shows that the latency of CoAP increases at faster rate as compared to XMPP. The reason is justified from Figure 13B where the iteration count (number of events generated during that particular scenario) for CoAP is higher as compared with XMPP. Since CoAP is a light-weight protocol, the battery also goes for longer time as compared with XMPP (see Table 4). The full-battery drainage hour using both the protocols is given in Table 4.

Figure 14 shows the average energy consumption by the edge device for our test case. As we can see, the energy consumption of edge device increases with increasing number of IoT devices. However, the energy consumption for CoAP is higher as compared to XMPP. This is because of the fact that using CoAP, more number of messages can be processed as shown in Figure 13B. Also, the energy always increases with increasing number of IoT devices regardless of the response time values.

4.3 | Case 3: Capacity planning for road side units

Self-driving cars[23] is an upcoming technology, in which each car has an embedded sensor that communicates with the microcontrollers of road side units (RSUs) to cooperatively maintain a smooth traffic flow and ensure the road safety. The RSUs can be smart edge devices that can process the data received from car and respond it back to the car to make some runtime decisions. The range of cars and RSUs are limited and a car is always moving with some velocity in a particular direction on the road. The coverage area of an RSU is dependent on the underlying transmission protocol. It is most probable at a point of time that the car losses its connection (handoff) with the previously connected RSU. The handoff may be hard or soft depending on the speed of car and distribution of the RSUs. Since the processed information from the previous RSU is required to make certain decision, an RSU-to-RSU (edge-to-edge) communication is also established. Based on the direction of the car movement, one RSU sends data to another RSU, which is further delivered to the specified car. Since the range and processing capacity of RSU are limited, it is necessary to analyze how many car data can it process without losing any information.[24] Also, it is necessary to guarantee the QoS requirement, for example, response time of the application.

We modeled this scenario using our simulator. Figure 15 shows the schematic diagram of the model considered by our simulator. At time *t*1, the car found a new RSU. It first establishes a connection with the new RSU, RSU 1, and then starts

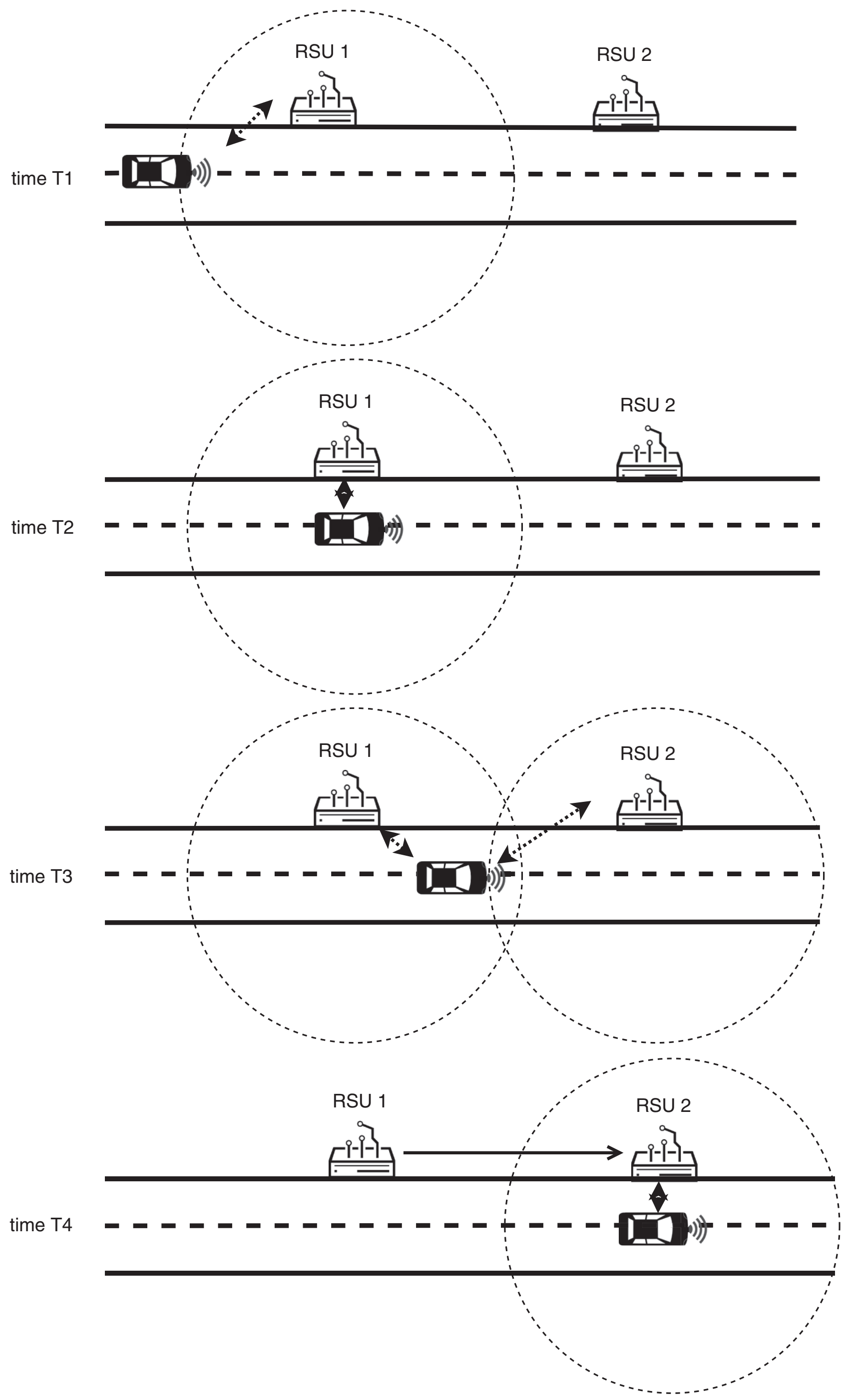


FIGURE 15 Schematic diagram showing the car movement captured by the road side units (RSUs)

TABLE 5 Configuration file for Case 3

IoT Device		Edge Device 1	
Current location	0, 0, 0	Type	Raspberry Pi
IoT type	car	Location	0, 0, 0
Movable	true	Movable	False
Data frequency	1	Signal range	50 m
Data gen. time	1	Max IoT device capacity	10 000
Network protocol	Wi-Fi (802.11p)	MIPS	10 000
IoT Protocol	XMPP	RAM	10 000 MB
Max battery capacity	70 units	Bandwidth	10 000 Mbps
Battery drainage rate	1 unit/h		
Number of entity	"variable"		
Velocity	0.5 m/s		

Abbreviations: IoT, Internet of Things; MPI, message passing interface; XMPP, extensible messaging and presence protocol.

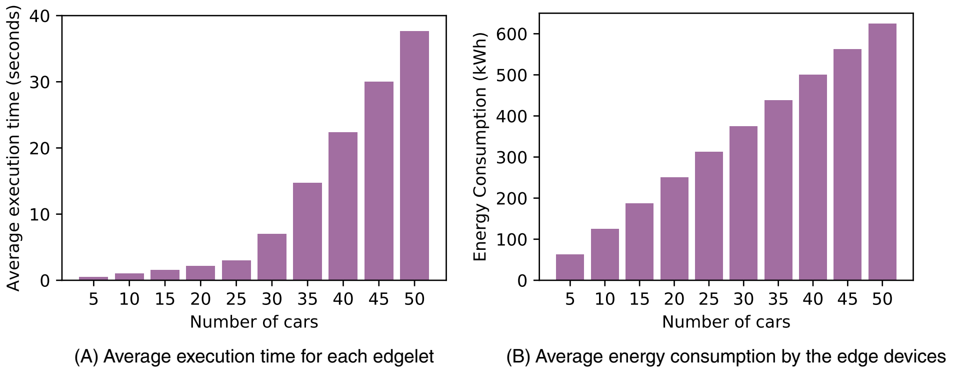


FIGURE 16 Simulation result for road-side units (RSUs). A, Average execution time for each edgelet. B, Average energy consumption by the edge devices [Color figure can be viewed at wileyonlinelibrary.com]

sending data. At time $t3$, it reaches at a position where it is in the range of two RSUs. Based on its movement, it starts making connection with RSU 2 while still sending data to RSU 1. When it lost connection with RSU 1, it starts sending data to RSU 2. RSU 1 figures out that the car is now not in their range and based on the velocity information, it transmits the processed data to RSU 2, which delivers the information to the car as shown for time $t4$. This scenario validates the mobility feature as well as cooperative edge communication feature of our simulator. Table 5 gives the specific simulation configuration of the specified case. Both the edge devices are identical except the location ($E1$ is at 0, 0, 0, while $E2$ is at 50, 0, 0).

The simulation result is presented in Figure 16. The result in Figure 16A shows the average execution time with respect to the number of cars. The execution time increases as the number of car increases. With the increase in the number of cars, the number of connection also increases. Since the number of edge devices is constant and the processing is done in a time-shared manner, requests are queued before processing, which leads to the increased execution time. Also, with the mobility of car, if the car moves away from the range of one RSU, then that RSU has to send the processed data to other RSUs, which further sends to the car. Since this is also added to the actual processing time, the execution time is higher. To process increased number of requests from car, edge will consume more energy as verified by our simulation (see Figure 16B), which shows that the average energy consumption of edge device increases with increasing number of cars (IoT).

4.3.1 | Simulation time and memory consumption

The average processing time and memory consumption for Case 2 are shown in Figure 17A, and B, respectively. Figure 17A shows that the simulation time increases with increasing number of IoT devices. However, the simulation can be completed in a reasonable time (4552.4 ms) for 50 IoT sensors. The result for memory consumption is bit different from the

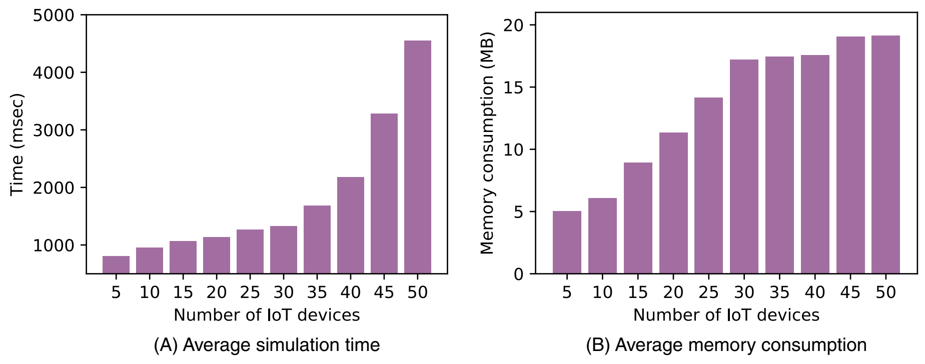


FIGURE 17 Simulation complexity. A, Average simulation time. B, Average memory consumption [Color figure can be viewed at wileyonlinelibrary.com]

simulation time. Figure 17B shows that the average memory consumption does not increase considerably with increasing number of IoT devices. Maximum memory consumption is 19.14 MB for 50 IoT devices, which is acceptable.

5 | RELATED WORK

With the surge in interest in IoT and edge computing, numerous simulation tools are developed in the past few years. Some of the tools are extended from the existing network and cloud simulators; however, there is a gap in the existing simulators and the realistic modeling of edge and IoT environment. In this section, first, we discuss the currently available simulation tools for the network, cloud, and IoT environment and how they are not able to model the existing IoT-Edge environment. Furthermore, we show how our simulation framework is able to satisfy the available challenges in a holistic way.

5.1 | Network simulators

Several simulation tools have been proposed for simulating computer network in the past decade. Among them, here we discuss a few well-known network simulation tools. OMNeT++[25] is a C++-based discrete-event simulation framework that can simulate a distributed processing environment with network communications. It supports parallel simulation as well as real-life protocol implementation in the simulation models. It also supports the simulation of some of the network protocols but does not support edge communication protocols. Castalia[26] is developed on top of OMNet++ platform to simulate body area network and other related networks that consist of lower energy-powered devices. It also endures in simulating a large number of mobile nodes dynamically. By extending this, David et al[27] proposed GreenCastalia to support energy-harvesting techniques in the simulator. However, this does not support the modeling of different communication protocols. TOSSIM[28] is an open-source even-driven simulator to simulate wireless sensor networks (WSN). It uses the NesC programming language to develop the simulation environment. TOSSIM supports features such as sophisticated network connectivity, bridging, and scalability. However, this simulator does not support energy consumption and mobility modeling. To support the modeling of power consumption in WSN, another simulator was proposed, which is known as PowerTOSSIMz[29] by extending TOSSIM. PowerTOSSIMz incorporates the nonlinear behavior of the battery model simulation. However, it does not support the mobility feature. NS-3[30] is another C++-based discrete-event simulation tool, which has a Python scripting interface. This tool can simulate a distributed environment with virtualization support. However, NS3 is not suitable for IoT simulation at edge level since it does not support the scheduling and application composition features.

5.2 | Cloud simulators

Many simulation tools have been proposed so far for the simulation of the cloud computing environment. Among them, CloudSim[6] is the most used cloud simulator in the research community. CloudSim is event-driven simulation tools which support both behavior and system modeling of cloud environment components. However, CloudSim does not support simulation and modeling of IoT and edge environment. The proposed IoTSim-Edge is developed on top of CloudSim simulator by extending it.

iCanCloud[31] is another cloud simulator that handles large-scale simulation with the support of communication and physical models. This simulator has customizable global hypervisor that can support any cloud brokering policy. It also contains Amazon public cloud instances and supports message passing interface. NetworkCloudSim[32] is another simulator that allows us to model the network behavior of cloud datacenters. However, these simulators do not support IoT and edge simulation.

Another cloud simulator is GreenCloud,[7] which is extended from the NS2 simulation tool. GreenCloud is a packet-level simulation tool, which can measure the energy consumption of datacenter components. This simulator only focuses on the calculation of energy consumption to ensure energy-aware placement. It also does not support IoT and edge simulation. DCSim[33] is another cloud simulator, which is specifically focused on infrastructure as a service (IaaS) cloud environment simulation. It supports modeling and simulation of the datacenter, host, and virtual machine with a limited number of application and resource management policies. However, simulation of IoT and edge environment is not supported by this simulator.

5.3 | IoT and edge simulators

Numerous simulation tools have been proposed for simulating either IoT or edge environment. SimIoT[34] is a simulation tool that models the communication between IoT devices and cloud datacenter but they did not incorporate edge devices in the simulation. This simulator enabled experimentation of multiuser submission dynamically in the IoT environment. However, this tool did not consider the heterogeneity of IoT devices. Also, the energy consumption of IoT devices is ignored here. Fredrik et al proposed a cross-level sensor network simulator called COOJA,[35] which can be used to simulate different deployment levels (machine code instruction set level, operating system level, and network level). This tool is suitable for heterogeneous network nodes simulation but is designed only for the Contiki operating system.

iFogSim[9] supports modeling of Fog and IoT environments and measures the impact of resource management techniques in network congestion, latency, cost, and energy consumption. This simulation tool considers edge devices as Fog devices. However, they did not consider edge communication protocols in their simulation tool. The IoTSim[36] supports simulation of map-reduce process, which is known as a big data processing paradigm. This simulation tool is only used for modeling map-reduce processes. EdgeCloudsim[8] is another edge simulation tool build on the top of CloudSim toolkit. It supports computational and networking aspects of edge computing paradigm. EdgeCloudSim provides network link model, the mobility model, and edge server model to evaluate the aspects of edge computing. However, it is not able to model the application composition of IoT applications. DIScrete event baSed Energy Consumption simulaTor for Clouds and Federations[37] is another simulation tool built on top of the cloud computing concept, which can evaluate the energy consumption of IaaS, model scheduling, and internal infrastructure behaviors. However, Gabor et al[5] stated that "although the integrated sensor models are generic, they might still not be inapplicable in future IoT scenarios."

In summary, all the above-discussed simulators do not support edge communication protocols and energy calculation (battery power). Also, most of them are not able to define the application composition in IoT environment. These features are essential for any IoT application. A composite simulation environment for IoT-Edge is crucial to help researchers and industries to gain the real potential of edge processing. Our proposed simulator, IoTSim-Edge, covers all these features in a holistic manner. The advantage of our IoTSim-Edge as compared to the existing simulators is clearly visible from Table 6 as the proposed simulator is able to satisfy more features.

6 | CONCLUSION AND FUTURE WORKS

The recent advances in IoT and edge computing offer numerous services for the users near to the edge. To enable these services, we need to develop new methods and techniques, which is required to be tested before offering to the users. The deployment of these methods and techniques for IoT application in the real environment is a complex, time-consuming task and also not cost-effective. Moreover, the application requires massive data collection and processing at the autonomous end devices, which need proper validation before deploying in the real environment. We also need to test the scalability and usability of the new methods and techniques.

Because of the efforts of the researchers in distributed computing domain, there are many simulators currently available for network, cloud, and IoT simulation. However, these simulators are not perfectly suitable for IoT and edge due to some exclusive features of edge computing. These features are including highly heterogeneous devices, communication

TABLE 6 Comparison of various open-source simulation tools that are currently using for IoT or edge simulation

Simulator Name	Features							
	Network Comm.	Heterogeneity	Edge Comm. Protocols	Network Protocols	Mobility	Battery Power	IoT Devices	Edge Processing
Omnet++[25]	✓	✓		✓				
TOSSIM[28]	✓	✓		✓				
PowerTOSSIMz[29]	✓	✓		✓		✓		
CloudSim[6]	✓	✓						
iCanCloud[31]	✓	✓						
GreenCloud[7]	✓	✓						
DCSim[33]	✓	✓						
NS-3[30]	✓	✓		✓				
SimIoT[34]	✓						✓	
COOJA[35]	✓	✓					✓	
iFogSim[9]	✓						✓	✓
IoTSim[36]							✓	
EdgeCloudSim[8]	✓				✓			✓
DISSECT-CF[37]							✓	
IoTSim-Edge (proposed)	✓	✓	✓	✓	✓	✓	✓	✓

Abbreviation: DISSECT-CF, DIScrete event baSed Energy Consumption simulaTor for Clouds and Federations; IoT, Internet of Things.

protocols, and mobility. To incorporate all these exclusive characteristics, we proposed a novel IoTSim-Edge simulator that models numerous features, including the following: (i) device heterogeneity; (ii) application composition; (iii) variety of IoT communication protocols; (iv) device movement and mobility; and (v) battery features. It benefits researchers to develop their own prototype and test in a scalable simulation environment. They can identify the bottlenecks and test the performance of their methods and techniques with no cost, which will help them to improve the usability and performance of their proposed techniques.

We validated the effectiveness of the simulator by considering three test cases: healthcare system, smart building, and capacity planning for RSUs. The result showed the varying capability of IoTSim-Edge in terms of application composition, battery-oriented modeling, heterogeneous protocols modeling, and mobility modeling along with the resources provisioning for IoT application in an efficient manner.

As the future works, we will develop an IoT-Edge emulator that consists of all the above features and integrate with the IoTSim-Edge. This can help to test all these functionality in a realistic environment.

CONFLICT OF INTEREST

The authors declare no potential conflict of interests.

ORCID

Devki Nandan Jha https://orcid.org/0000-0003-1322-2588

REFERENCES

1. Yannuzzi M, Milito R, Serral-Gracià R, Montero D, Nemirovsky M. Key ingredients in an IoT recipe: fog computing, cloud computing, and more fog computing. Paper presented at: Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2014 19th IEEE International Workshop on; 2014, 325–329.
2. Shi W, Dustdar S. The promise of edge computing. *Computer*. 2016;49(5):78-81.
3. Kalantarian H, Sideris C, Mortazavi B, Alshurafa N, Sarrafzadeh M. Dynamic computation offloading for low-power wearable health monitoring systems. *IEEE Trans Biomed Eng*. 2017;64(3):621-628.
4. Dolui K, Datta SK. Comparison of edge computing implementations: fog computing, cloudlet and mobile edge computing. Paper presented at: 2017 IEEE Global Internet of Things Summit (GIoTS); 2017, 1–6.

5. Kecskemeti G, Casale G, Jha DN, Lyon J, Ranjan R. Modelling and simulation challenges in internet of things. *IEEE Cloud Comput*. 2017;4(1):62-69.
6. Calheiros RN, Ranjan R, Beloglazov A, De Rose CAF, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Pract Exper*. 2011;41(1):23-50.
7. Kliazovich D, Bouvry P, Khan SU. GreenCloud: a packet-level simulator of energy-aware cloud computing data centers. *J Supercomput*. 2012;62(3):1263-1283.
8. Sonmez C, Ozgovde A, Ersoy C. Edgecloudsim: an environment for performance evaluation of edge computing systems. *Trans Emerg Telecommun Technol*. 2018;29(11):e3493.
9. Gupta H, Vahid Dastjerdi A, Ghosh SK, Buyya R. iFogSim: a toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Pract Exper*. 2017;47(9):1275-1296.
10. Yazici M, Basurra S, Gaber M. Edge machine learning: enabling smart internet of things applications. *BDCC*. 2018;2(3):26.
11. Al-Fuqaha A, Guizani M, Mohammadi M, Aledhari M, Ayyash M. Internet of things: a survey on enabling technologies, protocols, and applications. *IEEE commun surv tutor*. 2015;17(4):2347-2376.
12. Villari M, Fazio M, Dustdar S, Rana O, Jha DN, Ranjan R. Osmosis: the osmotic computing platform for microelements in the cloud, edge, and internet of things. *Computer*. 2019;52(8):14-26.
13. Nidal N, Ahmed H, Hossam H. Handoffs in fourth generation heterogeneous networks. *IEEE Commun Mag*. 2006;44(10):96-103.
14. Härri J, Fiore M, Filali F, Bonnet C. Vehicular mobility simulation with VanetMobiSim. *Simulation*. 2011;87(4):275-300.
15. Carnevale L., Celesti A., Galletta A., Dustdar S., Villari M.. From the cloud to edge and IoT: a smart orchestration architecture for enabling osmotic computing. Paper presented at: 2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA); 2018, 419-424.
16. Vasileios K, Periklis C, Francisco V-G, Jesus A-Z. A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud computing*. 2015;3(1):11-17.
17. Hassan G, Roozbeh J. Physical movement monitoring using body sensor networks: a phonological approach to construct spatial decision trees. *IEEE Trans Ind Informat*. 2011;7(1):66-77.
18. Nabil A, Wenyao X, Liu Jason J, et al. Designing a robust activity recognition framework for health and exergaming using wearable sensors. *IEEE J Biomed Health Inform*. 2014;18(5):1636-1646.
19. Michalák P, Watson P. PATH2iot: a holistic, distributed stream processing system. Paper presented at: 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom); 2017, 25–32.
20. Thomas B, Cook D. Activity-aware energy-efficient automation of smart buildings. *Energies*. 2016;9(8):624.
21. Balani R, Energy Consumption Analysis for Bluetooth, WiFi and Cellular Networks. University of California at Los Angeles, Technical Report TR-UCLA-NESL-200712-01; 2007.
22. Rocha P, Siddiqui A, Stadler M. Improving energy efficiency via smart building energy management systems: a comparison with policy measures. *Energ Buildings*. 2015;88:203-213.
23. Farzeen M, Shoaib A, Ishfaq HM, Muqeem SA, Moongu J. Autonomous vehicle: the architecture aspect of self driving car. Paper presented at: Proceedings of the 2018 ACM International Conference on Sensors, Signal and Image Processing SSIP 2018; 2018; New York, NY, 1–5.
24. Tonguz Ozan K, Wantanee V. Cars as roadside units: a self-organizing network solution. *IEEE Commun Mag*. 2013;51(12):112-120.
25. Varga A, Hornig R. An overview of the OMNeT++ simulation environment. Paper presented at: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops; 60ICST (Institute for Computer Sciences, Social-Informatics and âĂé; 2008.
26. Boulis A. Castalia, *A simulator for Wireless Sensor Networks and Body Area Networks*. Australia: NSW; 2013. https://github.com/boulis/Castalia.git.
27. Benedetti D, Petrioli C, Spenza D. GreenCastalia: an energy-harvesting-enabled framework for the Castalia simulator. Paper presented at: Proceedings of the 1st ACM International Workshop on Energy Neutral Sensing Systems; 2013, 7.
28. Levis P, Lee N, Welsh M, Culler D. TOSSIM: accurate and scalable simulation of entire TinyOS applications. Paper presented at: Proceedings of the 1st ACM International Conference on Embedded Networked Sensor Systems; 2003, 126–137.
29. Perla E, Catháin AÓ, Carbajo RS, Huggard M, Mc Goldrick C. PowerTOSSIM z: realistic energy modelling for wireless sensor network environments. Paper presented at: Proceedings of the 3rd ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks; 2008, 35–42.
30. Henderson Thomas R, Mathieu L, Riley George F, Craig D, Joseph K. Network simulations with the ns-3 simulator. *SIGCOMM Demonstration*. 2008;14(14):527.
31. Núñez A, Vázquez-Poletti JL, Caminero AC, Castañé GG, Carretero J, Llorente IM. iCanCloud: a flexible and scalable cloud infrastructure simulator. *J Grid Comput*. 2012;10(1):185-209.
32. Garg SK, Buyya R. Networkcloudsim modelling parallel applications in cloud simulations. Paper presented at: Proceedings of 2011 4th IEEE International Conference on Utility and Cloud Computing; 2011, 105–113.
33. Tighe M, Keller G, Bauer M, Lutfiyya H. DCSim: a data centre simulation tool for evaluating dynamic virtualized resource management. Paper presented at: Proceedings of 2012 8th IEEE International Conference on Network and Service Management (Cnsm) and 2012 Workshop on Systems Virtualiztion Management (SVM); 2012, 385–392.
34. Sotiriadis S, Bessis N, Asimakopoulou E, Mustafee N. Towards simulating the internet of things. Paper presented at: Proceedings of 2014 28th IEEE International Conference on Advanced Information Networking and Applications Workshops; 2014, 444–448.

35. Österlind F, Dunkels A, Eriksson J, Finne N, Voigt T. Cross-level sensor network simulation with cooja. Paper presented at: Proceeding of 1st IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006); 2006.
36. Zeng X, Garg SK, Strazdins P, Jayaraman PP, Georgakopoulos D, Ranjan R. IOTSim: a simulator for analysing IoT applications. *J Syst Archit*. 2017;72:93-107.
37. Gabor K. DISSECT-CF: a simulator to foster energy-aware scheduling in infrastructure clouds. *Simul Model Pract Theory*. 2015;5:188-218.

How to cite this article: Jha DN, Alwasel K, Alshoshan A, et al. IoTSim-Edge: A simulation framework for modeling the behavior of Internet of Things and edge computing environments. *Softw Pract Exper*. 2020;1–24. https://doi.org/10.1002/spe.2787