Received: 22 October 2016 | Revised: 3 May 2017 | Accepted: 5 May 2017
DOI: 10.1002/spe.2509

SPECIAL ISSUE PAPER

WILEY

iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments

Harshit Gupta[1,2] | Amir Vahid Dastjerdi[1] | Soumya K. Ghosh[3] | Rajkumar Buyya[1]

[1]Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Parkville, VIC, Australia

[2]School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA

[3]Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Kharagpur, India

Correspondence
Harshit Gupta, School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA.
Email: harshitg@gatech.edu

Summary

Internet of Things (IoT) aims to bring every object (eg, smart cameras, wearable, environmental sensors, home appliances, and vehicles) online, hence generating massive volume of data that can overwhelm storage systems and data analytics applications. Cloud computing offers services at the infrastructure level that can scale to IoT storage and processing requirements. However, there are applications such as health monitoring and emergency response that require low latency, and delay that is caused by transferring data to the cloud and then back to the application can seriously impact their performances. To overcome this limitation, Fog computing paradigm has been proposed, where cloud services are extended to the edge of the network to decrease the latency and network congestion. To realize the full potential of Fog and IoT paradigms for real-time analytics, several challenges need to be addressed. The first and most critical problem is designing resource management techniques that determine which modules of analytics applications are pushed to each edge device to minimize the latency and maximize the throughput. To this end, we need an evaluation platform that enables the quantification of performance of resource management policies on an IoT or Fog computing infrastructure in a repeatable manner. In this paper we propose a simulator, called *iFogSim*, to model IoT and Fog environments and measure the impact of resource management techniques in latency, network congestion, energy consumption, and cost. We describe two case studies to demonstrate modeling of an IoT environment and comparison of resource management policies. Moreover, scalability of the simulation toolkit of RAM consumption and execution time is verified under different circumstances.

KEYWORDS

Edge computing, Fog computing, Internet of Things (IoT), modeling and simulation

1 | INTRODUCTION

The Internet of Things (IoT) paradigm promises to make “things”—including consumer electronic devices or home appliances such as medical devices, fridge, cameras, and sensors—part of the Internet environment. This paradigm opens the doors to new innovations that will build novel types of interaction among things and humans and enables the realization of smart cities, infrastructures, and services for enhancing quality of life and use of resources. The IoT envisions a new world of connected

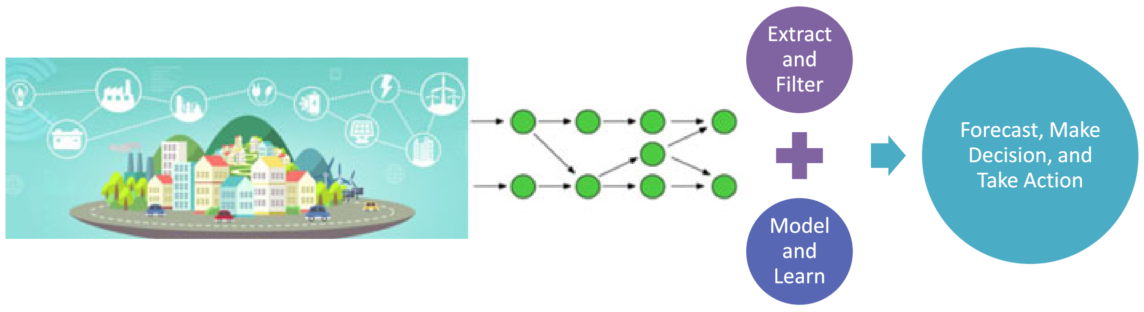


FIGURE 1 A typical IoT environment with basic modules. [Colour figure can be viewed at wileyonlinelibrary.com]

devices and humans in which quality of life is enhanced, by supporting intelligent analytics on data generated by devices affecting our daily lives, making management of infrastructure less cumbersome and disaster recovery more efficient. On the basis of bottom-up analysis for IoT applications, McKinsey estimates that the IoT has a potential economic impact of $11 trillion dollar per year by 2025,which would be equivalent to about 11% of the world economy.[1] They also expect 1 trillion IoT devices will be deployed by 2025.

Although technologies and solutions enabling connectivity and data delivery are growing rapidly, not enough attention has been given to real-time analytics and decision making as one of the major objectives of IoT (Figure 1). Majority of current IoT data processing solutions transfer data collected from IoT devices to cloud for long-term processing. This is mainly because existing data analytics approaches are designed to deal with large volume of data, but not real-time data processing and dispatching. With millions of things generating data, transferring all of that to the cloud is neither scalable nor suitable for real-time decision making. The dynamic nature of IoT environments and its associated real-time requirements and increasing processing capacity of edge devices (entry point into provider core networks, eg, gateways)[2] has lead to the evolution of the Fog computing paradigm. Fog computing[3] extends cloud services to the edge of networks, which results in latency reduction through geographical distribution of IoT application components, and increased scalability for handling large-scale deployments.

Many IoT applications (eg, stream processing) are naturally distributed and are often embedded in an environment with numerous connected computing devices with heterogeneous capabilities. As data travel from its point of origin (eg, sensors) towards applications deployed in cloud virtual machines, it passes through many devices, each of which is a potential target of computation offloading. Therefore, it is important to take advantage of computational and storage capabilities of these intermediate devices. One of the main challenges in using in-network resources is efficient application design. An application built for running on a Fog infrastructure should be partitioned in a way that it can leverage the real-time response from edge devices and use the immense resource availability of the cloud—both at the same time. Suboptimal application design can lead to poor user experience (in perceived latency) or overuse of edge devices. Hence, applications need to be broken down into components on the basis of the kind of guarantees they demand from the underlying infrastructure. Another challenge lies in designing resource management policies, which handle scheduling of application components in the pool of fog devices—stretching from the network edge to the cloud—to meet application level quality-of-service (QoS) requirements such as end-to-end latency or privacy requirements while minimizing resource and energy wastage. Such policies have been an integral part of cloud-based systems and possess a greater complexity in fog computing because of the heterogeneity, large scale, and loosely coupled nature of fog infrastructure.

To foster innovation and development enabling real-time analytics in fog computing, we require an evaluation environment for exploring different application designs and resource management policies including operator (operator and application module have been used interchangeably in the paper) and task placement, migration, and consolidation. A real IoT environment as a testbed, although desirable, in many cases is too costly and does not provide repeatable and controllable environment. To address this shortcoming, we propose a simulator called *iFogSim* that enables the simulation of resource management and application scheduling policies across edge and cloud resources under different scenarios and conditions.

In this paper, we discuss the architecture of iFogSim along with its design and implementation. The framework is designed in a way that makes it capable of evaluation of resource management policies applicable to fog environments with respect to their impact on latency (timeliness), energy consumption, network congestion, and operational costs. iFogSim also allows application designers to test the design of their application against metrics like cost, network use, and perceived latency. It simulates edge devices, cloud data centers, and network links to measure performance metrics. The major application model considered for iFogSim is the *Sense-Process-Actuate* model, wherein sensors publish measured data either periodically or in an event-based manner, applications running on fog devices subscribe to and process data coming from sensors, and finally, insights obtained are translated to actions forwarded to actuators. In addition, we present a simple IoT simulation recipe and two case studies to demonstrate how one can model an IoT environment and plug in and compare resource management policies. Finally, we evaluate the scalability of iFogSim in memory consumption and simulation execution time.

The paper is structured as follows: A formal definition of fog computing, its concepts, and benefits are presented in Section 2. Section 3 discusses the architecture of iFogSim followed by its implementation details, sample resource management policies, and a generic simulation recipe in Section 4. Case studies along with their application models and network topologies are expounded in Section 5. Section 6 presents results of scalability tests on iFogSim and compares two basic resource management policies considering metrics such as latency, energy consumption, and network use. Finally, Section 8 concludes the paper and discusses the future directions.

2 | FOG COMPUTING—DEFINITION AND CONCEPTS

Fog computing is a term coined by Cisco and defined as a distributed computing paradigm that extends the services provided by the cloud to the edge of the network.[3] It enables the seamless convergence of infrastructure stretching from the cloud datacenter to devices on the network edge (including intermediate devices like ISP gateways, cellular base stations, and private cloud deployments) into a continuum of resources, to be provisioned to multiple tenants for hosting applications (see Figure 2). Components of an application are hence able to run in a geo-distributed fashion using the services provided by the distributed infrastructure, wherein the resource management policies control and manage the resource services offered by each device. Bonomi et al identified the defining characteristics of fog computing as edge location, dense geographical distribution, large-scale of deployment, support for mobility, resource and interface heterogeneity, and interplay with the cloud properties that can address requirements of mobile applications that need low latency with a wide and dense geographical distribution. Fog computing takes advantages of the interplay between the edge and the cloud—while it benefits from edge devices' close proximity to the endpoints, it also leverages the on-demand scalability of cloud resources.

There are a number of benefits associated with adopting the Fog computing paradigm over the *use-the-cloud* approach. The first benefit is the reduction of traffic sent to the backbone network—as uncontrolled increase in network traffic may lead to congestion and result in increased latency. Fog computing provides a platform for filtering and analysis of the data generated by sensors by using resources of edge devices. This drastically reduces the traffic being sent to the cloud by allowing the placement

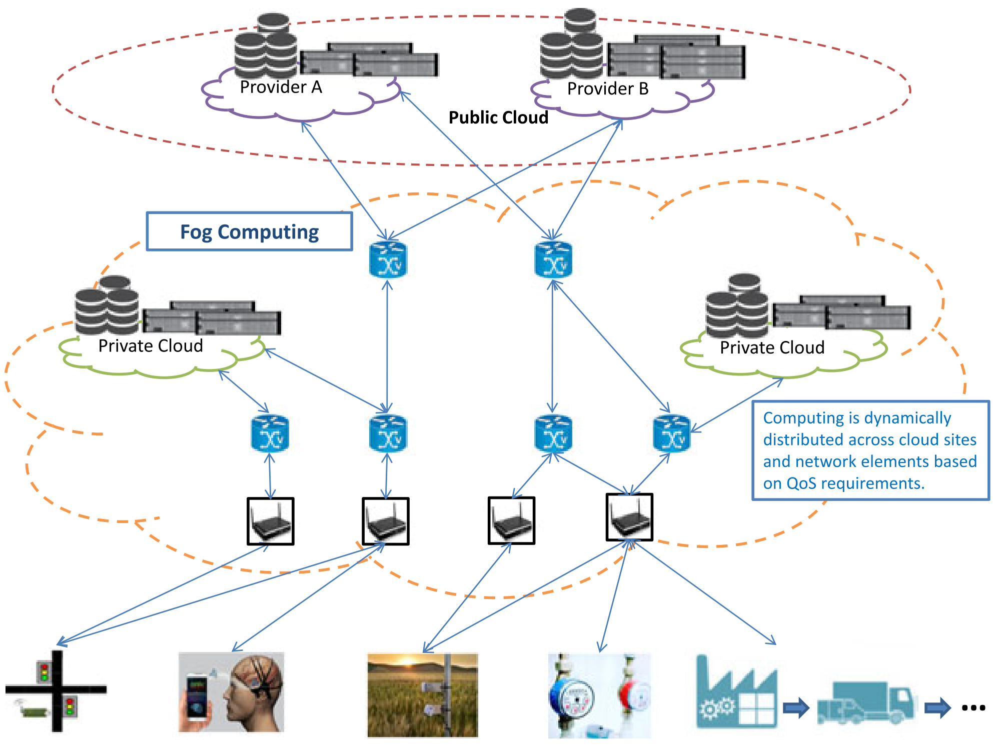

Provider A
Provider B
Public Cloud
Fog Computing
Private Cloud
Private Cloud
Computing is dynamically distributed across cloud sites and network elements based on QoS requirements.
...

FIGURE 2 Distributed data processing in a Fog computing environment. [Colour figure can be viewed at wileyonlinelibrary.com]

of filtering operators close to the source of data. Considerable reduction in propagation latency is the next important advantage of using Fog computing paradigm especially for mission critical applications that require real-time data processing. Some of the best examples of such applications are cloud robotics, control of fly-by-wire aircraft, or antilock brakes on a vehicle. In the case of cloud robotics, the effectiveness of motion control is contingent on the timeliness of data collection by the sensors, processing of the control system, and feedback to the actuators. Having the control system running on the cloud may make the sense-process-actuate loop slow or unavailable because of communication failures. This is where Fog computing helps by performing the processing of the control system very close to the robots—thus making real-time response possible. Finally, cloud computing paradigm, even with it is virtually infinite resources, can become a bottleneck if all the raw data generated by end devices (sensors) is sent to a centralized cloud. Fog computing is capable of filtering and processing considerable amount of incoming data on edge devices, making the data processing architecture distributed and thereby scalable.

3 | ARCHITECTURE

The architecture of Fog computing environment in iFogSim (presented in Figure 3) is comprised of multiple layers, with each layer responsible for specific tasks to facilitate operation of higher layers. In the architecture, the bottommost layer comprises of IoT devices that is those that interact with real world and are the source or sink of data. *IoT sensors* act as the source of data for applications and are distributed in different geographical locations, sensing the environment and emitting observed values to upper layers via gateways for further processing. Similarly, *IoT actuators* operate at the bottommost layer of the architecture and are responsible for controlling a mechanism or system. Actuators are usually designed to respond to changes in environments that are dictated by applications on the basis of information captured by sensors. Every device in the IoT is either a source or a sink of data and hence can be modeled by a sensor or an actuator, respectively. The authors of the previous work[4] identify 5 types of input technologies, namely, sensors, smart readers, cameras, microphones, and collectors. Any device belonging to these kind has a certain data-emission characteristics, for example, interemission time or size of data chunk emitted. In iFogSim, a sensor is associated with certain data-emission characteristics, which can be customized to simulate any kind of data-emitting IoT device, ranging from smart cameras (shown in case study 2) to wearable, environmental sensors to mobile

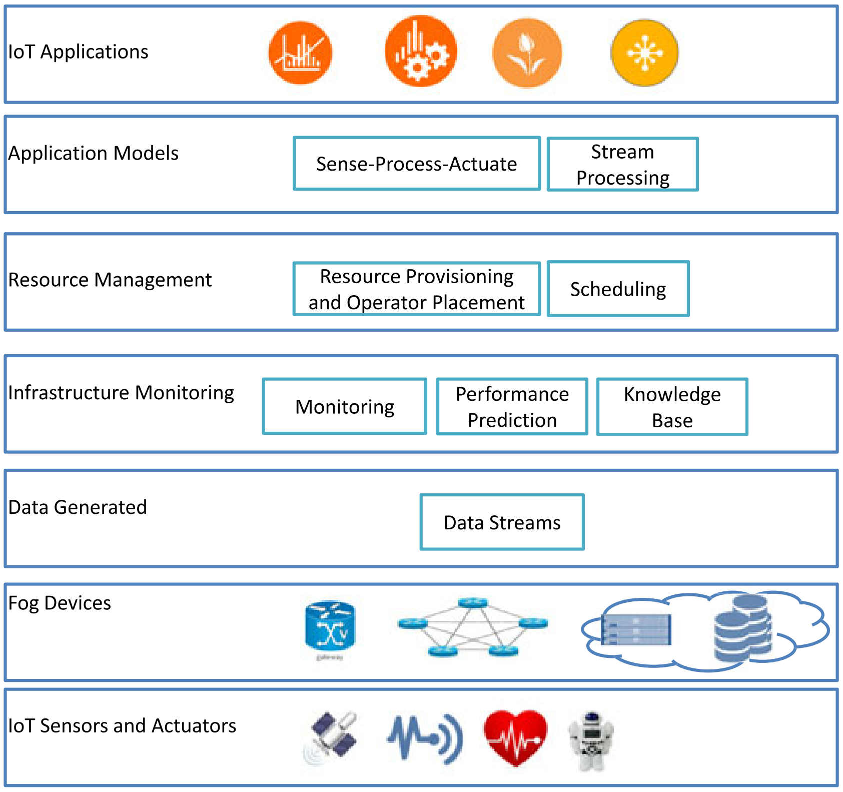


FIGURE 3 Fog computing architecture. [Colour figure can be viewed at wileyonlinelibrary.com]

vehicles, encompassing those identified by the aforementioned paper. Same with actuators, it can be customized to simulate the effects of received information from applications.

As iFogSim does not deal with the low-level network issues such as interference management between densely colocated devices, the users need to abstract these low-level issues to high-level attributes like latency or bandwidth of connection between IoT devices and gateways. Thorough profiling can enable the user to build a model of physical level behavior of wireless characteristics of IoT devices, which can be plugged into iFogSim to simulate those effects.

Fog device is any element in the network that is capable of hosting application modules. Fog devices that connect sensors to the network are generally called gateways. Fog devices also include cloud resources that are provisioned on-demand from geographically distributed data centers. The fog device layer encompasses the entire resource continuum (mentioned in previous section) stretching from edge devices to the cloud. Devices are arranges in a hierarchical topology with direct communication possible only between a parent-child pair in the hierarchy. An application module running on a fog device is responsible for processing all the data generated from elements below the device in the hierarchy.

iFogSim is based on the definition of fog computing that presents it as an infrastructure having similar characteristics as cloud computing but placed close to the edge of the network. It does not support device-to-device communication as it assumes a hierarchical organization of fog devices. Application placement happens in a north-south direction, and it is not possible (at present) to offload modules to another device on the same level of hierarchy. Hence, scenarios such as smartphone-to-smartphone offloading are not possible with the current version. In addition to that, direct communication between two devices at the level of hierarchy is also not possible in the current version because of the hierarchical organization. We are working towards getting rid of this hierarchical organization to allow more flexible communication patterns.

IoT Data Streams, which are sequences of values (referred to as *tuple* in *iFogSim*) emitted by devices, form the next layer of the architecture. These streams can be emitted by sensors (in which case they contain raw data) or may be transmitted from an application module to another or even from an application module to actuators. Data streams are also generated by fog devices, in the form of resource use details, which are processed by the monitoring layer for gaining insight into the state of devices.

Monitoring layer keeps track of the resource use, power consumption, and availability of sensors, actuators, and fog devices. Monitoring components supply this information to the resource management layer and can provide it to other services as required. For the sake of simplicity of iFogSim, relatively complicated monitoring layer components like performance prediction and knowledge base have not been included in the current version. These components can, however, be realized by writing entities that process resource use statistics generated by fog devices and availability messages emitted by all fog entities respectively.

Resource management is the core component of the architecture and consists of components that coherently manage resources of the fog device layer in such a way that application level QoS constraints are met and resource wastage is minimized. To this end, placement and scheduler components play a major role by keeping track of the state of available resources (information provided by the monitoring service) to identify the best candidates for hosting an application module and allocating the device's resources to the module. This layer functions on the basis of the quality of services requirements posed by the application layer, so that individual application components can experience the quality that they demand. The resource management policy can be complicated enough to allow migration of components and dynamic changes in allocation of device resources to components, or be as simple as statically provisioning components on a fog device. Furthermore, the implementation of the resource management layer can be distributed (with each device managing its own resources without global knowledge) or centralized (with all device sending resource information to a central resource manager), or a hybrid of both. The current version of iFogSim, however, provides a static application placement policy—with application modules being statically allocated to fog devices. This policy can be replaced by dynamic policies that can migrate modules to other fog devices based on criteria like energy consumption and perceived latency.

Application (programming) models. The applications developed for deployment in the fog are based on the distributed data flow (DDF) model.[5] An application is modeled as a collection of modules, which constitute the data processing elements. Data generated as output by module *i* may be used as input by another module *j*, giving rise to data dependency between module *i* and *j*. This application model allows us to represent an application in the form of a directed graph, with the vertices representing application modules and directed edges showing the flow of data between modules. Later, we present two sample applications modeled as DDF.

One of the major drivers of fog computing into existence has been the need of real-time response and increased scalability concomitant with the proliferation of IoT. The IoT applications tend to have sensors as sources of data, which are often in the form of tuples. The proposed iFogSim architecture supports two models used for IoT applications.

1. Sense-process-actuate model. The information collected by sensors is transmitted as data streams, which is acted upon by applications running on fog devices and the resultant commands are sent to actuators.
2. Stream-processing model. The stream-processing model has a network of application modules running on fog devices that continuously process data streams emitted from sensors. The information mined from the incoming streams is stored in data centers for large-scale and long-term analytics.

We consider *stream processing* model as a subcategory of the sense-process-actuate model. These models can, however, be extended to cater to use cases other than IoT applications. The fog can also be used to run batch applications, and the simulator can be easily extended to cater to this use case. Particularly, by setting the appropriate tuple lengths and frequency of tuple emission from sensors, the developer can simulate characteristics of batch processing. In such scenarios, sensors would be representing other sources of data and not IoT sensors per se.

4 | DESIGN AND IMPLEMENTATION

For implementing functionalities of iFogSim architecture, we leveraged basic event simulation functionalities found in CloudSim.[6] Entities in CloudSim, like data centers, communicate between each other by message passing operations (sending events, to be more precise). Hence, the core CloudSim layer is responsible for handling events between Fog computing components in iFogSim. The main classes of iFogSim are depicted in Figures 4 and 5. In this section, we present the details of these classes and their interactions. The implementation of iFogSim is constituted by simulated entities and services. First, we describe how the elements of architecture are modeled as iFogSim classes.

- **FogDevice**. This class specifies hardware characteristics of fog devices and their connections to other fog devices, sensors, and actuators. Having been realized by extension from the *PowerDatacenter* class in CloudSim, the major attributes of the FogDevice class are accessible memory, processor, storage size, uplink, and downlink bandwidths (defining the communication capacity of fog devices). Methods in this class define how the resources of a fog device are scheduled between application modules running on it and how modules are deployed and decommissioned on them. Overriding these methods enables developers to plug-in custom policies for the above-mentioned functions.
- **Sensor**. Instances of the sensor class are entities that act as IoT sensors described in the architecture. The class contains attributes representing the characteristics of a sensor, ranging from its connectivity to output attributes. The class contains a reference attribute to the gateway fog device to which the sensor is connected and the latency of connection between them. Most importantly, it defines the output characteristics of the sensor and the distribution of tuple intertransmission, which identifies the tuple arrival rate at the gateway. By setting appropriate values of these attributes, devices like smart cameras and connected cars can be simulated.
- **Actuator**. This class models an actuator by defining the effect of actuation and its network connection properties. The class defines a method to perform an action on arrival of a tuple from an application module, which can be overridden to implement

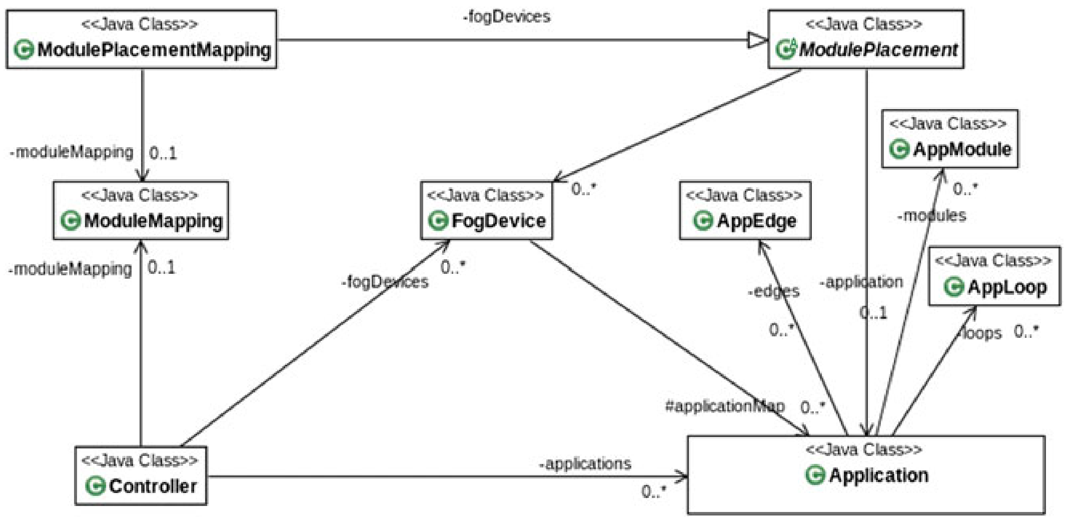


FIGURE 4 Fundamental classes of *iFogSim*. [Colour figure can be viewed at wileyonlinelibrary.com]

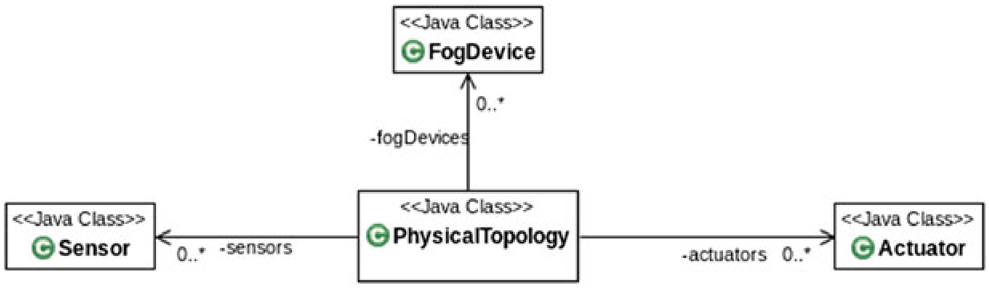


FIGURE 5 *iFogSim* physical topology classes. [Colour figure can be viewed at wileyonlinelibrary.com]

custom effects of actuation. An attribute in the class refers to the gateway to which the actuator is connected and the latency of this connection.

- **Tuple**. Tuples form the fundamental unit of communication between entities in the Fog and realize the data stream layer in the architecture. Tuples are represented as instances of tuple class in iFogSim, which is inherited from the *Cloudlet* class of CloudSim. A tuple is characterized by its type and the source and destination application modules. The attributes of the class specify the processing requirements [defined as million instructions (MI)] and the length of data encapsulated in the tuple.
- **Application**. The application design in iFogSim follows the DDF model, in which an application is modeled as a directed graph, the vertices of the directed acyclic graph (DAG) representing modules that perform processing on incoming data and edges denoting data dependencies between modules. These entities are realized using the following classes.
  - **AppModule**. Instances of *AppModule* class represent processing elements of fog applications and realize the vertices of the DAG in DDF model. *AppModule* is implemented by extending the class *PowerVm* in CloudSim. For each incoming tuple, an *AppModule* instance processes it and generates output tuples that are sent to next modules in the DAG. The number of output tuples per input tuple is decided using a selectivity model—which can be based on a fractional selectivity or a bursty model.
  - **AppEdge**. An *AppEdge* instance denotes the data dependency between a pair of application modules and represents a directed edge in the DDF application model. Each edge is characterized by the type of tuple it carries, which is captured by the *tupleType* attribute of AppEdge class along with the processing requirements and length of data encapsulated in these tuples. *iFogSim* supports two types of application edges—periodic and event based. Tuples on a periodic *AppEdge* are emitted at regular intervals. A tuple on an event-based edge $e = (u, v)$ is sent when the source module u receives a tuple and the selectivity model of u allows the emission of tuples carried by e.
  - **AppLoop**. *AppLoop* is an additional class, used for specifying the process-control loops of interest to the user. In iFogSim, the developer can specify the control loops to measure the end-to-end latency. An AppLoop instance is fundamentally a list of modules starting from the origin of the loop to the module where the loop terminates.

A sequence diagram demonstrating tuple emission and subsequent execution is shown in Figure 6. A tuple is generated by a sensor and sent to the gateway the sensor is connected to. The callback function for handling an incoming tuple *processTupleArrival()* is called once the tuple reaches the fog device (gateway). In case the tuple needs to be routed to another Fog device, it is sent immediately without processing. Otherwise, if the application module on which the tuple needs to be executed is placed on the receiving fog device, the tuple is submitted for execution. The function *checkCloudletCompletion()* is called on the fog device on completion of execution of the tuple.

In addition to the basic tuple processing functionalities, simulated services available in *iFogSim* are as follows:

- **Monitoring service**. In the current version of iFogSim, each device monitors and maintains its current resource use statistics. The *executeTuple()* method in the FogDevice class contains the tuple processing logic where the device updates its resource use. These statistics can also be encapsulated in a tuple and sent to the resource management layer for running use-aware resource management policies. Such information may be useful to the user to study performance of the application on fog infrastructure and can be obtained as logs to be analyzed offline. However, the current version of the simulator does not present the raw use values to the user.

  These resource use values are fed into a related power model to calculate the power consumption of the device, which is reported at the end of the simulation. Each fog device (a FogDevice instance) is associated with a power model (eg, PowerModelLinear), which estimates the power consumption at a given CPU use.

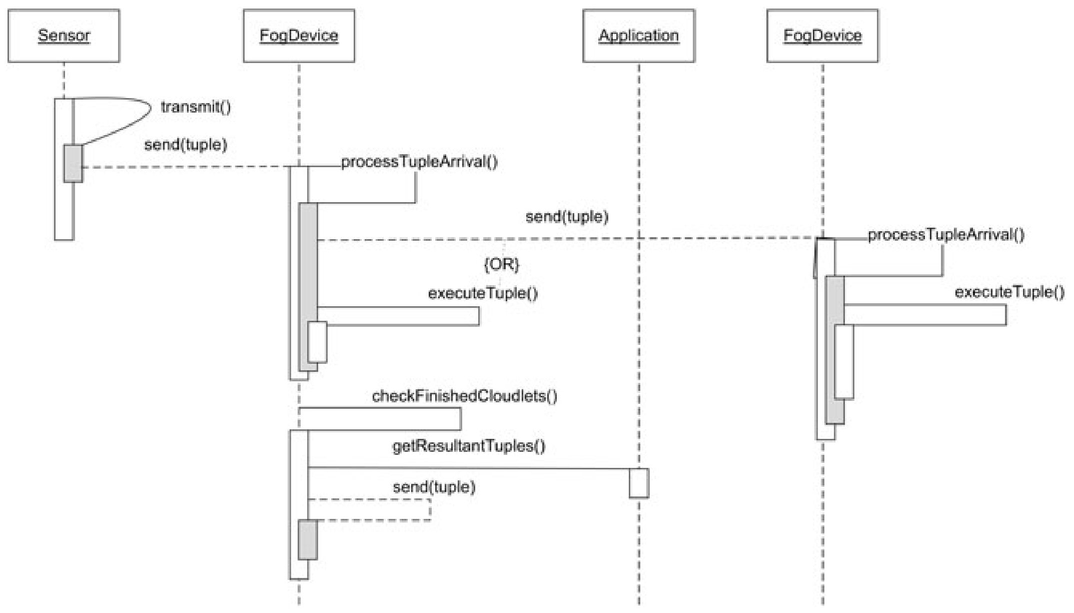


FIGURE 6 Sequence diagram of the generation and execution

- **Resource management service**. iFogSim has two levels of resource management for applications—placement and scheduling—which are abstracted as separate policies to facilitate extension and customization.

  1. Application placement. The placement policy determines how application modules are placed across Fog devices upon submission of application. The placement process can be driven by objectives such as minimizing end-to-end latency, network use, operational cost, or energy consumption. The class *ModulePlacement* is the abstract placement policy that needs to be extended for integrating new policies.
  2. Application scheduling. Scheduling resources of the host fog device to application modules forms the second level of resource management. The default resource scheduler equally divides a device's resources among all active application modules. The application scheduling policy can be customized by overriding the method *updateAllocatedMips* inside the class *FogDevice*.

4.1 | Properties inherited from CloudSim

iFogSim has been implemented as an extension of CloudSim, and hence inherits a number of features from CloudSim which are important to note.

- **Coexecution of multiple applications**. *iFogSim* allows the execution of multiple applications on the infrastructure at the same time. Each application would have to create distinct sensors/actuators and an application model, which will be placed in the fog infrastructure using the module placement policy. The user can also encode different behavior for different applications in the placement policy, thus enabling application specific module placement policies.
- **Migration of application modules**. *iFogSim* supports migration of application modules from one fog device to another, because *AppModule* is just an extension of CloudSim's *VM* class. The fog devices involved in the migration need to perform relevant bookkeeping to register the migration and direct traffic to the migrating module.
- **Pluggable resource management policies**. Resource management policies in *iFogSim* can be implemented as pluggable modules which are decoupled from the simulation. This allows easy repeatable experimentation of same scenario with multiple resource management policies.

4.2 | Built-in module placement strategies

iFogSim is packaged with two application module placement strategies—*cloud-only placement* and *edge-ward placement*. The objective of discussing such algorithms was to throw light on the ability of *iFogSim* to support development and evaluation of custom module placement algorithms and their evaluations. Recently, researchers from the University of Cardiff and Rutgers University have used iFogSim in modeling of electroencephalography (EEG) tractor beam game and its deployment in fog computing environment at large scale.[7] They demonstrated the ability of iFogSim in supporting evaluation of advance resource provisioning algorithm for their applications.

1. **Cloud-only placement**. The *cloud-only* placement strategy is based on the traditional cloud-based implementation of applications where all modules of an application run in data centers. The sense-process-actuate loop in such applications are implemented by having sensors transmiting sensed data to the cloud where it is processed and actuators are informed if action is required.
2. **Edge-ward placement**. Edge-ward placement strategy favors the deployment of application modules close to the edge of the network. However, devices close to the edge of the network—like routers and access points—may not be computationally powerful enough to host all operators of the application. In such a situation, the strategy iterates on fog devices towards cloud and tries to place remaining operators on alternative devices. This strategy (shown in Algorithm 1) demonstrates the interplay between the fog and the cloud by placing modules both near the network edge and the cloud and is described below. For the sake of simplicity, we introduce terms *north* and *south* to denote devices towards and away from the cloud, respectively.

Algorithm 1: Edge-ward module placement

```
for $p \in PATHS$ do  Across all paths
    $placedModules \leftarrow \{\}$;
    for Fog device $d \in p$ do                                    ▷ leaf-to-root traversal
        $modulesToPlace \leftarrow \{\}$;
        for module $w \in app$ do                                  ▷ find modules ready for placement on device d
            if all predecessors of w are in placedModules then     ▷ if all predecessors are placed
                add $w$ to $modulesToPlace$ ;
            end
        end
        for module $\theta \in modulesToPlace$ do
            if $d$ already has instance of $\theta$ as $\theta'$ then
                if $CPU_{\theta}^{req} \geq CPU_{d}^{avail}$ then  ▷ device d does not have CPU capacity to host θ
                    $\tilde{\theta} \leftarrow merge(\theta, \theta')$ ;
                    $f \leftarrow parent(d)$ ;
                    while $CPU_{\tilde{\theta}}^{req} \geq CPU_{f}^{avail}$ do   ▷ find device north of d for hosting θ
                        $f \leftarrow parent(f)$ ;
                    end
                    Place $\tilde{\theta}$ on device $f$ ;
                    add $\theta$ to $placedModules$ ;
                end
                else                                                ▷ device d can host θ
                    Place $\theta$ on device $d$ ;
                    add $\theta$ to $placedModules$ ;
                end
            end
            else if no device north of $d$ has an instance of $\theta$ then
                if $CPU_{\theta}^{req} \leq CPU_{d}^{avail}$ then  ▷ if not, will be handled by subsequent iterations
                    Place $\theta$ on device $d$ ;
                    add $\theta$ to $placedModules$ ;
                end
            end
        end
    end
end
```

The edge-ward placement algorithm iterates over all leaf-to-root paths in the physical network topology and places modules on each such path. For each path, the algorithm iterates from the leaf (south-most device)—typically en edge device—to the root (north-most device)—typically the cloud—and incrementally places modules on them. For each device in a path, the modules that can be placed on it are identified. A module θ can be placed on a device d only if all other modules that should be placed to the south of θ (predecessors) are already placed in the leaf-to-root path.

Once the modules to be placed are identified, the algorithm tries to place them one-by-one on the device. Consider the case of module θ being placed on device d. The algorithm first checks if an instance of θ is already placed on d (as it may be a part of some other leaf-to-root path). If so, these instances are merged together and placed on d if it can accommodate the merged instances. Otherwise, the algorithm searches for devices north of d to place the merged instance. However, if neither device d nor any device north of d has an instance of θ, the module is placed on d. In case any device north of d had an instance of θ, it would be handled by the earlier steps of the algorithm in a subsequent iteration.

Because iFogSim assumes that a module m placed on device d will handle all incoming tuples from devices south of d in the hierarchy, multiple instances of a module are merged and migrated north by this placement strategy (as discussed in previous paragraphs). Consider devices d_p and d_c where d_p is the parent of d_c in the topology and both host instances of the same module m. In such a case, any incoming traffic for module m on d_c would be processed there and not sent northwards.

4.3 | Graphical user interface

To facilitate description of the physical network topology, a GUI has been built over the iFogSim application logic. The GUI allows the user to draw physical elements such as fog devices, sensors, actuators, and connecting links. Defining characteristics of these entities can be fed to the topology using the GUI. The drawn topologies can be saved and reloaded by converting the topology to and from JSON file format. The physical topologies can be built both through GUI and programmatically through Java APIs. Figure 7 shows a sample physical topology, which includes a sensor, a gateway, a cloud virtual machine, and their connections. The JSON representation of this physical topology is shown in Figure 8.

4.4 | A recipe for simulating IoT environments and testing resource management techniques

This section lays down the high-level steps to simulate an IoT/Fog environment in iFogSim to analyze the performance of applications and resource management policies.

1. First, physical entities need to be created and their capabilities and configurations specified. These include sensors, gateways, and cloud virtual machines, and the links that describe how these entities are connected. As mentioned earlier, this can be achieved either by using the GUI or programmatically using aforementioned iFogSim classes. To model the workload of the system, first, we should set the tuple transmit rates of sensor using *transmitDistribution* (an attribute in Sensor class).

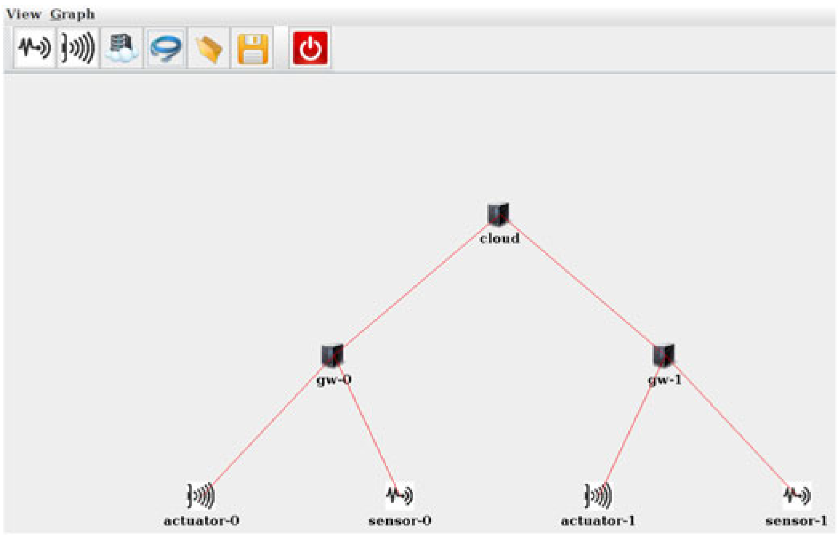
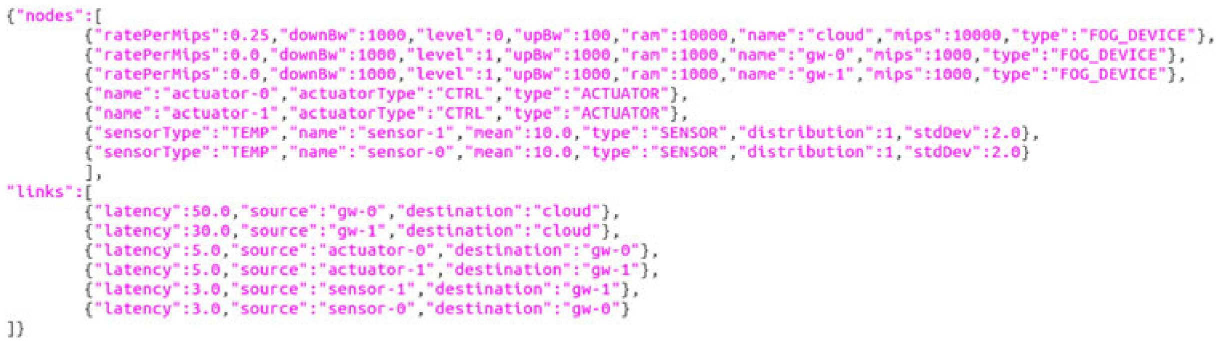


FIGURE 7 *iFogSim* GUI for building network topology. [Colour figure can be viewed at wileyonlinelibrary.com]



```
{"nodes":[
        {"ratePerMips":0.25,"downBw":1000,"level":0,"upBw":100,"ram":10000,"name":"cloud","mips":10000,"type":"FOG_DEVICE"},
        {"ratePerMips":0.0,"downBw":1000,"level":1,"upBw":1000,"ram":1000,"name":"gw-0","mips":1000,"type":"FOG_DEVICE"},
        {"ratePerMips":0.0,"downBw":1000,"level":1,"upBw":1000,"ram":1000,"name":"gw-1","mips":1000,"type":"FOG_DEVICE"},
        {"name":"actuator-0","actuatorType":"CTRL","type":"ACTUATOR"},
        {"name":"actuator-1","actuatorType":"CTRL","type":"ACTUATOR"},
        {"sensorType":"TEMP","name":"sensor-1","mean":10.0,"type":"SENSOR","distribution":1,"stdDev":2.0},
        {"sensorType":"TEMP","name":"sensor-0","mean":10.0,"type":"SENSOR","distribution":1,"stdDev":2.0}
        ],
"links":[
        {"latency":50.0,"source":"gw-0","destination":"cloud"},
        {"latency":30.0,"source":"gw-1","destination":"cloud"},
        {"latency":5.0,"source":"actuator-0","destination":"gw-0"},
        {"latency":5.0,"source":"actuator-1","destination":"gw-1"},
        {"latency":3.0,"source":"sensor-1","destination":"gw-1"},
        {"latency":3.0,"source":"sensor-0","destination":"gw-0"}
]}
```

FIGURE 8 Network topology JSON file. [Colour figure can be viewed at wileyonlinelibrary.com]

Next step in modeling workload is identifying how much resources are required to process the tuples. For defining resource use including CPU and RAM, necessary variables are defined in the tuple class.

2. Second, we need to model applications. As we stated earlier an application is modeled as a DAG, and built via 3 classes of AppModule, AppEdge, and AppLoop.
3. Finally, we need to define placement and scheduling policies that map application modules to fog devices. The policies may consider range of criteria including end-to-end processing latency, throughput, cost, power consumption, and devices constraints. As we mentioned earlier, ModulePlacement and Controller classes are where the placement logic is implemented.

5 | APPLICATION CASE STUDIES

In the this section, we provide 2 simulation case studies, namely, a latency-sensitive online game and intelligent surveillance through distributed camera networks.

5.1 | Case study 1—a latency-sensitive online game

The latency-critical application in the case study is a human-vs-human game (*EEG Tractor Beam Game*)[8] that involves augmented brain-computer interaction. To play the *EEG Tractor Beam game*, each player needs to wear an MINDO-4S wireless EEG headset that is connected to his smartphone. The game runs as an Android application on a user's smartphone. The application performs real-time processing of the EEG signals sensed by the EEG headset and calculates the brain state of the user.

On the application's display, the game shows all the players on a ring surrounding a target object. Each player can exert an attractive force onto the target in proportion to his level of concentration (estimated using a ratio of the average power spectral density in the EEG α, β, and θ bands of the player). To win the game, a player should try to pull the target toward himself by exercising concentration while depriving other players of their chances to grab the target.

Real-time processing requires that the application be hosted very close to the source of data—preferably on the smartphone itself. However, such a deployment would not allow global coverage, which typically requires deploying the application in the cloud. Such a mix of conflicting objectives makes this application a typical use case for Fog computing.

Application model. As illustrated in Figure 9, the application *EEG Tractor Beam Game* consists of 3 major modules that perform processing—*Client*, *Concentration Calculator*, and *Coordinator*. The application modules are modeled in iFogSim using AppModule class. As depicted in Figure 9, there are data dependencies between modules, and these dependences are modeled using AppEdge class in iFogSim. Finally, the control loop of interest for EEG application is modeled in iFogSim using AppLoop class. The application is fed EEG signals by a sensor *EEG* and an actuator *DISPLAY* displays the current game-scene to the user. The functions of the above-mentioned modules are as follows:

1. **Client**. Client module interfaces with the sensor and receives raw EEG signals. It checks the received signal values for any discrepancy and discards any seemingly inconsistent reading. If the sensed signal value is consistent, it sends the value to the *Concentration Calculator* module to get the concentration level of the user from the signal. On receiving the concentration level, it displays it by sending the value to the actuator *DISPLAY*.

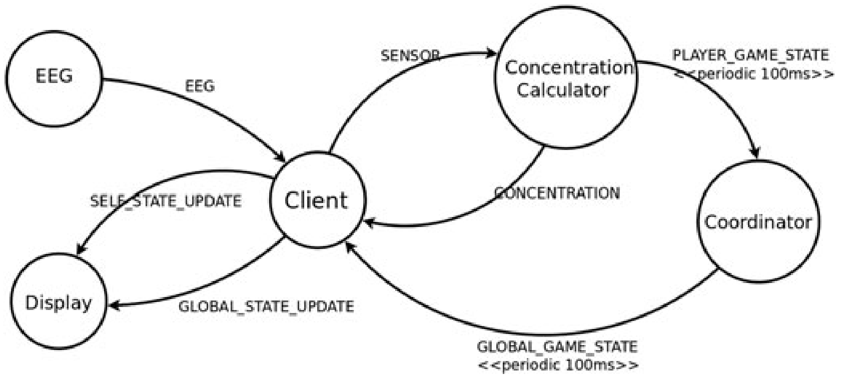


FIGURE 9 Application model of the EEG game

TABLE 1 Description of intermodule edges in the EEG tractor game application

Tuple type	CPU length (MIPS)	N/W length
EEG	2000 (A) / 2500 (B)	500
_ SENSOR	3500	500
PLAYER_ GAME_ STATE	1000	1000
CONCENTRATION	14	500
GLOBAL_ GAME_ STATE	1000	1000
GLOBAL_ STATE_ UPDATE	1000	500
SELF_ STATE_ UPDATE	1000	500

TABLE 2 Configuration of fog devices for EEG tractor game

Device type	CPU, GHz	RAM, GB	POWER, W
Cloud VM	3.0	4	107.339(M) 83.433(I)
WiFi gateway	3.0	4	107.339(M) 83.433(I)
Smartphone	1.6	1	87.53(M) 82.44(I)
ISP gateway	3.0	4	107.339(M) 83.433(I)

2. **Concentration Calculator**. The concentration calculator module is responsible for determining the brain-state of the user from the sensed EEG signal values and calculating the concentration level. This module informs the *Client* module about the measured concentration level so that the game state of the player on the display can be updated.
3. **Coordinator**. Coordinator works at the global level and coordinates the game between multiple players that may be present at geographically distributed locations. The *Coordinator* continuously sends the current state of the game to the *Client* module of all connected users.

The properties of tuples (modeled using Tuple class) carried by edges between the modules in the application are described in Table 1.

5.1.1 | Physical network

For the case study, we have considered a physical topology with 4 fog devices. Table 2 illustrates the configurations[9] of the different types of fog devices used in the topology. Moreover, two different types of EEG headsets have been used—each emitting tuples of different properties, as shown in Table 3. The physical topology of the case study is modeled in *iFogSim* via FogDevice, Sensor, PhysicalTopology, and Actuator classes.

5.2 | Case study 2—intelligent surveillance through distributed camera networks

Distributed system of cameras surveilling an area has garnered a lot of attention in recent years particularly by enabling a broad spectrum of interdisciplinary applications in areas of the likes of public safety and security, manufacturing, transportation, and health care. However, monitoring video streams from the system of cameras manually is not practical. Hence, we need tools

TABLE 3 Configuration of sensors for EEG tractor game

Headset	Tuple CPU length	Average interarrival time, ms
A	2000 million instructions	10
B	2500 million instructions	5

that automatically analyze data coming from cameras and summarize the results in a way that is beneficial to the end-user. The requirements of such a system have been listed down as follows.

- **Low-latency communication**. For effective object coverage, the Pan-tilt-zoom (PTZ) parameters of multiple cameras need to be tuned in real-time on the basis of the captured image. This requires low-latency communication between the cameras and the set of camera control strategies.
- **Handling voluminous data**. Video cameras continuously send captured video frames for processing, which causes a huge traffic, especially when all cameras in a system are taken into account. It is necessary to handle such a large amount of data without burdening the network into a state of congestion.
- **Heavy long-term processing**. The camera control strategy needs to be updated constantly so that it learns the optimal PTZ parameter calculation strategy. This requires analysis of the decisions taken by the control strategy over a long period, which makes this analysis computationally intensive.

Centralized tools for analyzing camera-generated data are not desirable primarily because of the huge amount of data that needs to be sent to the central processing machine. This would not only lead to high latency in the system but also consume more of available bandwidth. Hence, processing the video streams in a decentralized fashion is a more advisable method of analysis.

The Intelligent Surveillance system aims at coordinating multiple cameras with different fields of view (FOVs) to surveil a given area. Coordination between cameras involves coordinated tuning of PTZ parameters so that the best view of the area can be obtained. Furthermore, the system alerts the user in case of irregular events—which may demand attention of the security authorities.

The smart camera detects motion in its FOV and starts sending a video stream to the *Intelligent Surveillance* application. The application locates the moving object in the video stream sent and initiates tracking. Tracking of moving objects is done by constantly tuning the PTZ parameters of the cameras at that site so as to obtain the best view of all the tracked objects. Furthermore, in the event of detection of an event of interest, the application notifies the system user and sends captured video streams to him through the Internet.

Application model. As depicted in Figure 10, the *Intelligent Surveillance* application consists of 5 major modules that perform processing—*Motion Detector*, *Object Detector*, *Object Tracker*, *PTZ Control*, and *User Interface*. This application model has been inspired by the work by Hong et al,[10] wherein they proposed an API for fog applications and modeled a CCTV-based vehicle tracking system using that it. The surveillance application is fed live video streams by a number of CCTV cameras and the PTZ control in each camera continuously adjusts the PTZ parameters. The functions of the above-mentioned modules are as follows:

1. **Motion detection**. This module is embedded inside the smart cameras used in the case study. It continuously reads the raw video streams captured by the camera to find motion of an object. In the event of detection of motion in the camera's FOV, the video stream is forwarded upstream to the object detection module for further processing.
2. **Object detection**. The object detection module receives video streams in which the smart cameras detect motion of an object. The module extracts the moving object from the video streams and compares them with previously discovered objects, which are active in the area currently. In case the detected object has not been in the area before, tracking is activated for this object. In addition, it calculate the coordinates of the objects.
3. **Object tracker**. The object tracker module receives the last calculated coordinates of the currently tracked objects and calculates an optimal PTZ configuration of all the cameras covering the area so that the tracked objects can be captured in the most effective manner. This PTZ information is conveyed to the PTZ control of cameras periodically.
4. **PTZ control**. This module runs on each smart camera and adjusts the physical camera to conform to the optimal PTZ parameters sent by the object tracker module. This module serves as the actuator of the system and is embedded in the smart cameras itself.
5. **User interface**. The application presents a user interface by sending a fraction of the video streams containing each tracked object to the user's device. For this use case, it requires such filtered video streams from the object detector module.

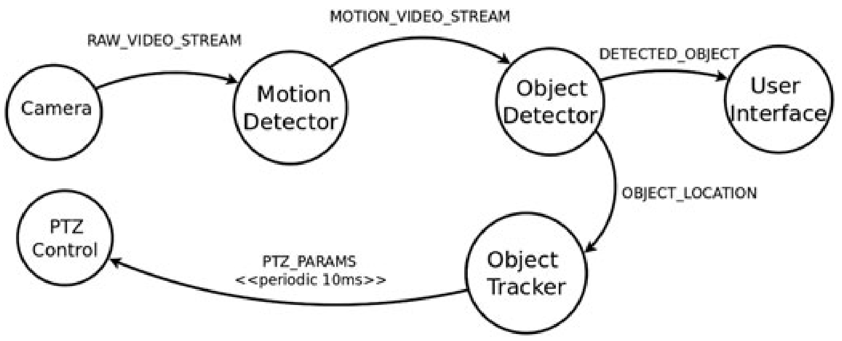


FIGURE 10 Application model of the intelligent surveillance case study

TABLE 4 Description of intermodule edges in the Intelligent Surveillance application

Tuple type	CPU length	N/W length
RAW_ VIDEO_ STREAM	1000	20 000
MOTION_ VIDEO_ STREAM	2000	2000
DETECTED_ OBJECT	500	2000
OBJECT_ LOCATION	1000	100
PTZ_ PARAMS	100	100

TABLE 5 Configuration of sensor for Intelligent Surveillance

CPU length	NW length	Average interarrival time
1000 million instructions	20 000 bytes	5 ms

Similar to the previous case studies the application modules, data dependencies, and the control loop of interest are modeled using AppModule, AppEdge, and AppLoop classes, respectively.

The properties of tuples carried by edges between the modules in the application are described in Table 4.

Physical topology. The physical topology for the second case study is similar to the first case study as described in Section 5.1.1.

Table 5 shows the configuration of the sensors involved in the case study. Here, the cameras that recording live video feeds act as sensors and provide input data to the application. Similar to the previous case study, the physical topology is modeled in iFogSim via FogDevice, Sensor, PhysicalTopology, and Actuator classes. Interested readers can check the examples in iFogSim package for more details on implementation of the case studies.

6 | PERFORMANCE EVALUATION

In this section, we have simulated a Fog computing environment for the application case studies. Then, we evaluated efficiencies of the two placement strategies (ie, cloud-only and edge-ward) in latency, network use, and energy consumption for each case study. Finally, we evaluated scalability of iFogSim in RAM use and execution time for different simulation scenarios.

6.1 | Evaluation of case study 1—a latency-sensitive online game

The simulation of this case study was performed for a period of 3 hours, and the various metrics reported by iFogSim were collected. The results of the simulation demonstrate how different input workloads and placement strategies impacts the network use and end-to-end latency.

Each headset is connected to a smartphone via Bluetooth communication link. Smartphones gain access to the Internet through WiFi gateways that are connected to the ISP gateway. For the purpose of testing iFogSim's performance on varying topology sizes, we have varied the number of WiFi gateways keeping the number of smartphones connected to each gateway constant. Five configurations of physical topology have been simulated—*Config 1, Config 2, Config 3, Config 4, and Config 5*—having

TABLE 6 Description of network links for EEG tractor game

Source	Destination	Latency, ms
EEG headset	Smartphone	6
Smartphone	WiFi gateway	2
WiFi gateway	ISP gateway	4
ISP gateway	Cloud DC	100

1, 2, 4, 8, and 16 WiFi gateways, respectively, with each gateway connected to 4 smartphones playing the EEG tractor beam game. Performance of an application on the fog depends on latencies of the links connecting the fog devices. In the simulation topology, different kinds of devices have different latencies between them, which is listed down in Table 6.

6.1.1 | Average latency of control loop

The most important control loop in the EEG tractor game application—in latency of response—is the loop responsible for transforming the brain-state of the user into his game state on the smartphone's display. This requires real-time communication between the smartphone and the device hosting the brain-state classification module along with efficient processing on the classification module. Lag in this loop will severely mar user experience as it affects entities that the user directly interacts with. Figure 11 illustrates the average delay in execution of this control loop.

Figure 11 shows that control loop execution delay dramatically decreases for Edge-ward placement strategy where fog devices are used for processing. This reduction is even more pronounced when topology sizes and tuple emission rate increases (headset B), because of the increase in congestion in the network that in turn affects delay in response. Results show that computing in the fog not only cuts down response time but also safeguards the application from scalability issues (leading to delayed responses) arising due to large topologies and high data-generation rates.

6.1.2 | Network use

Figure 12 shows the network use of the EEG tractor beam game application. Increase in the number of devices connected to the application significantly increases the load on the network where only cloud resources used. As shown in Figure 12, when fog devices are considered, the network use considerably decreased.

This result can also be interpreted as a demonstration of scalability of fog-based applications. Uncontrolled growth of network use in case of cloud-based execution can lead to network congestion and lead to further degradation of the application's performance. Such situations can be better avoided if fog-based deployment is adopted, and information is preprocessed closer to the source of data. Notably, a large amount of communication in this application takes place between *Client* and *Concentration Calculator* modules. With cloud-only placement, these modules are separated by long-latency links—leading to high degree

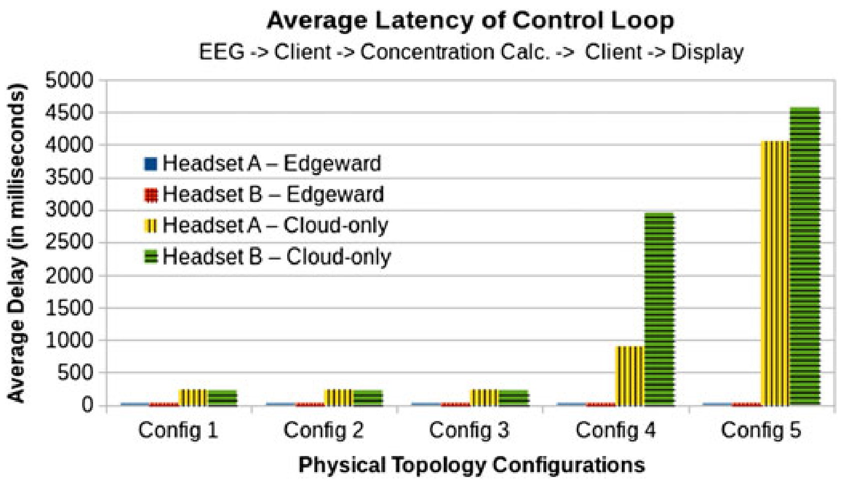


FIGURE 11 Average delay of control loop. [Colour figure can be viewed at wileyonlinelibrary.com]

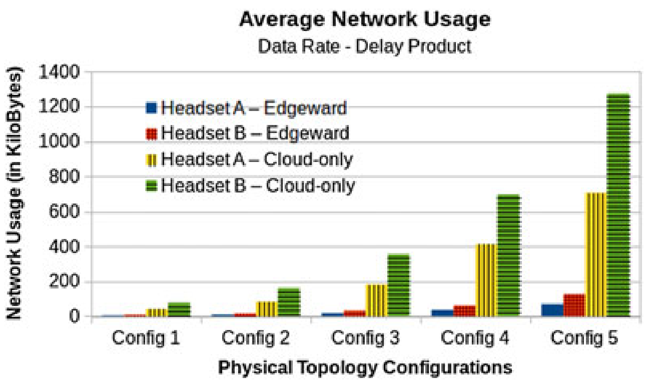


FIGURE 12 Comparison of network use. [Colour figure can be viewed at wileyonlinelibrary.com]

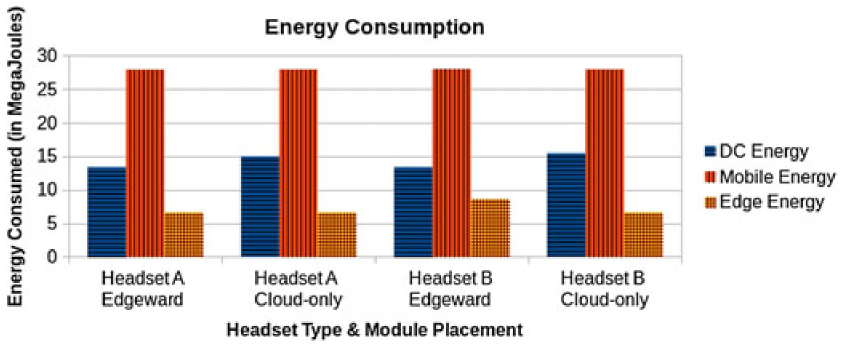


FIGURE 13 Energy consumption of devices in cloud and Fog execution. [Colour figure can be viewed at wileyonlinelibrary.com]

of network use; however, the edge-ward placement places the concentration calculator on the gateways—thereby reducing the effective network use.

As can be seen from Table 3, the frequency and length of tuples emitted by headset B are higher than that of headset A, which explains the increased network use for scenarios using headset B.

6.1.3 | Energy consumption

Figure 13 portrays the energy consumed by different classes of devices in the simulation.

As shown in Figure 13, using fog devices in Edge-ward placement strategy reduces energy consumption of cloud data centers while slightly increases energy consumption of edge devices. A fact worth noting is that the energy consumed by edge devices when modules are placed by edge-ward policy is greater for headset B than headset A. This is because the output characteristics of headset B are such that it emits tuples with greater CPU requirements and at a higher frequency than headset A, thus leading to increased use of edge devices hosting *Concentration Calculator* modules, and causing greater energy consumption.

Mobile devices host only the client module in all scenarios, and hence, their energy consumption is fixed. Note that the values reported in 13 pertains to topology configuration *Config 5*, and the energy consumption of each device type is the sum of energy consumption of all devices belonging to that type.

6.2 | Evaluation of case study 2—intelligent surveillance through distributed camera networks

For demonstrating the flexibility of iFogSim, the Intelligent Surveillance application has been evaluated on a number of physical infrastructure configurations. The number of surveilled areas has been varied from 1 to 16. Note that each surveilled area has 4 smart cameras monitoring the area. These cameras are connected to an area gateway that manages the activity in that surveilled area. In the simulated topologies, each surveilled area has 4 smart cameras connected to an area gateway, which is responsible for providing the Internet access to them. The number of surveilled areas is varied across physical topology configurations Config 1, Config 2, Config 3, Config 4, and Config 5, having 1, 2, 4, 8, and 16 surveilled areas, respectively. The network latencies between devices are listed in Table 7.

TABLE 7 Description of network links in the physical topology for Intelligent Surveillance

Source	Destination	Latency, ms
Camera	Area Switch	2
Area GW	ISP Gateway	2
ISP Gateway	Cloud DC	100

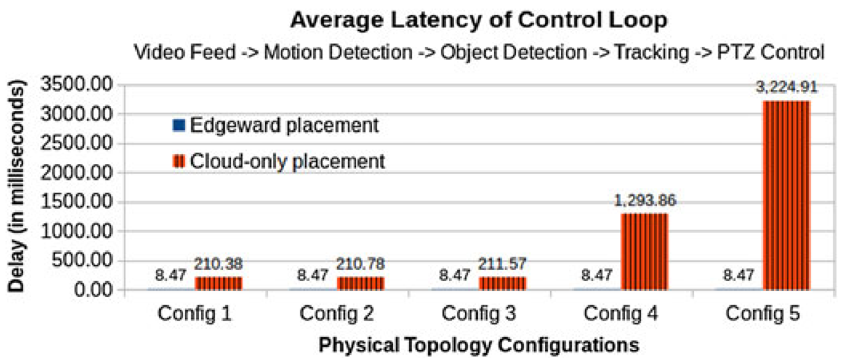


FIGURE 14 Average delay of control loop. [Colour figure can be viewed at wileyonlinelibrary.com]

On the basis of the aforementioned configurations of entities, a physical topology is designed. The topology has the cloud data center at the apex and smart cameras at the edge of the network. Smart cameras are fed live video streams in the form of tuples for performing motion detection and the PTZ control of the camera has been modeled as an actuator. Similarly, two placement strategies, namely, *cloud-only* and *Edge-ward* are used for placing application modules on the physical network. In case of *cloud-only* placement, all operators in the application are placed on the cloud data center except the *Motion Detector* module, which is bound to the smart cameras. However, in the *Edge-ward* placement, the *Object Detector* and *Object Tracker* modules are pushed to WiFi gateways connecting the cameras in a surveilled area to the Internet. The simulation of this case study was performed for a period of 1000 seconds.

6.2.1 | Average latency of control loop

Figure 14 demonstrates the average processing latency of sensing-actuation control loop.

In the case of *cloud-only* placement strategy, as shown in Figure 14, cloud data centers turned to a bottleneck in execution of the modules, which caused a notably significant increase in latency. On the other hand, the *Edge-ward* placement succeeds in maintaining low latency, as it places the modules critical to the control loop close to the network edge.

6.2.2 | Network use

Figure 15 shows the network use of the Intelligent Surveillance application for the placement strategies. As number of devices connected to the application increases, the load on the network increases significantly in the case of cloud-only deployment in contrast with edge-ward deployment.

This observation can be attributed to the fact that in the *Fog-based* execution, most of the data-intensive communication takes place through low-latency links. Hence, modules like Object Detector and Object Tracker are placed on the edge devices, which substantially decrease the volume of data sent to a centralized cloud data center.

6.2.3 | Energy consumption

Figure 16 shows the energy consumed by different category of devices in the simulation. Deployment of application on Fog devices has been compared to deployment only on the cloud data centers. Cameras perform motion detection in the captured video frames, which drains out a large amount of power. Therefore, as shown in Figure 16, when areas under surveillance increases, energy consumption in these devices increases, too. Furthermore, like the previous case study, the energy consumption in the cloud data center decreases when operators are pushed to fog devices.

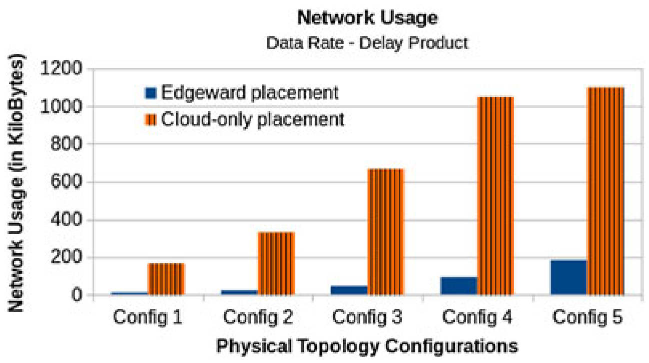


FIGURE 15 Comparison of network use. [Colour figure can be viewed at wileyonlinelibrary.com]

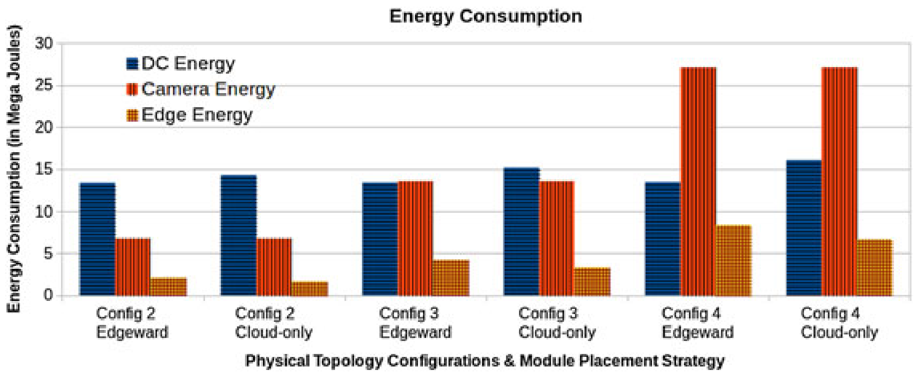


FIGURE 16 Energy consumption of devices in Fog execution. [Colour figure can be viewed at wileyonlinelibrary.com]

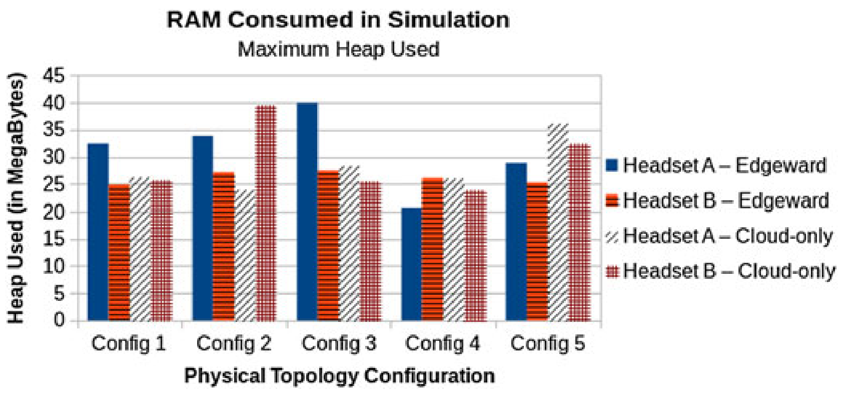


FIGURE 17 The RAM use of simulation for varying sizes of topology and input workload. [Colour figure can be viewed at wileyonlinelibrary.com]

6.3 | iFogSim execution footprint analysis

The scalability of a simulator depends on it resource use (in particular RAM) and the time it takes for simulation to execute. The following metrics have been collected for various sizes of simulation and reported in this section. We have used the configuration of the first case study, and the headset B emits tuples at twice the rate of headset A.

6.3.1 | RAM use

The *massif* heap profiler available in *Valgrind* tool suite[11] was used to measure the heap allocations during simulations of various topology sizes and input workloads.

As depicted in Figure 17, the heap allocation does not increase considerably with increasing workload and physical topology size. The figure shows that *iFogSim* scales with minimal memory overhead when the number of sensors (smart phones) and gateways increases from 4 to 64 and 1 to 16.

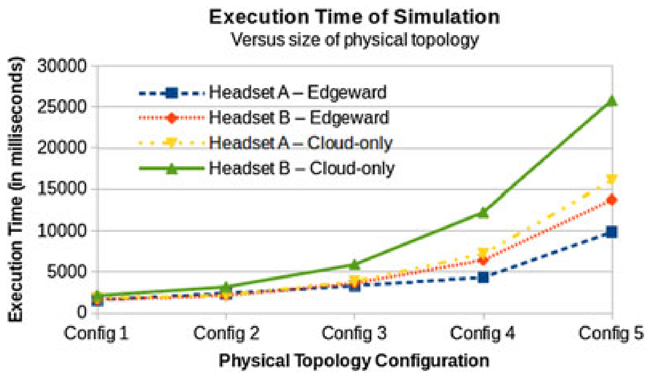


FIGURE 18 Execution time of simulation for varying sizes of topology and input workload. [Colour figure can be viewed at wileyonlinelibrary.com]

6.3.2 | Simulation time

Simulation time for various topologies and input workloads was measured and reported in Figure 18.

Figure 18 shows that the execution time increases when number of devices and transmission rate increases. However, the increase in simulation is almost linear, and consequently, simulation can be executed in an acceptable time (25 s) even if a considerable number of gateways are added.

7 | RELATED WORK

In IoT environments,[12] to enable real-time decision making,[13] distributed stream-processing systems have to push query operators to nodes located closer to the source of data. To this end, Cisco introduced fog computing.[3] Cisco's offering for fog computing, known as IOx,[14] is a combination of networking operating system, IOS, and the most popular open-source operating system, Linux. Ruggedized routers running Cisco IOx make compute and storage available to applications hosted in a *Guest Operating System* running on a hypervisor alongside the IOS virtual machine. Cisco provides an app store that allows users to download applications to IOx devices and an app-management console for controlling and monitoring the performance of an application. Using device abstractions provided by Cisco IOx APIs, applications running on the fog can communicate with IoT devices that use verities of protocols. In iFogSim also we have considered the similar concepts of apps and devices to model a fog environment and controllers, which act similar to app-management consoles in Cisco environments.

The FIT IoT-LAB[15] is a testbed equipped with thousands of wireless nodes located in 6 different sites across France. It allows users to evaluate and test their novel ideas ranging from low-level protocols to advanced analytic and services in a very large-scale wireless IoT environment. Major services offered by IoT-LAB include the following: (1) remote access to sensors and gateways: the testbed provides users with APIs to flash any firmware, design, build, and compile applications; (2) access to the serial ports of reserved IoT devices; (3) Internet access for nodes with end-to-end IP connection using IPv6 and 6LoWPAN; (4) power consumption monitoring per device; and (5) robots to test and improve real-time decision making in IoT context.

SmartSantander[16] is a project of the Future Internet Research and Experimentation initiative of the European Commission. It uniquely offers a city-scale experimental research facility with support of services of a smart city. The testbed comprises a large number of IoT devices deployed in several urban locations mainly in Santander City. SmartSantander has conceived a 3-tiered architecture as follows: (1) IoT nodes, responsible for sensing the environment parameter such as temperature and noise; 2) repeaters, these nodes are placed between sensors and gateways, to behave as forwarding nodes; and (3) gateways, IoT nodes and repeaters are configured to send all captured data via 802.15.4 protocol to gateways.

Simulators are essential tools during the design of IoT systems. A real IoT testbed —which can be built using Cisco solutions or combination of open-source solutions such as FIT IoT Lab—although desirable, in many cases is too costly and does not provide repeatable and controllable environment. Therefore, simulation can be considered as cost-effective first step before real experimentation to eliminate ineffective policies and strategies.

The WSNet[17] is an event-driven simulator for wireless networks, which can as well be used for IoT. It is capable of simulating nodes with different energy sources, mobility models, radio interfaces, applications, and routing protocols. Environment simulation is also supported by WSNet; in fact, it offers the opportunity for modeling and simulation of physical phenomena (eg, fire) and physical measures (eg, temperature and humidity). These values (eg, temperature) can be observed by the nodes and can also impact nodes.

The SimpleIoTSimulator[18] is a commercial simulator for creating IoT environments consisting of many sensors and gateway. The SimpleIoTSimulator supports common IoT protocols including CoAP and MQTT as a publish/subscribe-based protocol. The SimpleIoTSimulator objective is enabling IoT platform and gateway vendors to improve product quality with the focus on communication protocols. Our simulators also models publish/subscribe-based protocols; however, our focus is on the analysis of application design and resource management policies. In addition, the SimpleIoTSimulator dose not model fog environments where services can be deployed both on edge and cloud resources.

Because traditional wireless sensor networks and IoT simulators do not focus on modeling of large-scale deployments, Giacomo et al[19] proposed a simulation methodology for IoT systems with a large number of interconnected devices. It is designed to study low-level networking aspects. In summary, the main advantages of their approach are (1) simulation of IoT systems with geographically distributed devices and (2) simulation of are IoT devices with multiple network interfaces and protocols, as well as different mobility, network, and energy consumption models.

The OASIS standard *Devices Profile for Web Services* (DPWS) aims at enabling the deployment of web services on constrained devices. To accelerate the development of DPWS enabled applications, Han et al proposed DPWSim,[20] a simulation toolkit that allows developers to design, develop, and test service-based IoT applications using the DPWS technology without the presence of physical sensors and gateways.

CloudSim is developed as an extensible cloud simulation toolkit that enables modeling and simulation of cloud systems and application provisioning environments.[6] This toolkit provides both system and behavior modeling of cloud computing components such as virtual machines (VMs), data centers, and users. However, CloudSim and other cloud environment simulators such as GloudSim,[21] DCSim,[22] and GroudSim[23] do not model IoT devices and stream-processing applications.

Sarkar et al[24] have performed a rigorous analysis comparing the performance of fog and cloud computing paradigms in the context of the IoT by mathematically formulating the parameters and characteristics of fog computing. Their experiments show that fog computing paradigm outperforms cloud computing as the fraction of applications demanding real-time response increases. However, in scenarios where many applications do not have real-time response requirements, fog computing is observed to be an overhead compared to cloud-based execution. However, a mathematical model of a system lacks the dynamic nature of the environment, wherein applications can be migrated to and from the cloud on the basis of traffic surge. Furthermore, it is difficult to reproduce the experiments performed by the authors for extending the study and incorporating custom applications and resource management policies.

In summary, although there are few simulators that model IoT environments, iFogSim is uniquely designed and implemented to model fog environment along with IoT and cloud. This enables innovation and performance evaluation of resource management policies for IoT applications such as real-time stream processing in a comprehensive end-to-end environment.

8 | CONCLUSIONS AND FUTURE DIRECTIONS

Fog and Edge computing are emerging as an attractive solutions to the problem of data processing in the IoT. Rather than outsourcing all operations to cloud, they also use devices on the edge of the network that have more processing power than the end devices and are closer to sensors, thus reducing latency and network congestion. In this paper, we introduced iFogSim to model and simulate IoT, Fog, and Edge computing environments. In particular, iFogSim allows investigation and comparison of resource management techniques on the basis of QoS criteria such as latency under different workloads (tuple size and transmit rate). We described two case studies and demonstrated effectiveness of iFogSim for evaluating resource management techniques including cloud-only application module placement and a technique that pushes applications towards edge devices when enough resources are available. Moreover, the scalability of simulation is verified. Our experiment results demonstrated that iFogSim is capable of supporting simulations on the scale expected in the context of IoT. We also believe that the availability of our simulator will energize rapid development of innovative resource management policies in the areas of IoT and fog computing with end-to-end modeling and simulation.

There are a number of future directions that can enhance iFogSim capabilities and resource management strategies in the context of IoT:

- **Support for mobility**. A large fraction of devices used today are mobile, with Gartner predicting that smartphones would be the Internet access device of choice by 2018.[25] Support for mobility, a key characteristic of fog computing, will become a necessity for users of the Internet. Mobility support, although not included in current version of iFogSim, can be incorporated by based on works like Ottenwälder et al[26] that study complex-event processing with mobile users. From the implementation perspective, sensors and actuators will need to be extended so as to make them location-aware, fog devices should take suitable actions (like migration of application state) in the event of migration of a connected device. Similarly, fog applications will have to be made mobility aware by adding event handlers that react to change in location of connected endpoints.

- **Power-aware resource management policies**. One of the biggest challenges that most of fog computing solutions face is how to get extra bit of battery life for fog devices. To this end, future studies can look into new policies that dynamically and on the basis of the battery life of devices migrate the operators. Questions such as which operator to migrate, when to migrate, and where to migrate need to be addressed in by these policies.
- **Priority-aware resource management strategies for multitenant environments**. Looking into scheduling policies for an environment where multiple application instances (DAGs of operators) share the same pool of resources and are assigned different service level objectives is another promising research direction.
- **Modeling failures of fog devices**. Future research can focus on extracting failure models for the dominant failures in IoT and fog devices. The developed models can be used to evaluate and compare reliability-aware scheduling and recovery policies for a wide range of applications.
- **Dynamic priority and SLA aware flow placement and resource scheduling (joint Edge-network resource optimization)**. In IoT environments, heterogeneous network and sensing resources have to be often shared with multiple applications or services with different and dynamic quality of service requirements. Therefore, the joint Edge-network resource scheduling problem is another problem that we are going to investigate.
- **Modeling and comparison different virtualization techniques of IoT environments**. Future research studies can also consider and compare the performance of full virtualization, para-virtualization (as instances of hardware-level virtualization), and operating system level virtualization such as containers.
- **Scheduling of application modules**. The current version of iFogSim assumes that a fog application module is responsible for execution of data generated from all the devices under its coverage. In that case, if a module is moved north, it will not receive input data as they would be handled by the lower modules. As part of future work, we will extend the design to allow more flexible scheduling.

9 | SOFTWARE AVAILABILITY

iFogSim is available for download from the CLOUDS Lab website: http://www.cloudbus.org/cloudsim.

ACKNOWLEDGEMENTS

This work is (i) supported by Melbourne-Chindia Cloud Computing Research Network and (ii) initiated as part of the first author's visit to CLOUDS Lab at the University of Melbourne. It is also partially supported by the ARC Future Fellowship. We thank Redowan Mahmud for his contributions towards improving both the iFogSim simulator and this paper. We also thank other external users of our iFogSim for their feedback and reviewers for their comments on improving the quality of our work.

REFERENCES

1. Institute MG. Unlocking the potential of the Internet of Things. http://www.mckinsey.com/business-functions/business-technology/our-insights/the-internet-of-things-the-value-of-digitizing-the-physical-world
2. Chang C, Srirama SN, Mass J. A middleware for discovering proximity-based service-oriented industrial internet of things. In: *2015 IEEE International Conference on Services Computing (SCC)*, IEEE, New York, NY, USA, 2015;130-137.
3. Bonomi F, Milito R, Natarajan P, Zhu J. 2014. Fog computing: a platform for internet of things and analytics. In: *Big Data and Internet of Things: A Roadmap for Smart Environments* Springer; 169-186.
4. Yousfi A, Bauer C, Saidi R, Dey AK. UBPMN: a BPMN extension for modeling ubiquitous business processes. *Inf Softw Technol*. 2016;74:55-68.
5. Giang NK, Blackstock M, Lea R, Leung V. Developing IoT applications in the fog: a distributed dataflow approach. In: *2015 5th International Conference on the Internet of Things (IOT)*, IEEE, Seoul, South Korea, 2015;155-162.
6. Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw Pract Experience*. 2011;41(1):23-50.
7. Bittencourt LF, Diaz-Montes J, Buyya R, Rana OF, Parashar M. Mobility-aware application scheduling in fog computing. *IEEE Cloud Comput*. April, 2017 March;4(2):34-43.
8. Zao JK, Gan TT, You CK, et al. Augmented brain computer interaction based on fog computing and linked data. In: *2014 International Conference on Intelligent Environments (IE)*, IEEE, Shanghai Jiao Tong University China, 2014;374-377.

9. Guérout T, Monteil T, Da Costa G, Calheiros RN, Buyya R, Alexandru M. Energy-aware simulation with DVFS. *Simul Modell Pract Theory*. 2013;39:76-91.
10. Hong K, Lillethun D, Ramachandran U, Ottenwälder B, Koldehofe B. Mobile fog: a programming model for large-scale applications on the Internet of Things. In *Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing*, ACM, Hong Kong, China, 2013;15-20.
11. Seward J, Nethercote N, Fitzhardinge J. Valgrind, an open-source memory debugger for x86-GNU/Linux, 2004. http://www.ukuug.org/events/linux2002/papers/html/valgrind
12. Gubbi J, Buyya R, Marusic S, Palaniswami M. Internet of Things (IoT): a vision, architectural elements, and future directions. *Future Gener Comput Syst*. 2013;29(7):1645-1660.
13. Stojmenovic I, Wen S, Huang X, Luan H. An overview of fog computing and its security issues. *Concurrency Comput Pract Experience*. 2016;28(10):2991-3005.
14. Iox overview. https://developer.cisco.com/site/iox/documents/developer-guide/?ref=overview (Accessed on 05/11/2016).
15. Adjih C, Baccelli E, Fleury E, Harter G, Mitton N, Noel T, Pissard-Gibollet R, Saint-Marcel F, Schreiner G, Vandaele J, Watteyne T. Fit IoT-Lab: a large scale open experimental iot testbed. In: *Proceedings of the 2nd IEEE World Forum on Internet of Things (WF-IoT)*, Milan, Italy, 2015;459-464.
16. Sanchez L, Muñoz L, Galache JA, et al. Smartsantander: IoT experimentation over a smart city testbed. *Comput Netw*. 2014;61:217-238.
17. Chelius G, Fraboulet A, Fleury E. WSNET: a modular event-driven wireless network simulator, 2006.
18. Simpleiotsimulator: the internetofthings simulator. http://www.smplsft.com/SimpleIoTSimulator.html (Accessed on 05/11/2016).
19. Brambilla G, Picone M, Cirani S, Amoretti M, Zanichelli F. A simulation platform for large-scale internet of things scenarios in urban environments. In *Proceedings of the First International Conference on Iot in Urban Space*, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Rome, Italy, 2014;50-55.
20. Han SN, Lee GM, Crespi N, et al. DPWSIM: a simulation toolkit for iot applications using devices profile for web services. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, Seoul, South Korea, 2014;544-547.
21. Di S, Cappello F. GloudSim: Google trace based cloud simulator with virtual machines. *Softw Pract Experience*. 2015;45(11):1571-1590
22. Tighe M, Keller G, Bauer M, Lutfiyya H. DCSim: a data centre simulation tool for evaluating dynamic virtualized resource management. In: *Proceedings of the 2012 8th International Conference on Network and Service Management (CNSM) and 2012 Workshop on Systems Virtualiztion Management (SVM)*, Las Vegas, NV, USA, 2012;385-392.
23. Ostermann S, Plankensteiner K, Prodan R, Fahringer T. 2011. GroudSim: an event-based simulation framework for computational grids and clouds. In: *Proceedings of the 2010 Conference on Parallel Processing*, Euro-Par 2010 Springer-Verlag: Berlin, Heidelberg; 305-313.
24. Sarkar S, Chatterjee S, Misra S. Assessment of the suitability of fog computing in the context of Internet of Things, 2015.
25. Institute MG. Gartner says by 2018, more than 50 percent of users will use a tablet or smartphone first for all online activities. http://www.gartner.com/newsroom/id/2939217
26. Ottenwälder B, Koldehofe B, Rothermel K, Hong K, Lillethun D, Ramachandran U. Mcep: a mobility-aware complex event processing system. *ACM Trans Internet Technol (TOIT)*. 2014;14(1):6.

How to cite this article: 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. *Softw Pract Exper*. 2017;47:1275–1296. https://doi.org/10.1002/spe.2509