iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments

Harshit Gupta * †, Amir Vahid Dastjerdi *, Soumya K. Ghosh†, and Rajkumar Buyya *

SUMMARY

Internet of Things (IoT) aims to bring every object (e.g. smart cameras, wearable, environmental sensors, home appliances, and vehicles) online, hence generating massive amounts 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 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 a 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 terms of 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 in terms of RAM consumption and execution time is verified under different circumstances.

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 type of interactions among things and humans and enables the realization of smart cities, infrastructures, and services for enhancing the quality of life and utilization of resources. It supports integration, transfer, and analytics of data generated by smart devices (e.g. sensors). IoT envisions a new world of connected devices and humans in which quality of life is enhanced, because management of city and its infrastructure is less cumbersome, health services are conveniently accessible, and disaster

∗H. Gupta, A.V Dastjerdi, and R Buyya are with Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Australia.
Email: {amir.vahid,rbuyya}@unimelb.edu.au,
†H. Gupta and S.K. Ghosh are with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India.
Email: harshitgupta1337@gmail.com, skg@iitkgp.ac.in

recovery is more efficient. Based on 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 percent of the world economy. They also expect one 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 the data to cloud for 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, e.g. gateways) [1] has lead to the evolution of the Fog computing paradigm. Fog computing [2] extends cloud services to the edge of networks, which results in latency reduction through geographical distribution of IoT application components, and provides support for mobile mobility.
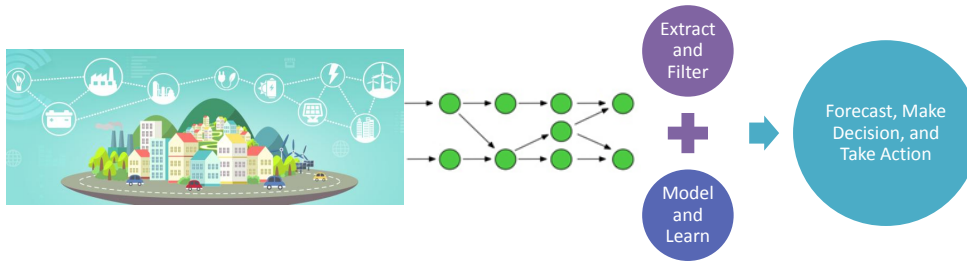

Figure 1. IoT Environment Common Objectives.

Many IoT applications (e.g. stream processing) are naturally distributed and are often embedded in an environment with numerous connected computing devices with heterogeneous capabilities. As data travels from its point of origin (e.g. 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. The main challenge lies in scheduling application components (operators in terms of stream processing or independent short-lived IoT tasks) in the pool of devices — 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.

Resource management policies to ensure Quality of Service (QoS), avoid energy wastage, and resource fragmentation are an integral part of IoT systems. To foster innovation and development enabling real-time analytics in Fog computing, we require a an evaluation environment for exploring different resource management and scheduling techniques 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. 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. In such models, sensors publish data to IoT networks, 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 terms of 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 usage. Finally, Section 8 concludes the paper and discusses the future directions.

2. FOG COMPUTING — DEFINITION AND CONCEPTS

We define Fog computing as a distributed computing paradigm that extends the services provided by the cloud to the edge of the network. It enables seamless leveraging of cloud and edge resources along with its own infrastructure (see Figure 2). It facilitates management and programming of compute, networking and storage services between data centers and end devices. Fog computing essentially involves components of an application running both in the cloud as well as in devices between endpoints and the cloud, i.e. smart gateways and routers. Fog computing supports mobility, resource and interface heterogeneity, interplay with the cloud, and distributed data analytics to addresses requirements of applications that need low latency with a wide and dense geographical distribution. Fog computing takes advantages of both edge and cloud computing — 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 Fog computing that assures its success. The first benefit is the reduction of network traffic as uncontrolled increase in network traffic may lead to
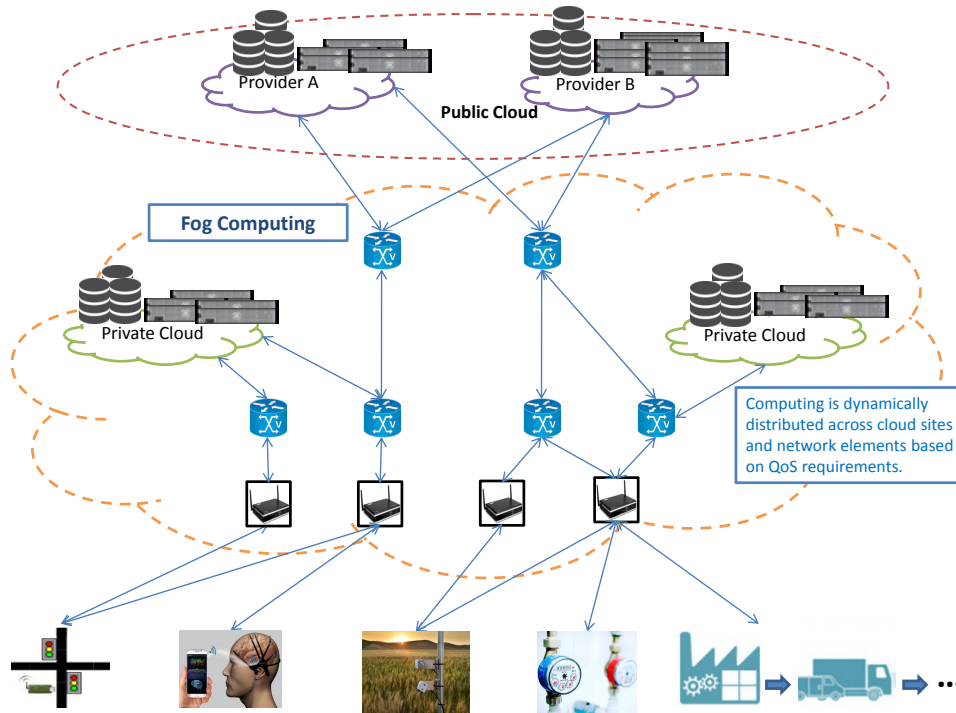

Figure 2. Distributed Data Processing in a Fog Computing Environment.

congestion, and results in increased latency. Fog computing provides a platform for filtering and analysis of the data generated by sensors by utilizing resources of edge devices. This drastically reduces the traffic being sent to the cloud by allowing the placement of filtering operators close to the source of data. Considerable reduction in propagation latency is the next important advantage of utilizing 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 anti-lock 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 as a result 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's 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 as presented in Figure 3 involves a hierarchical arrangement of Fog nodes throughout the network between sensors and the cloud at the core of the network. In the architecture, **IoT sensors** are placed at the bottommost layer of the architecture and distributed in different geographical locations, sensing the environment, and emitting observed values to upper layers via gateways for further processing and filtering. 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 captured by
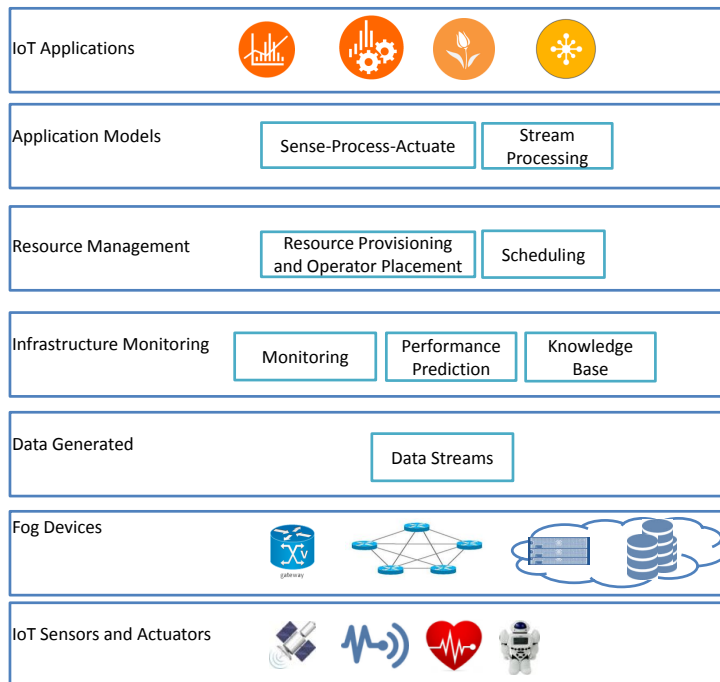

Figure 3. Fog Computing Architecture.

sensors. **IoT Data Streams** are made of a sequence of immutable values emitted by sensors. In the architecture any element in the network that is capable of hosting application modules is called **Fog Device**. Fog devices that connect sensors to the Internet are generally called gateways. Fog devices also include cloud resources that are provisioned on-demand from geographically distributed data centers.

In addition, the architecture defines three main services and two application models for Fog and IoT environments that are described below.

Monitoring components keep track of the resource utilization and availability of sensors, actuators, Fog devices and network elements. They keep track of the applications and services deployed on the infrastructure by monitoring their performance and status. Monitoring components supply this information to other services as required.

Resource management is the core component of the architecture and consists of components that coherently manage resources 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.

Power monitoring: One of the toughest challenges that most IoT solutions face is utilization of resources of IoT nodes while considering constraints on energy consumption. In contrast to cloud data centers, Fog computing encompasses a large number of devices with heterogeneous power consumption, making energy management difficult to achieve. Hence, evaluating the impact of applications and resource management policies on energy consumption is crucial before deployment in production environments. Therefore, we require a power monitoring component in the architecture that is responsible for monitoring and reporting the energy consumption of Fog devices in the simulation.

Application (programming) models: The applications developed for deployment in the Fog are based on the Distributed Data Flow (DDF) model [3]. An application is modelled 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.

The architecture supports two models used for IoT applications:

1. Sense-Process-Actuate Model: The information collected by sensors is emitted 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.

4. DESIGN AND IMPLEMENTATION

For implementing functionalities of iFogSim architecture, we leveraged basic event simulation functionalities found in CloudSim[4]. 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.
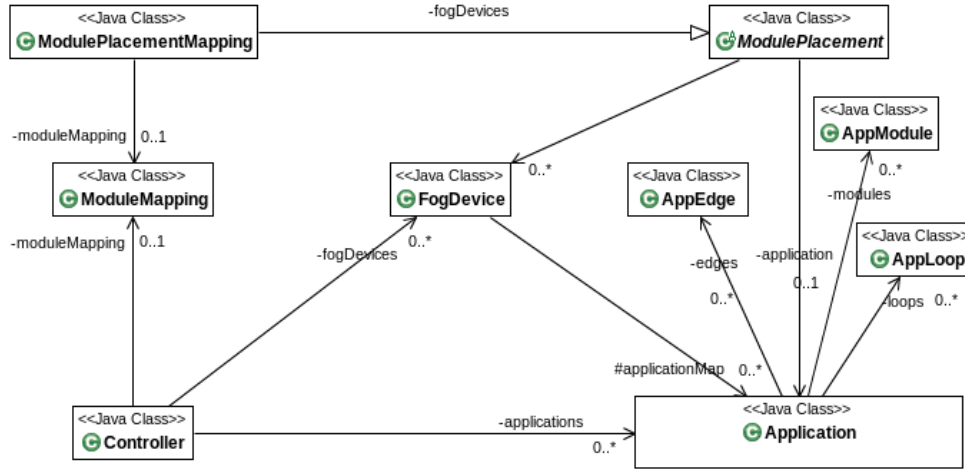
Figure 4. Main classes of iFogSim.

- **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 abovementioned 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 inter-transmission or inter-arrival time - which identifies the tuple arrival rate at the gateway.
- **Tuple:** Tuples form the fundamental unit of communication between entities in the Fog. 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.
- **Actuator:** This class models an actuator by defining its network connection properties. An attribute in the class refers to the gateway to which the actuator is connected and the latency
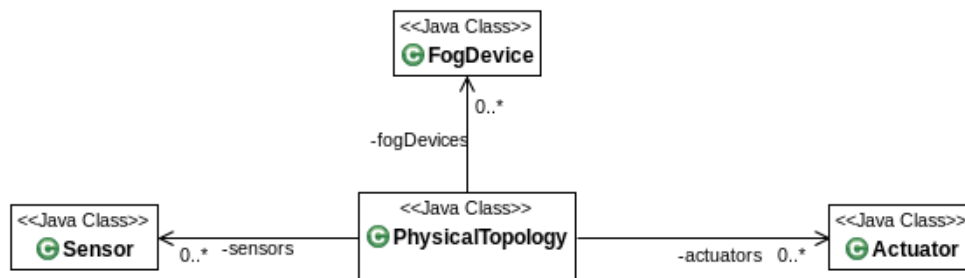

Figure 5. iFogSim physical topology classes.

of this connection. The class defines a method to perform an action on arrival of a tuple from an application module.

- **Application:** An application is modeled as a directed graph, the vertices of the 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. *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 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.
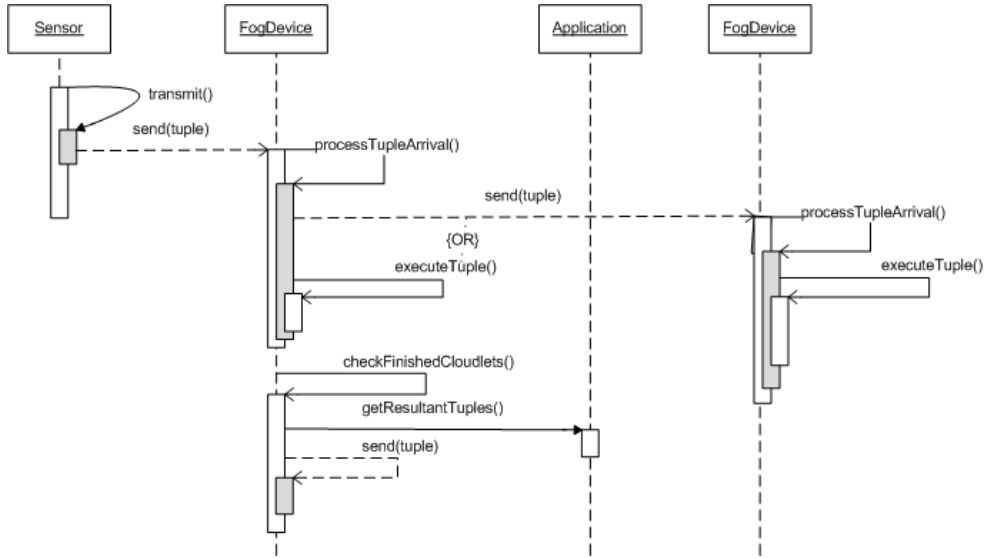

Figure 6. Sequence diagram of the generation and execution.

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:

- **Power Monitoring Service:** Each Fog device (a FogDevice instance) is associated with a power model (e.g. PowerModelLinear). The *executeTuple()* method in the *FogDevice* class contains the tuple processing logic where the related power model is used to update the device power consumption based on the changes in the resource utilization.
- **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 usage, 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. Built-in Module Placement Strategies

iFogSim is packaged with two application module placement strategies — *cloud-only placement* and *edge-ward placement*.

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 favours 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.

4.2. Graphical User Interface

In order 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 re-loaded 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.

4.3. 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

Algorithm 1: Edge-ward module placement

---

for $p \in PATHS$ **do** *Across all paths*
  $placeList := \{\}$;
  for *Fog device* $d \in p$ **do** leaf-to-root traversal
    for *module* $w \in app$ **do**
      if *all predecessors of* w *are placed* **then**
        add w to $placeList$;
      end
    end
    for *module* $\theta \in placeList$ **do**
      if θ *is already placed on device* $f \in p$ **then**
        Merge θ with its upstream instance ;
        $f :=$ device holding merged instance ;
        while $CPU_\theta^{req} \geq CPU_f^{avail}$ **do**
          $f := parent(f)$;
        end
        Place θ on device f ;
      end
      else if $CPU_\theta^{req} \leq CPU_d^{avail}$ **then**
        Place θ on device d ;
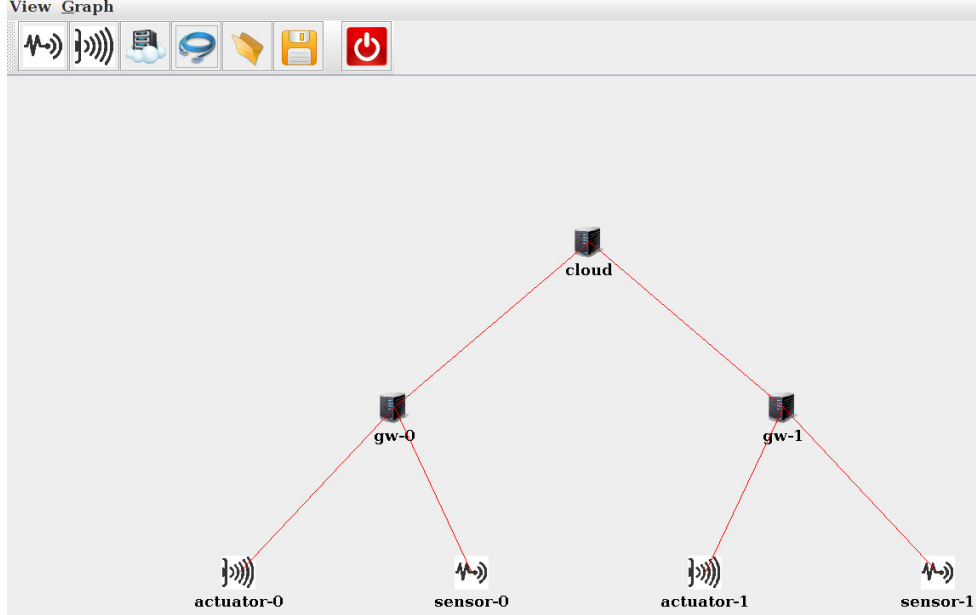   **end**
    end
  end
end

Figure 7. iFogSim GUI for building network topology.

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). Next step in modeling workload is identifying how much resources

```
{"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.

    are required to process the tuples. For defining resource utilization 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 directed acyclic graph (DAG), and built via three 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 two 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*) [5] that involves augmented brain-computer interaction. In order to play the *EEG Tractor Beam game*, each player needs to wear a 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). In order 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 three major modules which 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, 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:

Table I. Description of inter-module 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

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*.
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.
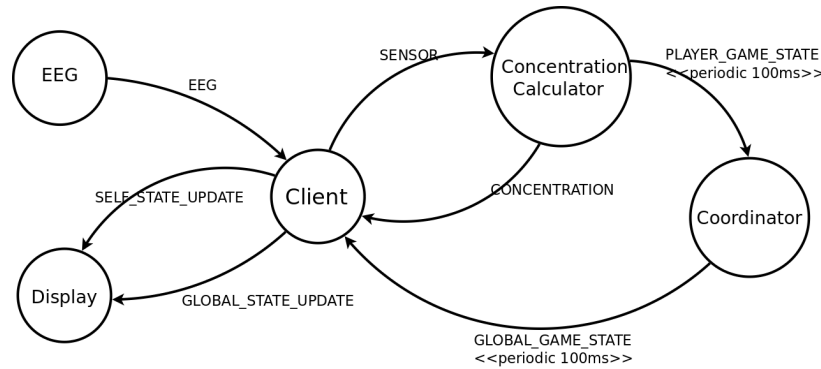

Figure 9. Application model of the EEG game.

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

5.1.1. Physical Network For the case study, we have considered a physical topology with 4 Fog devices. Table II illustrates the configurations [6] 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 III. 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

Table II. 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)

Table III. Configuration of sensors for EEG Tractor Game

HEADSET	TUPLE CPU LENGTH	Average Inter-arrival Time
A	2000 Million Instructions	10 milliseconds
B	2500 Million Instructions	5 milliseconds

of public safety and security, manufacturing, transportation, and healthcare. However, monitoring video streams from the system of cameras manually is not practical. Hence we need tools 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 based on 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 of time, 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 would also consume mote 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 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 five major modules which perform processing — *Motion Detector*, *Object Detector*, *Object Tracker*, *PTZ Control* and *User Interface*. The 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:

Table IV. Description of inter-module edges in the Intelligent Surveillance application

Tuple Type	CPU Length	N/W Length
RAW_ VIDEO_ STREAM	1000	20000
MOTION_ VIDEO_ STREAM	2000	2000
DETECTED_ OBJECT	500	2000
OBJECT_ LOCATION	1000	100
PTZ_ PARAMS	100	100

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.

Similar to the previous case studies The application modules , data dependencies, the control loop are modeled using AppModule, AppEdge, and AppLoop classes respectively. Finally, t of interest for EEG application is modeled in iFogsim using class.
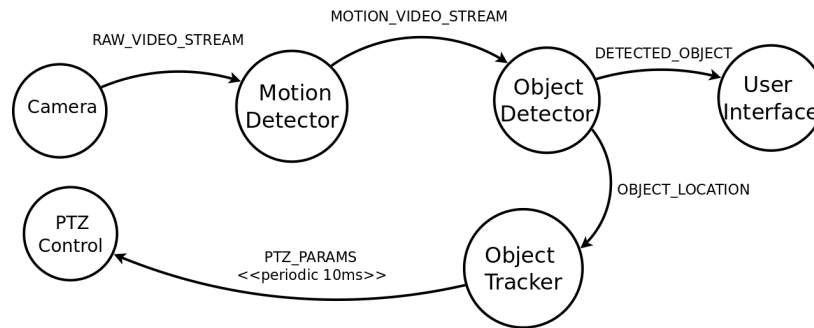

Figure 10. Application Model of the Intelligent Surveillance Case Study.

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

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 V 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

Table V. Configuration of sensor for Intelligent Surveillance

CPU Length	NW Length	Average Inter-arrival Time
1000 Million Instructions	20000 bytes	5 milliseconds

Table VI. Description of network links for EEG Tractor Game

Source	Destination	Latency (in ms)
EEG Headset	Smartphone	6
Smartphone	WiFi Gateway	2
WiFi Gateway	ISP Gateway	4
ISP Gateway	Cloud DC	100

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 (i.e. cloud-only and edge-ward) in terms of latency, network usage, and energy consumption for each case study. Finally, we evaluated scalability of iFogSim in terms of RAM usage 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 carried out 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 usage 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 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 VI.

6.1.1. Average Latency of Control Loop.
The most important control loop in the EEG Tractor game application — in terms of 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 utilized for processing. This reduction is even more pronounced when topology sizes and tuple emission rate increases (Headset B).
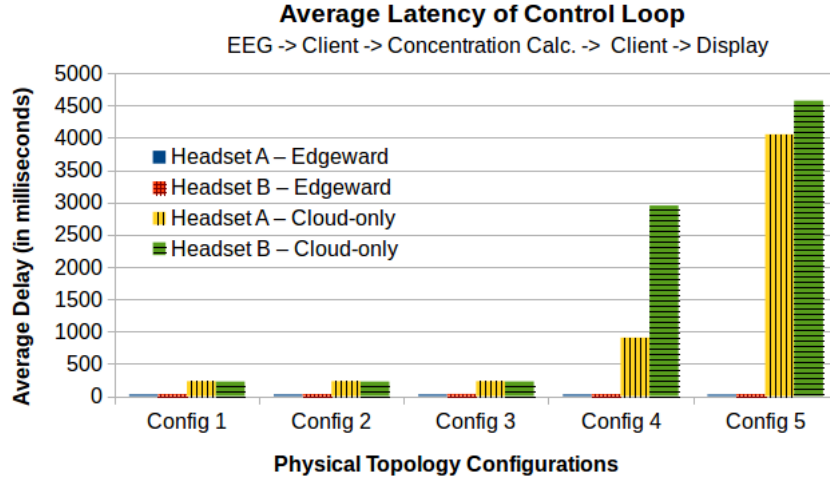
Average Latency of Control Loop

EEG -> Client -> Concentration Calc. -> Client -> Display

Figure 11. Average delay of control loop.

6.1.2. Network Usage. Figure 12 shows the network usage 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 Figure 12 shows, when Fog devices are considered, the network usage considerably decreased.

Average Network Usage
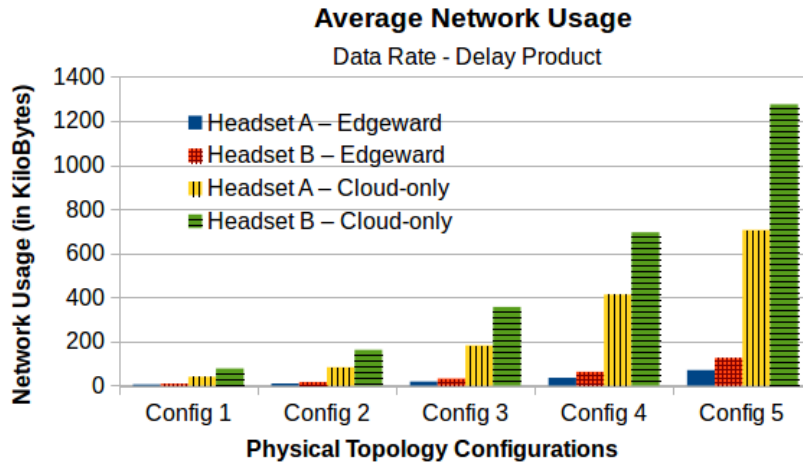
Data Rate - Delay Product

Figure 12. Comparison of network usage.

This result can also be interpreted as a demonstration of scalability of Fog-based applications. Uncontrolled growth of network usage 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.

6.1.3. Energy Consumption. Figure 13 portrays the energy consumed by different classes of devices in the simulation. As Figure 13 shows, utilizing Fog devices in Edge-ward placement strategy reduces energy consumption of cloud data centers while slightly increases energy consumption of edge devices.
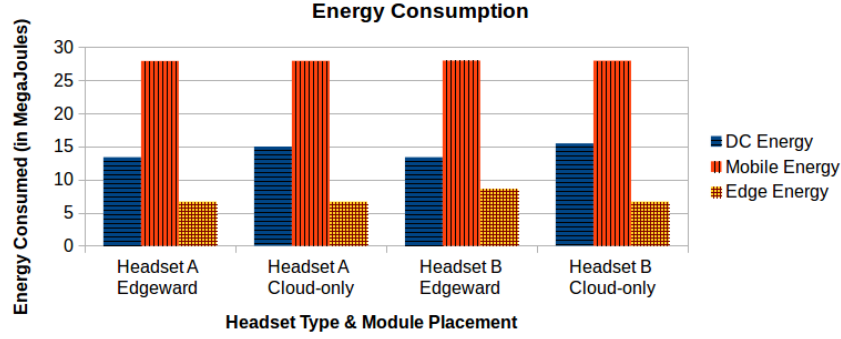
Figure 13. Energy consumption of devices in cloud and Fog execution.

Table VII. 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

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 four 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 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 VII.

Based on 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 carried out 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 Figure 14 shows, 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.
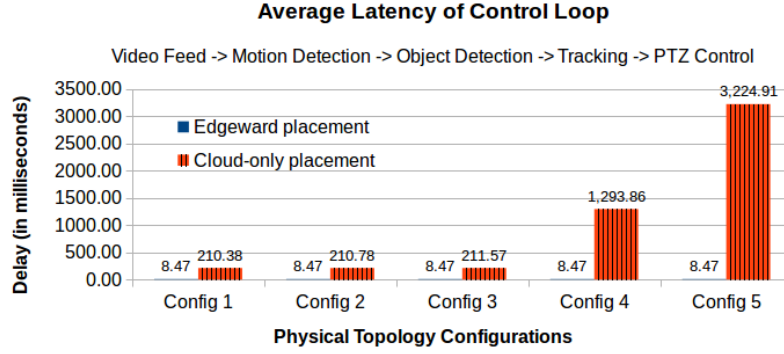
Figure 14. Average delay of control loop.

6.2.2. Network Usage. Figure 15 shows the network usage 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 to edge-ward deployment.
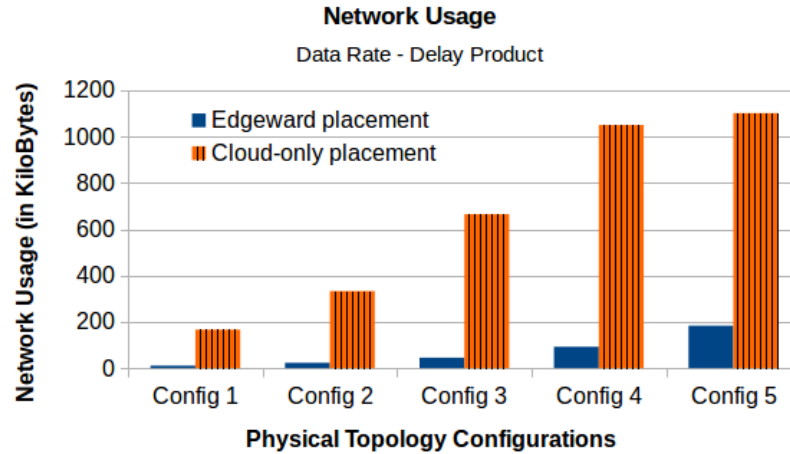

Figure 15. Comparison of network usage.

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 Figure 16 shows, when areas under surveillance increases, energy consumption in these devices increase too. Furthermore, like the previous case study, the energy consumption in the cloud data center decreases when operators are pushed to Fog devices.
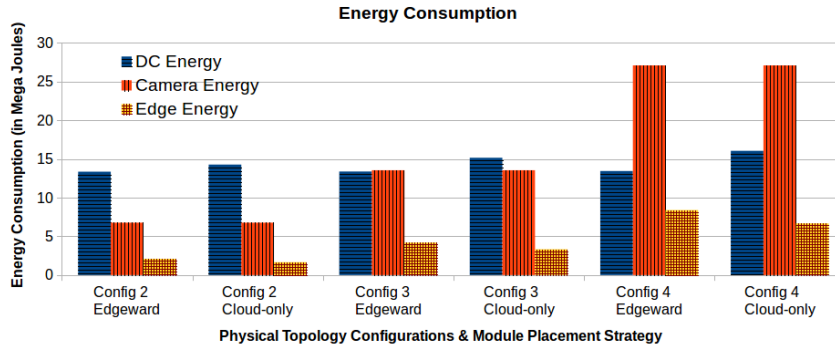
Figure 16. Energy consumption of devices in Fog execution.

6.3. iFogSim Execution Footprint Analysis

The scalability of a simulator depends on it resource usage (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 Usage. The *massif* heap profiler available in *Valgrind* tool suite [7] was used to measure the heap allocations during simulations of various topology sizes and input workloads. As
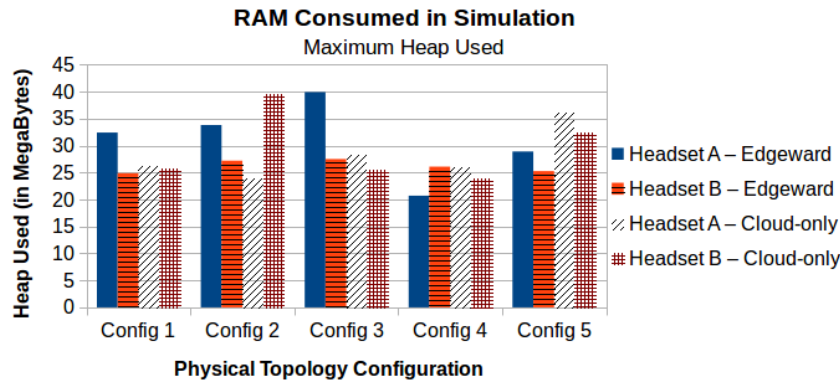

Figure 17. RAM usage of simulation for varying sizes of topology and input workload.

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.

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 seconds) even if a considerable number of gateways is added.
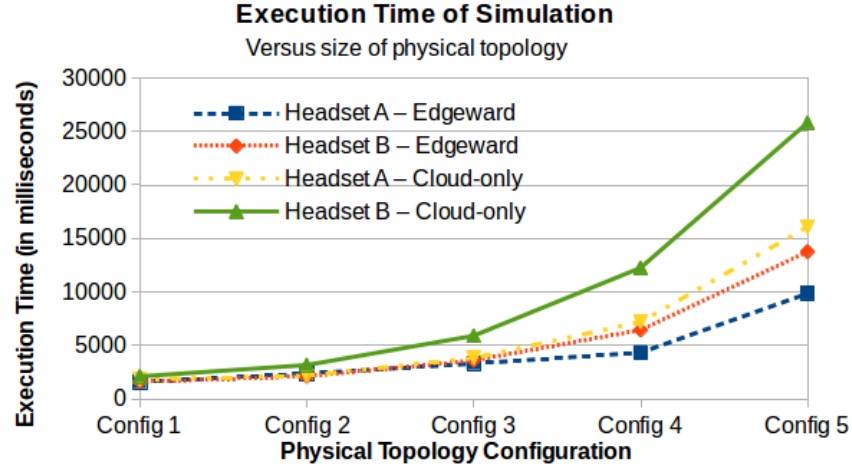
Figure 18. Execution time of simulation for varying sizes of topology and input workload.

7. RELATED WORK

In IoT environments [8], in order to enable real-time decision making [9], 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 [2]. Cisco's offering for Fog computing, known as IOx [10], 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 which 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 [11] is a testbed equipped with thousands of wireless nodes located in six 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: 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; 5) and robots to test and improve real-time decision making in IoT context.

SmartSantander [12] 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 Internet of Things 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, in order 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.

WSNet [13] 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 modelling and simulation of physical phenomena (e.g. fire) and physical measures (e.g. temperature, humidity). These values (e.g. temperature) can be observed by the nodes, and can also impact nodes.

SimpleIoTSimulator [14] is a commercial simulator for creating IoT environments consisting of many sensors and gateway. SimpleIoTSimulator supports common IoT protocols including CoAP and MQTT as a publish/subscribe based protocol. 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.

Since traditional wireless sensor networks and IoT simulators do not focus on modeling of large scale deployments, Giacomo et al. [15] 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; 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 [16], 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 [17]. This toolkit provides both system and behaviour modeling of cloud computing components such as virtual machines (VMs), data centers, and users. However, CloudSim and other cloud environment simulators such as GloudSim [18], DCSim [19], and GroudSim [20] do not model IoT devices and stream processing applications.

In summary, although there 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 Internet of Things. Rather than outsourcing all operations to cloud, they also utilize 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 based on 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 techniques that pushes applications towards edge devices when enough resources are available. Moreover, 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:

- **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 based on 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 multi-tenant 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 (SLO) 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.

SOFTWARE AVAILABILITY

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

ACKNOWLEDGEMENTS

REFERENCES

1. Chang C, Srirama SN, Mass J. A middleware for discovering proximity-based service-oriented industrial internet of things. *Services Computing (SCC), 2015 IEEE International Conference on*, IEEE, 2015; 130–137.
2. Bonomi F, Milito R, Natarajan P, Zhu J. Fog computing: A platform for internet of things and analytics. *Big Data and Internet of Things: A Roadmap for Smart Environments*. Springer, 2014; 169–186.
3. Giang NK, Blackstock M, Lea R, Leung V. Developing iot applications in the fog: a distributed dataflow approach. *Internet of Things (IOT), 2015 5th International Conference on the*, IEEE, 2015; 155–162.
4. 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. *Software: Practice and Experience* 2011; **41**(1):23–50.
5. Zao JK, Gan TT, You CK, Rodriguez Mendez SJ, Chung CE, Te Wang Y, Mullen T, Jung TP. Augmented brain computer interaction based on fog computing and linked data. *Intelligent Environments (IE), 2014 International Conference on*, IEEE, 2014; 374–377.
6. Guérout T, Monteil T, Da Costa G, Calheiros RN, Buyya R, Alexandru M. Energy-aware simulation with dvfs. *Simulation Modelling Practice and Theory* 2013; **39**:76–91.
7. Seward J, Nethercote N, Fitzhardinge J. Valgrind, an open-source memory debugger for x86-gnu/linux. *URL: http://www. ukuug. org/events/linux2002/papers/html/valgrind* 2004; .

8. Gubbi J, Buyya R, Marusic S, Palaniswami M. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems* 2013; **29**(7):1645–1660.
9. Stojmenovic I, Wen S, Huang X, Luan H. An overview of fog computing and its security issues. *Concurrency and Computation: Practice and Experience* 2015; .
10. Iox overview. `https://developer.cisco.com/site/iox/documents/developer-guide/?ref=overview`. (Accessed on 05/11/2016).
11. Adjih C, Baccelli E, Fleury E, Harter G, Mitton N, Noel T, Pissard-Gibollet R, Saint-Marcel F, Schreiner G, Vandaele J, *et al.*. Fit iot-lab: A large scale open experimental iot testbed. *Proceedings of the 2nd IEEE World Forum on Internet of Things (WF-IoT)*, 2015.
12. Sanchez L, Muñoz L, Galache JA, Sotres P, Santana JR, Gutierrez V, Ramdhany R, Gluhak A, Krco S, Theodoridis E, *et al.*. Smartsantander: Iot experimentation over a smart city testbed. *Computer Networks* 2014; **61**:217–238.
13. Chelius G, Fraboulet A, Fleury E. Wsnet: a modular event-driven wireless network simulator 2006; .
14. Simpleiotsimulator: The internetofthings simulator. `http://www.smplsft.com/SimpleIoTSimulator.html`. (Accessed on 05/11/2016).
15. Brambilla G, Picone M, Cirani S, Amoretti M, Zanichelli F. A simulation platform for large-scale internet of things scenarios in urban environments. *Proceedings of the First International Conference on IoT in Urban Space*, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014; 50–55.
16. Han SN, Lee GM, Crespi N, Luong NV, Heo K, Brut M, Gatellier P. Dpwsim: A simulation toolkit for iot applications using devices profile for web services. *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, 2014; 544–547.
17. 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. *Software: Practice and Experience* 2011; **41**(1):23–50.
18. Di S, Cappello F. GloudSim: google trace based cloud simulator with virtual machines. *Software: Practice and Experience* 2014; .
19. Tighe M, Keller G, Bauer M, Lutfiyya H. DCSim: a data centre simulation tool for evaluating dynamic virtualized resource management. *Proceedings of the 2012 8th international conference on Network and service management (cnsm) and 2012 workshop on systems virtualiztion management (svm)*, 2012; 385–392.
20. Ostermann S, Plankensteiner K, Prodan R, Fahringer T. GroudSim: an event-based simulation framework for computational grids and clouds. *Proceedings of the 2010 Conference on Parallel Processing*, Euro-Par 2010, Springer-Verlag: Berlin, Heidelberg, 2011; 305–313.